x

GCE Computer Science (7517)

The Practical Project

|  |  |  |
| --- | --- | --- |
| Centre number |  | Centre name |
| 22133 | The Perse School |

|  |  |  |
| --- | --- | --- |
| Candidate’s full name |  | Candidate number |
| Sebastian Rouse | 1249 |

|  |
| --- |
| Project title |
| Digital Scrabble |

Contents

[*1* Analysis 3](#_Toc126583535)

[1.1 Background to/ Identification of the problem 3](#_Toc126583536)

[1.2 Research carried out 3](#_Toc126583537)

[1.3 Identification of the prospective user(s) 6](#_Toc126583538)

[1.4 Modelling diagrams 9](#_Toc126583539)

[1.5 Numbered measurable, appropriate specific objectives of the project 11](#_Toc126583540)

[2 Design 14](#_Toc126583541)

[2.1 System design overview 14](#_Toc126583542)

[2.2 Data structures 23](#_Toc126583543)

[2.3 Algorithms 32](#_Toc126583544)

[2.4 File structure and organisation 42](#_Toc126583545)

[2.5 Database design 44](#_Toc126583546)

[2.6 SQL queries 47](#_Toc126583547)

[2.7 User interface design 51](#_Toc126583548)

[2.8 User guide 57](#_Toc126583549)

[*3* Technical Solution 58](#_Toc126583550)

[3.1 Completeness Section 58](#_Toc126583551)

[3.2 Technical Skills Section 59](#_Toc126583552)

[3.2.1 Overview Guide 59](#_Toc126583553)

[3.2.2 Code listing 63](#_Toc126583554)

[*4* System testing 127](#_Toc126583555)

[4.1 Test plan 127](#_Toc126583556)

[4.2 Testing Videos 133](#_Toc126583557)

[5 Evaluation*)* 134](#_Toc126583558)

[5.1 Comparison of project performance against the objectives 134](#_Toc126583559)

[5.2 Effectiveness of the solution 134](#_Toc126583560)

[5.3 Analysis of user feedback 134](#_Toc126583561)

[5.4 Possible improvements 135](#_Toc126583562)

[6 Appendix 137](#_Toc126583563)

[6.1 Git log 137](#_Toc126583564)

# Analysis

## Background to/ Identification of the problem

Scrabble is a word game in which 2–4 people compete against each other by scoring points by placing tiles which contain a single letter onto a 15x15 game board. The tiles must form a word reading that read across (from left to right) or down. The score for each turn is the sum of the letter values for each word. The winner is the player with the most points.

One main problem with scrabble is carrying around the board and the pieces is very cumbersome meaning that it makes a chore if you want to play scrabble anywhere outside of home. This problem can easily be fixed by making a computer version of scrabble. The portability of a laptop eliminates any prior portability issues with the game and means that you can play the game anywhere you want.

Another issue with scrabble is that, due to it being a board game, you can’t resume games that you didn’t finish earlier (unless you write down where each piece was on the board, which is extremely inconvenient and cumbersome. This can by having a computer version of scrabble that has a “save game” feature that allows you to save the state of an ongoing game and then be able to return to it at a later date.



## Research carried out

As part of my background work, I had a look at various existing software that allowed users to play Scrabble.

**Scrabble Online**

A picture containing scatter chart

Description automatically generated

This is a version of Scrabble from https://www.scrabblegames.info/play/scrabble-online . This version of the game does not have an option to explain the rules at all and what each star on the board means is not clear making the game unsuitable for beginners who do not know the rules of the game. It’s also very limited in terms of gameplay as you can only play a 1 vs 1 game against an AI meaning you cannot play local multiplayer against a friend or be able to play against multiple opponents. The game does not have any login functionality meaning players cannot keep track of their progress or replay matches.

On the positive side the game has selectable difficulty levels making it suitable to players of all skill levels. The interface is also clear and responsive and shows all relevant data to the users. The game also displays the words played and their definitions allowing you to learn new words to use for Scrabble.

**Internet Scrabble Club**

Chart

Description automatically generated

This is a version of Scrabble from <https://www.isc.ro/>. This version also doesn’t explain the rules of scrabble however has a help section where you can “ask a helper” for help. This version is very limited in terms of game modes as you can only play online multiplayer scrabble vs one other person. In this version each player has an overall time limit to make their moves making it so games don’t drag on. This version has a login system used to record your “skill rating” as well as your game history. This version also has a customizable UI that allows you to change the colour of the board and pieces and it also allows you to play scrabble in different languages. This version also has an chat feature allowing you to communicate whit other players. The user interface can be a bit confusing at times and is not easy to navigate. For example you can only view your statistics by clicking on your profile photo when in a game which is a bit cumbersome and it’s never made clear that you can do this meaning many players might play this version without realising that it records their statistics. The points each player has while playing aren’t displayed in a clear way and what the different coloured squares on the board mean are also never explained making it confusing for beginners trying to learn the game

**Play Scrabble**

Graphical user interface

Description automatically generated with low confidence

This version of scrabble is from <https://playscrabble.com/>. This version also does not explain the rules however it is much more suitable for beginners as it has an online forum where you can get help from other players, has a section where it lists all the words out in the scrabble dictionary as well as having a “word finder” which will find valid words based on the tiles you have which is useful for beginners who are struggling with forming words based on their tiles. This version also allows you to play scrabble in multiple languages. The UI is very clear and easy to navigate. This version also has a login system and record your game history and statistics. This version also allows you to play online or against the computer. The computer has multiple selectable difficulties making it accessible to multiple skill levels. The main draw back from this version is that you can only play against one opponent as well as lacking local multiplayer.

All of these versions of Scrabble have very visual and interactive user interfaces which is something that I liked and definitely would be something I would like to have in my implementation of the game. (Objectives 28-56). I especially liked how all these versions allow the user to drag and drop the tiles and is something I would plan on implementing (Objective 51). The log in/accounts features were good in some versions however lacking in others and I feel could be improved upon (Objectives 22-26). Some of the version also implement an AI that the user could play against. This is useful if you don’t want to play online against another player. The versions that implemented a playable AI allowed user to adjust the difficulty of the AI meaning that it caters for players of all skill levels(Objectives 19-21). What was lacking in all of these version though, was the ability to play against another human player on the same computer (Objectives 1-17)

## Identification of the prospective user(s)

**Primary user – Zakk:**

*-Uh, hi Zakk.. Thanks for coming to this interview.*

*-You're welcome. Seb, anytime.*

*-Uh, so, uh, have you played scrabble before?*

*-Yeah, I'm actually quite a big Scrabble fan. I play quite a lot.*

*-Uh, so like with the board version of the game, like what problems do you have when they're playing Scrabble?*

*-So I don't always have enough time to finish.*

*-So I would like to be able to like, save it and play over a few sessions.*

*-We can definitely help with that. Any more problems?*

*- carrying the board or the pieces around is a little bit of a pain. I'd much prefer to play on the laptop if that's possible.*

*-We can help with that. So who do you often play Scrabble with?*

*-So I generally play against my friends and family, although I think it'd be interesting to be able to play against people that I don't know, because I don't have this option when I'm playing physically.*

*-Okay. Yeah. Uh, do you normally like win against these people or, you know, like how often do you want to lose?*

*-Oh yeah. I mean, I'd say, I'd say I'm all right. I win some games but also lose quite a few*

*-Do you ever keep track of you wins and losses?*

*-Well, I would like to, but I just find it kind of inconvenient. So, I definitely like to keep track of some kind of record wins and losses against*

*-Do you ever find that playing a game of Scrabble takes too long?*

*-Yeah, I often find that I don’t have enough time to finish the game.*

*-Would you like a feature that allows you to save games that you can’t finish and return to them later to finish them?*

*-Definitely, that would be amazing.*

*-When playing like games on your computer with friends do you prefer it across one computer or, would you like it across multiple computers?*

*-Well I think it's all right to have it on one computer. It would be nice to play across multiple computers, but it doesn't really matter too much because we tend to be in the same room.*

*-Yeah. Yeah. Do you often play games against an AI? Does it interest you to play against an AI opponent?*

*-Yes. I love playing games against AI. I've never played Scrabble against AI, but it would allow me to play games when I've got no one to play with and kind of practice how good I am.*

*-Yeah. How good do you think the AI should be?*

*-Well, again, this there's got to be a balance. Like if I'm, if I'm a beginner, I wouldn't want to get destroyed every time. But at the same time, if I'm an expert, I'd find it boring. If they, I wasn't very good.*

*-So it would definitely have to be a bit of a mix a, how would you like it to look? Do you like the look of current scrabble?*

*-Like the board game version of it? Oh, I do like the classic Scrabble work. That's. I mean, it would be -nice to be able to customize it, but that's definitely not something that's super important*

*-And just that interest, like, are you bilingual?*

*-Um, I can speak Spanish.*

*-Uh, have you ever played Scrabble in Spanish?*

*-I've not because it's not really available in terms of like the right dictionaries in facilities for that. I would definitely like to, if that's a possibility.*

*-Okay.*

*-Thank you.*

**Secondary User:**

*I really like playing Scrabble as I think it helps improve my vocabulary. Not just in English (my native language) but it has also helped me learn other languages such as Spanish. Of course, a full Scrabble board is quite cumbersome and it is easy to lose pieces, but I often find myself wanting to play (or continue) a quick game when I am on the train or bus. It would be nice if I could resume a game I was playing earlier, but it might also be nice if there was a way to play a game that would last less time – maybe there is a way of changing to rules to have a quick, 10 minute game? Sometimes I find myself wanting to play a game of Scrabble but have no one to play with at that current time so a playable AI that I could play against would be nice. Sometimes I’ve played with people who have really become stuck with the letters they have, so perhaps some form of ‘hint’ feature could be implemented to offer a small amount of help to these people? That could be something as simple as stepping back one or more moves because they realise they made a silly mistake*.

**Conclusions form User feedback:**

Both prospective users provided useful feedback on the project and I have taken their feedback into consideration when outlining the objectives in the section below. Both users seemed very enthusiastic about a digital version of Scrabble as it would solve a lot of the pre-existing they have with the game such as the pieces and the board being a hassle to carry around. (Objectives 1 -17) Both users mentioned how they sometimes struggle to finish a game of Scrabble I believe this problem can be fixed by introducing a log in system where the user can log in to an account.(Objectives 22-26) Once logged into the account, while playing a game, the users should be able save the current state of that game (to a database) and exit the game and then be able to return to that game and finish whenever they want if they are logged in.(objectives 22-26, 50,45) The primary user also mentioned about how a feature that keeps track of the wins and losses in a game would be nice which I believe could also be implemented through a log in system, where the game will keep track of the games each account wins and losses. Both users also mentioned they would like it if there was an AI that the players could play against and the primary user mentioned how it would be useful if the AI had multiple difficulty levels. (Objectives 19 - 21)Both users also mentioned how it would be nice if there was a possibility to play Scrabble in another language.(Objectives 47,49) The secondary user also mentioned the ability to perhaps undo moves. I think that this would unbalance the game a bit too much and wouldn’t really work for scrabble however it did give me the idea of an undo action that is more to help navigate the UI by undoing any drag and drop actions made during a turn. (Objectives 48 and 55 )

## Modelling diagrams

**GUI Mock-up:**

This is a diagram showing the layout of the different windows in the User interface and the flow diagram showing how each window interact and open other windows

Diagram, schematic

Description automatically generated

**Initial Class Relationship Diagram:**

This is a diagram showing the initial design for the classes in the code

Diagram

Description automatically generated

## Numbered measurable, appropriate specific objectives of the project

**MVP – Terminal Mode**

1. The program must be runnable with a ‘t’ option to run in the terminal or ‘g’ option to run in a GUI. Running without one of these should report an error
2. The program should be able to access and open the text file containing the “scrabble dictionary” which contains all valid words
3. When running in the terminal mode a 15 x 15 scrabble board should be printed to the terminal
4. At the start of the game 7 letters should be printed and assigned to a player
5. Each player should be asked to input their moves each turn
6. If the player tries to use a tile that is not on his rack then an error should be reported
7. Each new word formed each turn should be searched for in the dictionary. If the word is not in the dictionary an error should be reported and the player should have redo his turn
8. A player should be able to pass their turn
9. Once a player has played his turn, the player should be assigned more letters so that he has 7 letters
10. The number of points obtained by the player that turn should be added to their total.
11. Each turn the board should be displayed and the number of points each player has should be displayed
12. Once the bag is empty no more tiles should be assigned to players.
13. Once a player has used his last letter/tile the game should end
14. The game should end if both players pass on consecutive turns
15. After the game has ended the sum of each players unplayed letters should be deducted from their respective scores, if a player used all of their letters the sum of the other player’s unplayed letters should be added to the total score
16. The winner of the game (the player with most points) should be reported
17. In case of 2 or more players have the same number of points, the player with the most points before adding or subtracting points due to unplayed letters should be declared as the winner

**Hashing**

1. A hashing function should be implemented where it is able to return a hash of a given string

**AI**

1. An AI should be able to be added to the game in place of a player. This a AI should be able to make moves
2. The AI should calculate all the possible moves and find the best ones.
3. The AI should have 5 different difficulty settings

**Login**

1. Users’ Usernames and Passwords should be stored in an external database
2. The password should be hashed before entering the database
3. Another table should store the state of saved games as well as which users played them
4. New users usernames and passwords should be inserted into the database
5. The number of wins and losses by the user should be tracked

**Langauge**

1. The game should be playable in both English and Spanish

**GUI**

1. When run with the ‘g’ command line option, the program should present a GUI with buttons to display ‘Play Game’, ‘Account’ ,‘Create Account’, ’Login’ or ‘Quit’
2. Pressing the ‘Quit’ button should cause the GUI window to disappear and the program to end.
3. Pressing the ‘Login’ button should open window with 2 entry boxes asking the user to enter their username and password, a ‘Login’ button and a ‘Quit button
4. Pressing the ‘Quit button should close the Login window
5. Pressing the login button should cause the program to validate the username and password entered
6. If the username and password is valid the Login window should close and cause the user to be logged in
7. If the username and password is invalid an error message should be displayed
8. Pressing the ‘Create Account’ button should open window with 2 entry boxes asking the user to enter their username and password, a ‘Create Account’ button and a ‘Dismiss’ button
9. Pressing the ‘Quit’ button should close the Create Account window
10. Pressing the Create Account button should cause the program to validate the username entered
11. If the username is already in the database an error message should be displayed
12. Otherwise a new record should be created in the database with the username and password entered by the user
13. Pressing the ‘Account’ button while the user is not logged in should cause a new window to open displaying the message ‘Previous games cannot be viewed unless you are logged in’ as well as a dismiss button
14. Pressing the ‘Account’ should open a window displaying the user’s username and stats of the user as well as 2 buttons labelled “Load Games” and “Quit”
15. Clicking the Quit button should close the window
16. Clicking the “Load Games” button should open a window display all of the uses saved games as buttons as well as a button labelled “Quit” at the bottom of the window
17. Clicking the Quit button should close the window
18. Clicking on one of the previous games should open a new window causing that game to resume
19. Pressing the Dismiss button should cause the Previous Games window to close
20. Pressing The ‘Play Game’ button should open a window with 2 buttons on it that say “English” and “Spanish” and should prompt the User to select langauge
21. Clicking either of should cause the ‘Setup Game’ window to disappear and a new window to open displaying 15x15 scrabble grid, a ‘Save Game and Quit’ button, a ‘Next Turn’ button, a ’Pass’ button ,an ‘Undo’ button ,a 7x1 grid displaying the letters/tiles assigned to the player and the number point each player has
22. Clicking either of those language buttons should make it so the game is played in that language
23. Pressing the ‘Save Game and Quit’ should cause the game to be saved and the game window to be closed
24. The letters/tiles should be able to be dragged and dropped onto the board
25. Pressing the next turn button should cause the next turn to occur
26. The points total displayed should be updated after each turn
27. Pressing the Pass button should cause the next turn to occur without that player making any moves
28. Pressing the Undo button should cause the last drag and drop action to be undone
29. When a player has won the game a new window should appear stating that that player has won

# Design

## System design overview

**Flow Diagram**

This is the flow diagram showing the logic behind the actual Scrabble game, showing how the Scrabble game flows

**Diagram

Description automatically generated**

**Class Diagram:**

This is a class diagram showing the all the classes as well as the relationship between the classes in the program

Diagram

Description automatically generated

**Classes**

**Game**

**Text

Description automatically generated**The game class is responsible for the main logic and storing the state of the scrabble game. The board property stores the current state of the board as a 2D array and the boardPoints property stores where the double letter score, triple letter score, double word score and triple word score multipliers are. The number of turns is stored in the NoOfTurn property and which player’s turn it is, is stored in the pTurn property. The tiles/letters that are left in the bag is stored in the currBag property and the pointsDict property stores the values of each letter/tile in a dictionary. The game class also has an instance of the Dictionary class in dict property. This allows the game to set and switch language which it does in the updateLanguage method. The players property is an array containing instances of the Players class. This is so the game class can have access to player’s rack and points so it can update them when the player makes a move. It does this with the calculatePoints and the updatePlayerRack methods. The language method stores what language the game is currently being played in. This is so that it can be passed into the database when the game is saved so that when that game is reloaded, it knows which language it is being played in. The user property stores an instance of the Account class. This instance of the account class is passed into the Game class when it is instantiated. The Game class has an instance of the Account class so that it knows whether or not a user is logged so that it knows whether to save a game. The SaveGame method will format and pass data to the account class so that it can be saved to the database. The LoadGame method will take data about that has been taken from the database (and passed through the account class) and update all of the properties with that data. This allows games to be saved mid game and then reloaded and finished later. The isGameOver method checks whether the game has finished or not and the findWinnner method finds the winner of the game. The validateTurn method will take a set of moves (stored in a 2D array) as a parameter and will return True or False whether that move is valid according to the laws of the game.

