Scrabble

By Louis Terrell

Contents

Contents

[Analysis 2](#_Toc7101989)

[*Project Outline* 3](#_Toc7101990)

[*Research* 4](#_Toc7101991)

[Board: 5](#_Toc7101992)

[The end game: 7](#_Toc7101993)

[The Scoring: 7](#_Toc7101994)

[Validation: 7](#_Toc7101995)

[Other ways of playing Scrabble: 8](#_Toc7101996)

[Feedback: 9](#_Toc7101997)

[Recipient 1: 10](#_Toc7101998)

[Recipient 2: 11](#_Toc7101999)

[Conclusion: 12](#_Toc7102000)

[Design 12](#_Toc7102001)

[Series of Events – Original Scrabble 13](#_Toc7102002)

[User Interface 17](#_Toc7102003)

[Pseudo Code: 22](#_Toc7102004)

[UML for Forms 24](#_Toc7102005)

[Database 25](#_Toc7102006)

[Users Table: 25](#_Toc7102007)

[ERD 29](#_Toc7102008)

[Pre-Game Settings Menu: 30](#_Toc7102009)

[Inside the Debug Folder 31](#_Toc7102010)

[Technical Analysis 33](#_Toc7102011)

[Game Form: 33](#_Toc7102012)

[PlayMenu Form: 47](#_Toc7102013)

[Starts Menu: 48](#_Toc7102014)

[Testing 49](#_Toc7102015)

[Selecting/Deselecting and Placing/Removing Letters Test: 49](#_Toc7102016)

[Assigning Player’s Hands Test: 51](#_Toc7102017)

[Determining Which Player Goes First Test: 54](#_Toc7102018)

[Placing Letters Test: 57](#_Toc7102019)

[Ability to Change the Quantities and Values of Each Letter: 59](#_Toc7102020)

[User Feedback 63](#_Toc7102021)

[Feedback Form 1 63](#_Toc7102022)

[Feedback Form 2: 64](#_Toc7102023)

[Feedback Form 3: 65](#_Toc7102024)

[Evaluation 66](#_Toc7102025)

[Objectives: 66](#_Toc7102026)

[Core: 66](#_Toc7102027)

[Extras: 66](#_Toc7102028)

[User Feedback: 68](#_Toc7102029)

[ User requirements: 68](#_Toc7102030)

[ Usability: 68](#_Toc7102031)

[ Performance: 68](#_Toc7102032)

Analysis

Introduction

The project I have chosen to set myself is the board game scrabble. I’ve always found scrabble a fun game to play with others. It’s a game of simple rules but found that these rules will be much harder to recreate on the VB platform. I know from the get-go, that I will have to find words (within the scrabble dictionary) in quick succession as to not delay the game and in turn bore the user. Another hurdle I will have to get over is identifying words on the board itself. This alone will take a while to implement as I have to also make sure that the user is putting the letters in a legal position on the board. An illegal position being, a word not found in the dictionary or not being a part of the same grouping as previously placed letters, it will also have to be conjoined to another letter (except at the start of the game). I believe these steps alone will make it a fun challenge that I hope to complete and possibly use to play with others in the future.

The fact that the game itself is well known will make peer assessment that much easier. Peer assessment will consist of people trying out my program at regular stages to make sure that it is running correctly and that I haven’t missed any features out. I plan for the board to be laid out the same way as the traditional set but subject to change with ease if need be.

I have thought about implementing a network within the game so that players don’t have to play on a local system. However, this is only an idea and will not delve too deeply into it if the program becomes a struggle. I would in later years like to implement this code but am unsure on whether to attempt as this project develops.

*Project Outline*

Before the game begins the user will be able to change the games default rules, I am unsure of how far I want to go with this feature. If possible, I would like to give the user, the ability to change the board and even change its dimensions. The settings menu will have a ‘Game Duration’, ‘Score Cap’, ‘Board Width and Height’ and even the proportion of letters within the “bag”.

The game will be ‘two-player’ and without AI. Each player will take their turn like any other game of Scrabble till one player runs out of letters or either can’t go anymore. The victor is decided by who has the most points. Each move will be validated to see if its legal and will come up with a message once the “end turn” button is pressed telling the player of the foul they’ve committed. Legal words are decided by the scrabble dictionary which will be listed in a text file.

I would like to implement a Leaderboard system to my game of scrabble. The system will hold relevant information on previous games played. It will include a listing of the highest scoring games with the Player’s usernames (if defined), ‘Highest Scoring Word: ’, ‘Shortest Game Duration: ’. This is all up to change and other records may be added.

Objectives:

1. A ‘Game-Menu’ where the user can start a game and modify the settings themselves. For example, the board can be changed as well as the score and chance of letters.
2. The hand is dealt randomly from the ‘bag’, each player receiving seven letters. The player who goes first is then decided.
3. Each player then takes their turn. The letters played are combined, validated and the scores are tallied up.
4. A ‘hint’ system may be implemented at a later stage which will give the user recommended words.
5. The first player to either ‘forfeit’ or use up all their tiles (in hand and the ‘bag’) ends the game. Final scoring takes place, congratulating the winner.
6. Scores are then saved to a database including a few other statistics, such as: ‘Longest Word: ‘ etc. Will be viewable by the user on the menu.
7. A save system will be implemented so the players can carry on from where they left off.

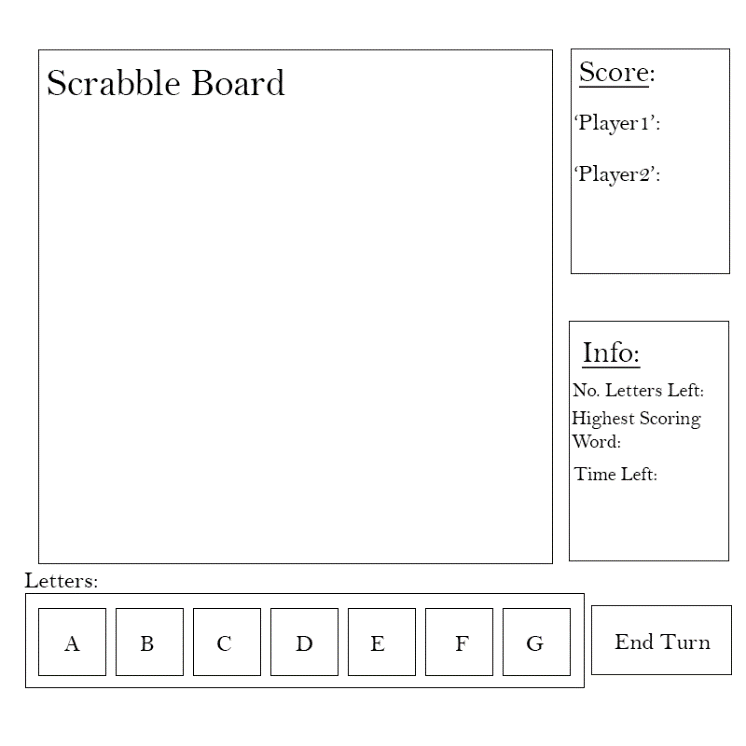
Extras:

* Introduction of networking so players on separate systems can play with one another through a server.
* A range of ‘game-modes’ that the user can pick (before the game) in the settings menu. These are listed below in more detail.

*Research*

The first thing I did with the project was make sure I knew the rules. The page I used: <https://en.wikibooks.org/wiki/Scrabble/Rules>. I will be keeping the standard Scrabble layout:

* 15x15 Scrabble board
* Set of 100 tiles
* A single rack per player

The set of tiles itself was another aspect I had to get right. The site I used for this <https://en.wikipedia.org/wiki/Scrabble_letter_distributions>. The distribution consists of (Following the standard game rules):

A concept for the final Scrabble game design and its GUI.

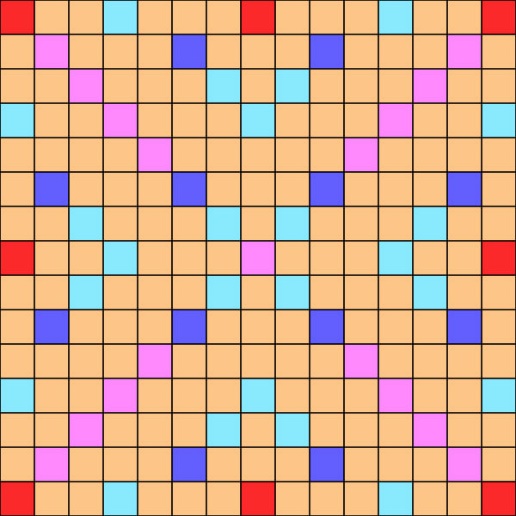
|  |  |  |
| --- | --- | --- |
| **Letter** | **Point(s)** | **Count** |
| A | 1 | 9 |
| B | 3 | 2 |
| C | 3 | 2 |
| D | 2 | 4 |
| E | 1 | 12 |
| F | 4 | 2 |
| G | 2 | 3 |
| H | 4 | 2 |
| I | 1 | 9 |
| J | 8 | 1 |
| K | 5 | 1 |
| L | 1 | 4 |
| M | 3 | 2 |
| N | 1 | 6 |
| O | 1 | 8 |
| P | 3 | 2 |
| Q | 10 | 1 |
| R | 1 | 6 |
| S | 1 | 4 |
| T | 1 | 6 |
| U | 1 | 4 |
| V | 4 | 2 |
| W | 4 | 2 |
| X | 8 | 1 |
| Y | 4 | 2 |
| Z | 10 | 1 |
| BLANK | 0 | 2 |

Board:

I would like to have two board layouts by default. The first of which being the traditional scrabble board layout. The other will be Facebook’s “Words with Friends” board type, this is because the platform I tend to use to play scrabble is through Facebook and its online-multiplayer system. If I was to implement a network within the program I would like to do it in a similar sense, without the Facebook friend system.

The board layout I will be using is the universal standard 15x15 board with the classic tile layout. The columns can be defined as ‘A’ to ‘O’ and the rows being 1 to 15. The board itself will be based off this:





Bonus Tile Key:

|  |  |
| --- | --- |
| Red | Triple Word Score (x3 Total Word) |
| Light Blue (Cyan) | Double Letter Score (x2 Letter Score) |
| Dark Blue | Triple Letter Score (x3 Letter Score) |
| Pink (Magenta) | Double Word Score (x2 Word Score) |
| Plain | Regular Word and Letter Score |

The centre tile as usual is the start point for the player. To decide which of the two players goes first I will be using the National School Scrabble Championship rule, this is where Both players put all the tiles into one bag and mix them up. I will be randomly sorting them to produce this. Each player then chooses one tile, and the letter closest to A or a blank tile goes first. The players are then randomly distributed 7 letters each.

To begin the game a word must be played in the centre. It has to be at least two letters long and covering the centre square. The second word played must be conjoining the others.

The player has three options for each turn:

* Pass, forfeiting the turn and scoring nothing
* Exchange one or more tiles for an equal number from the bag, scoring nothing, an option available only if at least seven tiles remain in the bag
* Play at least one tile on the board, adding the value of all words formed to the player's cumulative score

The end game:

To decide the end of the game there are several ways. Under North American rules:

1. If a player plays every tile on their rack with no tiles remaining in the bag. It does not consider the opponents rack; the game just ends.
2. A player decides to end the game after six successive scoreless turns.
3. If a player uses more than 10 minutes of overtime.

My game of scrabble will stick closely to the rules of ‘end game’. However, I would like to add an overall time limit to the game and a shorter length of overtime. Both can be set before the game begins by the user.

Once the game comes to an end, each Player’s score is reduced by the sum of all their unplayed letters. However, if a player has used up all their letters (a “going out” end game) then all unplayed opponent’s tiles are added to that players score; in tournament play a player who ‘goes out’ will receive twice that score, plus neither player is penalized if tiles are left over.

The Scoring:

The scoring, as shown in the table above, is made up of 100 letter tiles each of which having given points. Vowels having a point of 1, with letters such as “Q” and “X” showing up less in the pack being worth 8-10 points. The board is split up into 225 separate tiles with some more valuable than others. For instance, the double letter score (cyan) and triple letter score (dark-blue) tiles mean that whatever letter is placed on that tile its points are doubled, and “Q” would now be worth “20” points if placed on a double letter score tile and “30” on a triple. This is known as a letter bonus. The double word score (Magenta) and the triple word score (Red), is the same in sense that it multiples by two and three but instead of multiplying a single letter it does the total score of the word. An example of this would be the word “WORDS” which would normally give a score of (4+1+1+2+1) “9”, if crossing a triple word score would now equal “27”. If the word includes both a word bonus and a letter bonus the letter bonus is applied first.

The centre square is also a Double Word Score, so the word is doubled.

Once the Bonus letter squares have been placed upon, they no longer have any effect on subsequent words.

If a play covers two Bonus word tiles the score is doubled and then doubled again (Multiplying the score by 4). The same can be done with two Triple Word tiles.

When a player uses all seven tiles on their rack in a single play, they receive an extra 50 points, plus the score for the word. This is commonly known as a ‘bingo’ or a ‘bonus’.

Validation:

In the program I will need to setup a system to validate words played. This will be a multistep process and will be the most challenging part of the program. To begin with I will need a real-time check of the tiles and where they are on the board. To do this I will need to check for the position of the tile placed and whether it is valid. Validation:

* If the tile(s) placed are conjoining. Not separated from the rest of the board.
* The order of the tiles (Letters) create a valid word within ‘ScrabbleWords.txt’.
* The word itself starts from left-to-right or top-to-bottom. The words cannot be spelt backwards.
* A word has to be made up of adjacent letters, not diagonal and within the same row or column of the first that’s been placed.
* Offensive words are allowed for I am mostly following the tournament rules. Though if wanted or you would need to do is remove these words from the ‘ScrabbleWords.txt’ file.