**Players**

The main purpose of the Players class is to hold all the game information relating to the players playing the game. The attribute Players.\_\_points stores the number of points that player has, Players.\_\_rack is an array of length 7 which stores the tiles that the player has in their rack and Players.\_\_preEndPoints stores the number of points the player had before the points from the left over tiles on the player’s rack had been taken away from their points total. Players.\_\_preEndPoints is used if the number of points the players’ have ends the game in a draw so the points each player had before points were taken away is used as the tie break. The updateRack, getRack(), updatePoints(), getPoints, updatePreEndPoints() and getPreEndPoints() functions, either get or update the points, rack or preEndPoints attributes respectively

**Text

Description automatically generated**

**Dictionary**

**Text

Description automatically generated**The Dictionary class stores all of the data relating to the langauge that the game is currently being played in. The Dictionary.\_\_Dawg attribute stores an instance of the Dawg class. This instance of the Dawg class relates to/stores the Dawg for the list of valid words for the language currently that the game is being played in. The Dictionary.\_bag attribute stores a list of all the letters/tiles that would be in a full bag for the language the game is being played in. Dictionary.\_\_pointsDict stores a dictionary which stores how many points each letter/tile is worth for the language that the game I being played in. Dictionary.\_\_languages stores the languages in a dictionary. The name of the language is the key and an instance of the class relating to that language is the value. The getBag() and getPointsDict() functions returns Dictionary.\_bag and Dictionary.\_\_pointsDict respectively. updateLanguage() takes a string as a parameter and updates the language (by updating the bag, pointsDict and Dawg) to the one specified by the string. search() allows a word to be searched in the Dawg and will return True if the word is in the Dawg and False if it’s not. CreateDawg() creates a Dawg for the language that the game is currently being played in. It does this by creating an instance of the Dawg class and then inserting the words line by line(which are from a text file where file name is stored in the class relating to the language the game is being currently played in) into the Dawg by using the Dawg.insert() function

**Languages**

**Text

Description automatically generated**

The languages class is the base class for the English and Spanish classes. It has 3 attributes: bag, pointsDict and filename. These are all set to None in the languages class. The returnBag(), returnPointsDict() and returnFilename() methods all return their respective property respectively.

**English**

**Text

Description automatically generated**

The English class stores all the data relating to the English version of scrabble. The bag attribute stores the tiles/letters in the full version of the bag in the English version of scrabble. The pointsDict attribute stores the points each letter/tile is worth and the filename attribute stores the name of the filename where all the valid English scrabble words are stored. The English class is also a subclass of the Languages class so it has the Languages.returnBag(), Languages.returnPointsDict() and languages.returnFilename() functions

**Spanish**

**A picture containing text

Description automatically generated**

Like the English class, the Spanish class stores all the data relating to the Spanish version of scrabble. The bag attribute stores the tiles/letters in the full version of the bag in the Spanish version of scrabble. The pointsDict attribute stores the points each letter/tile is worth and the filename attribute stores the name of the filename where all the valid Spanish scrabble words are stored. The Spanish class is also a subclass of the Languages class so it has the Languages.returnBag(), Languages.returnPointsDict() and languages.returnFilename() functions

**DawgNode**

**Text

Description automatically generated**

DawgNode stores an individual node of the DAWG. The ID attribute is used as a way to identify each node. The EndOfWord attribute is a boolean which says whether that node represents the end of a word or not. The Children attribute is a dictionary that stores all the nodes that are child nodes of that nodes. A letter value (which represents the letter value of an edge of the graph/DAWG) is used as the key while another instance of the DawgNode class is the value to that key. This is how the program represents the Dawg as each node points to other nodes which point to more nodes building up a whole graph

Text

Description automatically generated**Dawg**

This class that manages and represents the DAWG. The root attribute is an instance of the DawgNode class and is the root node which means that that node is a child of no other node and that starting from that node you will be able to reach all the other nodes in the graph. The previousWord attribute stores the previous word inserted into the DAWG. The uncheckedNodes attribute stores all the nodes that have not been checked whether they can be minimised or not and the minimisedNodes attribute stores all the nodes that have been checked whether they can be minimised. The insert() function will insert a specified word into the DAWG (more detail on how that is done is in the data structures and key algorithms section). The minimize function will check all nodes, going back from the last node of the word just inserted into the DAWG to the node where the last inserted word and the previous word before that stop sharing a prefix and sees if there is any node within the graph that shares the same attributes as that node. It will then change the edge pointing to that node to the node already within the graph that has the same attributes. The finish() function calls the minimize function where the prefix length is 0 as the last word inserted into the DAWG will have not been minimized so the program so the program would need to minimize all the nodes from the last node of the word going back to the root. The search() function will search the DAWG for a given word and return True if the word is in the DAWG or False if it’s not.

**Account**

**Text

Description automatically generated**

The account class’s main purpose is to store the username of the user that is currently logged in. This is stored in the username property. If no user is logged in the username property will equal None. The database property stores an instance of the Database class. The Account class also provides a level of abstraction from the Database class to the rest of the code. The getUsername method will return the value of the username property. The other methods relate to methods in the Database class. In these methods the account class will modify any of the data that goes to/from the database so that it will be in a format that the database/rest of the program will be able to process. The login method will log the user in and update the username property if the username and password of the user are correct.

**Database**

**Text

Description automatically generated**The Database class is the class responsible for managing and inserting and extracting data from the database. This class is where all the SQL statements are. In the Sha256 property it also has an instance of the Sha256 class. This is so that the database can hash the password of the user before it is entered. The PasswordHash function will take a given password and return the hashed version of it by calling the HashAndDigest method in Sha256 class. In the classes \_\_init\_\_ the CreateTables method is called. This method will create the database and the tables in the database if they do not already exist. This ensures that the database and the tables do exist when an instance of this class is created which ensures that data isn’t being passed into a database/tables that do not already exist. The CreateAccount method will create an account user given that their username isn’t already in use by some other account. The Login method will check if a given username and password hash are in the database and will return True or False accordingly. The GetWinLoss method will get the number of wins, losses and games played a given user has. The SaveGame and LoadGame method will respectively and insert data about a scrabble game or get data about a specific scrabble game from the database. This allows users to save games and return to them later to finish them.

**Sha256**

Table

Description automatically generated with medium confidence

The Sha256 class contains the code for hashing a string. The StrToBin method converts a given string to a binary value. The rightRotate method will rotate right a given binary value a given number of bits. So if were to rightRotate a binary value by 1 bit, the bit furthest on the right would then become the bit furthest on the left and the rest of the bits would shift to the right by one place. The Hash method is where the bulk of the code for actually hashing a binary value is. The digest method then converts this hash to a string which represents a hex value and returns it.

**UI**

**Rectangle

Description automatically generated with medium confidence**

The UI class is the base class for the Terminal and GUI classes. It has zero properties and only contains one method. The run method will simply print out “UI not implemented”

**Terminal**

**Table

Description automatically generated with medium confidence**

The terminal class is responsible for the terminal version of the game. The terminal class is responsible for displaying the game in the terminal as well as handling users inputs into the terminal. The terminal class has two methods. The run method is responsible for setting up the game. It will create an instance of the game class as well, set the language to English using the game class and set it to be a 2 player game. It will then call the turn method. The turn method takes an instance of the game class as a parameter. The turn method is responsible for handling the turn by turn gameplay of the terminal version of the game. One run through of the turn method would be one turn of the game. It will display the board and the racks of the players and allow them to make their move. It will also display the winner of the game if the game if the game has ended. The turn method is called recursively until the game ends.

**Text

Description automatically generatedUI**

The GUI class is responsible for handling the GUI of the program. The main method in the GUI class is the run method. This method opens up the main menu which is the root window of the program (in tkinter) and is where the rest of the windows will be accesible from. The PlayGame method is the method responsible for the turn by turn running of the game in the GUI. This method allows users to drag and drop the tiles onto the board as well as handles the buttons in the GUI version of the game (such as the Next Turn button). The drawBackground method is responsible for drawing out the board and the tiles and the HandleBlank method is responsible for allowing the user to input the value of the blank tile if they place a blank tiles. Most of the methods in the GUI classing are responsible for displaying a separate window (e.g. the SelectLanguage method is responsible for displaying the select language window). The exceptions to this are PlayGameEnglish and PlayGameSpanish which are responsible for setting up the game in their respective language, LoadGame which is responsible for setting up and drawing a game that was loaded from the database, CheckLogin which is responsible for getting the details the user entered into the Login window and checking if the Login is valid, CheckCreateAccount which is responsible for getting the details the user entered into the create account window and checking if the user is able to create an account with that username and quitCreateAccount and quitLogin which are responsible for quitting the Login and Create Account windows. The GUI class also has the property account which has an instance of the Account class so that the GUI can know which user is logged in. The property blank is used in getting the value of what a player wants the blank tile to be. The rest of the properties relate to objects within the UI such as windows (properties which are of the tkinter.Tk, tkinter.Toplevel and pygame.display types) and entry boxes( properties which are of the tkinter.Entry types). For the windows, this is so that they can be modified are closed from methods in which they are not created in and for the entry boxes, this so the values typed into the entry boxes can be accessed from methods in which they are not created in. All of these properties that are related GUI objects are initially set to None (except for GUI.\_\_root) when an instance of the class is created.

## Data structures

**Board:**

The state of the board at any time is stored in 2D arrays. I used two 2D arrays to store the state of the board. One array stores where all the double/triple letter/word tiles are and the other stores where the letters have been played.

**DAWG (Directed Acrylic Word Graph):** For storing and implementing the scrabble dictionary I decided to use a Direct Acrylic Word Graph. The DAWG is similar to a trie in that it is used to store a large list of words/strings. However in a DAWG after a word has been entered in the DAWG a minimisation function is called which iterates through the letters/nodes of the end of the word that has just been entered into the DAWG to see if there are any similar nodes in the DAWG

Shape, circle

Description automatically generatedBelow shows an example of how a DAWG with the wors cat, cats, fact, facts, facet and facets would be created

1. First we start with a root node

Diagram

Description automatically generated

1. Diagram

   Description automatically generatedFrom the current node we add a new node. This new nodes is connect to the current node by an edge labelled with a character.
2. C was previously added so now we add A in order to insert the word cat

Diagram

Description automatically generated

1. We now add T to finish inserting the word cat. Once we reach the end of a word we mark the node as terminal. (In this diagram a terminal node is marked by a nested circle)

Diagram

Description automatically generated

1. We now move on to adding thew word cat. Cats shares the prefix “cat” with last word that was inserted. Since the common prefix size is the same as our current depth we don’t have to minimize any nodes. Then we add a new edge as “S” and mark the new node as a terminal node
2. A picture containing sky, map

   Description automatically generatedNow we add the word ‘fact’. Since there are no nodes in common to the last word added we return to the root when adding this word. Since there are no branches yet, no changes were made when minimising the existing nodes. The nodes and edges for the word fact are then added

A picture containing text, sky, map

Description automatically generated

1. Next we add the word ‘facts’. Much like ‘cats’ with ‘cat’, all we need to do for ‘facts’ is add a new node with an edge marked “S” pointing to it

A picture containing text, sky, map

Description automatically generated

1. Next we add ‘facet’. ‘facet’ shares the common prefix ‘fac’ with the previously added word so we will minimise nodes while traveling backwards from our current node to that point. On our first step back, we arrive at a node whose only edge is labelled 'S' and points to a terminal node with no further edges. We shrink the graph by transferring our edge to an identical existing node.

A picture containing sky, map

Description automatically generated

1. On the next step back we arrive at a node we arrive at a node with one edge labelled ‘T’ which points to a terminal node that has one edge labelled ‘S’ which points to terminal node with no further edges. An identical node like that already exists within the DAWG so we move the edge to point to that node

A picture containing text, sky, map

Description automatically generated

1. Having made our way back to the node reached from the root by ‘F’ ‘A’ ‘C’, we can add the new word ‘facet’

Diagram

Description automatically generated

1. As we’ve already done twice before, when the common prefix for the new word is simply just the entire previous word then we just have to add the new nodes and edges.

Diagram

Description automatically generated

1. A picture containing sky, map

   Description automatically generatedNow that we have finished inserting all the words, we will minimize all the nodes back to the root. On our first step back we arrive at a node that has one edge labelled “S” that points to terminal node with no further edges. Since a node like that already exists within the we move the edge to point to that node
2. Again we repoint the edge to an already existing identical node
3. Diagram

   Description automatically generatedFinally, we repoint he edge to an already existing identical node. Since none of the other nodes have identical nodes within the DAWG we go back to the root and have finished creating the DAWG for these words

In my program the DAWG class is stored through the use of the Dawg and DawgNode class. Each node of the DAWG is represented by an instance of the DawgNode class. The DawgNode class stores which nodes are its children in a dictionary and whether tha node is the end of the word. The letters of each edge in the DAWG are stored as the keys in the dictionary which store which nodes are children of the current node. The Dawg class stores the root node of the DAWG in the attribute Dawg.root. The Dawg class has 4 functions: the insert function, which allows a word to be inserted into the DAWG, the minimise function, which will check though all of the unchecked nodes that weren’t in the previous minimized word and will check them against the already minimised nodes and if an already minimised node exists with the same properties of an unchecked it will replace the already unchecked node with that already minimised node, the finish function, which calls the minimize function with a common prefix length of 0 and the search function, which takes a string as a parameter and return whether that string is in the DAWG or not.

**Miscellaneous Data Structures**

Game.\_currBag/Dictionary.\_bag – A standard array of strings that stores all of the letter tiles that have not been played

e.g. [“A”, “A”, “A”, “B”, “C”, “E”, “E”…. “Z”]

Game.\_pointsDict/Dictionary.\_pointsDict– A dictionary that stores the value of each letter tile

e.g. {"blank":0,"A":1,"E":1,"I":1,"L":1,"N":1,"O":1,"S":1,"T":1,"U":1,"D":2,"G":2,"B":3,"C":3,"M":3,"P":3,"F":4,"H":4,"V":4,"W":4,"Y":4,"K":5,"J":8,"X":8,"Q":10,"Z":10}

currMoves – An 2D list that contains the letters and the coordinates of the moves that have just been played by the player. The array inside the list always is in the same format, with the letter in the 0 index, the y coordinate in the 1 index and the X coordinate in the 2 index

e.g.[[“A”, 5, 6],[“C”, 5, 5],[“T”, 5, 7]]

Player.\_rack – A standard array used to store the letter tiles in the players rack

e.g. [“C”, “A”, “F”, “Z”, “U”, “J”, “H”]

Game.players – A list that stores instances of the Players class. These represent the players playing the gam

movesStack – A stack within the GUI class that contains all the drag and drops a user has made that turn. The items within the stack are stored as tuple which contain the instance of the pygame.Rectangle object of the rectangle used to draw that tile, as well as the original and final coordinates of the rectangle. When the undo button is pressed an item is popped of the stack and the rectangle in that tuple is set back to its original position

## Algorithms

Dawg.insert – is used to insert a new word into the DAWG. It will check if this word shares any letters with the previous word inserted (as the words should be inserted in alphabetical order into the DAWG) and then minimise (See below) all the nodes/letters that don’t share a prefix with the new word. It will then insert the new word into the DAWG creating a new node for each new letter and adding that node to the list of unchecked nodes so it can be minimised later.

Graphical user interface, text, application

Description automatically generated

Figure - Pseudocode for the insert function

Dawg.minimise – This function goes though the unchecked nodes and will check if any already checked nodes share the same properties as one of the unchecked nodes if so it will get rid of that unchecked node and what previously pointed to the unchecked node will point to similar already checked node.

Text

Description automatically generated

Figure - Pseudocode for the minimize function

Sha256 – The hash function implemented in the program was the Sha256 hash function. A hash function is a function that converts any input data into an output string of a fixed number of characters. Hashing functions are irreversible/incredibly hard to reverse without an infeasible amount of computational power. A hash function was used in my program to hash the passwords of the users before they are entered into the database. This is so that if the data in the database was hacked, the hackers would not be able to login to the accounts of the users as they would not know the passwords of the users as the passwords had been hashed.

In the program the Sha256 hash function is implemented in the Sha256 class. The HashAndDigest method is overarching method that will take a string and return the hashed version of that string. It does this by first calling the StrToBin method withing the class, which will convert a string to a binary value. It then calls the Hash method and passes the binary value of the string as a parameter. The Hash method is where the bulk of hashing algorithm is takes place. Finally it calls the Digest method which returns the hashed value of the string. The other method within the class is the RightRotate method which rotates the bits of a binary value right an n amount of times. This method is called multiple times within the Hash method.

Below is the pseudocode that the implementation was based from (sourced from the Sha-2 Wikipedia page):

Text

Description automatically generatedText

Description automatically generated

Game.validateTurn() – This method is responsible for validating the moves played by the player in their turn. It will return True if the moves are valid and false if the moves are not valid. The method takes one parameter, currMoves, which is a 2d array which stores the letters and coordinates of the moves.

* In this method, first a copy of player’s rack is stored in the variable pRack
* A copy of the board is taken and stored in the variable boardCopy
* Next it will loop through all the moves in currMoves using a for loop..
  + It will check if the letter of that move is in pRack
    - If it isn’t the method will return false
    - Otherwise it will remove that letter from pRack
  + It will then check if the x and y coordinate of that move is within the range of the board
    - If it they aren’t the method will return false
  + It will then check if the move was played through the centre square of the board
    - If a move was played through the centre square of the board a variable called playedCentre will be set to true
* If the turn is the first turn of the game where a move is made and playedCentre equals false
  + The method will return false
* Next the program checks that all the tiles placed are placed in line in one direction
* To do this it first sets 2 variables called vert and horz equal to true
* Using a for loop, looping from 0 to len(currMoves)-1 which allows it to loop though all the currMoves by looping through the indexes
  + If the x coordinate of that move at the current index is equal to the x coordinate of the move at the next index and horz equals true
    - vert will be set to false
  + If the y coordinate of that move at the current index is equal to the y coordinate of the move at the next index and vert equals true
    - horz will be set to false
  + else
    - False will be returned
* This also allows the program to identify in which direction the move was placed (whether it was horizontal or vertical)
* A for loop, looping for 0 to len(currMoves)
  + If boardCopy[currMoves[i][1]][currMoves[i][2]] != “ “
    - False is returned
    - This checks that the square on the board where the player is trying to make his move to is already empty
  + boardCopy[currMoves[i][1]][ currMoves[i][2]] is then set to the letter of that move (currMoves[i][0]
* Next the program validates all the words that have been formed by the tiles placed on the board
* First we check in the direction the move was played in
* A variable called word is assigned to be an empty string
* If vert equals True
  + We loop around a for loop going from currMoves[0][1] (the y value of one the moves) to 15
    - If boardCopy[i][currMoves[0][2]]== “ “
      * We break from the loop
    - Else
      * The letter in that square is added to the front of the word string
  + Then we loop around a for loop going from currMoves[0][1] -1 to 0 stepping down in increments of -1
    - If boardCopy[i][currMoves[0][2]]== “ “
      * We break from the loop
    - Else
      * The letter in that square is added to the end of the word string
  + This gets us the word was formed in the vertical direction when the tiles were placed
  + This word is searched for in the DAWG
    - And if it’s not in the DAWG, False is returned as that means the word is invalid
  + Next we loop around a for loop, with variable i, a len(currMoves) amount of time
    - Word is set to be an empty string
    - We loop around a for loop, with variable j, going from currMoves[i][2] to 15
      * If boardCopy[currMoves[i][1]][j] equals “ “
        + We break from the loop
      * Else
        + The letter in that square is added to the front of the word string
    - We loop around a for loop, with variable j, going from currMoves[i][2]-1 to 0 stepping down in increments of -1
      * If boardCopy[currMoves[i][1]][j] equals “ “
        + We break from the loop
      * Else
        + The letter in that square is added to the end of the word string
    - One loop of the for loop will get us one of the words formed in the horizontal direction when the moves were made
    - The word is searched for in the DAWG
    - If len(word) > 1 and the word is not in the DAWG
      * False is returned
  + This whole for loop finds and validates all the horizontal words created when the moves were made
* If horz equals True
  + We loop around a for loop going from currMoves[0][2] (the x value of one the moves) to 15
    - If boardCopy[currMoves[0][1]][i]== “ “
      * We break from the loop
    - Else
      * The letter in that square is added to the front of the word string
  + Then we loop around a for loop going from currMoves[0][2] -1 to 0 stepping down in increments of -1
    - If boardCopy[currMoves[0][1]][i]== “ “
      * We break from the loop
    - Else
      * The letter in that square is added to the end of the word string
  + This gets us the word was formed in the horizontal direction when the tiles were placed
  + This word is searched for in the DAWG
    - And if it’s not in the DAWG, False is returned as that means the word is invalid
  + Next we loop around a for loop, with variable i, a len(currMoves) amount of time
    - Word is set to be an empty string
    - We loop around a for loop, with variable j, going from currMoves[i][[1] to 15
      * If boardCopy[j][currMoves[i][2]] equals “ “
        + We break from the loop
      * Else
        + The letter in that square is added to the front of the word string
    - We loop around a for loop, with variable j, going from currMoves[i][2]-1 to 0 stepping down in increments of -1
      * If boardCopy[j][currMoves[i][2]][ equals “ “
        + We break from the loop
      * Else
        + The letter in that square is added to the end of the word string
    - One loop of the for loop will get us one of the words formed in the vertical direction when the moves were made
    - The word is searched for in the DAWG
    - If len(word) > 1 and the word is not in the DAWG
      * False is returned
  + This whole for loop finds and validates all the vertical words created when the moves were made
* If the program reaches this point that means the move is valid and has passed all the checks
* Game.\_board is updated to equal boardCopy
* The player’s rack is updated to equal pRack
* The value True is returned

Game.CalculatePoints – this method is found within the game class. It is used to calculate the points of given moves and adds them to the player’s total. It takes the parameter currMoves which is a 2d array that contains the character(s) and the coordinates of the moves made.

* First the algorithm checks whether the move was made in a horizontal or vertical direction
* If len(currMoves) == 1
  + The variable vert is set to True
  + The variable horz is set to false
  + This is just arbitrary for turns were only 1 move is made as it should not matter whichever of vert and horz are True as a move were only 1 move is made has no real direction
* Else
  + It compares the y coordinate of the first item/move in currMoves to y coordinate of the second item in currMoves (currMoves[0][1] == currMoves[1][1])
    - If they are equal that means the move was made in a horizontal direction
      * The variable horz is set to true
      * The variable vert is set to false
    - If they are not equal
      * The variable horz I set to false
      * The varaiable vert is set to true
* The variable totalPoints is set to 0
* If vert is true
  + The variable vMultiplier is set to 1
    - The variable word is set to be an empty string
    - Next we loop around a for loop, with variable i, from 0 to len(currMoves)
    - The variable wordPoints is set to 0
    - The variable hMultiplier is set to 1
    - The variable lMutiplier is set to 1
    - Next we check if the square that currMoves[i] is in is multiplier square by checking the value of Game.\_boardPoints[currMoves[i][1]][currMoves[i][2]]
      * If it’s in a letter multiplier square, lMultiplier is set to the value of that multiplier
      * If it’s in a word multiplier square, vMultiplier and hMultiplier are set to the value of that multiplier
    - Next we loop around a for loop, with variable j, going from currMoves[i][2]+1 to 15
      * If self.\_board[currMoves[i][1]][j] equals “ “
        + We break from the loop
      * Elif the letter in self.\_board[currMoves[i][1]][j] is lower case
        + The letter in that square is added to the front of the word string
        + However no points are added as if it’s a lower case letter it represents a blank tile
      * Else
        + The letter in that square is added to the front of the word string
        + The value of self.\_pointsDict[ self.\_board[currMoves[i][1]][j] is added to wordPoints
    - We loop around a for loop, with variable j, going from currMoves[i][2]-1 to 0 stepping down in increments of -1
      * If self.\_board[currMoves[i][1]][j] equals “ “
        + We break from the loop
      * Elif the letter in self.\_board[currMoves[i][1]][j] is lower case
        + The letter in that square is added to the end of the word string
        + However no points are added as if it’s a lower case letter it represents a blank tile
      * Else
        + The letter in that square is added to the end of the word string
        + The value of self.\_pointsDict[ self.\_board[currMoves[i][1]][j] is added to wordPoints
      * If len(word) == 1
        + Then wordPoints is set to 0
      * Else
        + wordPoints equals wordPoints \* hMultiplier
      * The value of wordPoints is added to the value of totalPoints and stored in totalPoints
      * One loop of this loop gets us the points for one of the horizontal words formed when this move was played
  + wordPoints is then set to 0
  + word is set to be an empty string
  + lMultiplier is set to equal 1
  + We loop around a for loop going from currMoves[0][1] (the y value of one the moves) to 15
    - If self.\_board[i][currMoves[0][2]]== “ “
      * We break from the loop
    - Elif self.\_board[i][currMoves[0][2]] is lower case
      * The letter in that square is added to the front of the word string
      * But no points are added for that letter as it’s a blank tile
    - Else
      * We check if the coordinates of this square are in currMoves
        + If they are we then check if this is a letter multiplier square

If it is lMultiplier is set to the value of the multiplier

* + - * The letter in that square is added to the front of the word string
      * The value of self.\_pointsDict[self.\_board[i][currMoves[0][2]]]\*lMultiplier is added to wordPoints
  + Then we loop around a for loop going from currMoves[0][1] -1 to 0 stepping down in increments of -1
    - If self.\_board[i][currMoves[0][2]]== “ “
      * We break from the loop
    - Elif self.\_board[i][currMoves[0][2]] is lower case
      * The letter in that square is end to the front of the word string
      * But no points are added for that letter as it’s a blank tile
    - Else
      * We check if the coordinates of this square are in currMoves
        + If they are we then check if this is a letter multiplier square

If it is lMultiplier is set to the value of the multiplier

* + - * The letter in that square is added to the end of the word string
      * The value of self.\_pointsDict[self.\_board[i][currMoves[0][2]]]\*lMultiplier is added to wordPoints
  + If len(word) = =1
    - Then wordPoints is set to 0
  + wordPoints is set to wordPoints \* vMultiplier
  + This gets us the points for the vertical word formed
  + The value of wordPoints is added to totalPoints
* If horz is true
  + We’d go through the same process detailed above but instead the loop detailed first would get the points for the vertical words and then the other loop would get the points for the horizontal word
* If len(currMoves) is 7
  + Then 50 points is added to totalPoints
* totalPoints is then added to the player’s total

Game.isGameOver() – this algorithm is a simple algorithm that is crucial for the running of the game as it checks if the game is over.

It will check if self.\_currBag is empty and len(self.players[self.\_pTurn].getRack()) == 0. This checks whether the bag containing all the tiles is empty and if the player who played the last turn’s rack is empty as well.

It will also check if self.\_passFlag is True. self.\_passFlag is only True when both players have passed their turn on 2 consecutive turns

If either of these conditions are true then it will return True (flagging that the game has ended) otherwise it will return false

Game.findWinner() – this algorithm is another simple but crucial algorithm to the running of the game. This algorithm finds the winner of the game and returns an integer relating to the index of Game.players property that the instance of the player class of the winner is stored in.

Game.deductPoints() – this method is called once a game has ended and will deduct points from the player based on the tiles they have on their rack.

It first calls a for loop, with variable I, that loops form 0 to self.\_numPlayers. It saves the points total before the poinst have been deducted in self.players[i].preEndPoints. It will then check if the length of the players rack equal 0 (by using the statement: if len(self.players[i].displayRack()) == 0 ). If it’s not equal to 0 then the variable a variable called playerPoints is set to 0. Then a for loop iterating through the players rack (self.players[i].displayRack()) with variable j will. Each iteration the value of self.\_pointsDict[j] will be added to playerPoints. Then the method self.players[i].updatePoints will be called with the negative value of player points being passed as the parameter. This cause the value of playerPoints to be taken away form the player’s points

## File structure and organisation

**File Structure Diagram**

Below is the file structure diagram for the files in the program. The arrows show which files import/read data and classes from which files (where the tail of the arrow is the file imported/read from and the head of the arrow is the file imported/read to). The arrows have show

Diagram

Description automatically generated

**Files**

Below is a table that lists all the files in the program as well as what they are used for in the program

|  |  |
| --- | --- |
| **File** | **Use** |
| Scrabble.py | This is the file where the game is launched |
| UI.py | This file has the code for how the program will be displayed in both terminal mode and GUI mode. It has the Terminal class which is responsible for the UI in terminal mode and GUI class which is responsible for the UI in GUI mode |
| Game.py | This file contains the code relating to the main logic behind the game. This includes the Game class. |
| Database.py | This file contains the code involved in managing and getting data to and from the database |
| ScrabbleDatabase.db | This is the database where all the data relating to the users and the games will be stored |
| Hashing.py | This file contains all the code relating to hashing |
| Languages.py | This file contains all the code relating to the different languages of the game. It can creates the DAWG for both languages as well as handling switching languages. |
| Spanish.txt | This file contains all the possible Spanish words in the Spanish Scrabble dictionary |
| Spanish.txt | This file contains all of the Spanish words in the Spanish Scrabble dictionary |
| DAWG.py | This file contains all the code relating to implementing the DAWG. The DAWG is how all the words (in both text files) will be stored and accessed by the program |

**Classes stored in files**

Below is a table that shows which classes are in which files in the program

|  |  |
| --- | --- |
| **File** | **Classes** |
| Scrabble.py | N/A |
| UI.py | UI  Terminal GUI |
| Game.py | Game  Player |
| Database.py | Account  Database |
| Hashing.py | Sha256 |
| Languages.py | Dictionary  Language  Spanish  Spanish |
| DAWG.py | Dawg  DawgNode |

## Database design

The database contains three tables: Players, Games, GamesPlayed

The Players table is used to store the all the data relating to the user such as the username, password and number of wins and losses the user has

The Games stable is used to store the data for the saved games so that users are able to return to them and finish them at a later date

The GamesPlayed table is used to store which users have played which games.

**Fields in each database**

**Players** (Username, Password, NumGames, Wins, Losses)

|  |  |
| --- | --- |
| Username(VARCHAR 255, PRIMARY KEY) | Stores a unique player username which is used as a primary key to uniquely identify each account |
| Password(VARCHAR 255) | Stores a hashed password used to Login the user |
| NumGames(INT) | Store the number of games the player has played |
| Wins(INT) | Stores the number of wins the player has |
| Losses(INT) | Stores the number of losses the player has |

**Game** (GameID, TurnNo, Board, ScoreP1,ScoreP2, Bag, Language)

|  |  |
| --- | --- |
| GameID(INTEGER, PRIMARY KEY, AUTOINCREMENT) | Stores a number (which is unique to each record) which is used as the primary key. This field is an autoincrement field which means, each time a record is inserted into the database, a unique value for this field for that recorded will be automatically generated |
| TurnNo(INT) | Stores which players turn it is |
| Board(VARCHAR 255) | Stores the state of the board as a string |
| Scores(VARCHAR 255) | Stores the scores of the players |
| RackP1 (VARCHAR 255) | Stores the rack of player 1 |
| RackP2 (VARCHAR 255) | Stores the rack of player 2 |
| Bag(VARCHAR 255)) | Stores all the remaining letters/tiles that have not yet been played |
| Language(VARCHAR 255)) | Stores which language the game is being played in |

**GamesPlayed**(Username, GameID)

|  |  |
| --- | --- |
| Username(VARCHAR 255, PRIMARY KEY, FOREIGN KEY)) | Stores the username of the user who played the game |
| GameID(INTEGR, PRIMARY KEY,FOREIGN KEY) | Stores the GameID of the game that was played |

**Database Relationship Diagram**

Below is a diagram showing the how each table relates to the others.

The Games table has a one to many relationship with the Games Played table and a many to many relationship with the Players table. The Players table also has a one to many relationship with the Games Played table.Diagram

Description automatically generated

## SQL queries

These are all the SQL queries used within the code. They have been divided up by what function they are in:

**Database.CreateTables()**

"""CREATE TABLE IF NOT EXISTS Players (

Username VARCHAR(255) PRIMARY KEY,

Password VARCHAR(255),

NumGames INT,

Wins INT,

Losses INT)"""

This is SQL statement is used to create the Players table if the table does not already exist within the database. The Players table is used to store the details of users/players. In the Players table the Username is the primary key. The Password field stores the hash of the password of the user , the NumGames field stores the number of games the user has played and the Wins and Losses fields store the number Wins and Losses that user has respectively.

"""CREATE TABLE IF NOT EXISTS Games(

GameID INTEGER PRIMARY KEY AUTOINCREMENT,

TurnNo INT,

Board VARCHAR(255),

Scores VARCHAR(255),

Bag VARCHAR(255),

Language VARCHAR(255),

RackP1 VARCHAR(255),

RackP2 VARCHAR(255));"""

This SQL statement is used to create the Games table if it does not already exist in the database. This table is used to store the details of unfinished so that they can be loaded from the database and finished. The GameID field acts as the primary key and is an autoincrementing primary key. This means that every time a new record is inserted into the database, the database will automatically generate the value of GameID. The Board field stores the sate of the board, the Scores field stores the scores of the players, the Bag field stores the tiles left in the bag, the Language field stores what language the game is being played in and RackP1 and RackP2 store the rack of player 1 and player 2 respectively.

"""CREATE TABLE IF NOT EXISTS GamesPlayed(

Username VARCHAR(255) ,

GameID INT ,

PRIMARY KEY(Username,GameID),

FOREIGN KEY (Username)

REFERENCES Players (Username),

FOREIGN KEY(GameID)

REFERENCES Games (GameID)

);"""

This SQL statement creates the GamesPlayed table if it does not already exist within the database. This table is used to store which players have played which games. The primary keys of this table are Username and GameID. The Username field relates to the Username field in the players table and GameID field relates to the GameID field in the Games table.

**Database.CreateAccount()**

"SELECT Username FROM Players WHERE Username = ?",(*username*,)

This is SQL statement simply checks the Players table to see if there is a Username in the table with the same username as the one the player entered. This is to make sure that the username the user entered to create his account is unique and if its not the program is able to handle it accordingly and display an appropriate message.

"INSERT INTO Players(Username,Password,NumGames,Wins,Losses) VALUES (?,?,0,0,0)",(*username*,pHashed)

This SQL statement creates a new record for a new user in the Players table. The username and the hashed version of the password the player entered are entered into the table under the Username and Password records respectively and the number of games the user has played and the number of wins and losses the user has had is set to 0.

**Datatbase.Login()**

"SELECT Username FROM Players WHERE Username = ? AND Password = ? ",(*username*,pHashed)

This SQL selects any statements where the Username and (hashed) Password are equal to the username and hashed password the user entered. This SQL statement is used to verify the Login of users trying to Login to an account. If the select statement returns a record then we know that there is an account with that user and password combination and the program can then “log the user in”. If the select statement returns zero record then we know the user has entered the details to their account incorrectly and thus handle this accordingly.

**Database.UpdateWin()**

"UPDATE players SET Wins = Wins + 1 WHERE Username = ? ",(*username*,)

This function is used whenever a logged in user wins a game in order to update their statistics. This SQL statement updates the amount of wins that user has won by 1

"UPDATE players SET NumGames = NumGames + 1 WHERE Username = ? ",(*username*,)

This SQL statement updates the amount of games that user has played by 1

**Database.UpdateLoss()**

"UPDATE players SET Losses = Losses + 1 WHERE Username = ? ",(*username*,)

This function is used whenever a logged in user losses a game in order to update their statistics. This SQL statement updates the amounts of losses that user has lost by 1.

"UPDATE players SET NumGames = NumGames + 1 WHERE Username = ? ",(*username*,)

This SQL statement updates the amount of games that user has played by 1

**Database.GetWinLoss()**

"SELECT NumGames,Wins,Losses FROM Players WHERE Username = ?",(*username*,)

This SQL statement gets the number of games the user has played as well as the number of wins and losses that user has from the database.

**Database.SaveGame()**

"INSERT INTO Games (TurnNo,Board,Scores,Bag,Language,RackP1,RackP2)

VALUES (?,?,?,?,?,?,?)",(*TurnNo*,*Board*,*Scores*,*Bag*,*Language*,*RackP1*,*RackP2*)

This function is used to save a game to the database. This SQL statement creates a new record in the Games table and inserts the turn number, state of the board, the scores of the players, the tiles remaining in the bag, the language the game is being played in and the rack of player 1 and player 2. Since GameID is an AUTOINCREMENT field the value for that field does not need to be inserted as the GameID will automatically be generated and inserted into the database by the database. (The Game ID will be the same value as the row number as the row it is inserted into)

"SELECT last\_insert\_rowid()"

This SQL statement gets the rowid/row number where the last insert was made. Since the GameID field is an AUTOINCREMENT field the rowid will be equal to the value of GameID of the game that was just inserted into the database. This GameID is needed as it needs to be inserted into the GamesPlayed table so the program knows which players played which games.