Other ways of playing Scrabble:

Clabbers/Anagrams:

The game has the same rule set as Scrabble except the words played can be anagrams. The player can ask their opponent for the “real” word that the anagram makes, following the ‘double challenge’ rule.

Double challenge rule – if one or more of the challenged words is not in the dictionary (in this instance: anagram is not a word) the player misses their turn.

Take Two:

Also known as Bananagrams is played without a board. The first player to use all of their face-up tiles, they must shout “take two” meaning every other player must take two more tiles. Once all the tiles have been taken, the first player to use all his or her tiles is the winner.

Polyglot:

Where you can play words in any language. Keeping the same rules as Scrabble.

Fours:

Played the same as Scrabble but only four tiles can be played at once. The game can be altered to ‘Fives’ or ‘Threes’.

Kajawah:

Kajawah again follows the same rule set as Scrabble and plays the same in every way. However, all players decide on a ‘target’ word – In this case it is Kajawah. If a player spells out this word, they get an extra 50 points and the game ends. The winner is whoever has the most points.

Kajawah – a camel’s litter.

Nouns:

You may only play nouns and you gain bonus points when playing proper nouns which are also valid in the Scrabble words. For example, ‘Sandwich’ which is also an area in Kent.

These are a few variations of Scrabble I would like to try and implement as a sort of ‘game mode’. The mode will be decided in the game settings at the start of the game. Not all will be implemented into the game as some would not work, like ‘Bananagrams’, but others such as ‘Polyglot’ would be fun and fairly simple to add due to it just being a larger text document.

Site used:

<https://www.collinsdictionary.com/word-lovers-blog/new/scrabble-addict-16-ways-to-mix-up-your-word-game-addiction,130,hcb.html>

Feedback:

Recipient 1:

[Me]: What would you like to see in this Scrabble based game?

[Other]: I think you should keep the traditional Scrabble board and rules, to use as a substitute to the physical board game. Will this be played online?

[Me]: I would like to implement an online system but is unlikely to be put into the project. I will see how the code develops before researching into VB networking. The game will be played on a local system – on the same computer.

[Other]: Won’t that mean the player can see the other letters?

[Me]: Yes, it will. However, a timer can be introduced between turns so that the device can be swapped. This can be, say, five seconds long. Would you be happy with a regular scoring system?

**\*Subsequently, if the user decides to add a timer for the game this will need to be paused while this five second timer is counting down. \***

[Other]: Scoring system should be kept the same, any changes may break the play of the game. I’m assuming any changes will be hard to make.

[Me]: Simple changes like the points of certain letters, E being equal to 1 could be changed to 3. Changing the ratio and number of tiles would involve a few more lines of code. I guess, it can be added to the Pre-Game Settings menu, in a list of such where the points and amount of tiles can be changed.

[Other]: Yeah sounds great, I like the amount of customizability. Will make each game feel unique. Have you thought of adding a Combo-like system?

[Me]: What do you mean?

[Other]: Say you get three words in a row you get a score bonus. Or you get a word over seven letters long you receive an extra score multiplier.

[Me]: I don’t think I will add that into the game as it would give an unfair advantage to a player with less fortunate letters. It would also be hard to implement and don’t really have any ideas of what it will involve. I do like the idea of longer receiving a bonus but that’s generally the case anyway. I just think it will be too confusing for the player to get to grips with and may be unfair. However, I will keep it in mind.

Recipient 2:

[Me]: What are you looking for in a scrabble based game?

[Other]: What do I look for in a scrabble game? Well, I like to stick to the traditional rules and scoring system. What style of board you thinking of going with? Just the traditional layout?

[Me]: I was going with the traditional layout as well as Facebook’s ‘Words with friends’ style. Have not thought of any others.

[Other]: You thought of ever randomly generating the board? Making each game unique.

[Me]: I have thought about randomly generating the board. Randomly generating a quadrant and then mirroring it onto the other three. Therefore, creating an even board. The only problem with this is that you may get a Triple Word Score tile next to the starting tile, giving the first player an unfair advantage for the start of the game.

**\*Again, I would have to determine how many bonus letter tiles I will need to use. Most likely the default number as only the positions will change\***

[Other]: True, true…

[Me]: I could add rules to the generation as to limit these effects and create a fair board. Most likely this will be used as an extra setting and will not be used over the other scrabble board layouts. I would like to have this as an option.

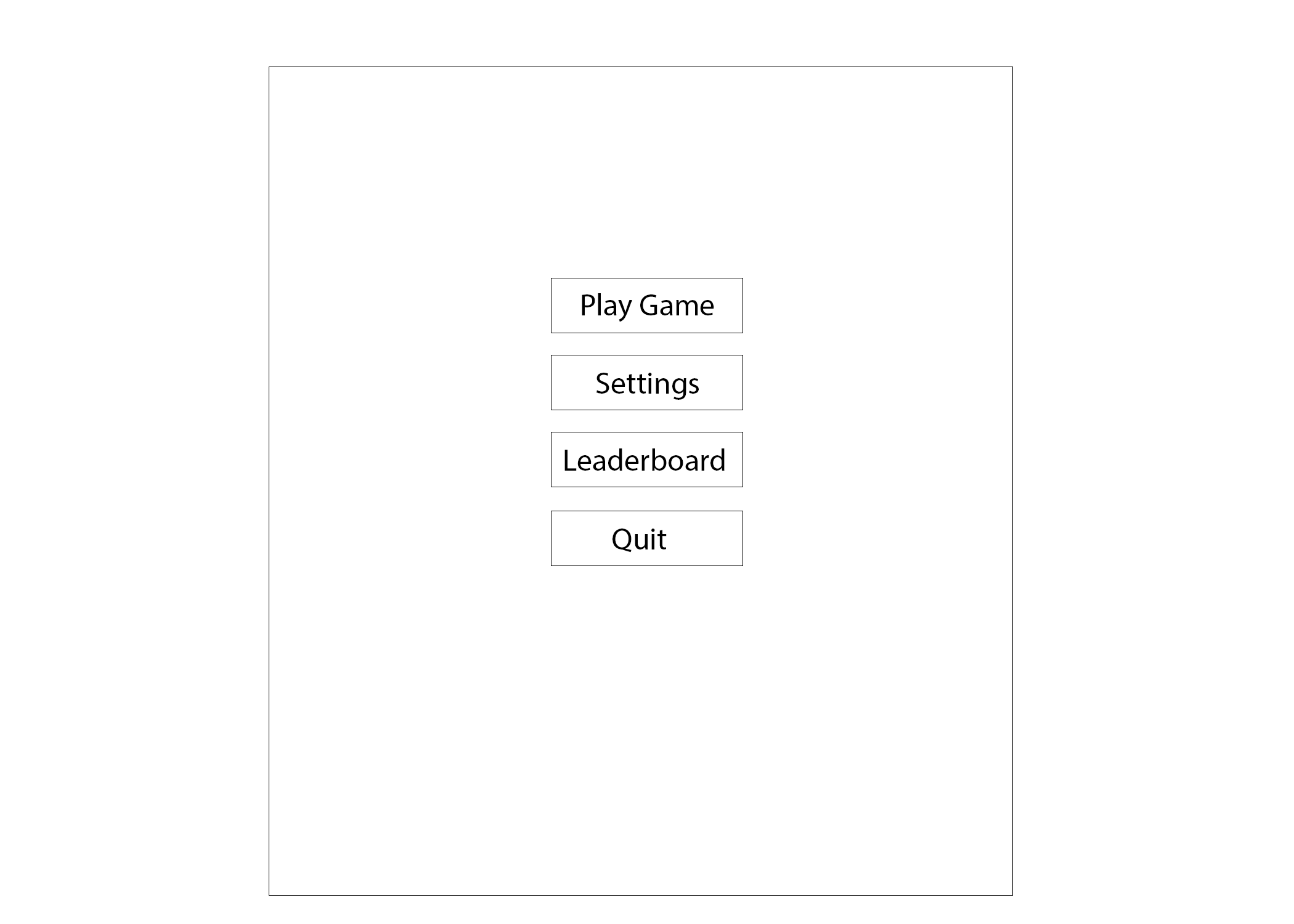
[Me]: Would you like to see and variations in ‘Game Modes’? Say ‘Polyglot’, a game mode made by Scrabble players where you can use words in any language.

[Other]: I’ve never played these from experience, however does sound fun. Can’t say I’ll be any good. Sounds fun though, good for re-playability.

[Me]: Of course, this will just be an option and the user and toggle whether they want to play or not. Will also make the game itself a little easier.

Conclusion:

In conclusion on whether the people I spoke to about the game and what they would like to see. Both recipients said they wanted to keep the default rulesets but would like to see options to change the Board, possible addition of other ‘game modes’ or even the scoring of certain letters and the chance of them popping up (ratio or number of tiles). It has also highlighted some issues with the program, the ability to view the opponent’s tiles. A potential solution to this is adding a five second timer in-between turns that will pause game. The other issue was with the possible addition of a randomly-generated board, which could give an unfair advantage to the starting player. To fix I would need to add certain regulations to tiles.



Design

The Design stage will consist of the GUI for the Program. A lot of this program is visual and is essential that I know how the program will be laid out and work. This section will contain the following:

* The Main Menu
* Buttons for each of the corresponding menus.
* A ‘Quit’ button so the user can exit the program.
* The Pre-Game Menu
* Can define the chance of tiles (How many there are).
* The points of tiles (Can keep it default).
* The board type/Layout.
* ‘Game modes’ of scrabble.
* The Settings Menu
* Will mostly be a placeholder for it is unlikely that sound or resolution of the program will be implemented.
* The Leaderboard Menu

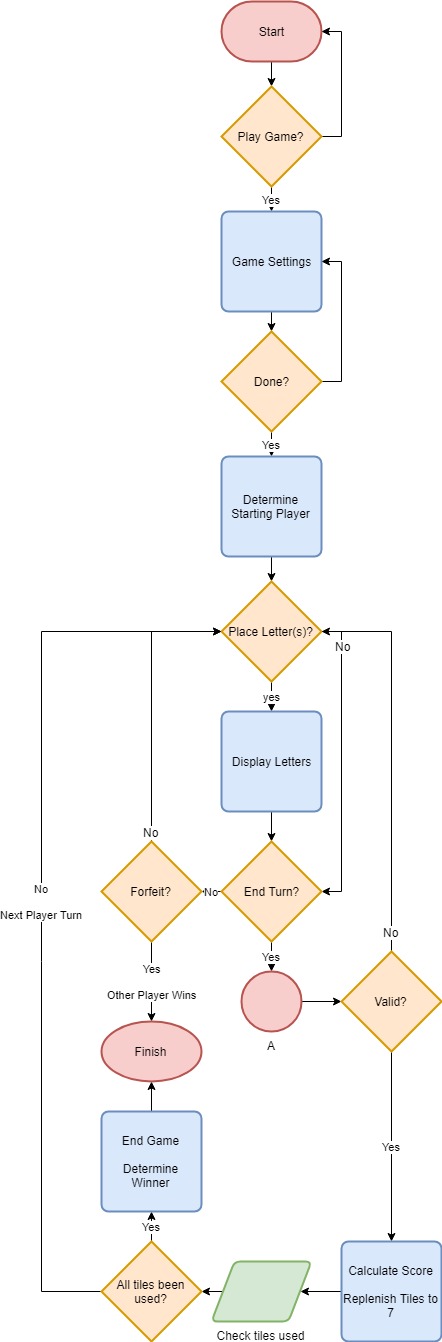
- Will hold the ‘End-game’ scores of users, ranking them from highest to lowest.

- Hold stats like ‘Longest word’ and ‘Highest scoring turn’.

- All saved onto a database.

* Letter Tiles (A-Z)
* Scrabble Game
* Scores of both players.
* Scrabble board
* Player’s ‘rack’ of letters
* End turn button

Series of Events – Original Scrabble



The flowchart outlines the events and actions the program will have to take in a basic form. When the program is run it will begin at the ‘Start’. In this example it goes straight into game settings, in more detail there will be a choice of ‘Leaderboards’, ‘Options’ and ‘Start Game’. For the case of simplicity, it goes straight onto ‘Game Settings’. Game Settings is where the rules of the game are determined: Board type, Game Mode, Letter settings. Once the changes have been made it moves onto the game itself.

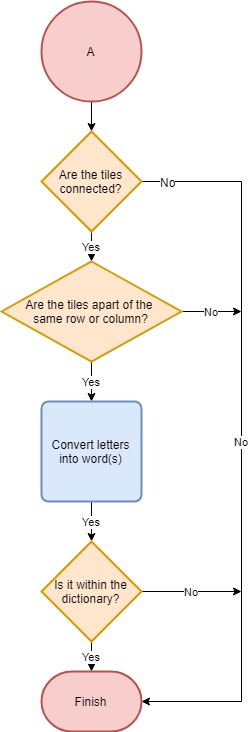
Determine Starting Player, is the action where the starting player is determined. This is done through the Player’s starting letters – The player with the closest letter to A or a BLANK tile. If undecided, the player is picked from a ‘coin-flip’.