"INSERT INTO GamesPlayed (Username,GameId) VALUES (?,?)",(*username*,GameID)

This SQL statement inserts the username of the user who played the game as well as the GameID of the game into the GamesPlayed table so that the program knows that that user played that game.

**Database.LoadGame()**

"SELECT TurnNo,Board,Scores,Bag,Language,RackP1,RackP2 FROM Games

WHERE GameID = ?",(*GameID*,)

This SQL statement gets the turn number, the state of the board, the scores of the players, the tiles left in the bag, the language the game is being played in and the racks of player 1 and player 2 from the Games table where the GameID is equal to the GameID of that game. This so that that game can be reloaded and continued from where it was saved.

**Database.GetGames()**

"""SELECT Games.GameID, Games.TurnNo,Games.Scores, Games.Language

FROM Games

INNER JOIN GamesPlayed ON GamesPlayed.GameID = Games.GameID

WHERE GamesPlayed.Username = ?""",(*username*,)

This SQL statement gets all the GameIDs , number of turns, scores and languages of the games that that user has played from the Games table. This is used in order to display all of the games that user has played in the load games window in the GUI .This SQL statement uses an inner join as the Username field (which stores the usernames of the users) is not in the Games table. The Username field however is in the GamesPlayed table which also contains the GameID field which is in the Games table. So an inner join must be used (on the GameID field in both tables) in order to select records from the Games table.

## User interface design

**Terminal**

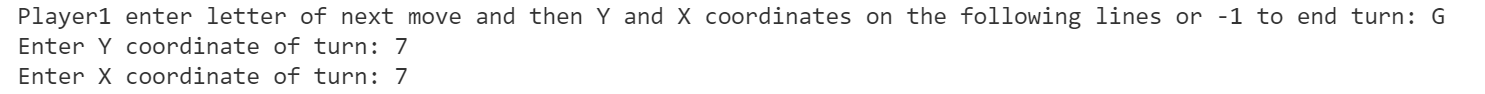
When a game is launched in terminal mode, each turn an interface that looks like this should be displayed:  
Table

Description automatically generated

In this interface the board is displayed, along with the rack of the player whose turn it is, the number of tiles in the bag and the number of points each player has.

The player should be prompted to input their turn. To input a move the player should first enter the letter when prompted and then y coordinated of where they want to play the letter followed by the x coordinate of where they want to play the letter.

This should like something like the screenshot below:



The user should keep making moves until they want to end their turn. To end their turn a user simply needs to type in -1 when prompted to enter a letter. A sample turn should look something like this:

Text

Description automatically generated with medium confidence

This should cause the board to update on the next turn if the move is valid.

Background pattern

Description automatically generated

If the turn is invalid then an error message should be displayed and the user should be asked to re-enter their moves.

To pass their turn the user simply needs to enter -1 when initially prompted to enter a letter

Once the game has ended then the game should display the winner which should look something like this:

Text

Description automatically generated with medium confidence

**Gui**

Table

Description automatically generatedWhen launching the GUO a window containing a menu of multiple buttons should launch.

Table

Description automatically generatedClicking the Play Game should close down the menu and open up a window that looks like this:

The window should say “Select Language” and have two buttons on it that say “English” and “Spanish”.

Clicking either of create a new window that looks like the wireframe below that allows users to play a game in the language tha they selected

A picture containing table

Description automatically generated

This window should display a grid representiing the scrabble board. It should also diplay which player’s turn it is, how many points the players have, how many tiles are left in the pack and what tiles are in the players rack. It should also display the buttons labelled “Next Turn”, “Undo”,”Pass” and “Save Game and Exit” .Users should be able to drag and drop the tiles from their rack on to the squares on the board. The user should then click next turn once they have made their move. If an invalid move was made or if the player hadn’t made a move when the Next Turn button was pressed an error message should appear and the program will make the player replay their move. If a valid move was played, the points for the move will be calculated and updated at the top of the window as well as the text displaying whose turn it is. Clicking the Pass button should pass the that players turn and move on to the next turn. Clicking the undo button after the user has dragged and dropped some tiles onto the board, should move the last tile the user played back to its previous position. When the game has finished the winner of the game should be displayed. Clicking the Save Game and Exit should cause the window to close an return to the main window as well as the game being saved to the database

Text, letter

Description automatically generatedClicking the Account button when the user is logged in should display a window that looks like this: (otherwise it should display an error window)

Diagram

Description automatically generatedThe Accounts window should display the players statistics as well as having two buttons that say “Load Games” and “Quit”. Clicking the Quit button should close the window. Click the Load Games button should open up a window that looks like this:

This window should say “Load Games” at the top of the window. In the middle of the window there should be buttons that represent saved games. To the right of the window there should be a scroll bar that allows the user to scroll through saved games if there are too many to fit in the window. At the button of the window there should be a button labelled “Quit”. Clicking this button should close the window. Clicking any of the buttons that represent saved games should open the window that allows the user to play those games.

Clicking the Login button in the main menu should open a window that looks like this:

Graphical user interface

Description automatically generated

Clicking the Login button should check the database if there is an account with the same username and password as the one entered into the entry boxes. If there is, the player will be logged in and the window should close and the main menu window should open. If there isn’t, an error message should be displayed and the user should be asked to re-enter their details. Clicking the quit button should close the window

Graphical user interface, application

Description automatically generatedClicking the Create Account button in the main menu should open a window that looks like this:

The window should have two entry boxes labelled “Enter Username” and “Enter Password” as

well as two buttons that say “Create Account” and “Quit”. Clicking the “Create Account” button should check the database if there is a record with the same username as the one entered in the entry box. If there is an error message should be displayed. Otherwise the a new record should be created with the username and password entered into the entry box

Clicking the Quit button on the main meu should close the window and exit the program.

## User guide

For this program the user would need to have installed to tkinter and Pygame. To open the GUI version of the game the user will need to enter ‘g’ on the command line which should launch the GUI. To then enter a game of scrabble the user should click the “Play Game” button at the top of the menu. This should open up a window prompting the user to select langauge. Selecting either language should open up a window with a scrabble board on it. To play a move the user simply has to drag and drop tiles from their rack onto the board and then click the next turn button. The user should then click the button of the language they want to switch to on this new menu to change the language. To open the terminal version of the game the user should enter ‘t’ on the command line and a game on 1 vs 1 scrabble should then commence in their terminal.

# Technical Solution

## Completeness Section

All the objectives of the “Minimal Viable Product” were completed providing a reliable version of the game in terminal mode. The terminal mode allows users to play against another person on the same device.

All objectives of the “Graphical User Interface” were met allowing the user to play against another person through mouse clicks. The GUI also allows users to Login to an account and load saved games from the database.

All objectives of the Login system were met. This allows users to create an account and Login to an account and allows them to save unfinished games to finish later on.

The Hashing and Languages objectives were also able to be met

The “AI” objectives weren’t able to be met.

Though a few objectives weren’t able to be met, the overall product still meets the original background description of the project to create an electronic version of scrabble.

## Technical Skills Section

### Overview Guide

**Technical Skills**

|  |  |  |  |
| --- | --- | --- | --- |
| **Skill Group** | **Skill Demonstrated** | **Purpose** | **Location** |
| A | List Operations | Used throughout the code e.g. to update the rack of a player | Languages.py,game.py,DAWG.py,UI.py |
| A | DAWG | Used to store the scrabble dictionary | DAWG.py |
| A | Graph Traversal | Searching for a word within the DAWG and the minimise function | DAWG.py |
| A | Complex user defined algorithms | Used to check if a word played by a player is valid or not | Game.py |
| A | Complex OOP – dynamic creation of objects | Creating Nodes in the DAWG | DAWG.py |
| A | Complex User defined algorithm | Calculating the points score of a player | Game.py |
| A | Complex OOP - polymorphism | The run method will run no matter which UI | UI.py |
| A | Aggregate SQL functions | Used to get the games a user has played | Database.py |
| A | Hashing | The Sha256 class is an implementation of the Sha256 hashing algorithm | Hashing.py |
| A | Recursion | The turn method in the Terminal class is called recursively so that turns can keep being played | UI.py |
| A | Complex OOP - inheritance | The English and Spanish classes inherit the returnBag, returnPointsDict, returnFilename methods from the Language class | Languages.py |
| A | Complex OOP - composition | The Dictionary class creates instances of the English and Spanish class in its \_\_init\_\_ function and are store in a property of the Dictionary class | Languages.py |
| A | Complex OOP - composition | The Database class creates an instance of the Sha256 class in its \_\_init\_\_ are is stored in a property of the Database class | Database.py |
| A | Complex OOP - composition | The Account class creates an instance of the Database class in its \_\_init\_\_ and is stored in a property of the Account class | Database.py |
| A | Complex OOP - composition | The GUI class creates an instance of the Account class in its \_\_init\_\_ and is stored in a property of the GUI class | UI.py |
| A | Complex OOP - composition | The Game class creates and stores instances of the Players class in a property of the Game class | Game.py |
| A | Complex OOP - composition | The game class creates an instance of the Dictionary class in its \_\_init\_\_ and stores it in a property with the game class | Game.py |
| A | Complex OOP - composition | The Dawg class creates and stores an instance of the DawgNode class in a property of the Dawg class | DAWG.py |
| A | Dynamic generation of objects | The rectangle objects generated when the board is drawn depend on the users action | UI.py |
|  |  |  |  |
| B | Multi-Dimensional Array | Used to store the state of the board | Game.py |
| B | Multi-Dimensional Array | Used to store the location of the multiplier squares | Game.py |
| B | Reading form files | Loading the list of words | Languages.py |
| B | Simple user-defined algorithm | Adding a word to the DAWG | DAWG.py |
| B | Simple user-defined algorithm | Checking if a word is in the DAWG | DAWG.py |
| B | Simple use-defined algorithm | Minimisation algorithm in the DAWG | DAWG.py |
| B | Dictionaries | A dictionary is used to store the points of each letter | Languages.py |
| B | Simple data model in database | 3 interlinked tables to store the Users and the Saved Games | Database.py |
| B | Records | Used to store the details of Users and the details of saved games | Database.py |
| B | Single table SQL | Used to log users in | Database.py |
| B | Single table SQL | Used to Save a game | Database.py |
| B | Single table SQL | Used to create an account | Database.py |
| B | Single table SQL | Used to get the number of wins, losses and games played the user has | Database.py |
| B | Single table SQL | Used to update the number wins and gamesPlayed a user has | Database.py |
| B | Single table SQL | Used to update the number of losses and gamesPlayed a user has | Database.py |
| B | Single table SQL | Used to create the tables in the databse | Database.py |
| B | Simple user defined algorithms | Used to find winner of the game | Game.py |
| B | Simple user defined algorithms | Used to check if the game is over | Game.py |
|  |  |  |  |
| C | Single Dimensional Arrays | Single dimensional arrays are used throughout the code e.g. to store the letters currently in the players rack | All files |

### Code listing

#### Scrabble.py

from UI import Terminal,GUI,UI

from sys import argv

def usage():

    print(f"""

Usage: {argv[0]} [g | t]

g : play with the GUI

t : play with the Terminal""")

    x =input()

    if x =='g':

        ui = GUI()

        ui.run()

    elif x =='t':

        ui = Terminal()

        ui.run()

    else:

        raise ValueError("Input should be either 'g' or 't'")

usage()

#### Database.py

import sqlite3

import hashlib

from Hashing import Sha256

class Account():

   def \_\_init\_\_(self):

      self.\_Account = None

      ###########################################

      #Group A skill - Complex OOP - Composition#

      ###########################################

      self.\_database= Database()

   def Login(self,Username,Password):

      result = self.\_database.Login(Username,Password)

      if result==False:

         return False

      else:

         self.\_Account= result[0][0]

         return True

   def getAccount(self):

      return self.\_Account

   def CreateAccount(self,username,password):

      result= self.\_database.CreateAccount(username,password)

      print(result)

      return result

   def GetWinLoss(self):

      result = self.\_database.GetWinLoss(self.\_Account)

      return result

   def SaveGame(self,TurnNo,Board,Scores,Bag,Langauge,rackP1,rackP2):

      self.\_database.saveGame(self.\_Account,TurnNo,Board,Scores,Bag,Langauge,rackP1,rackP2)

   def UpdateWin(self):

      self.\_database.updateWin(self.\_Account)

   def UpdateLoss(self):

      self.\_database.updateLoss(self.\_Account)

   def SaveGame(self,TurnNo,Board,Scores,Bag,Language,rack1,rack2):

      self.\_database.saveGame(self.\_Account,TurnNo,Board,Scores,Bag,Language,rack1,rack2)

   def LoadGame(self,GameID):

      return self.\_database.loadGame(GameID)

   def GetGames(self):

      return self.\_database.GetGames(self.\_Account)

######################################

#Group B skill - simple databse model#

#Group B skill - records             #

######################################

class Database():

   def \_\_init\_\_(self):

      ###########################################

      #Group A skill - Complex OOP - Composition#

      ###########################################

      self.\_Sha256 = Sha256()

      self.CreateTables()

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def CreateTables(self):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("""CREATE TABLE IF NOT EXISTS Players (

      Username VARCHAR(255) PRIMARY KEY,

      Password VARCHAR(255),

      NumGames INT,Wins INT,

      Losses INT)""")

      c.execute("""CREATE TABLE IF NOT EXISTS Games(

         GameID INTEGER PRIMARY KEY AUTOINCREMENT,

         TurnNo INT,

         Board VARCHAR(255),

         Scores VARCHAR(255),

         Bag VARCHAR(255),

         Language VARCHAR(255),

         RackP1 VARCHAR(255),

         RackP2 VARCHAR(255));""")

      c.execute("""CREATE TABLE IF NOT EXISTS GamesPlayed(

        Username VARCHAR(255) ,

        GameID INT ,

        PRIMARY KEY(Username,GameID),

        FOREIGN KEY (Username)

           REFERENCES Players (Username),

        FOREIGN KEY(GameID)

            REFERENCES Games (GameID)

         );""")

      con.commit()

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def CreateAccount(self,username,password):

      pHashed= self.PasswordHash(password)

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("SELECT Username FROM Players WHERE Username = ?",(username,))

      result = c.fetchall()

      if len(result) == 0:

         c.execute("INSERT INTO Players(Username,Password,NumGames,Wins,Losses) VALUES (?,?,0,0,0)",(username,pHashed))

         con.commit()

         return True

      else:

         return False

   def PasswordHash(self,password:str):

      p=self.\_Sha256.HashAndDigest(password)

      return p

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def Login(self,username:str,password:str):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      pHashed= self.PasswordHash(password)

      c.execute("SELECT Username FROM Players WHERE Username = ? AND Password = ? ",(username,pHashed))

      result = c.fetchall()

      if len(result) == 0:

         return False

      else:

         return result

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def updateWin(self,username:str):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("UPDATE players SET Wins = Wins + 1 WHERE Username = ? ",(username,))

      c.execute("UPDATE players SET NumGames = NumGames + 1 WHERE Username = ? ",(username,))

      con.commit()

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def updateLoss(self,username:str):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("UPDATE players SET NumGames = NumGames + 1 WHERE Username = ? ",(username,))

      c.execute("UPDATE players SET Losses = Losses + 1 WHERE Username = ? ",(username,))

      con.commit()

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def saveGame(self,username,TurnNo,Board,Scores,Bag,Language,RackP1,RackP2):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("INSERT INTO Games (TurnNo,Board,Scores,Bag,Language,RackP1,RackP2) VALUES (?,?,?,?,?,?,?)",(TurnNo,Board,Scores,Bag,Language,RackP1,RackP2))

      c.execute("SELECT last\_insert\_rowid()")

      GameID =c.fetchall()[0][0]

      print(GameID)

      c.execute("INSERT INTO GamesPlayed (Username,GameId) VALUES (?,?)",(username,GameID))

      con.commit()

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def loadGame(self,GameID):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("SELECT TurnNo,Board,Scores,Bag,Language,RackP1,RackP2 FROM Games WHERE GameID = ?",(GameID,))

      result = c.fetchall()

      return result

   ##################################

   #Group B skill - single table SQL#

   ##################################

   def GetWinLoss(self,username):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("SELECT NumGames,Wins,Losses FROM Players WHERE Username = ?",(username,))

      result = c.fetchall()

      return result

   ##################################

   #Group A skill - multi table SQL#

   ##################################

   def GetGames(self,username):

      con = sqlite3.connect("ScrabbleDataBase.db")

      c = con.cursor()

      c.execute("""SELECT Games.GameID, Games.TurnNo,Games.Scores, Games.Language

      FROM Games

      INNER JOIN GamesPlayed ON GamesPlayed.GameID = Games.GameID

      WHERE GamesPlayed.Username = ?""",(username,))

      result = c.fetchall()

      return result

#### Hashing.py #########################

#Group A skill - Hashing#

#########################

class Sha256:

    def StrToBin(string:str):

        bi = ""

        for i in range(len(string)):

            nextbi=ord(string[i])

            nextbi= bin(nextbi)

            bi+= str(nextbi)[2:]

        return bi

    def rightRotate(num:int,shift:int):

        size =32

        return (num>>shift)|(num<<(size-shift))

    def \_\_init\_\_(self):

        self.\_h=[None]\*8

        self.\_B32 = 0xFFFFFFFF

    def digest(self):

        hash =""

        for i in range(len(self.\_h)):

            temph =hex(self.\_h[i])

            temph=str(temph)[2:]

            hash+=temph

        return hash

    def Hash(self,message:str):

        h0 = 0x6a09e667

        h1 = 0xbb67ae85

        h2 = 0x3c6ef372

        h3 = 0xa54ff53a

        h4 = 0x510e527f

        h5 = 0x9b05688c

        h6 = 0x1f83d9ab

        h7 = 0x5be0cd19

        k=[0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]

        binMessage = Sha256.StrToBin(message)

        messageLength = bin(len(binMessage))

        messageLength=str(messageLength[2:])

        for i in range(64-len(messageLength)):

            messageLength="0"+messageLength

        count =0

        while True:

            value = str(binMessage)+"1"+"0"\*count+messageLength

            if len(value) %512 ==0:

                break

            count+=1

        for i in range(len(value)//512):

           chunk =value[512\*i:512\*(i+1)]

           w=[0]\*64

           for i in range(16):

                bitW=str(chunk[i\*32:(i+1)\*32])

                w[i]=int(bitW,base=2)

        for i in range(16,64):

            s0 = Sha256.rightRotate(w[i-15],7)^Sha256.rightRotate(w[i-15],18)^(w[i-15]>>3)

            s1 = Sha256.rightRotate(w[i-2],17)^Sha256.rightRotate(w[i-2],19)^(w[i-2]>>10)

            w[i]=(w[i-16]+s0+w[i-7]+s1) & self.\_B32

        a = h0

        b = h1

        c = h2

        d = h3

        e = h4

        f = h5

        g = h6

        h = h7

        for i in range(64):

            s1= Sha256.rightRotate(e,6)^Sha256.rightRotate(e,11)^Sha256.rightRotate(e,25)

            ch = (e&f)^((~e)&g)

            temp1= h + s1 + ch+ k[i]+ w[i]

            s0 = Sha256.rightRotate(a,2)^Sha256.rightRotate(a,13)^Sha256.rightRotate(a,22)

            maj = (a & b)^(a&c)^(b&c)

            temp2=s0+maj

            h=g

            g=f

            f=e

            e=(d+temp1)&self.\_B32

            d=c

            c=b

            b=a

            a=(temp1+temp2)&self.\_B32

        self.\_h[0] = (h0 +a)&self.\_B32

        self.\_h[1] =(h1+b)&self.\_B32

        self.\_h[2] =(h2+c)&self.\_B32

        self.\_h[3] =(h3+d)&self.\_B32

        self.\_h[4] =(h4+e)&self.\_B32

        self.\_h[5] =(h5+f)&self.\_B32

        self.\_h[6] =(h6+g)&self.\_B32

        self.\_h[7] =(h7+h)&self.\_B32

    def HashAndDigest(self,string:str):

        self.Hash(string)

        hash=self.digest()

        return hash

#### Dawg.py

######################

#Group A skill - DAWG#

#####################

class DawgNode:

    NextID = 0

    def \_\_init\_\_(self):

        self.id  = DawgNode.NextID

        DawgNode.NextID += 1

        self.EndOfWord = False

        #############################

        #Group B skill - Dictionaries#

        #############################

        self.children={}

class Dawg:

    def \_\_init\_\_(self):

        ###########################################

        #Group A skill - Complex OOP - Composition#

        ###########################################

        self.root = DawgNode()

        self.previousWord= ""

        self.uncheckedNodes=[]

        self.minimisedNodes={}

    ###############################################

    #Group B skill - simple user defined algorithm#

    ###############################################

    def insert(self, word):

        LengthOfCommonPrefix = 0

        length = min(len(word),len(self.previousWord))

        for i in range(length):

            if word[i] == self.previousWord[i]:

                LengthOfCommonPrefix += 1

            else:

                break

        self.minimize(LengthOfCommonPrefix)

        if len(self.uncheckedNodes) == 0:

            CurrentNode = self.root

        else:

            CurrentNode = self.uncheckedNodes[-1][2]

        ##############################################################

        #Group A skill - Complex OOP - dynamic generation of objects#

        ##############################################################

        for l in word[LengthOfCommonPrefix:]:

            NextNode= DawgNode()

            CurrentNode.children[l] = NextNode

            self.uncheckedNodes.append( (CurrentNode,l,NextNode) )

            CurrentNode =NextNode

        CurrentNode.EndOfWord=True

        self.previousWord=word

    ###############################################

    #Group B skill - simple user defined algorithm#

    #Group A skill - graph traversal              #

    ###############################################

    def minimize(self, min):

        for i in range(len(self.uncheckedNodes)-1,min-1,-1):

            parent, letter, child = self.uncheckedNodes.pop()

            if child in self.minimisedNodes:

                parent.children[letter]= self.minimisedNodes[child]

            else:

                self.minimisedNodes[child]= child

    def finish(self):

        self.minimize(0)

    ###############################################

    #Group B skill - simple user defined algorithm#

    #Group A skill - graph traversal              #

    ###############################################

    def search(self,word):

        node=self.root

        for l in word:

            if l not in node.children:

                return False

            node = node.children[l]

        return node.EndOfWord

#### Languages.py

from DAWG import Dawg

from DAWG import DawgNode

from pathlib import Path

class Dictionary:

    def \_\_init\_\_(self):

        self.\_Dawg=None

        ###########################################

        #Group A skill - Complex OOP - Composition#

        #Group B skill - Dictionaries            #

        ###########################################

        self.\_languages={"English":English(),"Spanish":Spanish()}

        self.\_pointsDict=None

        self.\_bag=[]

    ############################################################

    #Group A skill - Dynamic generation of complex OOP objects#

    #Group B skill - Reading from text files                  #

    ###########################################################

    def CreateDawg(self,filename:str):

        self.\_Dawg=Dawg()

        file1 = open(filename,"r")

        lines = file1.readlines()

        for line in lines:

            self.\_Dawg.insert(str(line).strip().upper())

    def search(self,word :str):

        match =self.\_Dawg.search(word.upper())

        return match

    def updateLanguage(self,l:str):

        self.\_bag = self.\_languages[l].returnBag()

        self.\_pointsDict = self.\_languages[l].returnPointsDict()

        self.CreateDawg(self.\_languages[l].returnFilename())

    def getBag(self):

        return self.\_bag

    def getPointsDict(self):

        return self.\_pointsDict

class Language:

    def \_\_init\_\_(self):

        self.\_bag=None

        self.\_pointsDict=None

        self.\_filename=None

    def returnBag(self):

        return self.\_bag

    def returnPointsDict(self):

        return self.\_pointsDict

    def returnFilename(self):

        return self.\_filename

#######################################

#Group A skil -Complex OOP inehritance#

#######################################

class English(Language):

    def \_\_init\_\_(self):

        self.\_bag=[]

        #################################

        #Group A skill - list operations#

        #################################

        for i in range(12):

            self.\_bag.append("E")

            if i<9:

                self.\_bag.append("I")

                self.\_bag.append("A")

            if i<8:

                self.\_bag.append("O")

            if i<6:

                self.\_bag.append("N")

                self.\_bag.append("R")

                self.\_bag.append("T")

            if i <4:

                self.\_bag.append("D")

                self.\_bag.append("L")

                self.\_bag.append("S")

                self.\_bag.append("U")

            if i <3:

                self.\_bag.append("G")

            if i<2:

                self.\_bag.append("B")

                self.\_bag.append("C")

                self.\_bag.append("F")

                self.\_bag.append("H")

                self.\_bag.append("M")

                self.\_bag.append("P")

                self.\_bag.append("V")

                self.\_bag.append("W")

                self.\_bag.append("Y")

                self.\_bag.append("blank")

            if i<1:

                self.\_bag.append("J");self.\_bag.append("K");self.\_bag.append("Q");self.\_bag.append("X");self.\_bag.append("Z")

        ##############################

        #Group B skill - Dictionaries#

        ##############################

        self.\_pointsDict = {"blank":0,"A":1,"E":1,"I":1,"L":1,"N":1,"O":1,"S":1,"T":1,"U":1,"D":2,"G":2,"B":3,"C":3,"M":3,"P":3,"F":4,"H":4,"V":4,"W":4,"Y":4,"K":5,"J":8,"X":8,"Q":10,"Z":10,"R":1}

        self.\_filename= "English.txt"

###########################################

#Group A skill - complex OOP - inheritance#

##########################################

class Spanish(Language):

    #################################

    #Group A skill - list operations#

    #################################

    def \_\_init\_\_(self):

        self.\_bag=[]

        for i in range(12):

            self.\_bag.append("A")

            self.\_bag.append("E")

            if i < 9:

                self.\_bag.append("O")

            if i <6:

                self.\_bag.append("S")

                self.\_bag.append("I")

            if i <5:

                self.\_bag.append("D")

                self.\_bag.append("U")

                self.\_bag.append("N")

                self.\_bag.append("R")

            if i <4:

                self.\_bag.append("C")

                self.\_bag.append("L")

                self.\_bag.append("T")

            if i<2:

                self.\_bag.append("B")

                self.\_bag.append("G")

                self.\_bag.append("H")

                self.\_bag.append("M")

                self.\_bag.append("P")

                self.\_bag.append("blank")

            if i<1:

                self.\_bag.append("CH")

                self.\_bag.append("F")

                self.\_bag.append("J")

                self.\_bag.append("LL")

                self.\_bag.append("Ñ")

                self.\_bag.append("Q")

                self.\_bag.append("RR")

                self.\_bag.append("V")

                self.\_bag.append("X")

                self.\_bag.append("Y")

                self.\_bag.append("Z")

        ##############################

        #Group B skill - Dictionaries#

        ##############################

        self.\_pointsDict = {"blank":0,"Z":10,"Y":4,"X":8,"V":4,"U":1,"T":1,"S":1,"RR":8,"R":1,"Q":5,"P":3,"O":1,"Ñ":8,"N":1,"M":3,"LL":8,"L":1,"J":8,"I":1,"H":4,"G":2,"F":4,"E":1,"D":2,"CH":5,"C":3,"B":3,"A":1}

        self.\_filename="Spanish.txt"

        self.\_\_TrimSpanishDict()

    def \_\_TrimSpanishDict(self):

        path = Path('./Spanish.txt')

        if path.is\_file():

            pass

        else:

            file=open("Spanish Untrimmed.txt","r",encoding="utf-8",errors="ignore")

            spanish = open("Spanish.txt","a")

            lines=file.readlines()

            for i in range(1,len(lines)):

                flag = False

                line =""

                for j in range( len(lines[i])):

                    char =lines[i][j]

                    if char == "ñ":

                        print("Check")

                        flag=True

                    if char== "á":

                        char= "a"

                    elif char == "é":

                        char="e"

                    elif char ==  "í":

                        char="i"

                    elif char ==  "ó":

                        char="o"

                    elif char ==  "ú":

                        char="u"

                    line+=char

                if flag:

                    print(line)

                    print(line.upper())

                spanish.write(f"{line.upper()}")

#### Game.py

from Languages import Dictionary

import random

from database import Account

from typing import Optional

from copy import deepcopy

class Game:

    def \_\_init\_\_ (self,user :Optional [Account] or None):

        ##########################

        #Group B skill - 2D array#

        ##########################

        self.\_board = [[" "]\*15 for y in range (15)]

        self.\_boardPoints= [["TWS",None,None,"DLS",None,None,None,"TWS",None,None,None,"DLS",None,None,"TWS"],

        [None,"DWS",None,None,None,"TLS",None,None,None,"TLS",None,None,None,"DWS",None],

        [None,None,"DWS",None,None,None,"DLS",None,"DLS",None,None,None,"DWS",None,None],

        ["DLS",None,None,"DWS",None,None,None,"DLS",None,None,None,"DWS",None,None,"DLS"],

        [None,None,None,None,"DWS",None,None,None,None,None,"DWS",None,None,None,None],

        [None,"TLS",None,None,None,"TLS",None,None,None,"TLS",None,None,None,"TLS",None],

        [None,None,"DLS",None,None,None,"DLS",None,"DLS",None,None,None,"DLS",None,None],

        ["TWS",None,None,"DLS",None,None,None,"DWS",None,None,None,"DLS",None,None,"TWS"],

        [None,None,"DLS",None,None,None,"DLS",None,"DLS",None,None,None,"DLS",None,None],

        [None,"TLS",None,None,None,"TLS",None,None,None,"TLS",None,None,None,"TLS",None],

        [None,None,None,None,"DWS",None,None,None,None,None,"DWS",None,None,None,None],

        ["DLS",None,None,"DWS",None,None,None,"DLS",None,None,None,"DWS",None,None,"DLS"],

        [None,None,"DWS",None,None,None,"DLS",None,"DLS",None,None,None,"DWS",None,None],

        [None,"DWS",None,None,None,"TLS",None,None,None,"TLS",None,None,None,"DWS",None],

        ["TWS",None,None,"DLS",None,None,None,"TWS",None,None,None,"DLS",None,None,"TWS"]]

        self.\_numPlayers=2

        self.\_NoOfTurn=0

        self.\_pTurn=0

        ##########################################

        #Group A skill - Complex OOP -Composition#

        ##########################################

        self.\_dict= Dictionary()

        self.\_currBag =None

        self.players=[]

        self.formedWords=[]

        self.\_pointsDict ={}

        self.\_numPasses = 0

        self.\_language = None

        self.\_turnOfFirstPass=-9999

        self.\_gameEndFlag = False

        self.\_passFlag=False

        self.user=user

    def SaveGame(self):

        if self.user.getAccount() != None:

            board = deepcopy(self.\_board)

            for i in range(len(board)):

                board[i] =  ','.join(board[i])

            saveboard = '.'.join(board)

            points=[]

            for i in range(len(self.players)):

                points.append(str(self.players[i].getPoints()))

            savepoints=','.join(points)

            bag =",".join( self.\_currBag)

            rackP1 =",".join(self.players[0].displayRack())

            rackP2 =",".join(self.players[1].displayRack())

            self.user.SaveGame(self.\_NoOfTurn,saveboard,savepoints,bag,self.\_language,rackP1,rackP2)

    def LoadGame(self,GameID):

        self.addPlayers(2)

        result =self.user.LoadGame(GameID)

        turn,board,points,bag,language,rackP1,rackP2 = result[0]

        self.\_NoOfTurn=turn

        self.\_pTurn = self.\_NoOfTurn%self.\_numPlayers

        board = board.split(".")

        for i in range(len(board)):

            board[i]=board[i].split(",")

        self.\_board= board

        points = points.split(',')

        self.players[0].updatePoints(int(points[0]))

        self.players[1].updatePoints(int(points[1]))

        self.updateLanguage(language)

        self.\_currBag = bag.split(",")

        rackP1 = rackP1.split(",")

        rackP2 = rackP2.split(",")

        self.players[0].updateRack(rackP1)

        self.players[1].updateRack(rackP2)

    def increaseNumPasses(self):

        if self.\_numPasses ==0:

            self.\_turnOfFirstPass=self.\_NoOfTurn

        self.\_numPasses+= 1

    def GetPointsBoard(self):

        return self.\_boardPoints

    def lenBag(self):

        return len(self.\_currBag)

    def resetPasses(self):

        self.\_numPasses=0

        self.\_turnOfFirstPass=-9999

    def updateLanguage(self,langauge:str):

        self.\_dict.updateLanguage(langauge)

        self.\_currBag= self.\_dict.getBag()

        self.\_pointsDict=self.\_dict.getPointsDict()

        self.\_language = langauge

    def incrementTurn(self):

        self.updatePlayerRack(self.\_pTurn)

        self.\_NoOfTurn+=1

        self.\_pTurn = self.\_NoOfTurn % self.\_numPlayers

        if self.\_NoOfTurn-self.\_turnOfFirstPass==2:

            if self.\_numPasses//2 == 1:

                self.\_passFlag=True

            else:

                self.resetPasses()

    def ChangeNumPlayers(self,numPlayers):

        self.\_numPlayers= numPlayers

    def getTurnNo(self):

        return(self.\_NoOfTurn)

    def getPTurn(self):

        return(self.\_pTurn)

    def getBoard(self):

        return self.\_board

    def getPointsDict(self):

        return self.\_pointsDict

    def getNumPlayers(self):

        return self.\_numPlayers

    ##########################################

    #Group A skill - Complex OOP -Composition#

    ##########################################

    def addPlayers(self,numPlayers):

        for i in range(numPlayers):

            self.players.append(Player())

            self.updatePlayerRack(i)

        self.\_numPlayers= numPlayers

    #################################

    #Group A skill - list operations#

    #################################

    def updatePlayerRack(self,n):

        rack=[]

        rack = deepcopy(self.players[n].displayRack())

        for i in range(7-len(rack)):

            if self.\_currBag:

                x=random.randint(0,len(self.\_currBag)-1)

                rack.append(self.\_currBag[x])

                self.\_currBag.pop(x)

            else:

                break

        self.players[n].updateRack(deepcopy(rack))

    ###############################################

    #Group B skill -simple user defined algorithms#

    ###############################################

    def isGameOver(self):

        if not self.\_currBag and  len(self.players[(self.\_NoOfTurn-1)%self.\_numPlayers].displayRack())==0  or self.\_passFlag==True:

            self.deductPoints()

            return True

        else:

            return False

    ################################################

    #Group A skill - complex user defined algorithm#

   ################################################

    def validateTurn(self,currMoves):

        pRack =deepcopy(self.players[self.\_pTurn].displayRack())

        playedCentre = False

        for i in range (len(currMoves)):

            if currMoves[i][0] not in pRack:

                if "blank" not in pRack:

                    print("Letters not in rack")

                    return False

                else:

                    pRack.remove("blank")

            else:

                pRack.remove(currMoves[i][0])

            if currMoves[i][1] >= 15 or currMoves[i][2] >= 15 or currMoves[i][1]<0 or currMoves[i][2] < 0:

                print("Coordinates out of range")

                return False

            if currMoves[i][1] ==7 and currMoves[i][2] == 7:

                playedCentre = True

        if self.\_NoOfTurn == 0 or self.\_NoOfTurn==1 and self.\_numPasses==1:

            if not playedCentre:

                return False

        boardCopy = deepcopy(self.\_board)

        vert= True

        horz = True

        if len(currMoves) ==1:

            if boardCopy[ currMoves[0][1]+1][currMoves[0][2]] != " " or boardCopy[ currMoves[0][1]-1][currMoves[0][2]] != " ":

                horz = False

            elif boardCopy[ currMoves[0][1]][currMoves[0][2]-1] != " " or boardCopy[ currMoves[0][1]][currMoves[0][2]+1] != " ":

                vert= False

        for i in range(len(currMoves)-1):

            if horz and currMoves[i][1] == currMoves[i+1][1]:

                vert = False

            elif vert and currMoves[i][2] == currMoves[i+1][2]:

                horz = False

            else:

                print("Coordinates not in row")

                return False

        for i in range(len(currMoves)):

            if boardCopy[currMoves[i][1]][currMoves[i][2]] != " ":

                print("A piece already occupies that square")

                return False

            boardCopy[currMoves[i][1]][currMoves[i][2]] = currMoves[i][0]

        match =1

        count = 1

        if vert == True:

            while True:

                horizontalValue= currMoves[0][2]

                if currMoves[0][1] + count >= 15 or boardCopy[currMoves[0][1]+count][horizontalValue] == " ":

                    break

                for i in range(len(currMoves)):

                    if currMoves[0][1]+count == currMoves[i][1]:

                        match += 1

                count += 1

            count = 1

            while True:

                horizontalValue= currMoves[i][2]

                if currMoves[0][1] - count <0 or boardCopy[currMoves[0][1]-count][horizontalValue] == " ":

                    break

                for i in range(len(currMoves)):

                    if currMoves[0][1]-count == currMoves[i][1]:

                        match += 1

                count += 1

        elif horz == True:

            while True:

                verticalValue= currMoves[0][1]

                if currMoves[0][2] + count >= 15 or boardCopy[verticalValue][currMoves[0][2]+count] == " ":

                    break

                for i in range(len(currMoves)):

                    if currMoves[0][2]+count == currMoves[i][2]:

                        match += 1

                count += 1

            count =1

            while True:

                verticalValue= currMoves[0][1]

                if currMoves[0][2] - count <0 or boardCopy[verticalValue][currMoves[0][2]-count] == " ":

                    break

                for i in range(len(currMoves)):

                    if currMoves[0][2]-count == currMoves[i][2]:

                        match += 1

                count += 1

        if match != len(currMoves):

            print("Letters not connected")

            return False

        word = ""

        if vert == True:

            for i in range(currMoves[0][1],15):

                if boardCopy[i][currMoves[0][2]] == " ":

                    break

                else:

                    word = word + boardCopy[i][currMoves[0][2]]

            for i in range(currMoves[0][1]-1,-1,-1):

                if boardCopy[i][currMoves[0][2]] == " ":

                    break

                else:

                    word =  boardCopy[i][currMoves[0][2]] + word

            if self.\_dict.search(word) == False:

                print("Flag 1")

                return False

            for i in range(len(currMoves)):

                word =""

                for j in range(currMoves[i][2],15):

                    if boardCopy[currMoves[i][1]][j] == " ":

                        break

                    else:

                        word = word + boardCopy[currMoves[i][1]][j]

                for j in range(currMoves[i][2]-1,-1,-1):

                    if boardCopy[currMoves[i][1]][j] == " ":

                        break

                    else:

                        word = boardCopy[ currMoves[i][1] ][j] + word

                if len(word) > 1:

                    if self.\_dict.search(word)==False:

                        print("Flag 2")

                       # print(word)

                        return False

        elif horz == True:

            for i in range(currMoves[0][2],15):

                if boardCopy[currMoves[0][1]][i] == " ":

                    break

                else:

                    word = word + boardCopy[currMoves[0][1]][i]

            for i in range(currMoves[0][2]-1,-1,-1):

                if boardCopy[currMoves[0][1]][i] == " ":

                    break

                else:

                    word = boardCopy[currMoves[0][1]][i] + word

            if self.\_dict.search(word) == False:

                print(word)

                return False

            for i in range(len(currMoves)):

                word = ""

                for j in range(currMoves[i][1],15):

                    if boardCopy[j][currMoves[i][2]] == " ":

                        break

                    else:

                        word = word + boardCopy[j][currMoves[i][2]]

                for j in range(currMoves[i][1]-1,-1,-1):

                    if boardCopy[j][ currMoves[i][2] ] == " ":

                        break

                    else:

                        word =  boardCopy[j][currMoves[i][2]] + word

                if len(word)> 1:

                    if self.\_dict.search(word) == False:

                        return False

        if self.\_NoOfTurn == 0 or self.\_NoOfTurn==1 and self.\_numPasses==1:

            match = False

            for i in range(len(currMoves)):

                try:

                    x = [self.\_board[currMoves[i][1]][currMoves[i][2]+1],currMoves[i][1],currMoves[i][2]+1]

                    if x not in currMoves:

                        match = True

                        break

                except:

                    pass

                try:

                    x = [self.\_board[currMoves[i][1]][currMoves[i][2]-1],currMoves[i][1],currMoves[i][2]-1]

                    if x not in currMoves:

                        match = True

                        break

                except:

                    pass

                try:

                    x = [self.\_board[currMoves[i][1]+1][currMoves[i][2]],currMoves[i][1]+1,currMoves[i][2]]

                    if x not in currMoves:

                        match = True

                        break

                except:

                    pass

                try:

                    x = [self.\_board[currMoves[i][1]-1][currMoves[i][2]],[currMoves[i][1]-1],[currMoves[i][2]]]

                    if x not in currMoves:

                        match = True

                        break

                except:

                    pass

            if match == False:

                return False

        self.\_board = boardCopy

        self.players[self.\_pTurn].updateRack(pRack)

        return True

    ################################################

    #Group A skill - complex user defined algorithm#

   ################################################

    def calculatePoints(self,currMoves):

        if len(currMoves)==1:

            vert = False

            horz = True

        elif currMoves[0][1] == currMoves[1][1]:

            vert = False

            horz = True

        else:

            vert = True

            horz = False

        totalPoints=0

        if vert == True:

            vMultiplier = 1

            for i in range(len(currMoves)):

                wordPoints = 0

                hMultiplier = 1

                lMultiplier =1

                word = ""

                match self.\_boardPoints[currMoves[i][1]][currMoves[i][2]] :

                    case "TWS":

                        hMultiplier = hMultiplier\*3

                        vMultiplier= vMultiplier\*3

                    case "DWS":

                        hMultiplier =  hMultiplier\*2

                        vMultiplier = vMultiplier\*2

                    case "TLS":

                        lMultiplier = 3

                    case "DLS":

                        lMultiplier = 2

                if not self.\_board[currMoves[i][1]][currMoves[i][2]].islower():

                    wordPoints += self.\_pointsDict[ self.\_board [currMoves[i][1]] [currMoves[i][2]] ] \*lMultiplier

                word+= self.\_board [currMoves[i][1]] [currMoves[i][2]]

                for j in range(currMoves[i][2]+1,15):

                    if self.\_board[currMoves[i][1]][j] == " ":

                        break

                    elif self.\_board[currMoves[i][1]][j].islower():

                        word+=self.\_board[currMoves[i][1]][j]

                    else:

                        wordPoints+=self.\_pointsDict[ self.\_board[currMoves[i][1]][j]]

                        word+=self.\_board[currMoves[i][1]][j]

                for j in range(currMoves[i][2]-1,-1,-1):

                    if self.\_board[currMoves[i][1]][j] == " ":

                        break

                    elif self.\_board[currMoves[i][1]][j].islower():

                        word+=self.\_board[currMoves[i][1]][j]

                    else:

                        wordPoints+=self.\_pointsDict[ self.\_board[currMoves[i][1]][j]]

                        word+=self.\_board[currMoves[i][1]][j]

                wordPoints=wordPoints\*hMultiplier

                if len(word)<=1:

                    wordPoints=0

                totalPoints+=wordPoints

            wordPoints = 0

            for i in range(currMoves[0][1],15):

                lMultiplier = 1

                if self.\_board[i][currMoves[0][2]] == " ":

                    break

                elif self.\_board[i][currMoves[0][2]].islower():

                    pass

                else:

                    for j in range(len(currMoves)):

                        if currMoves[j][1]==i:

                            match self.\_boardPoints[i][currMoves[0][2]]:

                                case "TLS":

                                    lMultiplier = 3

                                case "DLS":

                                    lMultiplier = 2

                    wordPoints+=self.\_pointsDict[self.\_board[i][currMoves[0][2]]]\*lMultiplier

            for i in range(currMoves[0][1]-1,-1,-1):

                if self.\_board[i][currMoves[0][2]] == " ":

                    break

                elif self.\_board[i][currMoves[0][2]].islower():

                    pass

                else:

                    for j in range(len(currMoves)):

                        if currMoves[j][1]==i:

                            match self.\_boardPoints[i][currMoves[0][2]] :

                                case "TLS":

                                    lMultiplier = 3

                                case "DLS":

                                    lMultiplier = 2

                    wordPoints+=self.\_pointsDict[self.\_board[i][currMoves[0][2]]]\*lMultiplier

            wordPoints=wordPoints\*vMultiplier

            totalPoints+=wordPoints

        if horz == True:

            hMultiplier =1

            for i in range(len(currMoves)):

                wordPoints=0

                vMultiplier=1

                lMultiplier = 1

                word = ""

                match self.\_boardPoints[currMoves[i][1]][currMoves[i][2]] :

                    case "TWS":

                        vMultiplier = vMultiplier\*3

                        hMultiplier= hMultiplier\*3

                    case "DWS":

                        vMultiplier =  vMultiplier\*2

                        hMultiplier = hMultiplier\*2

                    case "TLS":

                        lMultiplier = 3

                    case "DLS":

                        lMultiplier = 2

                if not self.\_board[currMoves[i][1]][currMoves[i][2]].islower():

                    wordPoints += self.\_pointsDict[ self.\_board [currMoves[i][1]] [currMoves[i][2]] ] \*lMultiplier

                word+=self.\_board[currMoves[i][1]][currMoves[i][2]]

                for j in range(currMoves[i][1]+1,15):

                    if self.\_board[j][currMoves[i][2]] == " ":

                        break

                    elif self.\_board[j][ currMoves[i][2] ].islower():

                        word+= self.\_board[j][ currMoves[i][2]]

                    else:

                        wordPoints+=self.\_pointsDict[self.\_board[j][currMoves[i][2]]]

                        word +=self.\_board[j][currMoves[i][2]]

                for j in range(currMoves[i][1]-1,-1,-1):

                    if self.\_board[j][ currMoves[i][2]] == " ":

                        break

                    elif self.\_board[j][ currMoves[i][2] ].islower():

                        word+= self.\_board[j][ currMoves[i][2]]

                    else:

                        wordPoints+=self.\_pointsDict[self.\_board[j][currMoves[i][2]]]

                        word +=self.\_board[j][ currMoves[i][2]]

                if len(word) <= 1:

                    wordPoints=0

                wordPoints =wordPoints\*vMultiplier

                totalPoints+=wordPoints

            wordPoints = 0

            lMultiplier=1

            for i in range(currMoves[0][2],15):

                if self.\_board[currMoves[0][1]][i] == " ":

                    break

                elif self.\_board[currMoves[0][1]][i].islower():

                    pass

                else:

                    for j in range(len(currMoves)):

                        if currMoves[j][2]==i:

                            match self.\_boardPoints[currMoves[0][1]][i] :

                                case "TLS":

                                    lMultiplier = 3

                                case "DLS":

                                    lMultiplier = 2

                    wordPoints += self.\_pointsDict[self.\_board[currMoves[0][1]][i]]\*lMultiplier

            for i in range(currMoves[0][2]-1,-1,-1):

                if self.\_board[currMoves[0][1]][i] == " ":

                    break

                elif self.\_board[currMoves[0][1]][i].islower():

                    pass

                else:

                    for j in range(len(currMoves)):

                        if currMoves[j][2]==i:

                            match self.\_boardPoints[currMoves[0][1]][i] :

                                case "TLS":

                                    lMultiplier = 3

                                case "DLS":

                                    lMultiplier = 2

                    wordPoints += self.\_pointsDict[self.\_board[currMoves[0][1]][i]]\*lMultiplier

            wordPoints=wordPoints\*hMultiplier

            totalPoints+=wordPoints

        if len(currMoves)==7:

            totalPoints+=50

        self.players[self.\_pTurn].updatePoints(totalPoints)

        return totalPoints

    ###############################################

    #Group B skill -simple user defined algorithms#

    ###############################################

    def findWinner(self):

        Max = -99999999999

        index= 0

        flag = False

        for i in range (self.\_numPlayers):

            if self.players[i].getPoints() > Max:

                Max = self.players[i].getPoints()

                index = i

                flag = False

            elif self.players[i].getPoints() ==Max:

                flag = True

        if flag:

            Max = 0

            index = None

            for i in range (self.\_numPlayers):

                if self.players[i].getPreEndPoints() > Max:

                    Max = self.players[i].getPreEndPoints()

                    index = i

        self.updateWinsAndLosses(index)

        return index

    def getBoard(self):

        return self.\_board

    def deductPoints(self):

        points = 0

        index = None

        for i in range (self.\_numPlayers):

            self.players[i].updatePreEndPoints(self.players[i].getPoints())

            if len(self.players[i].displayRack()) == 0:

                self.players[i].updatePoints(0)

            else:

                playerPoints = 0

                for j in self.players[i].displayRack():

                    playerPoints += self.\_pointsDict[j]

                points += playerPoints

                self.players[i].updatePoints(-playerPoints)

    def updateWinsAndLosses(self,winner):

        if self.user != None:

            if self.user.getAccount()!=None:

                if winner == 0:

                    self.user.UpdateWin()

                else:

                    self.user.UpdateLoss()

    def printBoard(self):

        x = "   "

        for i in range(10):

            x +=(" "+str(i)+"   ")

        for i in range (10,15):

            x +=(" "+str(i)+"  ")

        print(x)

        for i in range(10):

            print(str(i)+" "+str(self.\_board[i]))

        for i in range(10,15):

            print(str(i)+str(self.\_board[i]))

class Player:

    def \_\_init\_\_(self):

        self.\_points = 0

        self.\_rack = []

        self.\_preEndPoints = 0

    def updateRack(self,NewRack):

        self.\_rack=NewRack

    def displayRack(self):

        return self.\_rack

    def updatePoints(self,points):

        self.\_points+= points

    def getPoints(self):

        return self.\_points

    def updatePreEndPoints(self,points):

        self.\_preEndPoints+= points

    def getPreEndPoints(self):

        return self.\_preEndPoints

#### UI.py

from Game import Game

from abc import ABC, abstractmethod

import pygame

from tkinter import \*

from database import Account

class UI(ABC):

    ############################

    #Complex OOP - polymorphism#

    ############################

    def run(self):

        raise NotImplementedError("UI not implemented")

class Terminal(UI):

    def \_\_init\_\_(self):

        pass

    #####################################

    #Group A skill - recursive algothims#

    #####################################

    def turn(self,game:Game):

        print(game.printBoard())

        print(f"Player 1: {game.players[0].getPoints()} points: ")

        print(f"Player 2: {game.players[1].getPoints()} points: ")

        print(f"Player {game.getPTurn()+1}'s turn")

        print(f"Player {game.getPTurn()+1}'s rack: {game.players[game.getPTurn()].displayRack()}")

        print(f"Bag : {game.lenBag()} tiles")

        while True:

            currMoves = []

            while True:

                while True:

                    l = str(input(f"Player{game.getPTurn()+1} enter letter of next move and then Y and X coordinates on the following lines or -1 to end turn: "))

                    if len(l)==1 or l == "-1" or l=="blank":

                        break

                if l == "blank":

                    l=str(input("Enter the letter the blank tile represents"))

                    l = l.lower()

                else:

                    l = l.upper()

                if str(l) == "-1":

                    break

                Y = int(input("Enter Y coordinate of turn: "))

                X = int(input("Enter X coordinate of turn: "))

                currMoves.append([l,Y,X])

            if len(currMoves) == 0:

                game.increaseNumPasses()

                break

            if not game.validateTurn(currMoves):

                print("Invalid Move")

            else:

                game.calculatePoints(currMoves)

                break

        game.incrementTurn()

        if game.isGameOver():

            winner = game.findWinner()

            print(f"Player 1: {game.players[0].getPoints()} points: ")

            print(f"Player 2: {game.players[1].getPoints()} points: ")

            print("\*"\*30)

            print(f"Player {winner+1} won")

            print("\*"\*30)

            exit()

        self.turn(game)

    ############################

    #Complex OOP - polymorphism#

    ############################

    def run(self):

        game = Game(None)

        numPlayers = 2

        game.updateLanguage("English")

        game.addPlayers(numPlayers)

        self.turn(game)

class GUI():

    def \_\_init\_\_(self):

        self.\_root=Tk()

        self.\_loginWindow = None

        self.\_LEntryU=None  #login window - username entry box

        self.\_LEntryP=None # login window - password entry box

        self.\_createAccountWindow=None

        self.\_CEntryU=None  # create account window - username entry box

        self.\_CEntryP=None  # create account window - password entry box

        ##########################################

        #Group A skill - Complex OOP -Composition#

        ##########################################

        self.\_account = Account()

        self.\_screen=None

        self.\_languageWindow = None

        self.\_blank=None

        self.\_handleBlankWindow = None

        self.\_blankEntry = None

        self.\_inavlidTurnFlag=False

    def loginErrorUsername(self):

        lError= Toplevel(self.\_root)

        lError.title("Scrabble-Login Error")

        lError.geometry("200x80")

        Label(lError,text="Error:\n Your username/password is\n incorrect or does not exist").pack()

        Button(lError,text ="Dismiss",command=lError.destroy).pack()

    def AccountError(self):

        aError = Toplevel(self.\_root)

        aError.title("Scrabble - Account Creation Error")

        aError.geometry("200x80")

        Label(aError,text="Error:\n An account with that\n username already exists").pack()

        Button(aError,text="Dismiss",command=aError.destroy).pack()

    def checkLogin(self):

        username = self.\_LEntryU.get()

        password = self.\_LEntryP.get()

        if username.strip() == "" or password.strip()=="":

            return False

        check =self.\_account.Login(username,password)

        if check:

            self.quitLogin()

            self.loginSuccesful()

        else:

            self.loginErrorUsername()

    def loginSuccesful(self):

        loginSuccesful= Toplevel(self.\_root)

        loginSuccesful.title("Scrabble - Login Succesful")

        loginSuccesful.geometry("200x80")