The program will then wait until the turn has been ended by the user. Moving it onto the ‘Validation’ phase. If the letters played create a valid word and don’t break any of the other rules, it calculates the Score. However, if invalid it outputs an error of the rules that have been broken, informing the user and restarting their turn.

Tiles are replenished back to seven, dependent on how many tiles have been played will determine the number removed from the ‘bag’. In turn, decreasing the number of tiles left within the bag by the number used. If the ‘bag is empty’ then the user remains with the same number of tiles and does not gain anymore, therefore, cannot replenish to seven.

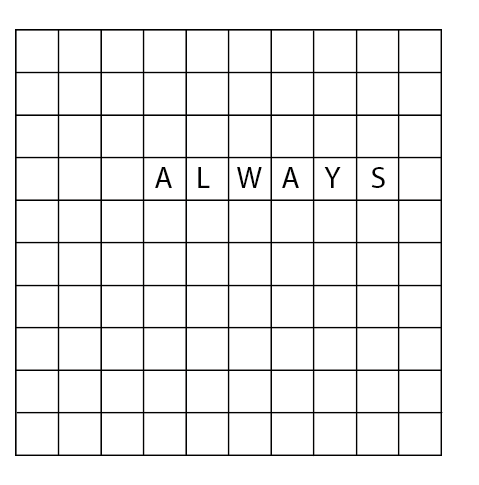
The Game Ends when one player uses all the tiles on their rack with the ‘bag’ being empty. Program then decides the distribution of points and declares the player with the highest score the winner. Ending the game.

Validation Flowchart:



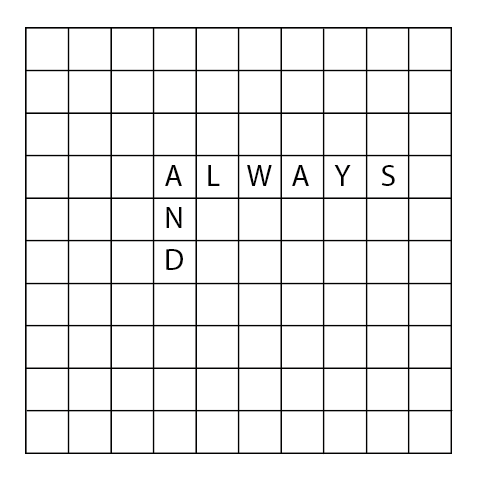
The validation flowchart outlines, in more detail, the steps the program will take to see if a play is valid. To begin with it checks the position of the tiles themselves to see if they’re in a valid spot, removing any lines of code that are not necessary to be run. If both are these steps are valid it then converts the letters into words.

Once the word(s) have been put together it goes through the final stage of Validation. It checks through the ‘Collins Scrabble Words’ for the word(s), if they are not within the text file then they are not valid and returns the user to the start of the turn. If valid then it calculates the Score of the word(s) and tallies them up onto your total score as shown above.

For example, If the image below was an input into the program. Each of the letters representing where the player has placed the letters in his hand and showing how the positioning of the letters matters.

Above shows an example of a valid input. When the turn is ended it will work out the scoring for each letter and the tiles it has been placed upon.

The following example involves two separate turns:

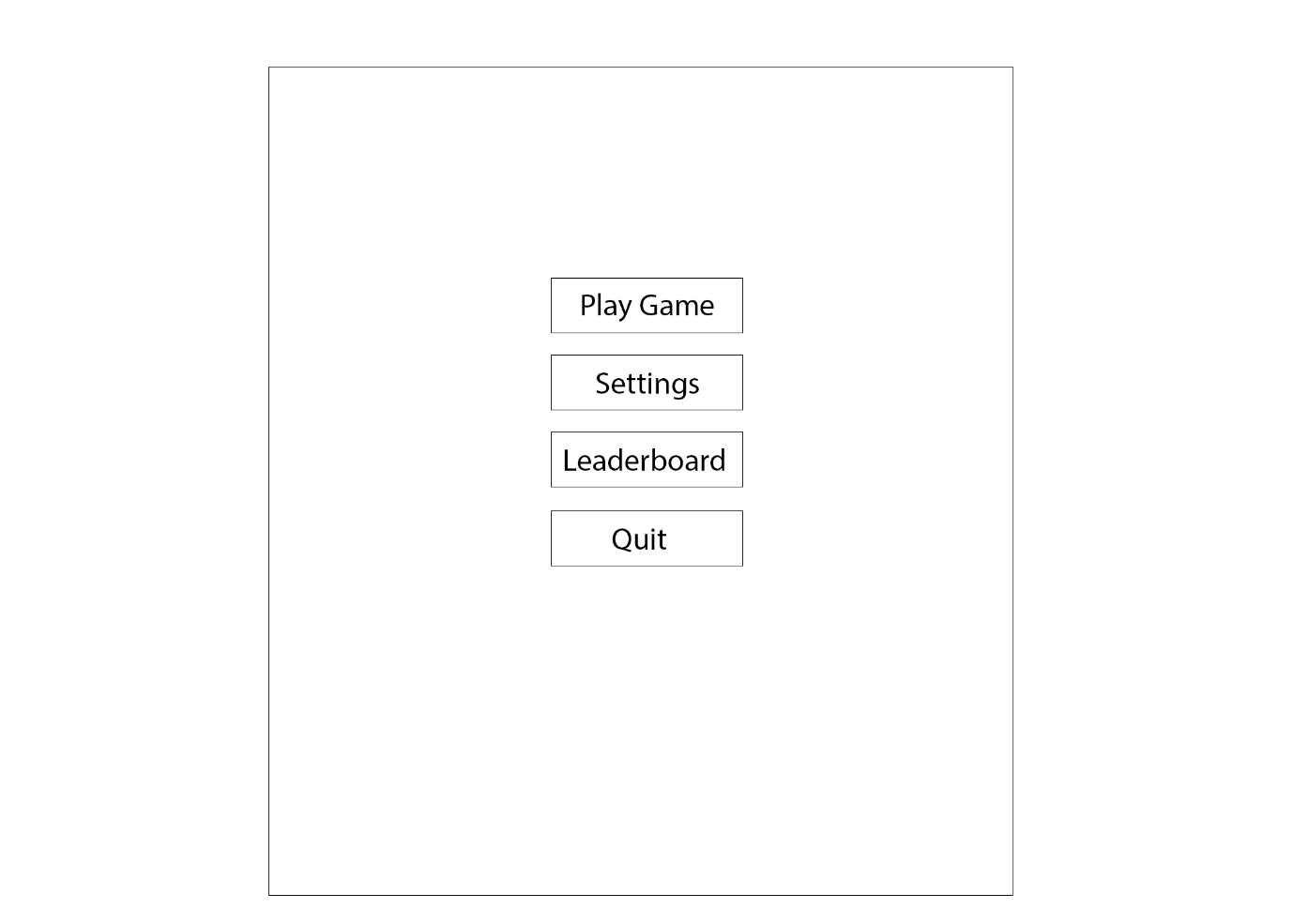


Turn 1: Player1 places the letters “ALWAYS” in that position.

Turn 2: Player2 places the “AND” in the following position.

Both of these turns are valid as all letters are adjacent to one another. They also create valid words, read from ‘top to bottom’ and ‘left to right’ within the Collins Oxford dictionary.

User Interface

The player will be met with a ‘Main-Menu’ UI as soon as the program is run. Dependent on which button is clicked will determine the form opened. For instance, if ‘Play Game’ is clicked the ‘Play Menu’ form will open this will hold another series of buttons as well as a return so the user can return to the previous form. Each form will have its own unique purpose.

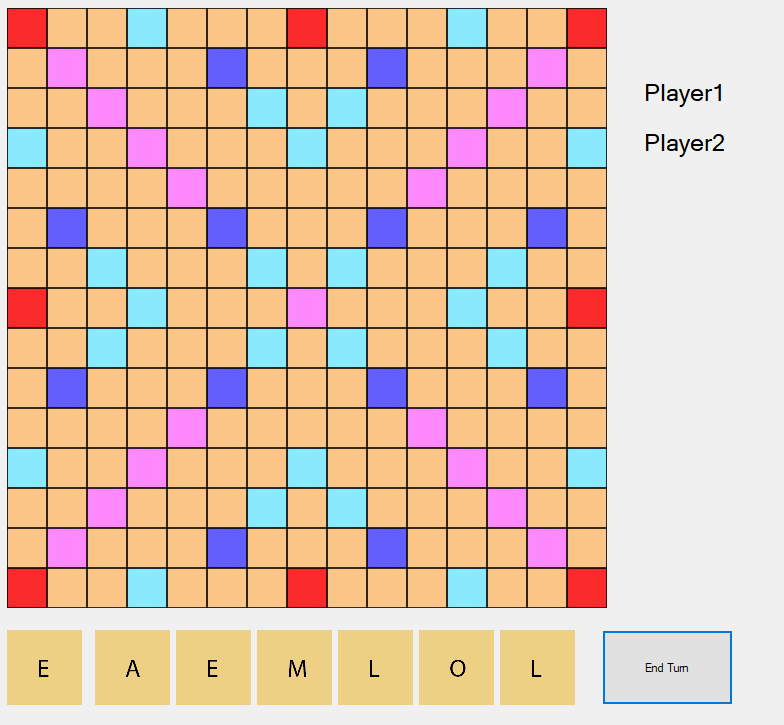
The Leaderboard will be linked to a database that holds each unique user’s highest score within a game. Both users scores being added at the end of a game. Going back to the unique users, each user will have a username and password that they can create on the menu. This option will appear on the ‘Play Menu’. If no username or password is entered, then it will give the database null values of ‘Player1’ and ‘Player2’ respective of that player’s score. The database ranks the scores, in order, from highest to lowest, possibly only showing the ten highest scores.

|  |  |  |
| --- | --- | --- |
| Username | High Score | Date |
| ChickenIsNice1 | 180 | 07/04/2019 |

What might also feature, is the highest score within a round. It would mean that separate leaderboards would have to be created. A template of how one of the leaderboards will look:

|  |  |  |
| --- | --- | --- |
| **Position:** | **User:** | **High Score** |
| 1 | ChickenIsNice1 | 180 |
| 2. | Player1 | 137 |
| 3. | Justcause2 | 117 |
| 4. | ScrabblerGod | 82 |

sThe layout for the game itself will consist of the player’s scores, the board, letters in ‘rack’, the current player’s turn and a button to end the turn. ‘Player1’ and ‘Player2’ labels are current placeholders but will be replaced with the usernames followed by the scores. Below shows the fundamental layout but is subject to change.



The letter tiles used to play scrabble are not currently set to any score values. This is because the user will be able to individually change the points of each letter prior to the game (within the ‘Play Menu’ form), that includes a button to reset them to their default values. As the letters are dealt into the Player’s racks, they will be assigned the corresponding scores with labels.

When a tile has been selected (clicked) it goes red. It clearly shows the user that they have successfully selected the letter and are now able to place on the board. If the tile is clicked again it will return to the original tile and the user may select another.



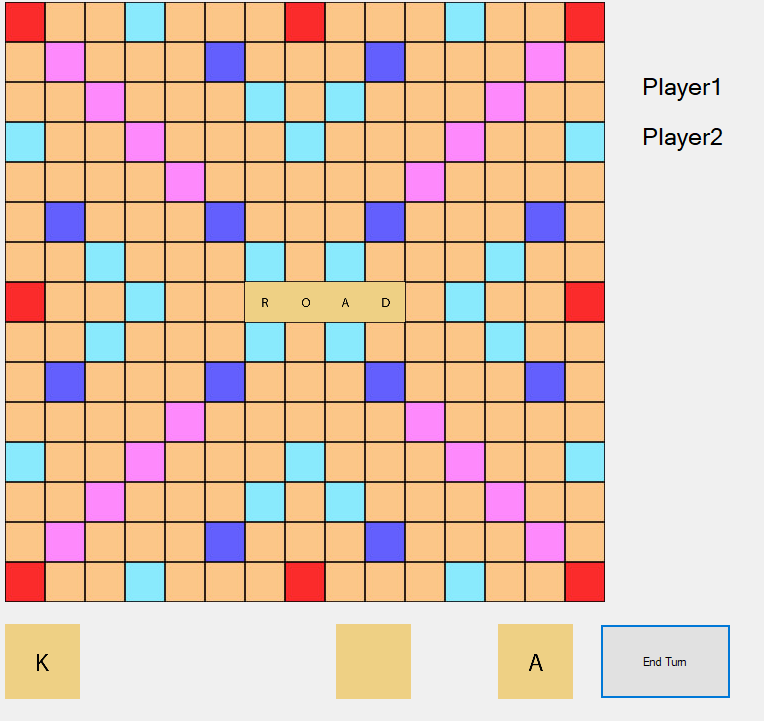
Once the letter is placed the tile goes invisible, meaning it can no longer be selected by the user.



Snapshot taken with a new set of letters. But as you can see the letter disappears once placed on the board. To place a letter, you click on the intended ‘empty’ tile within the board’s picturebox. In the future you will be able to click this tile again to put it back into your rack (if in the same turn). Once the ‘end turn’ button is clicked the turn will switch over to the next player, now showing their hand. The previous player’s hand will now be replenished with the number of letters they used in that turn (up to seven).

Placing a letter:



An example of using the letters ‘A’,’O’,’R’,’D’ to spell out ‘ROAD’. It is done by simply clicking the letter to select it and then clicking on the board where you would want it to go. The algorithm then auto aligns the letters based off the mouse cursor’s x and y coordinates. The program will flag up any illegal moves, say the letters placed aren’t in the same row or column.

When the ‘end turn’ button is clicked the score is tallied up. In this example, following the basic point system of scrabble ‘R’ (1) + ‘O’ (1) + ‘A’ (1) + ‘D’ (2) = 5 points multiplied by two (5 x 2) due to it being on a Double Word Score tile, giving a total score of 10. If it is player1’s turn, the label in the top right will be changed to “Player1: 10”. For example, the player’s turn can be determined by the colour of the label, if Player2 is green then it is their turn.

Player1: 10

Player2: 0

This is only a placeholder but does a great job of differentiating the Player’s turns.

Pseudo code:

First of all, we need the program to decide which player will go first. In the scrabble rules it’s the player with the closest letter to ‘A’ or ‘-‘. It means that we must first supply each player with seven letters. It will all be done within a ‘DecideFirst’ Function. The function will be used as we will return a true or false statement. If ‘true’ it will be Player1’s turn. However, an issue still stands, if both players have a letter the same distance away from ‘A’ then a coin flip is used. A quick and simple method of fixing the problem.

Dim LowestLetter\_Hand1 As Integer = ASCII(Player1\_Hand())

Dim LowestLetter\_Hand2 As Integer = ASCII(Player2\_Hand())

Loop until 7

If ASCII(Player1\_Hand(i)) is less than the current lowest value

“Then lowest value is now equal to that value”

LowestLetter\_Hand1 = ASCII(Player1\_Hand(i))

ElseIf ASCII(Player2\_Hand(i)) is less than the current lowest value

“Then lowest value is now equal to that value”

LowestLetter\_Hand2 = ASCII(Player2\_Hand(i))

End If

End Loop

If LowestLetter\_Hand1 < LowestLetter\_Hand2

“Player1 goes first”

Return True

ElseIf LowestLetter\_Hand2 < LowestLetterHand1

“Player2 goes first”

Return False

Else

Coin flip takes place giving each player a fair chance of going first

“Will be a 0 or a 1 and will be converted into a Boolean”

Return value

End If

The ASCII() is used to convert the characters ‘A’ to ‘Z’ to a decimal value, it is then easy to compare the values. The ‘-‘ value (which represents the BLANK tile) is already equal to ‘42’ in the ASCII table, which meant that no further values had to be set to make it less than ‘65’ (the ASCII value for the letter A). However, as mentioned before we need to already know the ‘racks’ of both players before we can decide who goes first. The ‘DealLetters’ subroutine will work this out for us.

First of all, we need to know the rack size which by default will be set to seven. User will be able to set this to another value in the pre-game menu. The rack size determines how many letters a player can have in his hand at one time. The next step is knowing how many of each tile exists and then taking one away each time the letter is dealt. A letter can be taken back by clicking on it, however this process is done in a separate subroutine (as long as it has been placed in the current turn). The final step is sending both players hands to the ‘Draw()’ subroutine that will add the GUI to the game. After the ‘DecideFirst()’ as mentioned above is declared.

Variables:

* The size of the ‘Hand’, by default seven letters maximum – ‘Dim HandSize’
* An array that holds all the letters. This could be done with ASCII values but thought it would be more clear to show.
* Another array that holds the information for the number of each letter are left. Say, a letter is put into a player’s hand, when this is done it needs to be taken away from the bag as to stop letters from being created out of thin air. The amounts will be listed in alphabetical order so the for loop can sort through and deduct.

The comments (in green) help to show this. Later in the pseudo will show this at work.

* The next few variables are to differentiate the Bag to the ‘newly’ created bag (after the letters have been dealt) to stop exceptions from happening: ‘NewBag’, ‘NewBagSize’, ‘BagSize’, ‘NewBagRnd’, ‘BagRnd’
* We also need to use a random number generator from 0 to 25 to pick a random letter within the alphabet. The ‘rndLetter’ variable as ‘New Random()’ does this.

### Pseudo Code:

Loop: to the ‘Bag’ length

Creates the new bag and lists all the possible letters

NewBag += Letter(i)

EndLoop

\*A later mistake I made with the program was not doing this, each letter had the same chance of coming up, so was very hard to string together words with very few vowels and the letter ‘Z’ having the same probability as the letter ‘E’\*

NewBagSize = The Length of the New Bag

BagSize = The length of the Bag (not the newly created one)

Loop: to the number of ‘Letters Played’.

Loop:

Picks a random number within the limits of ‘NewBagSize’ and ‘BagRnd’.

NewBagRnd = Random(0 to NewBagSize)

BagRnd = Random(0 to BagSize)

EndLoop: when Bag(BagRnd) is not equal to 0 – Checks that not all the letters have been used.

Now it takes the Letter that has been used away from the Bag

Bag(BagRnd) take-away one

After all of this, the letter is added onto Player1’s hand.

This is again repeated with Player2’s hand.

The code will be changed so both players don’t have changes made to their hands, this is only done for testing purposes in the first turn.

Player1\_Turn = DecideFirst(Player1\_Hand, Player2\_Hand)

LettersPlayed = 0

Draw(Player1\_Hand, Player2\_hand, PlayerTurn)

The next subroutine, is the ‘Draw()’. As the title suggests, it’s this that draws the GUI of the game, though a lot of it is already drawn out in the forms. However, it does not deal with all the GUI, only the basics. For instance, displaying each Player’s hands dependent on who’s turn it is and also displaying which Player’s turn it is. Compared to the other two it is much more basic to program so no pseudo code is needed for it.

The next part we need the program to do is actually placing the letters on the board and being able to take them back off again. To do this I have split it into two separate functions, one of which will have to detect when a letter has been selected and the other will have to detect where the user wants it to go. The first function will be ‘Letter\_Select()’, it will handle each of the seven letter images and detect when they are clicked. The image of the letter that’s been clicked will then show a graphical change, e.g. goes the colour red to signify that it has been selected. It will return back to its original image if deselected or go blank if placed on the board.

The Subroutine Handles each image of the letters:

‘Handles Letter1.Click, Letter2.Click, … ‘

A DirectCase is used to get the images name and to differentiate which one has been clicked. Differentiation is necessary as the function is called when any of the seven images is clicked so we have to tell the program which letter has actually been selected (to place).

‘Dim LetterClicked As PictureBox = DirectCast(sender, PictureBox)’

Select Case Name of Letter Clicked

Case “Letter1” – name of the first letter picturebox

LetterSelected = 0

Case “Letter2”

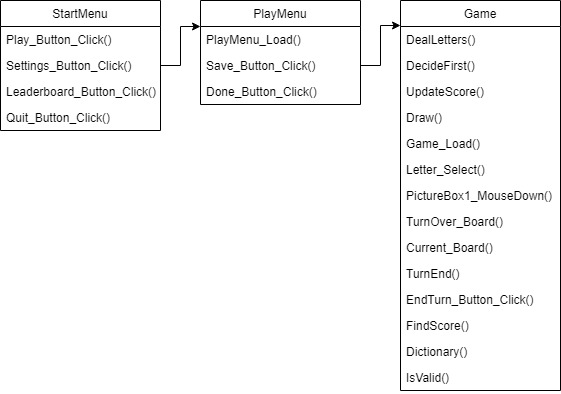
Letter Selected = 1

…

This is repeated up to “Letter7”.

A Boolean called ‘Selected’ will be used so we know whether we are selecting another letter or just deselecting the one we already have selected. For instance,

UML for Forms



Database

For my Scrabble program I will be using a database in SQL to record user’s high scores and other interesting values I’d like to compare. Examples of these have referenced before in the document, however I will clearly list them below:

* Highest scores players got at the end of a game. Ranked from highest to lowest but will only display the top ten scores.
* Highest scoring turn. Again, will rank them in the top ten but only updated at the end of the game to reduce references to the online database.
* Each user when running the program will be asked for a username and password. If either isn’t entered corrected then an exception will be made, setting each user to the default ‘Player1’ and ‘Player2’. When the scores are sent, they will be accompanied by these usernames.
* Ability to ‘save’ games to the database.

The database will be using ‘000webhost’ to save data to a cloud. This is so users can save their scores on any system, making the game easy-to-access.

<https://www.000webhost.com/>

Designing the databases themselves will require a bit of planning. A variety of tables will need to be used. Probably one for each bullet point. The main table, that will contain the high scores is displayed below:

### Users Table:

The ‘users’ table is the first table in the Database, it holds the inputs for the username, password and score for each user. The username and password then needs to be verified with the ‘account’ table. If the username doesn’t exist it will create a new user in the database and set its password. However, if the username does already exist the password will have to match that in the ‘account’ table, otherwise it will inform the user “The password you have entered is incorrect”.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Users | **Primary Key** | Username |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Username | TEXT | 16 | Primary Key. Each user has a different username hence why it is the Primary Key. This is so each user’s scores can be differentiated from one another. |
| Password | PASSWORD | 20 | So each user’s account is secure, no one can tamper with the user’s high score. |

Once the username has been verified as correct or created it will then compare the score to that of the top 10 high scores (if they all have values set) and put it in its appropriate position. To do this, the score will be compared to the ‘PointsLeaderboard’ table and once the position has been found it will set change the appropriate value of the ‘UsersLeaderboard’ table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Points\_Leaderboard | **Primary Key** |  |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Username | TEXT | 16 |  |
| HScore1 | INT | 4 | Each of the following fields holds the integer values of the scores. |
| HScore2 | INT | 4 |  |
| HScore3 | INT | 4 |  |
| HScore4 | INT | 4 |  |
| HScore5 | INT | 4 |  |
| HScore6 | INT | 4 |  |
| HScore7 | INT | 4 |  |
| HScore8 | INT | 4 |  |
| HScore9 | INT | 4 |  |
| HScore10 | INT | 4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Users\_Leaderboard | **Primary Key** |  |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Username | TEXT | 16 |  |
| HUser1 | TEXT | 16 | Each of the following fields holds the usernames for the corresponding (high-scores) in size order. |
| HUser2 | TEXT | 16 |  |
| HUser3 | TEXT | 16 |  |
| HUser4 | TEXT | 16 |  |
| HUser5 | TEXT | 16 |  |
| HUser6 | TEXT | 16 |  |
| HUser7 | TEXT | 16 |  |
| HUser8 | TEXT | 16 |  |
| HUser9 | TEXT | 16 |  |
| HUser10 | TEXT | 16 |  |

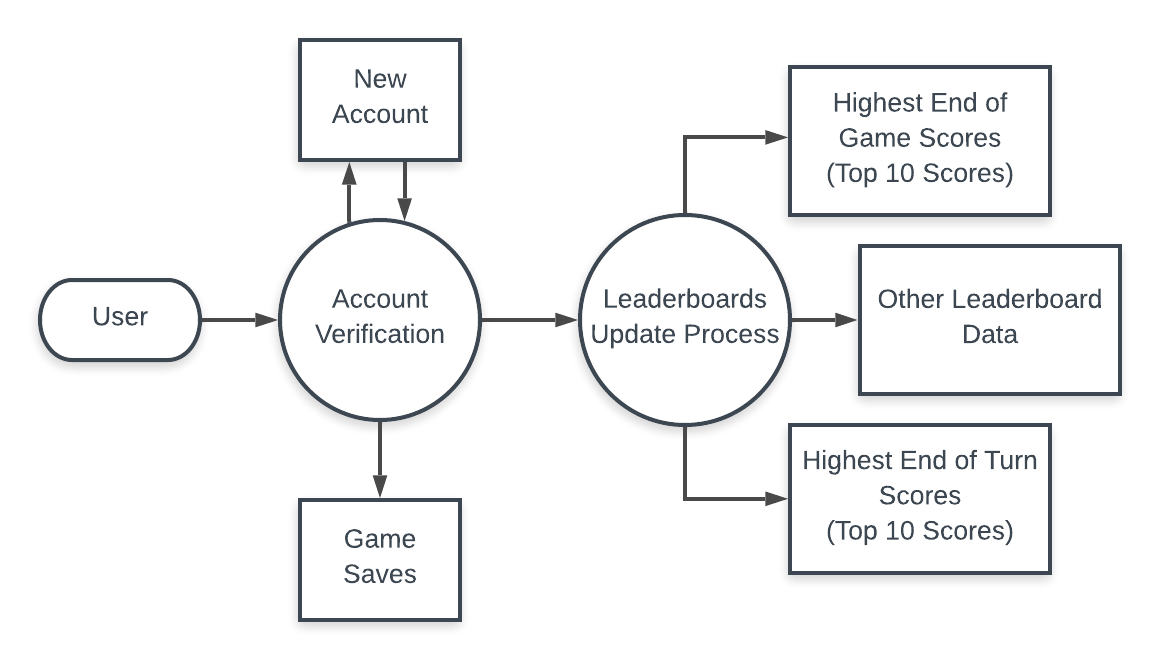
|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | User\_Saves | **Primary Key** | Username |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Username | TEXT | 16 | Primary Key |
| Player\_Hands | TEXT |  | Linked to a separate table “Save\_Hands” holding each user’s hand/rack of letters. |
| Letter\_Coords | INT |  | Linked to a separate table “Saves\_Coords” holding each letters coordinates on the board (placed). |
| Player\_Points | INT |  |  |
| Turn\_Number | INT |  |  |
| Bag | TEXT | 100 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Saves\_Hands | **Primary Key** |  |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Player1\_Hand | TEXT | 7 |  |
| Player2\_Hand | TEXT | 7 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Saves\_Coords | **Primary Key** |  |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Letter\_Coords\_X | INT | 15 |  |
| Letter\_Coords\_Y | INT | 15 |  |