        Label(loginSuccesful,text="Login \n Succesful \n").pack()

        Button(loginSuccesful,text="Dismiss",command=loginSuccesful.destroy).pack()

    def quitLogin(self):

        self.\_loginWindow.destroy()

        self.\_loginWindow=None

    def login(self):

        if self.\_loginWindow != None:

            return False

        self.\_loginWindow=Toplevel(self.\_root)

        self.\_loginWindow.title("Scrabble- Login")

        self.\_loginWindow.geometry("300x150")

        canvas1 = Canvas(self.\_loginWindow, width=300, height=150)

        label1 = Label(self.\_loginWindow,text ="Enter Username")

        label1.pack()

        canvas1.create\_window(150,10,window=label1)

        self.\_LEntryU = Entry(self.\_loginWindow, width = 35)

        self.\_LEntryU.pack()

        canvas1.create\_window(150,30, window=self.\_LEntryU)

        label2 = Label(self.\_loginWindow,text="Enter Password")

        label2.pack()

        canvas1.create\_window(150,50,window = label2)

        self.\_LEntryP = Entry(self.\_loginWindow, width= 35)

        self.\_LEntryP.pack()

        canvas1.create\_window(150,70,window=self.\_LEntryP)

        button1=Button(self.\_loginWindow,text="Login",width=10,command=self.checkLogin)

        button1.pack()

        canvas1.create\_window(150,100,window = button1)

        button2=Button(self.\_loginWindow,text="Quit",width=10,command=self.quitLogin)

        button2.pack()

        canvas1.create\_window(150,135,window = button2)

        canvas1.pack()

    def CreateAccountWindow(self):

        if self.\_createAccountWindow:

            return False

        self.\_createAccountWindow=Toplevel(self.\_root)

        self.\_createAccountWindow.title("Scrabble- Create Account")

        self.\_createAccountWindow.geometry("300x150")

        canvas1 = Canvas(self.\_createAccountWindow, width=300, height=150)

        label1 = Label(self.\_createAccountWindow,text ="Enter Username")

        label1.pack()

        canvas1.create\_window(150,10,window=label1)

        self.\_CEntryU = Entry(self.\_createAccountWindow, width = 35)

        self.\_CEntryU.pack()

        canvas1.create\_window(150,30, window=self.\_CEntryU)

        label2 = Label(self.\_createAccountWindow,text="Enter Password")

        label2.pack()

        canvas1.create\_window(150,50,window = label2)

        self.\_CEntryP = Entry(self.\_createAccountWindow, width= 35)

        self.\_CEntryP.pack()

        canvas1.create\_window(150,70,window=self.\_CEntryP)

        button1=Button(self.\_createAccountWindow,text="Create \n Account",width=10,command=self.checkCreateAccount)

        button1.pack()

        canvas1.create\_window(150,100,window = button1)

        button2=Button(self.\_createAccountWindow,text="Quit",width=10,command=self.quitCreateAccount)

        button2.pack()

        canvas1.create\_window(150,135,window = button2)

        canvas1.pack()

    def quitCreateAccount(self):

        self.\_createAccountWindow.destroy()

        self.\_createAccountWindow=None

    def checkCreateAccount(self):

        username = self.\_CEntryU.get()

        password = self.\_CEntryP.get()

        if username.strip() == "" or password.strip()=="":

            return False

        check = self.\_account.CreateAccount(username,password)

        if check:

            self.CreateAccountSuccesful()

            self.quitCreateAccount()

        else:

            self.AccountError()

    def CreateAccountSuccesful(self):

        createAccountSuccesful= Toplevel(self.\_root)

        createAccountSuccesful.title("Scrabble - Create Account Succesful")

        createAccountSuccesful.geometry("200x80")

        Label(createAccountSuccesful,text="Account Created \n Succesfuly \n").pack()

        Button(createAccountSuccesful,text="Dismiss",command=createAccountSuccesful.destroy).pack()

    def SelectLanguage(self):

        if self.\_languageWindow:

            return False

        self.\_languageWindow=Toplevel(self.\_root)

        self.\_languageWindow.title("Scrabble - Play Game - Select Language")

        self.\_languageWindow.geometry("200x250")

        Label(self.\_languageWindow,text = "Select language:").pack(pady=1)

        Button(self.\_languageWindow,text="English",width=20,height=2,command=self.PlayGameEnglish).pack(pady=20)

        Button(self.\_languageWindow,text="Spanish",width=20,height=2,command =self.PlayGameSpanish).pack()

    def PlayGameEnglish(self):

        self.\_languageWindow.destroy()

        game=Game(self.\_account)

        game.updateLanguage("English")

        game.addPlayers(2)

        self.PlayGame(game)

    def PlayGameSpanish(self):

        self.\_languageWindow.destroy()

        game=Game(self.\_account)

        game.updateLanguage("Spanish")

        game.addPlayers(2)

        self.PlayGame(game)

    def UndoError(self):

        UError=Toplevel(self.\_root)

        UError.title("Scrabble - Play Game - Undo Error")

        UError.geometry("200x60")

        Label(UError,text="Error:\n There are no moves left to undo").pack()

        Button(UError,text ="Dismiss",command=UError.destroy).pack()

    def AccountCheck(self):

        if self.\_account.getAccount():

            self.AccountWindow()

        else:

            self.NotLoggedIn()

    def NotLoggedIn(self):

        AError=Toplevel(self.\_root)

        AError.title("Scrabble - Account - Login Error")

        AError.geometry("200x80")

        Label(AError,text="Error:\n you are not logged in\n").pack()

        Button(AError,text ="Dismiss",command=AError.destroy).pack()

    def AccountWindow(self):

        accountWindow = Toplevel(self.\_root)

        accountWindow.title("Scrabble - Account")

        accountWindow.geometry("300x300")

        result = self.\_account.GetWinLoss()

        games,wins,losses = result[0]

        name =self.\_account.getAccount()

        Label(accountWindow,font=('Helvetica',44),text=f"{name}:").pack()

        Label(accountWindow,font=('Helvetica',12),text=f"Games:{games}").pack()

        Label(accountWindow,font=('Helvetica',12),text=f"Wins:{wins}").pack()

        Label(accountWindow,font=('Helvetica',12),text=f"Losses:{losses}\n").pack()

        Button(accountWindow,text="Load Games",font=('Helvetica',12),command=self.LoadGamesWindow,height=2,width=25).pack()

        Label(accountWindow,text=" ",font=('Helvetica',12)).pack()

        Button(accountWindow,font=('Helvetica',12),text="Quit",height=2,width=15,command=accountWindow.destroy).pack()

    def LoadGamesWindow(self):

        LoadGamesWindow=Toplevel(self.\_root)

        LoadGamesWindow.title("Scrabble - Account - Load Games")

        Label(LoadGamesWindow,font=('Helvetica',30),text="Load Games").pack()

        frameContainer=Frame(LoadGamesWindow)

        canvasCointainer = Canvas(frameContainer)

        frame2=Frame(canvasCointainer)

        scrollbar= Scrollbar(frameContainer,orient="vertical",command=canvasCointainer.yview)

        canvasCointainer.create\_window((0,0),window=frame2,anchor="nw")

        gameList=self.\_account.GetGames()

        print(gameList)

        for i in range(len(gameList)):

            GameID, TurnNo,Scores, Language =gameList[i]

            Button(frame2,text=f"Game {i}: Scores({Scores}) turn {TurnNo}: {Language}",width=50,height=1,font=('Helvetica',12),command=lambda:self.LoadGame(GameID)).pack(pady=5)

        frame2.update()

        canvasCointainer.configure(yscrollcommand=scrollbar.set,scrollregion="0 0 0 %s" % frame2.winfo\_height())

        canvasCointainer.pack(side=LEFT)

        scrollbar.pack(side=RIGHT,fill=Y)

        frameContainer.pack()

        Button(LoadGamesWindow,text="Quit",font=('Helvetica',12),width=15,height=1,command=LoadGamesWindow.destroy).pack(pady=20)

    def LoadGame(self,GameId):

        game = Game(self.\_account)

        game.LoadGame(GameId)

        self.PlayGame(game)

    ############################

    #Complex OOP - polymorphism#

    ############################

    def run(self):

        self.\_root.title("Scrabble - Main Menu")

        frame = Frame(self.\_root)

        frame.pack()

        Button(frame,text='Play Game',width = 40,height =3,command=self.SelectLanguage).pack(fill=X)

        Button(frame,text='Account',width = 40,height =3,command = self.AccountCheck).pack(fill=X)

        Button(frame,text='Create Acount',width = 40,height =3,command=self.CreateAccountWindow).pack(fill=X)

        Button(frame,text='Login',width = 40,height =3, command=self.login).pack(fill=X)

        Button(frame,text='Quit',command=self.\_root.quit,width = 40,height =3).pack(fill=X)

        self.\_root.mainloop()

    def drawBackround(self,game :Game):

        board\_tiles = [[0 for row in range(15)] for col in range(15)]

        block\_size = 50

        backboard=game.GetPointsBoard()

        board = game.getBoard()

        buttonColourLight = (229,229,229)

        buttonColourDark=(122, 122, 122)

        pointsFont = pygame.font.SysFont(None,14)

        font = pygame.font.SysFont(None, 24)

        tileFont =pygame.font.SysFont(None,36)

        titleFont=pygame.font.SysFont(None,48)

        titleFont.set\_underline(True)

        self.\_screen.fill((0,0,0))

        colour=(255,255,255)

        buttonColourLight = (229,229,229)

        for row in range(len(board\_tiles)):

            for col in range(len(board\_tiles[0])):

                if backboard[row][col]=="TWS":

                    txt = font.render("TWS",True,(0,0,0))

                    colour=(40,78,96)

                elif backboard[row][col]=="DWS" and row != 7 and col !=7:

                    colour=(49,155,118)

                    txt = font.render("DWS",True,(0,0,0))

                elif backboard[row][col]=="DLS":

                    txt = font.render("DLS",True,(0,0,0))

                    colour=(217,89,128)

                elif backboard[row][col] =="TLS":

                    txt = font.render("TLS",True,(0,0,0))

                    colour=(99,170,192)

                elif row==7 and col == 7:

                    txt = font.render("Start",True,(0,0,0))

                    colour =(249,155,69)

                else:

                    txt = font.render("",True,(0,0,0))

                    colour=(255,255,255)

                rect = pygame.Rect(col\*(block\_size+1), row\*(block\_size+1), block\_size, block\_size)

                pygame.draw.rect(self.\_screen, colour, rect)

                self.\_screen.blit(txt,((block\_size+1)\*col+6,(block\_size+1)\*row+15))#

                ##############################################

                #Group A skill - Dynamic generation of objects#

                ###############################################

                if board[row][col] != " ":

                    pygame.draw.rect(self.\_screen,buttonColourLight,rect)

                    txt = tileFont.render(board[row][col],True,(0,0,0))

                    self.\_screen.blit(txt,((block\_size+1)\*col+15,(block\_size+1)\*row+15))

                    pointsDict = game.getPointsDict()

                    if board[row][col].islower():

                        char = "0"

                    else:

                        char =str(pointsDict[board[row][col]])

                    txt = pointsFont.render(char,True,(0,0,0))

                    self.\_screen.blit(txt,(rect.x,rect.y))

        txt = font.render(f"{game.lenBag()} tiles left",True,(255,255,255))

        self.\_screen.blit(txt,(765,150))

        if self.\_account.getAccount()!= None:

            img = font.render(f"{self.\_account.getAccount()} : {game.players[0].getPoints()} points", True, (255,255,255))

        else:

            img = font.render(f"Player1:{game.players[0].getPoints()} points", True, (255,255,255))

        self.\_screen.blit(img, (765, 60))

        img=font.render(f'Player2 :{game.players[1].getPoints()} points',True,(255,255,255))

        self.\_screen.blit(img,(765,90))

        turn = game.getPTurn()+1

        if self.\_account.getAccount()!= None:

            if turn == 1:

                img = titleFont.render(f"{self.\_account.getAccount()}'s turn",True,(255,255,255))

            else:

                img = titleFont.render(f"Player{turn}'s turn",True,(255,255,255))

        else:

            img = titleFont.render(f"Player{turn}'s turn",True,(255,255,255))

        self.\_screen.blit(img,(865,10))

        if self.\_inavlidTurnFlag:

            img = titleFont.render(f"Invalid turn",True,(255,255,255))

            self.\_screen.blit(img,(865,250))

    def PlayGame(self,game:Game):

        winner = None

        rack=game.players[game.getPTurn()].displayRack()

        FPS =30

        pygame.init()

        self.\_screen = pygame.display.set\_mode((1200,765))

        movesStack =[]

        board=[[" "]\*15 for y in range (15)]

        pygame.display.set\_caption("Scrabble - Play Scrabble")

        running = True

        board\_tiles = [[0 for row in range(15)] for col in range(15)]

        block\_size = 50

        buttonColourLight = (229,229,229)

        buttonColourDark=(122, 122, 122)

        pointsFont = pygame.font.SysFont(None,14)

        font = pygame.font.SysFont(None, 24)

        tileFont =pygame.font.SysFont(None,36)

        titleFont=pygame.font.SysFont(None,48)

        titleFont.set\_underline(True)

        UIrack=[]

        for i in range(len(rack)):

            char = rack[i]

            if rack[i]=="blank":

                char = "\_"

            txt = tileFont.render(char,True,(0,0,0))

            rec = pygame.Rect(765+40+(block\_size+1)\*i,300,block\_size,block\_size)

            UIrack.append((txt,rec))

            pygame.draw.rect(self.\_screen,buttonColourLight,rec)

            self.\_screen.blit(txt,(765+40+(block\_size+1)\*i+15,300+10))

            pointsDict = game.getPointsDict()

            if rack[i].islower():

                char = "0"

            else:

                char = str(pointsDict[rack[i]])

            txt = pointsFont.render(char,True,(0,0,0))

            self.\_screen.blit(txt,(rec.x,rec.y))

        rectangle\_draging=False

        self.drawBackround(game)

        clock = pygame.time.Clock()

        TileDragged=None

        startCoords=None

        pygame.display.update()

        while running:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    running = False

                if event.type == pygame.MOUSEBUTTONDOWN:

                    ###drag and drop##

                    for i in range(len(UIrack)):

                        txt,rectangle=UIrack[i]

                        if rectangle.collidepoint(event.pos):

                            TileDragged =i

                            rectangle\_draging = True

                            mouse\_x, mouse\_y = event.pos

                            offset\_x = rectangle.x - mouse\_x

                            offset\_y = rectangle.y - mouse\_y

                            startCoords = (rectangle.x,rectangle.y)

                    ####Undo button###

                    #######################################

                    #Group A skill - list/stack operations#

                    #######################################

                    if (883+200)>=mousePos[0]>=883 and (440+50)>= mousePos[1]>=440 :

                        if len(movesStack) != 0:

                            rectangle,endCoords,startCoords = movesStack.pop()

                            rectangle.x = startCoords[0]

                            rectangle.y = startCoords[1]

                    ####next turn button#####

                    if (883+200)>=mousePos[0]>=883 and (380+50)>= mousePos[1]>=380:

                        currMoves=[]

                        for i in range(len(rack)):

                            txt,rectangle=UIrack[i]

                            x = rectangle.x //(block\_size+1)

                            y= rectangle.y//(block\_size+1)

                            if 14>=x>=0 and 14>=y>=0:

                                char = rack[i]

                                if rack[i]=="blank":

                                    self.\_blank = None

                                    self.HandleBlank()

                                    char = self.\_blank.lower()

                                currMoves.append([char,y,x])

                        if game.validateTurn(currMoves):

                            game.calculatePoints(currMoves)

                            game.incrementTurn()

                            if game.isGameOver():

                                i = game.findWinner()

                                if i == 0 and game.user != None:

                                    winner = game.user.getAccount()

                                else:

                                    winner = f"Player {i+1}"

                                running = False

                            rack=game.players[game.getPTurn()].displayRack()

                            UIrack=[]

                            for i in range(len(rack)):

                                char=rack[i]

                                if rack[i]=="blank":

                                    char="\_"

                                txt = tileFont.render(char,True,(0,0,0))

                                rec = pygame.Rect(765+40+(block\_size+1)\*i,300,block\_size,block\_size)

                                UIrack.append((txt,rec))

                            self.\_inavlidTurnFlag= False

                        else:

                            self.invalidTurn()

                    #pass button#

                    if (883+200)>=mousePos[0]>=883 and (500+50)>= mousePos[1]>=500:

                        game.increaseNumPasses()

                        game.incrementTurn()

                        if game.isGameOver():

                            i = game.findWinner()

                            if i == 0 and game.user != None:

                                winner = game.user.getAccount()

                            else:

                                winner = f"Player {i+1}"

                            running = False

                        ###############################################

                        #Group A skill - Dynamic generation of objects#

                        ###############################################

                        rack=game.players[game.getPTurn()].displayRack()

                        UIrack=[]

                        for i in range(len(rack)):

                            char=rack[i]

                            if rack[i]=="blank":

                                char="\_"

                            txt = tileFont.render(char,True,(0,0,0))

                            rec = pygame.Rect(765+40+(block\_size+1)\*i,300,block\_size,block\_size)

                            UIrack.append((txt,rec))

                    ###Save Game and Exit Button####

                    if (883+200)>=mousePos[0]>=883 and (700+50)>= mousePos[1]>=700:

                        game.SaveGame()

                        running = False

                elif event.type == pygame.MOUSEMOTION:

                    if rectangle\_draging:

                        txt,rectangle=UIrack[TileDragged]

                        mouse\_x, mouse\_y = event.pos

                        rectangle.x = mouse\_x + offset\_x

                        rectangle.y = mouse\_y + offset\_y

                elif event.type == pygame.MOUSEBUTTONUP:

                    if rectangle\_draging:

                        txt,rectangle=UIrack[TileDragged]

                        mouse\_x, mouse\_y = event.pos

                        if  (mouse\_y )//(block\_size+1)\*(block\_size+1) <0 :

                            ycoord=0

                        elif (mouse\_y)//(block\_size+1)\*(block\_size+1) >=15\*(block\_size+1):