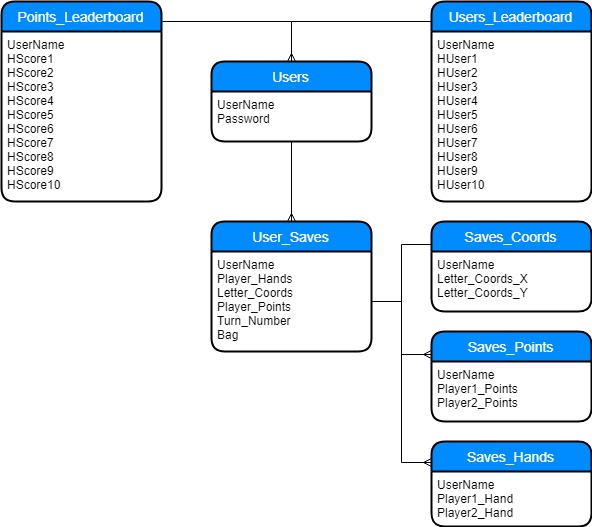
|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | Saves\_Points | **Primary Key** |  |
| **Field Name** | **Data Type** | **Field Size** | **Comments (can be left blank)** |
| Player1\_Points | INT | 4 |  |
| Player2\_Points | INT | 4 |  |

For example:



ERD

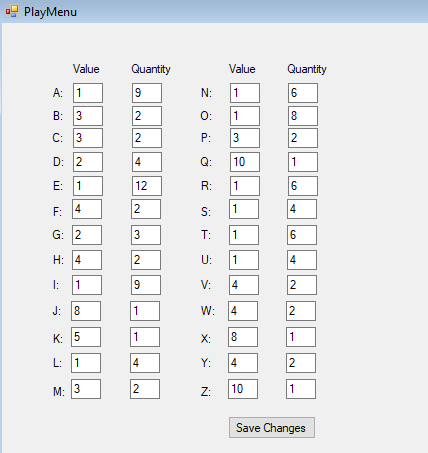
The normalised ERD helps to show how the database would be set-up when implemented into the program. As you can see there is a many to one relationship two both the ‘Users\_Leaderboard’ and the ‘Points\_Leaderboard’ tables, this is because the leaderboard holds many users. However, each user can have many saves, this is why a one-to-many relationship exists between ‘Users’ and ‘User\_Saves’.



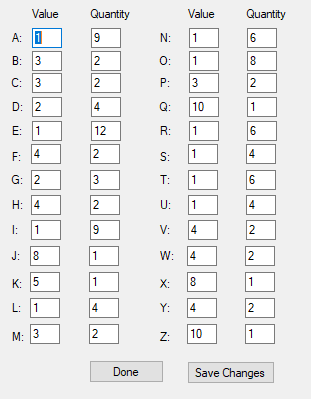
The Database itself is very simple to set-up and illustrate with most tables interconnecting with one another and all sharing the same key ‘UserName’.

## Pre-Game Settings Menu:

The idea of the ‘pre-game settings menu’ is so the user can modify the values of each letter and quantity of each. I wanted to use a simple UI so users could easily navigate the system. Below is the finished project. Later on, I hope to add different map types, each with their own tile types and layouts. As an ‘extra’ objective the possible implementation of separate game-modes, each having a unique ruleset. These are listed in the Analysis (pages 8-9).



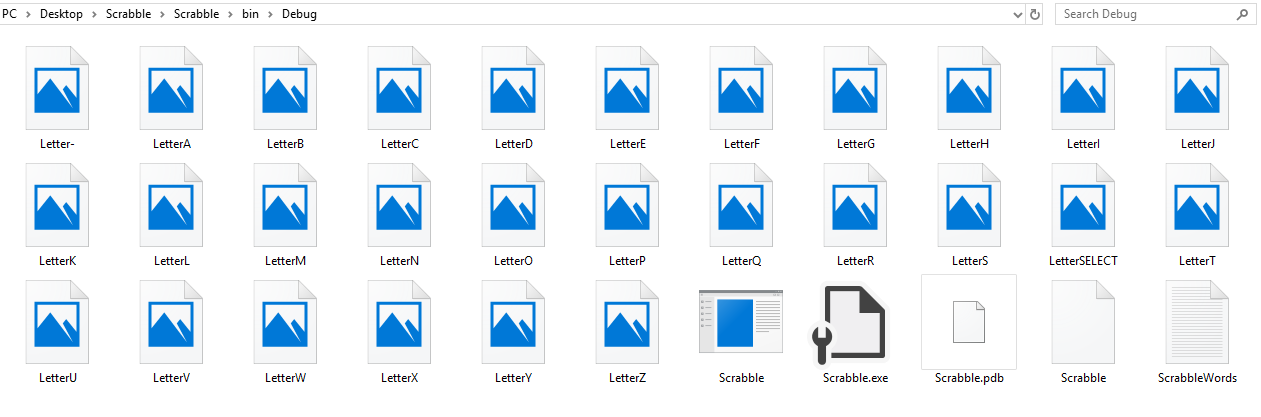
By default each letter is set to the standard scrabble value and quantity with 100 tiles in the ‘bag’. However, the user can change to this to what suits them. I later added a button called ‘Done’ so it could progress onto the actual ‘play’ aspect. Here’s how it looks:

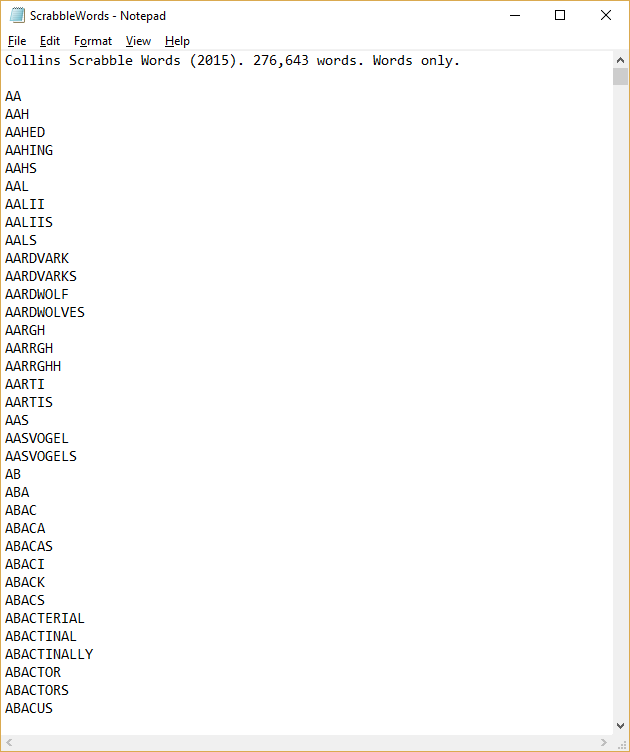


This will be further tested in the Testing chapter later on.

Inside the Debug Folder

Below shows the debug folder, found within the Scrabble\Scrabble\bin\Debug directory. Here holds each of the graphics for the Letters (“A” to “Z”, and “-“), also including the red graphic for when a letter is selected (LetterSELECT). However, the most important feature is the “ScrabbleWords.txt”. Inside this contains all the ‘viable’ words for a scrabble game and this is what the program uses to compare the words placed to the words in the dictionary.





Using the Collins Scrabble Dictionary.

Technical Analysis

## Game Form:

Private Player1\_TotalScore As Integer

Private Player2\_TotalScore As Integer

' Tracks which player's turn it is

Private Player1\_Turn As Boolean

' Each holds seven letters for both players

Private Player1\_Hand As String

Private Player2\_Hand As String

Public NewLetterPositions(14, 14) As Char ' letters placed in the current turn

Public FinalizedLetterPositions(14, 14) As Char ' Letters placed in previous turns

' Current letter selected

Private MovingLetter As Char = " "

Private PlayedLetters(6) As Char ' Number of letters used in a single turn

Private LetterSelected As Integer ' Letter selected in integer form, where it is placed in the hand

Public placedLetter(14, 14) As PictureBox

Private LetterPoints() As Integer = PlayMenu.IntLetterPoints ' Gets the values of each letter from the "PlayMenu" form

Private wordsInTurn As New List(Of String) ' All words created in the turn

Private Sub DealLetters(ByVal LettersPlayed As Integer) 'Change to values entered into the SettingsMenu later in development \* This has now been changed

Dim HandSize = 7 - LettersPlayed 'Will update after each turn - Subject to change

' Holds each letter

Dim Letter() As String = {"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", "-"}

' Gets the values from the "PlayMenu" form

Dim Bag() As Integer = PlayMenu.IntBag

' A slightly complex method used to create two separate bags

Dim NewBag As String

Dim NewBagSize As Integer

Dim BagSize As Integer

Static Dim rndLetter As New Random() ' Set as Static so each value is random. Doesn't work otherwise

Dim NewBagRnd As Integer

Dim BagRnd As Integer

Dim PlayerTurn As Integer

For i = 0 To Bag.Length - 1

For j = 0 To Bag(i) - 1

NewBag += Letter(i)

Next

Next

NewBagSize = NewBag.Length

BagSize = Bag.Length

For i = 0 To LettersPlayed - 1

Do

NewBagRnd = rndLetter.Next(0, NewBagSize - 1)

BagRnd = rndLetter.Next(0, BagSize - 1)

Loop Until Bag(BagRnd) <> 0

Bag(BagRnd) -= 1

Player1\_Hand &= NewBag(NewBagRnd)

Next

'!!Need to be differentiated between, dependent on which player's turn it is!! \* has been fixed

LettersPlayed = 7

For i = 0 To LettersPlayed - 1

Do

NewBagRnd = rndLetter.Next(0, NewBagSize - 1)

BagRnd = rndLetter.Next(0, BagSize - 1)

Loop Until Bag(BagRnd) <> 0

Bag(BagRnd) -= 1

Player2\_Hand &= NewBag(NewBagRnd)

Next

' Determines which player goes first (Only at the start of the game)

Player1\_Turn = DecideFirst(Player1\_Hand, Player2\_Hand)

' Not needed in the program but works well as a debugging tool \* will leave it in

MsgBox("Player1: " & Player1\_Hand & ", Player2: " & Player2\_Hand & ", Player1Turn: " & Player1\_Turn)

LettersPlayed = 0

' Function used to add the user interface

Draw(Player1\_Hand, Player2\_Hand, PlayerTurn)

'MsgBox(Player1\_Hand & " " & Player2\_Hand & " Player" & PlayerTurn)

End Sub

Private Function DecideFirst(ByVal Hand1 As String, ByVal Hand2 As String) ' Determines which player goes first (only in the first turn)

' Sets the first value in the hand to the lowest ASCII value. As the program runs it will determine the "actual" lowest value.

' The lowest value it can be is "-" or the "BLANK" tile that can be any letter, the second lowest is "A"... and so on

Dim Lowest1 As Integer = Asc(Hand1(0))

Dim Lowest2 As Integer = Asc(Hand2(0))

' Finds the lowest letter in terms of ASCII value. The lower the ASCII value the closer it is to the letter "A"

For i = 1 To 6

If Asc(Hand1(i)) <= Lowest1 Then

Lowest1 = Asc(Hand1(i))

ElseIf Asc(Hand2(i)) <= Lowest2 Then

Lowest2 = Asc(Hand2(i))

End If

Next

' Compares the lowest value in each hand. If Player1 has a lower value than Player2, Player1 goes first

If Lowest1 < Lowest2 Then

'Player1 goes first

Return True

ElseIf Lowest2 = Lowest1 Then

' If both hands have an equal value it goes to a 50/50 coinflip

Static Coin As New Random

Dim rnd As Integer

rnd = Coin.Next(0, 2)

Return rnd

Else

'Player2 goes first

Return False

End If

End Function

Public Sub UpdateScore(ByVal Score As Integer) ' This function updates the xcores for both players when a turn is ended

' Only one score needs to be updated, dependent on who's go it is

If Player1\_Turn Then

Player1\_TotalScore += Score

Else

Player2\_TotalScore += Score

End If

End Sub

Public Sub Draw(ByVal Player1\_Hand As String, ByVal Player2\_Hand As String, ByVal PlayerTurn As Integer) ' This subroutine "draws" the user interface

' The user interface includes: The player's hand (dependent on who's go it is) and displays who's go it is (colouring the font red)

'Contains each of the picturebox's within the player's hand

Dim Letters() As PictureBox = {Letter1, Letter2, Letter3, Letter4, Letter5, Letter6, Letter7}

Dim PlayersHands() As String = {Player1\_Hand, Player2\_Hand} ' Contains both Player1's and Player2's hands in string form

If Player1\_Turn Then

'If player1's turn it will change the colour of the text to red

Player1\_Score.ForeColor = Color.Red

Player2\_Score.ForeColor = Color.Black

For i = 0 To 6

Letters(i).Image = Image.FromFile("Letter" + Player1\_Hand(i) + ".png")

Next

Else

'If player2's turn it will change the colour of the text to red

Player1\_Score.ForeColor = Color.Black

Player2\_Score.ForeColor = Color.Red

For i = 0 To 6

Letters(i).Image = Image.FromFile("Letter" + Player2\_Hand(i) + ".png")

Next

End If

End Sub

Private Sub Game\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

' Subroutines starts up on load

' Hides the "PlayMenu" on load as it is no longer needed

PlayMenu.Hide()

For x = 0 To 14

For y = 0 To 14

' Sets all letter positions (x,y) to nothing

NewLetterPositions(x, y) = ""

FinalizedLetterPositions(x, y) = ""

Next

Next

Player1\_TotalScore = 0

DealLetters(7)

PictureBox1.SendToBack()

End Sub

Private Sub Letter\_Select(sender As Object, e As EventArgs) Handles Letter1.Click, Letter2.Click, Letter3.Click, Letter4.Click, Letter5.Click, Letter6.Click, Letter7.Click

' This subroutine runs when any of the seven pictureboxes (in the hand) are clicked

' Set as a Static variable as we need to save whether a letter has been selected or not, even when the subroutine isn't being run

Static Dim Selected As Boolean

' An array that holds each of the seven images. Makes it easier to hold them in this form

Dim Letters() As Image = {Letter1.Image, Letter2.Image, Letter3.Image, Letter4.Image, Letter5.Image, Letter6.Image, Letter7.Image}

' This variable checks which picturebox has been clicked as we need to differentiate it to the rest

' Very important that we do so

Dim LetterClicked As PictureBox = DirectCast(sender, PictureBox)

' This is needed so we can update the selected picturebox

LetterClicked.Image.Dispose()

' Informs the program which letter has been selected

Select Case LetterClicked.Name

Case "Letter1"

LetterSelected = 0

Case "Letter2"

LetterSelected = 1

Case "Letter3"

LetterSelected = 2

Case "Letter4"

LetterSelected = 3

Case "Letter5"

LetterSelected = 4

Case "Letter6"

LetterSelected = 5

Case "Letter7"

LetterSelected = 6

End Select

' These if statements determine whether the picturebox (that has been clicked) is being deselected

' .. or selected. Changing the way it looks

If Selected Then

Selected = False

MovingLetter = " "

If Player1\_Turn Then

' Changes the image of Player1's hand to the default letter image

' i.e deselecting it

LetterClicked.Image = Image.FromFile("Letter" + Player1\_Hand(LetterSelected) + ".png")

Else

' Same for Player2

LetterClicked.Image = Image.FromFile("Letter" + Player2\_Hand(LetterSelected) + ".png")

End If

Else

' This sets the picturebox (that was previously deselected) to selected

' changing its image

Selected = True

If Player1\_Turn Then

MovingLetter = Player1\_Hand(LetterSelected)

Else

MovingLetter = Player2\_Hand(LetterSelected)

End If

LetterClicked.Image = Image.FromFile("LetterSELECT.png")

End If

End Sub

Private Sub CursorPosition()

Dim CursorPosX = Cursor.Position().X

Dim CursorPosY = Cursor.Position().Y

'Limit = 12,572

'limit (150 - 697, 95 - 695)

'Grid = 36.333 per tile

End Sub

Private Sub PictureBox1\_MouseDown(sender As Object, e As MouseEventArgs) Handles PictureBox1.MouseDown

' This subroutine is used to place letters on the board

' Detects when the board is clicked, where it is clicked and what letter is currently selected

' Gets the current cursor position of where the user clicked on the board (X and Y)

' For this to all work, the program has to be in fullscreen-windowed mode, if not, the coords are innacurate

Dim CursorPosX As Integer = Cursor.Position().X

Dim CursorPosY As Integer = Cursor.Position().Y

' Each of these values represents when a tile starts and ends on the board

' Each tile is 40x40 pixels across

Dim LetterPositionX() As Integer = {12, 52, 92, 132, 172, 212, 252, 292, 332, 372, 412, 452, 492, 532, 572}

Dim LetterPositionY() As Integer = {12, 52, 92, 132, 172, 212, 252, 292, 332, 372, 412, 452, 492, 532, 572}

' These variables were never used, but in theory would work if I had time to fix

' This would've made it so that the program could be played in any position on the screen, and not only in fullscreen-windowed

Dim DifferenceLetterX As Double 'Aligns the coords to the window no matter where it is (Attempt)

Dim DifferenceLetterY As Double

' As done before, this puts each picturebox for the seven letters in one array

Dim Letters() As PictureBox = {Letter1, Letter2, Letter3, Letter4, Letter5, Letter6, Letter7}

' These are worked out later in the program. These basically lock the tiles in place, in a grid,

' so they don't overlap the lines (other tiles)

Dim CurrentLetterPositionX As Integer

Dim CurrentLetterPositionY As Integer

' This creates the "grid" so letters fit snuggly in each box

CurrentLetterPositionX = (CursorPosX - 12) \ 40

CurrentLetterPositionY = (CursorPosY - 35) \ 40

DifferenceLetterX = 0 - CurrentLetterPositionX

DifferenceLetterY = 0 - CurrentLetterPositionY

' Used to add the values to the letters

Dim lbl As Label = New Label

lbl.Size = New System.Drawing.Size(40, 40)

lbl.Location = New System.Drawing.Point(LetterPositionX(CurrentLetterPositionX), LetterPositionY(CurrentLetterPositionY))

lbl.Text = LetterPoints(Asc(MovingLetter) - 65)

lbl.BringToFront()

Me.Controls.Add(lbl)

' This if statement makes sure that there are no letters currently on the tile they want to place

' if there is then it will not allow the user to place the tile there

If NewLetterPositions(CurrentLetterPositionX, CurrentLetterPositionY) = Nothing And FinalizedLetterPositions(CurrentLetterPositionX, CurrentLetterPositionY) = Nothing Then

' Makes sure that the user has a letter selected

If MovingLetter <> " " Then

' This algorithm creates new pictureboxs

' i.e places letters on the board that can also be removed

' The variable "picturebox" is set to a "New PictureBox". The "New" means that others can be created from this

Dim picturebox As New PictureBox

' Sets the name of the newly created picturebox to the X and Y coordiantes of where it has been placed

' Each of these is unique as only can be set to each tile

picturebox.Name = Str(CurrentLetterPositionX) + "," + Str(CurrentLetterPositionY)

' This simply tells the picturebox image to fill the whole picturebox, otherwise it is disproportioned

picturebox.SizeMode = PictureBoxSizeMode.StretchImage

' The most crucial line of code, setting the newly created picturebox to the correct image

' All dependent on which letter has been placed

picturebox.Image = System.Drawing.Bitmap.FromFile(My.Computer.FileSystem.GetName("Letter" + MovingLetter + ".png"))

' This is used to place the picturebox in the correct place on the board using the rounded X and Y coords

picturebox.Location = New Point(LetterPositionX(CurrentLetterPositionX), LetterPositionY(CurrentLetterPositionY))

' Sets the picturebox width and height to that of the tiles width and height (40x40)

picturebox.Width = 40

picturebox.Height = 40

' Sets the picturebox to visible (so can actually be seen on the board

picturebox.Visible = True

AddHandler picturebox.Click, AddressOf PictureBox1\_MouseDown 'Crucial for allowing the program to access the board (behind the placed letters)

' This final line adds the picturebox to the form

Me.Controls.Add(picturebox)

' Makes sure that the picturebox is not hidden behind the board

picturebox.BringToFront()

' Tells the program what the current board looks like (in that turn)

Current\_Board(MovingLetter, CurrentLetterPositionX, CurrentLetterPositionY)

'Adds the letter placed to the "PlayedLetters" array

PlayedLetters(LetterSelected) = MovingLetter

' Visibly hides the picturebox of the letter that has been placed (in the hand) to signify that it can no longer be placed

' and has been removed from the hand

Letters(LetterSelected).Visible = False

placedLetter(CurrentLetterPositionX, CurrentLetterPositionY) = picturebox

Replace(Player1\_Hand, MovingLetter, "")

Else

End If

ElseIf Not NewLetterPositions(CurrentLetterPositionX, CurrentLetterPositionY) = Nothing Then

' Removes letters (that have been placed in that turn and not previously ended turns) from the board

placedLetter(CurrentLetterPositionX, CurrentLetterPositionY).Visible = False

placedLetter(CurrentLetterPositionX, CurrentLetterPositionY) = Nothing

placedLetter(CurrentLetterPositionX, CurrentLetterPositionY) = Nothing

End If

'pwetty colours

End Sub

Public Class Board

'Holds the dimensions of the board (in fullscreen-windowed mode) for both new and old coords

Public ExactFinalizedCoords(572, 572) As Char

Public ExactNewCoords(572, 572) As Char

Public NewLetterPositions(14, 14) As Char

Public FinalizedLetterPositions(14, 14) As Char

End Class

Sub TurnEnd()

' When the turn is ended all "NewLetterPositions" are reset

Dim board As Board

Dim Valid As Boolean

Valid = IsValid()

If Valid Then

'Adds the letters to the final board layout (Will be done if the isValid is true)

For x = 0 To board.NewLetterPositions.Length - 1

For y = 0 To board.NewLetterPositions.Length - 1

If board.NewLetterPositions(x, y) <> "" Then

board.FinalizedLetterPositions(x, y) = board.NewLetterPositions(x, y)

End If

Next

Next

Array.Clear(board.NewLetterPositions, 0, board.NewLetterPositions.Length)

Else

End If

End Sub

Function IsValid() 'Will be called after 'End Turn' button is clicked

'First step is checking whether it is in a legal position

Dim board As Board

Dim LetterInY As Boolean

Dim LetterInX As Boolean

Dim Connected As Boolean = False

Dim ValidWord As Boolean

Dim SameWord As Boolean = False

Dim word As String

Dim Letter As Char

Dim i As Integer = 0

Dim SpecialTiles(,) As String = {} ' Contains the coords of the special tiles

'Checks X and Y to see if the new letter is in the same axis as another

For x = 0 To 14

Dim LettersInXCount As Integer

Dim LettersInYCount As Integer

For y = 0 To 14

'At the moment it does not count single letters added on to previous words

If NewLetterPositions(x, y) <> "" Then

LettersInXCount += 1

LettersInYCount += 1

End If

If LettersInXCount > 1 Then

LetterInX = True

End If

If LettersInYCount > 1 Then

LetterInY = True

End If

LettersInYCount = 0

'Now I have to check whether the the letter is adjacent to another

'Now I have to determine the validity of a word

'I have to create a rough grid with all letters so I can compare everything

If FinalizedLetterPositions(x, y) = Nothing And NewLetterPositions(x, y) = Nothing And SameWord Then

'Checks that there aren't any single letter words (illegal play)

If word.Length > 1 Then

Dim check = Dictionary(word)

MsgBox(word + " " + Str(check))

word = ""

SameWord = False

Else Return False

End If

ElseIf FinalizedLetterPositions(x, y) = Nothing And NewLetterPositions(x, y) = Nothing Then

SameWord = False

ElseIf NewLetterPositions(x, y) <> "" Then

word = word & NewLetterPositions(x, y)

SameWord = True

ElseIf FinalizedLetterPositions(x, y) <> "" Then

word = word & Str(FinalizedLetterPositions(x, y))

SameWord = True

End If

Next

SameWord = False

LettersInXCount = 0

Next

' This part checks the validity of the words created on the board and whether they are in the scrabble dictionary that I am using

For y = 0 To 14

For x = 0 To 14

If FinalizedLetterPositions(x, y) = Nothing And NewLetterPositions(x, y) = Nothing And SameWord Then

' Loads up the Dictionary() function and sets the variable "check" to the output

Dim check = Dictionary(word)

MsgBox(word + " " + Str(check))

word = ""

SameWord = False

ElseIf FinalizedLetterPositions(x, y) = Nothing And NewLetterPositions(x, y) = Nothing Then

SameWord = False

ElseIf NewLetterPositions(x, y) <> "" Then

word = word & NewLetterPositions(x, y)

SameWord = True

ElseIf FinalizedLetterPositions(x, y) <> "" Then

word = word & Str(FinalizedLetterPositions(x, y))

SameWord = True

End If

Next

Next

If LetterInX Or LetterInY Then

Else

Return False

End If

End Function

Function Dictionary(ByRef word As String)

' Reads the "ScrabbleWords.txt" document which holds the valid words

' If they are not in this dictionary, they are not valid and will be rejected

Dim FileReader As System.IO.StreamReader

FileReader = My.Computer.FileSystem.OpenTextFileReader("ScrabbleWords.txt")

Dim stringReader As String

Dim words(276643) As String

For i = 0 To words.Length - 1

' Searches throught the dictionary using a brute force method (not the most efficient) but still works almost instantly

words(i) = FileReader.ReadLine

If words(i) = word Then

' If it finds the word, it is valid and returns true

wordsInTurn.Add(word)

Return True

End If

Next

' If it doesn't find the word it returns false

Return False

End Function

Function FindScore()

Static TotalScore As Integer

Dim currentWord As String

For i = 0 To wordsInTurn.Count - 1

currentWord = wordsInTurn(i)

'Need to check each letter and whether it is on a letter multiplier tile

'Check whether the word touches a total word multiplier tile

Next

End Function

Private Sub EndTurn\_Button\_Click(sender As Object, e As EventArgs) Handles EndTurn\_Button.Click

IsValid()

TurnEnd()

End Sub

End Class

## PlayMenu Form:

Public Class PlayMenu

' Holds both the Values and the Quantities of each letter

Public IntLetterPoints() As Integer

Public IntBag() As Integer

Private Sub PlayMenu\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

' Sets the quantity and values to default values, in case the user doesn't change any of the values

' Stops errors from coming up

IntLetterPoints = {1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10}

IntBag = {9, 2, 2, 4, 12, 2, 3, 2, 9, 1, 1, 4, 2, 6, 8, 2, 1, 6, 4, 6, 4, 2, 2, 1, 2, 1, 2}