                            ycoord = 15\*(block\_size+1)

                        else:

                            ycoord = (mouse\_y)//(block\_size+1)\*(block\_size+1)

                        if (mouse\_x + offset\_x)//(block\_size+1)\*(block\_size+1) <0:

                            xcoord =0

                        else:

                            xcoord=(mouse\_x )//(block\_size+1)\*(block\_size+1)

                        if 765-block\_size>=(mouse\_x )//(block\_size+1)\*(block\_size+1)>=0:

                            rectangle.x = xcoord

                            rectangle.y = ycoord

                        else:

                            rectangle.x =765+40+(block\_size+1)\*TileDragged

                            rectangle.y = 300

                        movesStack.append((rectangle,(rectangle.x,rectangle.y),(startCoords)))

                    rectangle\_draging = False

            self.drawBackround(game)

            for i in range(len(UIrack)):

                txt,rectangle=UIrack[i]

                pygame.draw.rect(self.\_screen,buttonColourLight,rectangle)

                self.\_screen.blit(txt,(rectangle.x+15,rectangle.y+10))

                pointsDict=game.getPointsDict()

                if rack[i].islower():

                    char = "0"

                else:

                    char = str(pointsDict[rack[i]])

                txt = pointsFont.render(char,True,(0,0,0))

                self.\_screen.blit(txt,(rectangle.x,rectangle.y))

            clock.tick(FPS)

            mousePos=pygame.mouse.get\_pos()

            ###Next Turn Button###

            if (883+200)>=mousePos[0]>=883 and (380+50)>= mousePos[1]>=380:

                pygame.draw.rect(self.\_screen,buttonColourDark,[883,380,200, 50])

            else:

                pygame.draw.rect(self.\_screen,buttonColourLight,[883,380,200, 50])

            txt =font.render('Next Turn',True,(0,0,0))

            self.\_screen.blit(txt,(945,397))

            ###Undo Button###

            if (883+200)>=mousePos[0]>=883 and (440+50)>= mousePos[1]>=440:

                pygame.draw.rect(self.\_screen,buttonColourDark,[883,440,200, 50])

            else:

                pygame.draw.rect(self.\_screen,buttonColourLight,[883,440,200, 50])

            txt = font.render("Undo",True,(0,0,0))

            self.\_screen.blit(txt,(960,457))

            ###Pass Turn Button###

            if (883+200)>=mousePos[0]>=883 and (500+50)>= mousePos[1]>=500:

                pygame.draw.rect(self.\_screen,buttonColourDark,[883,500,200, 50])

            else:

                pygame.draw.rect(self.\_screen,buttonColourLight,[883,500,200, 50])

            txt = font.render("Pass",True,(0,0,0))

            self.\_screen.blit(txt,(960,517))

            ###Save Game and Exit Button####

            if (883+200)>=mousePos[0]>=883 and (700+50)>= mousePos[1]>=700:

                pygame.draw.rect(self.\_screen,buttonColourDark,[883,700,200, 50])

            else:

                pygame.draw.rect(self.\_screen,buttonColourLight,[883,700,200, 50])

            txt= font.render("Save Game & Exit",True,(0,0,0))

            self.\_screen.blit(txt,(913,717))

            pygame.display.update()

        if winner != None:

            self.displayWinner(winner)

            pygame.display.update()

        else:

            pygame.quit()

            self.\_screen=None

            self.\_languageWindow = None

    def HandleBlank(self):

        running = True

        alphabet =["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]

        blocksize=45

        buttonColourLight = (229,229,229)

        buttonColourDark=(122, 122, 122)

        buttons=[]

        tileFont =pygame.font.SysFont(None,36)

        self.\_screen.fill((0,0,0))

        titleFont =pygame.font.SysFont(None,70)

        txt = titleFont.render("Click the letter that the blank should represent:",True,(255,255,255))

        self.\_screen.blit(txt,(50,300))

        for i in range(26):

            rect = pygame.Rect((blocksize+1)\*i,350,blocksize,blocksize)

            buttons.append(rect)

        while running:

            mousePos=pygame.mouse.get\_pos()

            for i in range(26):

                if 350<=mousePos[1]<=(350+blocksize) and (blocksize+1)\*i<=mousePos[0]<=(blocksize+1)\*(i+1):

                    pygame.draw.rect(self.\_screen,buttonColourDark,buttons[i])

                else:

                    pygame.draw.rect(self.\_screen,buttonColourLight,buttons[i])

                txt = tileFont.render(alphabet[i],True,(0,0,0))

                self.\_screen.blit(txt,((blocksize+1)\*i+15,350+15))

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    running = False

                if event.type == pygame.MOUSEBUTTONDOWN:

                    for i in range(26):

                        if 350<=mousePos[1]<=(350+blocksize) and (blocksize+1)\*i<=mousePos[0]<=(blocksize+1)\*(i+1):

                            self.\_blank =(alphabet[i])

                            running=False

            pygame.display.update()

    def displayWinner(self,winner):

            self.\_screen.fill((0,0,0))

            font = pygame.font.SysFont(None, 120)

            txt = font.render(f"!!!{winner} wins!!!",True,(255,255,255))

            self.\_screen.blit(txt,(300,380))

    def invalidTurn(self):

        self.\_inavlidTurnFlag = True

# System testing

## Test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Objective Tested** | **Test** | **Time stamp in video** | **Test Status** |
| 1. | The program must be runnable with a ‘t’ option to run in the terminal or ‘g’ option to run in a GUI. Running without one of these should report an error | Test that if you type “t” the terminal version of the game should open  If “g” is typed into the terminal the GUI version of the game should  Any other input should cause an error | 0:00 | Passed |
| 2. | When running in the terminal mode a 15 x 15 scrabble board should be printed to the terminal | Check that the board is printed every turn | 1:00 | Passed |
| 3. | At the start of the game 7 letters should be printed and assigned to a player | Check that the player rack is printed and is the correct length | 1:15 | Passed |
| 4. | The program should be able to access and open the text file containing the “scrabble dictionary” which contains all valid words | Check that the game allows the player to input their turn  Test that any invalid words or inputs should cause an error message to be displayed and then re-ask for the user’s input  Any valid word should be accepted and the points for that turn should be correctly calculated | 1:29 | Passed |
| 5. | Each player should be asked to input their moves each turn | Passed |
| 6. | If the player tries to use a tile that is not on his rack then an error should be reported | Passed |
| 7. | Each new word formed each turn should be searched for in the dictionary. If the word is not in the dictionary an error should be reported and the player should have redo his turn if the word is valid then the number of points for that word should be accurately calculated | Passed |
| 8. | The number of points obtained by the player that turn should be added to their total. | Passed |
|  | A player should be able to pass their turn | Check that players can pass their turns and that if a player passes their turn the next turn updates | 5:16 | Passed |
| 9. | Once a player has played his turn, the player should be assigned more letters so that he has 7 letters | Check that on the players next turn the player rack has been updated | 5:16 | Passed |
| 10. | Each turn the board should be displayed and the number of points each player has should be displayed | Check that the board and the number of points each player has is updated and displayed each turn | 1:00 | Passed |
| 11. | Once the bag is empty no more tiles should be assigned to players. | Test that no more tiles are added to the payers rack after the bag is empty  After this, once a player has played their final letter the game should end  Test that if both players pass their turns on consecutive turns then the game should end  …  And the correct winner of the game should be displayed | 7:00 | Passed |
| 12. | Once a player has used his last letter/tile the game should end | Passed |
| 14. | After the game has ended the sum of each players unplayed letters should be deducted from their respective scores, if a player used all of their letters the sum of the other player’s unplayed letters should be added to the total score | Passed |
| 15. | The winner of the game (the player with most points) should be reported | Passed |
| 16. | In case of 2 or more players have the same number of points, the player with the most points before adding or subtracting points due to unplayed letters should be declared as the winner | Passed |
|  | If both players pass on consecutive turns the game should end | Passed |
| 20. | Users’ Usernames and Passwords should be stored in an external database | Tested within the tests for the UI |  | Passed |
| 21. | Another database should store the state of saved games as well as which users played them |  | Passed |
| 22. | New users usernames and passwords should be inserted into the database |  | Passed |
| 23. | The number of wins and losses by the user should be tracked |  | Passed |
|  | When running the game in GUI mode a database called “ScrabbleDatabase.db” should be created if does not already exist | Check thar the database is created properly | 13:33 | Passed |
| 24. | When run with the ‘g’ command line option, the program should present a GUI with buttons to display ‘Play Game’, ’Previous Games’ ,‘Help’, ‘Options’ ,’Login’ or ‘Quit’ | Test that the GUI opens correctly when “g” is entered into the command line | 15:00 | Passed |
| 25. | Pressing the ‘Quit’ button should cause the GUI window to disappear and the program to end. | Check that Quit button closes the GUI | 15:00 | Passed |
|  | Pressing the ‘Create Account’ button should open window with 2 entry boxes asking the user to enter their username and password, a ‘Create Account’ button and a ‘Dismiss’ button | Check the Create account window opens correctly.  Check that it allows users to create an account if the username entered is not already in the database  If it is in the database check that an error message is diplayed | 16:00 |  |
|  | Pressing the ‘Quit’ button should close the Create Account window |  |
|  | Pressing the Create Account button should cause the program to validate the username entered |  |
|  | If the username is already in the database an error message should be displayed |  |
|  | Otherwise a new record should be created in the database with the username and password entered by the user |  |
| 32. | Pressing the ‘Login’ button should open window with 2 entry boxes asking the user to enter their username and password, a ‘Login’ button and a ‘Dismiss’ button | Check that the Login window opens and is displayed correctly when the Login button is pressed | 17:30 | Passed |
| 33. | Pressing the ‘Quit” button should close the Login window | Test that pressing the Dismiss button in the Login window should close the Window down | Passed |
| 34. | Pressing the Login button should cause the program to validate the username and password entered | Test that the Login system works:  If a valid Login is entered the user should be logged in and the Login window should close  If an invalid Login is entered an error message should pop up in a new window | Passed |
| 35. | If the username and password is valid the Login window should close and cause the user to be logged in | Passed |
| 36. | If the username and password is invalid an error message should be displayed | Passed |
| 37. | Pressing the Account button when not logged in should display an error message | Test that Account window only opens when the user is logged in and test that is displayed correctly  If the user is not logged in and tries to open the Account window then an error message should appear in a new window  The account window should display the stats of the user  The Quit button should quit the window | 20:00 | Passed |
| 38. | Pressing the ‘Account’ should open a window displaying the user’s username and stats of the user as well as 2 buttons labelled “Load Games” and “Quit” | Passed |
|  | Clicking the Quit button should close the window | Passed |
|  | The number of wins and losses by the user should be tracked | Passed |
|  | Clicking the “Load Games” button should open a window display all of the uses saved games as buttons as well as a button labelled “Quit” at the bottom of the window | The load games window should be correctly displayed  The quit button should quit the window | 20:10  25:00 | Passed |
|  | Clicking the Quit button should close the window | Passed |
| 44 | Pressing the ‘Save Game and Quit’ should cause the game to be saved and the game window to be closed | Check that the user is able to continue playing previous games |  | Passed |
| 39. | Clicking on one of the previous games should open a new window causing that game to resume | Passed |
| 43. | Pressing The ‘Play Game’ button should open a window with 2 buttons on it that say “English” and “Spanish” and should prompt the User to select langauge | Test that when clicking the Play Game button a window asking to select language should appear.  Upon clicking either button a new window displaying a Scrabble board should open | 21:40 | Passed |
|  | Clicking either of should cause the ‘Setup Game’ window to disappear and a new window to open displaying 15x15 scrabble grid, a ‘Save Game and Quit’ button, a ‘Next Turn’ button, a ’Pass’ | Passed |
|  | The game should be playable in both English and Spanish | Check that game recognises Spanish words if Spanish words if Spanish is selected and English words if English is selected | 21:40  27:30 | Passed |
| 44. | Pressing the ‘Save Game and Quit’ should cause the game to be saved and the game window to be closed | Clicking the Save and Quit button in the Scrabble window should cause the window to close | 25:00 | Passed |
| 45. | The letters/tiles should be able to be dragged and dropped onto the board | Check that letters and tiles can be dragged and dropped onto the board | 22:00 | Passed |
| 46. | Pressing the next turn button should cause the next turn to occur | Check that pressing the next turn button allows the next user to play their turn and updates the board and points total accordingly | 24:00 | Passed |
| 47. | The points total displayed should be updated after each turn | 24:00 | Passed |
|  | Pressing the Pass button should cause the next turn to occur without that player making any moves | Check that pressing the pass button passes the turn | 26:00 | Passed |
|  | Pressing the Undo button should cause the last drag and drop action to be undone | Check that the undo button is working correctly | 23:00 | Passed |
| 48. | When a player has won the game the winner should be displayed | Test that the game correctly displays a the winner | 26:30 | Passed |

## Testing Videos

**http://bitly.ws/zQkV**

# Evaluation*)*

## Comparison of project performance against the objectives

I met the first set of objectives which was to build a minimal viable product. This product runs entirely in the terminal and allows a user to play against one other human player. The board was displayed using ascii characters and users could enter their moves by entering the letter they wanted to place and then the coordinates they wanted to go. The overall minimal viable product, though it does its job and allows users to play Scrabble, it could be possibly described as ugly and cumbersome to use if used as the actual product and was primarily used for developing the base of my program.

This then allowed me to develop the user log in/account section of the program. This allows users to create or log in to pre-existing accounts using a unique username and password. Once a user is logged in, while in a game, they are able to quit and save that game and then return to that exact game at that point whenever they want.

I then developed the user interface and managed to meet all the objectives tied to the user interface. These objectives created a user friendly environment in which the users can play the game in a more visually appealing and engaging way than the terminal. It also allowed users to access the log in system easily.

I was not able to get around to developing an AI that can play the game so that section of the original objectives I missing in the program.

## Effectiveness of the solution

Overall the program is a relatively effective solution to the original problem. The user interface is responsive and well laid out making it easy to use and navigate. The Login/account features enable to users to easily save games and return to them later something which would be something difficult to in the original board version of the game. The game also allows you to swap the language of the game easily something not possible in the original game unless you buy a different version for each language. The program meets most of the user requirements however is missing a playable AI

## Analysis of user feedback

I was able to get feedback from my main user, Zakk:

Zakk

* General Feedback - What are your general thoughts on the software?
  + The software works really well and I like it a lot
  + I like the user interface
  + The game works well and the user interface is easy to use and navigate
  + All of the functionality that I have tested seems to work well
  + *Overall Zakk like the software and though it was visual and easy to navigate and use*
* Usability and accessibility of the software and features – Did you find the game easy to use and easy to access all its features?
  + The UI was clear and visual
  + The buttons were clearly and appropriately labelled so I did not have much of a problem navigating around the program
  + *The game was easy to use and navigate and all the features were accesible*
* Meeting of requirements – Does the game meet your original requirements and expectations?
  + The game definitely meets most of my initial requirements
  + However it is missing a playable AI which I think would be a good addition to the game
  + *The game meets most of the original requirements however is missing a playable AI*
* Use of Software – Do you think you could see yourself using this software, and if so when and where would use it?
  + Yes, I could definitely see myself using the software
  + I would probably use it every time I wanted to play Scrabble away when I am away from home as the board and pieces are difficult and annoying to carry around. So I definitely use the software often
  + *The software is useful as it is portable and doesn’t require a physical board and pieces. The primary user would use it often*
* Improvements – Are there any improvement you would like to see to this software?
  + A playable AI would definitely be great as it would allow you to play the game without needing another person to play with you
  + Customisation options for the UI would also be nice to see
  + Online functionality would be great as it would allow you to play with your friends even if you are in separate locations. Also having to people play over the same computer is a bit cumbersome
  + *There are a few improvements that have been suggested by Zakk. These have been acknowledged and are incorporated in the next section which discusses possible improvements for the software*

*Overall it seems that Zakk like the software and could see himself using it often. He particularly liked the user interface as it made the game easy to use and navigate. However Zakk did mention some improvements that could be made and did seem disappointed at the lack of a playable AI*

## Possible improvements

Sound effects

There could be sound effects every time the user drops a tile onto the board or clicks a button in the menus. This would help they game feel slightly interactive and less bare-bones .

This could be implemented by storing sound files and playing them for a short period of time whenever the relevant action happens

Implement an AI that can act as a player

There could be an AI that users could play against. This would allow users to play the game even if they are by themselves. The AI could have multiple difficulty levels which would allow users of multiple various skill levels to play against the AI without finding it too easy or too difficult

Add a hints button

Currently the program gives little to no help to beginners of the game and assumes you already know the rules. Adding a hint button while that users could use during the game could give some added help to beginners.

This hint button could highlight the letters in their rack that the user would need to play in order to play the highest scoring word. This hints button could be disabled in the menu so that more experienced players can play the game without having access to it

Adding board customisation options

Currently the board and the tiles are always the same colours. Adding customisation options could allow users to change the colour of the background, the board and the tiles. This would allow users to choose colours that match their preferences and would give the game a more personal feel.

The customisation options could be selected in the settings window and could be saved in a database relating to the account of the user, so that every time the user logs into their account their customisation options will automatically be applied

Add 3 and 4 player functionality

In the board game version of Scrabble the game can be played between 2-4 players however this version can only be played between 2 players.

To accommodate this the game class must be modified to take number of players as a parameter and rather than switching between player 1 and player 2 it would instead have to switch between even more players

Add more languages into the game

Currently the only languages in the game are Spanish and Spanish. Scrabble is available in many more langauges than this (e.g. German). To implement a new language a new class would have to be made, that is similar in design to the Spanish and Spanish classes, that stores the points for each letter as well as that languages Scrabble dictionary.

Add a client based network over a LAN

AT the moment, the program provides multiplayer functionality in that two humans can play against each other on the same machine. However it would be more useful to be able to play some over a LAN, so that both do not have to crowd over a single computer

# Appendix

## Git log

Graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

*Graphical user interface, application

Description automatically generated with medium confidence*

*Graphical user interface

Description automatically generated*

*Graphical user interface, text, application, email

Description automatically generated*

***Link to Git Hub Repo:***

***https://github.com/SebRouse/NEA***