End Sub

Public Sub Save\_Button\_Click(sender As Object, e As EventArgs) Handles Save\_Button.Click

' When the "Save Button" is clicked all the textbox's values (changed or not) are saved to the two arrays

Static Dim LetterPoints() = {AValue.Text, BValue.Text, CValue.Text, DValue.Text, EValue.Text, FValue.Text, GValue.Text, HValue.Text,

IValue.Text, JValue.Text, KValue.Text, LValue.Text, MValue.Text, NValue.Text, OValue.Text, PValue.Text, QValue.Text, RValue.Text,

TValue.Text, UValue.Text, VValue.Text, WValue.Text, XValue.Text, YValue.Text, ZValue.Text}

Static Dim Bag() = {AQuantity.Text, BQuantity.Text, CQuantity.Text, DQuantity.Text, EQuantity.Text, FQuantity.Text, GQuantity.Text, HQuantity.Text,

IQuantity.Text, JQuantity.Text, KQuantity.Text, LQuantity.Text, MQuantity.Text, NQuantity.Text, OQuantity.Text, PQuantity.Text, QQuantity.Text, RQuantity.Text, SQuantity.Text,

TQuantity.Text, UQuantity.Text, VQuantity.Text, WQuantity.Text, XQuantity.Text, YQuantity.Text, ZQuantity.Text}

' The "LetterPoints()" and "Bag()" variables are then converted to integers as they are saved as strings in the previous arrays

' Again stops any errors from coming up

IntLetterPoints = Array.ConvertAll(LetterPoints, Function(str) Int32.Parse(str))

IntBag = Array.ConvertAll(Bag, Function(str) Int32.Parse(str))

End Sub

Private Sub Done\_Button\_Click(sender As Object, e As EventArgs) Handles Done\_Button.Click

' When this button is clicked the game form is displayed

Game.Show()

End Sub

End Class

## Starts Menu:

Public Class StartMenu

Private Sub Play\_Button\_Click(sender As Object, e As EventArgs) Handles Play\_Button.Click

PlayMenu.Show() 'Change to PlayMenu later

End Sub

Private Sub Settings\_Button\_Click(sender As Object, e As EventArgs) Handles Settings\_Button.Click

' Loads the "SettingsMenu" form when the "Settings" button is clicked

Application.Run(SettingsMenu)

End Sub

Private Sub Leaderboard\_Button\_Click(sender As Object, e As EventArgs) Handles Leaderboard\_Button.Click

' Loads the "LeaderboardMenu" form when the "Leaderboard" button is clicked

Application.Run(LeaderboardMenu)

End Sub

Private Sub Quit\_Button\_Click(sender As Object, e As EventArgs) Handles Quit\_Button.Click

' Program closes when the "Quit" button is clicked

Application.Exit()

End Sub

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

End Sub

End Class

Testing

## Selecting/Deselecting and Placing/Removing Letters Test:

|  |
| --- |
| What it should do:  The program should select letters by clicking on the letters in their ‘hand’, these will go red when selected to clearly show that the user has selected the letter. If placed on a ‘clear’ tile on the board it will go blank (disappear) so it **can’t** be placed again. The user can deselect a selected tile either by clicking on the same tile or clicking another tile in the user’s hand. This will return the selected tile from red to the tile it was initially assigned, e.g. the letter ‘A’ with its assigned point value.  The user should also be able to remove a placed tile (within the same turn it had been placed) back into their hand. |
| What it does:   * When selected it goes red. ✓ * When deselected it returns to its initial value. ✓   + Clicking the same selected letter in hand. ✓   + Clicking another letter within the hand (which is now selected). ✓🗶   + Once it is placed on the board. 🗶 * Ability to remove a tile from the board that has been placed in the same turn. 🗶 * Able to place a letter on the board ✓   The reason why *Point 4* has both a tick and a cross is due to it working but not working the way I would’ve liked. When selecting a letter – if a letter is selected – it will take three subsequent clicks before the new letter will be selected – otherwise – it will only take one click, as intended. |
| How to fix:  Fortunately, the bug is easy to remove, having to add a line of code to reset the PictureBox that is selected to its original image.  The code:  If Selected Then  Selected = False  MovingLetter = " "  If Player1\_Turn Then  LetterClicked.Image = Image.FromFile("Letter" + Player1\_Hand(LetterSelected) + ".png")  Else  LetterClicked.Image = Image.FromFile("Letter" + Player2\_Hand(LetterSelected) + ".png")  End If  Else  Selected = True  If Player1\_Turn Then  MovingLetter = Player1\_Hand(LetterSelected)  Else  MovingLetter = Player2\_Hand(LetterSelected)  End If  LetterClicked.Image = Image.FromFile("LetterSELECT.png")  End If  The variable ‘LetterClicked’:  Dim LetterClicked As PictureBox = DirectCast(sender, PictureBox)  Used to collect information and differentiate between the seven letters that are clicked.  When a letter is placed on the board the tile does indeed go blank. However, it does not deselect the letter. In other words, the user can place as many letters of the same type down if they do not select a different letter within the hand. To fix this I will have to update the currently selected letter the same time the letter goes blank. Again, this is a simple bug to fix but a crucial one.    All I had to do was set MovingLetter to “ “, it’s equivalent to a null value, inside the “PictureBox1\_MouseDown” subroutine. The user can no longer place multiples of the same letter. |
| How to fix:  The other issue we were faced with was having to click another letter (in the hand) two-three subsequent times after the first to select it. It isn’t a game breaking bug like the previous but is a great annoyance to the user and even myself when testing the program. It also means that the program displays that two letters at once have been selected, when in fact only one has.  The fix will have to involve, remembering the previously selected letter so it can be returned to its original (non-selected) image and the next letter will have to be selected on the first click.  To fix this issue I removed two lines of code:    The commented lines (in green colour text) are the two lines I removed. I replaced it with two ‘static’ variables, ‘PrevImage’ and ‘PrevLetterSelected’, as I wanted to remember the previous two image values to recall back to when the function was next called. These are set when there are no letter tiles selected:    The problem isn’t completely fixed to how I would’ve liked but instead of three to four clicks to deselect a letter and to select a new one it now only takes two |

## Assigning Player’s Hands Test:

|  |
| --- |
| What it should do:  When the ‘Game’ form is first run, it will randomly assort each player seven tiles (dependent on what game-mode the user is on) from the bag. Tiles are then taken away from the bag array so that once all the tiles of a particular set, e.g. ‘E’, don’t keep being drawn. |
| What it does:   * Assigns each player a rack of seven letters. ✓ * Some letters are more likely to come up than others (higher chance). 🗶 * Takes letters drawn away from the bag array. ✓   Rather than the letters have a varying chance of coming up, they all have the same equal chance. The reason for having varying chances is so that players can make words, for instance vowels are much more common to come up than consonants. The chance is dependent on how many of each letter there are in the bag, the more letter there are the higher the chance the player will draw it. For example, the letter ‘E’ has the most letter tiles in the bag with twelve and ‘Q’ has one of the lowest with only one letter tile. The letter ‘E’ is the highest as it is the most common letter in the English language, the more common a letter is, the lower the point value is. However, in this program the user has full control of these values. |
| How to fix:  So what I need to do is create a string that lists all the letters in the bag into one single line and then pick from 0 to the list’s length. A for loop will list this for me. The program will also need to know the letters played by the player in the previous turn as to not give out too many letters into their hand/rack.  First of all, we need to create a ‘newBag’ string variable, differentiating from the ‘bag’ array. The for loop that will create this is:    I then need to create two separate for loops for each player as each ‘NewBagRnd’ needs to be different. The loops will pick a random letter from the ‘NewBag’ string and add it to the player’s hand it will also remove a letter from the ‘Bag()’ array from the correct letter.      I had to add an exception within the for loop with the ‘Do’ loop where if there are no longer any letters of that type left in the bag it will pick another random letter from the string. Once the separate hand’s for each player have been made the program will send the information onto the ‘DecideFirst’ function which will return the player who goes first based off of the letters drawn, and the ‘Draw’ subroutine which will display the players hand on the screen, dependant on who goes first.    To test whether it had worked I ran the program a few times to see what letters I would get. I judged it by how many times I had the letter ‘E’ in my hand:          It isn’t the best way to test, and you could argue that the letter ‘E’ is popping up too much but in this small sample it shows that the letters with a higher chance of coming up are in fact coming up more than those with a lower chance.  To double check. I ran the program several times to graph the values given in each starter hand. Here are the results:    And now I’m going to compare it to a graph of how it should look:    Clearly with a higher sample size the results would be a lot more accurate with the program only being running twenty-two times.  When you compare the letters from ‘most’ frequent to ‘least’ frequent you find that the two graphs are very similar. The letters ‘E’, ‘A’, ‘I’ and ‘O’ appear in the top four, though a slightly different order, it all comes down to the sample size. The percentages on the right again show the similarities in frequency. With the graphs having very similar results to one another I believe that this is great evidence to show that programs ‘DealLetter()’ function is working properly and to what was initially intended. |

## Determining Which Player Goes First Test:

|  |
| --- |
| What it should do:  Running the program for the first time, on a new game (cleared board), a ‘DecideFirst()’ function is used to decide which player will go first based off the letters in their hands. This is mentioned in more depth on page **\*FILL IN LATER\***. |
| What it does:  To check this, I had to know which player the program had selected to be first, to do this I used a simple *MsgBox* within the ‘DealLetters’ function:    If the program returned ‘True’ that means the program picked ‘Player1’ to go first, if returns ‘False’ then it is ‘Player2’ turn:      To verify whether the program is correct, I must compare the hands of both players and see if the one with the lower letter wins. If, however, they have letters of equivalent value it will go to a 50/50-coin flip, the simplest of ways to settle it. The program is run until each scenario is checked.  I modified the code slightly to make it clearer:    The first scenario is ‘Player1’ being selected to go first, when the program is run it displays all the variables. In this case, Player1’s hand has the letters “CYAVEER”, the letter ‘A’ being the lowest so that will be the value set for the hand. Player2’s hand however has the letters “ISTWCIE”, the letter ‘C’ being the lowest in this hand. Values of each letter are sorted with their equivalent ASCII (American Standard Code for Information Interchange), a character set much like Unicode but on a much smaller scale. The Letter ‘A’ for instance having the integer value of 65 and ‘Z’ having 91.    ‘Player1Turn’ returning true is the correct output. The only letter tile below ‘A’ is the ‘BLANK’ tile, this would’ve allowed Player2 to have gone first. The second scenario is when Player2 wins the draw by having a letter below that of Player1’s. The scenario being identical to that of the first, I just want to make sure the program is working properly:    As you can see, Player2 has the lowest letter tile of ‘A’ meaning that Player1 either needs a letter ‘A’ or a ‘BLANK’ which is represented as a ‘-‘ in the program. As I mentioned before, Player1 could have gotten the letter ‘A’, if the program has a letter of equivalent value in both hands it settles the decision with a simple coin-flip. To test whether the coin-flip is in fact random I ran the program several times, also adding a further line of code within the coin-flip to show that it was running:      Examples of a coin-flip taking place:        The examples show that the ‘DecideFirst()’ function is working properly and to the extent I would’ve liked. It also shows the right amount of randomness, showing that it is a 50/50 chance. |
| Conclusion:   * Picks the player to go first based off of their lowest letter in hand. ✓ * If Players share the same lowest-letter-value it will got to a coin-flip which will decide. ✓   Each point has been completed to the full and thoroughly tested. The program working exactly to how I wanted. |

## Placing Letters Test:

|  |
| --- |
| What it should do:  After selecting a tile from the Player’s hand, they should be able to place it where they click on the board. The letter will then hide itself once placed and appear on the board, to create the illusion that it has been placed. |
| What it does:   * When a letter in the hand is clicked, it is highlighted with a red colour. ✓ * When the board is clicked the letter in the hand hides itself. ✓ * Letter is placed where the user clicked. ✓🗶   + Scales to the appropriate board size. ✓   + Is fully visible. ✓   The reason why the third point has a tick and cross is because the coordinate system used to place the letters on the board only works when the game is in full screen. To solve this I’m going to have to force the game into fullscreen mode, instead of reprogramming the function |
| How to fix:  Fixing the problem is simple, only adding one line of the code to the ‘Game\_Load()’ private subroutine. The line of code used:    It simply forces the game, once loaded, into a maximized window. The user before, would have to manually put the game into a maximized mode, so to a new user it would be incompatible. |
| Conclusion:  Maximizing the window is not the solution I would’ve liked. Instead I would’ve used the local position of the PictureBox for the board to place the letters, rather than using the local positions of the screen itself and dividing it into its 15x15 tile layout.    Above is the code unchanged. |

## Ability to Change the Quantities and Values of Each Letter:

|  |
| --- |
| What it should do:  When the user wants to start a game and clicks the ‘Play Game’ button they will be met with a screen aptly called ‘PlayMenu’, or in other words the pre-game menu. Here, the user can change the values of letters and the quantity of them. So instead of the letter ‘A’ being worth a single point the user could change it to 10, and instead of it having a 9/100 of being selected for the users hand it could have a 1/20 chance simply by changing the quantities of each letter.  Different maps and game-modes are another possible addition to the program but due to time constraints it is unlikely that they will be implemented. |
| What it does:  When the user clicks the ‘Play Game’ button:    It loads the ‘PlayMenu’ as intended and shows it:    The user should then click the ‘Save Changes’ button (if any changes have been made). Now when the user clicks ‘Done’ the game should load.    To test whether changing the quantities works I’m going to set them ridiculously high:  Firstly, by setting Z, which usually has a quantity of ‘1’ to ‘1000’:      As you can see, all Player1’s hand is made up of Z’s. I will do this once more but with the letter ‘B’:    Again, majority of letters except one are ‘B’. It clearly shows that the system is working how it should.  An issue I came across when testing the system is that if you don’t click the ‘Save changes’ button then an error would occur. This error happens because no values have been set (still set to null) so an exception appears when calculating the ‘bag.length – 1’. Because the bag has no values set it returns an error when taking away 1.     * Ability to change the values of each letter. ✓ * Ability to change the quantity of each letter. ✓ * Ability to change maps and game-modes. 🗶 |
| How to fix:  There are two ways of dealing with this problem. The first is to remove the ‘Save Changes’ button and just have the ‘Done’ button, this means that when it is clicked the standard (even if not changed) get saved. The second option is checking if the user has saved, if not, it sets the values to their defaults.    Simply by adding:    Without clicking ‘Save Changes’ this time around worked perfectly with no exception errors:    The reason why the ability to ‘change levels and game-modes’ |

|  |
| --- |
| What it should do: |
| What it does: |
| How to fix: |

User Feedback

### Feedback Form 1

|  |  |
| --- | --- |
| **Feedback Form**  Name: Tom Hoddison | |
| **User Requirements** | |
| Anything you liked/disliked about the game? | It was very easy to use and liked the simple interface. |
| Did it work as well as other Scrabble games? | I’ve only played a little bit of scrabble before. It was mostly just the physical board game. And, yeah, I couldn’t differentiate it from other scrabble games I’ve played. Very easy and fun to play. |
| Did you win at all? | \*haha\*. Yes, in fact I did. Lost my second game though, blame my hand. |
| **Usability** | |
| Easy to use user interface? | Very clean and easy to use, however looks a little too rough in places. Could do with some flashier UI. Would have liked it so that the screen could be viewed at all sizes, little annoying that you had to play in maximized windowed mode. |
| What do you mean by ‘flashier UI’? | Just a stylised user interface, to make It stand out. |
| Response of controls? | Selecting letters was a little annoying, as had to sometimes click it twice. |
| Do all the buttons work? | Yes I believe that all the buttons did work. |
| Text easy to read? | Yeah the text was fine, didn’t really notice it. |
| What improvements can be made? | * Stylised user interface, so doesn’t look so basic. * More screen options, other than windowed maximised. * Possible sounds in the game or music. |
| **Performance** | |
| Did the game perform as you expected it to? | Yes it worked perfectly fine, just what I envisioned it to be. |
| Are there any errors within the game that you noticed? | No I didn’t notice any errors whatsoever. |
| Does the game load quickly? | The game is almost instant on loading up, and is perfectly fine after that. |

### Feedback Form 2:

|  |  |
| --- | --- |
| **Feedback Form**  Name: Jack Robinson | |
| **User Requirements** | |
| Anything you liked/disliked about the game? | Basic UI made it easy to use. |
| Did it work as well as other Scrabble games? | Never played Scrabble electronically before, maybe I should play more. |
| Did you win at all? | I played three games and one two. Very happy. |
| **Usability** | |
| Easy to use user interface? | Simple, basic, easy-to-use and very clear what is going on. |
| Point system working how it should? | I believe the points tallied up perfectly. No errors that I could see there. |
| Response of controls? | Controls were quick and easy to understand. Didn’t freeze at all. |
| Do all the buttons work? | Yeah, they all worked fine. |
| Text easy to read? | All legible, good size, easy to read. |
| What improvements can be made? | * Nice looking user interface. * Graphical settings to work with different computer systems, in case it gets demanding (very unlikely). * Addition of sound effects or music to pull users in. * Possible animations to make it seem more fluid. Like when the Letters are added to the player’s hands after each turn. |
| **Performance** | |
| Did the game perform as you expected it to? | Played just like regular scrabble with a couple twists. Enjoyed it very much. |
| Are there any errors within the game that you noticed? | No… no errors came up when playing the game. |
| Does the game load quickly? | Yeah very quick, no issues with lag whatsoever. |

### Feedback Form 3:

|  |  |
| --- | --- |
| **Feedback Form**  Name: George Owens | |
| **User Requirements** | |
| Anything you liked/disliked about the game? | Had a few good games, were fun. |
| Did it work as well as other Scrabble games? | Played just like Scrabble should. Couldn’t see any major differences from the board game. First time I’ve played a digital version. |
| Did you win at all? | \*haha\*. Yes, in fact I did. Lost my second game though, blame my hand. |
| **Usability** | |
| Easy to use user interface? | Very clean and easy to use, however looks a little too rough in places. Could do with some flashier UI. Would have liked it so that the screen could be viewed at all sizes, little annoying that you had to play in maximized windowed mode. |
| What do you mean by ‘flashier UI’? | Just a stylised user interface, to make It stand out. |
| Response of controls? | Selecting letters was a little annoying, as had to sometimes click it twice. |
| Do all the buttons work? | Yes I believe that all the buttons did work. |
| Text easy to read? | Yeah the text was fine, didn’t really notice it. |
| What improvements can be made? | * Stylised user interface, so doesn’t look so basic. * More screen options, other than windowed maximised. * Possible sounds in the game or music. |
| **Performance** | |
| Did the game perform as you expected it to? | Yes it worked perfectly fine, just what I envisioned it to be. |
| Are there any errors within the game that you noticed? | No I didn’t notice any errors whatsoever. |
| Does the game load quickly? | The game is almost instant on loading up, and is perfectly fine after that. |

Evaluation

Referring to the user feedback received and the number of objectives that have been completed I can evaluate my program. In the analysis (page 4), I listed the objectives at the core of the program and objectives, if there was time, that would provide more sustenance to the program. Below I will list the objectives I managed to complete:

Objectives:

Core:

1. A ‘Game-Menu’ where the user can start a game and modify the settings themselves. For example, the board can be changed as well as the score and chance of letters. ✓
2. The hand is dealt randomly from the ‘bag’, each player receiving seven letters. The player who goes first is then decided. ✓
3. Each player then takes their turn. The letters played are combined, validated and the scores are tallied up. ✓
4. A ‘hint’ system may be implemented at a later stage which will give the user recommended words. ✓🗶
5. The first player to either ‘forfeit’ or use up all their tiles (in hand and the ‘bag’) ends the game. Final scoring takes place, congratulating the winner. ✓
6. Scores are then saved to a database including a few other statistics, such as: ‘Longest Word:‘ etc. Will be viewable by the user on the menu. ✓🗶
7. A save system will be implemented so the players can carry on from where they left off. ✓

### Extras:

* Introduction of networking so players on separate systems can play with one another through a server. 🗶
* A range of ‘game-modes’ that the user can pick (before the game) in the settings menu. These are listed below in more detail. 🗶

From ticks and crosses above, signifying implemented and unimplemented respectively, I did not complete any of the ‘extra’ tasks. The tasks themselves were designed so that if there was plenty of time left there would be a chance to try and attempt this. Sadly, there was no time to implement this into my program or plan the steps needed to implement.

The introduction of a networking system so two clients could play on separate networks or the same network through a LAN was not worked upon greatly as I hadn’t researched the objective too greatly. The only networking that I had carried out before was within the ‘*Unity*’ engine using a networking plugin called *‘Photon*’. Photon made easy work of the networking, connecting the assets from my game to their cloud networking service with only a few clicks and drags. The ability to connect with my friends, show them my game and play it with them was an amazing feeling. I believe that still resonates with me today, hence the idea of implementing it within my Scrabble program.

Another perk of having the program work on separate systems is so that no cheating could go on. An example of cheating was the fact that players could in theory see the other’s letters. To help counter this I implemented a five-second transition timer, this was later increased to seven-seconds as five was not long enough. The transition meant that when a player finished his turn and clicked the ‘End turn’ button the game would be paused for seconds hiding that player’s letters. The seven second period providing the other enough time to switch seats without the letters being spied upon. I believe this worked as a good system to prevent any cheating without the implementation of a networking system.

The other feature I would’ve much rather implemented was the addition of ‘game modes.’ These were listed in the Analysis (pages 8 and 9). Again, these were listed under the ‘extra’ objectives heading. I have never personally played any of these variations of the game of scrabble but much like the networking system, would add more content to the program and more importantly, more for the user to do.

The user feedback could’ve been much greater and helpful if both had been implemented. Some users presenting similar ideas to these, it being interesting to hear these views and if I were to implement these at a later stage, how I would go about it.

Within the core objectives, some have both a tick and a cross, each for different reasons. Objective 4 was the implementation of a ‘**hint** system’. Each player would have the ability to click the ‘hint button’ that would tell the user where, what or if any letters they could place to create a word if they were stuck (from those within their hand). My hint system works to the extent that it finds words with the user’s hand but only exclusively to that hand, it will not compare it to that of the board. The hint system itself is rather stunted due to this, but as it was an objective that ‘may be implemented’ further down production I believe it still provides the user with a helpful guide.

The second tick-cross combination is objective 6. Objective 6 was the addition of a database system. The database as mentioned in the Design section (pages 24-27) was all planned out but did not see it as a necessity to the program. The statistics could be stored locally but I intended to use a cloud server with a simple login system to access my details. It meant that if I were to use a foreign machine, I could still get a hold of and update my scores and if I were to introduce a networking/multiplayer system so users from a variety of places could also update their results and share to their friends. If I were to implement this at a later stage, I would refer to the plan I have created on pages 24-27 as they set a great foundation for the database. Most likely using the free *webhost* site to create this.

I believe the objectives set for myself were too demanding to implement in the space of time that I had. If I were to try and implement a network to my scrabble program, I would have to research into it beforehand. Limitations of the college system itself meant that connecting to a third-party network, even an online database without an admin account, made it near to impossible to develop in college time and project lectures. If I were to do the project differently I would thoroughly research and plan my objectives, make sure that they were logistically possible, evaluate the time I have to implement these and whether it is enough.

In conclusion, I have managed to meet 6/7 of my core objectives, one of which isn’t exactly what was planned but works to a great enough degree to achieve its goal – to benefit/help the user. Neither of the ‘extra’ objectives were completed, the range of ‘game modes’ would’ve been the easier of the two to implement as each game could be split apart through different functions and subroutines. The time it would take to implement each of these would be too great looking back in retrospect so in terms of time taken to implement, the networking would be much easier if done the right amount of research. Implement a network into a game after it has been programmed is a lot more complicated than tailoring the code to suit a networked program, if I were to carry this program out again I would in fact network it from the get-go.

In terms of consulting my objectives, I believe I have done a great job. Completing nearly all core objectives but missing out the extra’s. The objectives themselves helped keep me on track but where the objectives provided a broad sense of the steps I had to take; the user feedback helped me view the program from a ‘third-person’. They could see problems that I could not and whenever the program seemed to branch off, they would put me back on track.

## User Feedback:

The user feedback found both in the *Analysis* (pages 10-12) and in the *Testing* (pages 50-52) were a great help in the development of my project and gave me several useful ideas I later implemented. It also greatly helped in debugging the program, many errors that I didn’t know existed with my playthroughs and testing.

The feedback forms were split up into three sections:

* User requirements:
  + Possible additions to improve the game itself.
  + Anything they disliked about the game.
  + How it compared to other scrabble games and whether they enjoyed it the same way.
* Usability:
  + Methods of ‘stylising’ the program, in terms of its GUI (Graphical User Interface); look more professional.
  + Whether it all worked like it should.
  + More user customizability. Greater number of options in the ‘*settings*’ menu.
* Performance:
  + Any errors that they may have come across whilst playing (very important for me).
  + How the game performed. E.g. if it crashed at all.
  + Load times or performance drops.

The three recipients I talked to were friends and close/immediate family members. Wish I had asked a broader range of people to try out my program, but just found it a whole lot easier to ask people I knew. The problem with this approach is that they may not speak their mind, even so, I was grateful for the criticism I did receive. To show the general consensus of the feedback I’m going to break it down into the three subsections listed above.

User Requirements:

This was the most basic of the three sections only touching on whether they liked/disliked the program, if it played as well as other scrabble games they had played and if they’d won at all. Overall, they all enjoyed the game, all playing a multitude of games against friends.

Usability of the Program:

This section focused on how the program looked, what could be done to improve it and if the GUI (Graphical User Interface) e.g. Buttons all worked as they should. In summary, the users asked for an improved GUI, in terms of stylisation, thinking it all looked very “plain” and “basic”. In addition to this, the use of sound effects came up often to not only entertain the user but to make it feel more like a polished game. The final asks of the possible implementation of a graphical slider so that users (not that the game is demanding in terms of graphics in any effect) could adjust the overall graphics of the game for their PC. In every other respect, the game ran perfectly according to the users.

Performance:

The performance of the program checks with the user if any errors did come up whilst running the program, whether it ran as expected and loading times of the program. The program was perfect in this respect: no errors sprung up; it worked as they expected it to – if not better; loading times were pretty much instant, except when compiling for the first run.

In conclusion, much like the Objectives it almost ticked every box. The only issues being possible additions to the program, the actual functionality of the program was perfect and worked how I would’ve liked. The overall fidelity of the program with the objectives I set myself were coherent to say the least. Looking at the feedback I did receive, like the possible addition of animations, sound and improved graphical interface. With more time I would’ve like to have implemented these or even added them to my objectives list, however I cared more about the program at its core instead of how it looks. I hope to improve upon these in my own time, to create in my eyes, the perfect scrabble experience.