**Language Trainer**

**Andrea Briffa**

**Designing and Developing Object Oriented Computer Programs (DDOOCP)**

**STC Higher Education Malta**

**Table of Contents**

[**Introduction** 4](#_Toc8771142)

[**Task 3 – Class Diagram** 5](#_Toc8771143)

[**Introduction** 5](#_Toc8771144)

[**Class Diagram** 6](#_Toc8771145)

[**Class Diagram Explanation** 7](#_Toc8771146)

[**Use Case Diagram** 10](#_Toc8771147)

[**Assumptions** 11](#_Toc8771148)

[**Required Features** 12](#_Toc8771149)

[**Additional Features** 13](#_Toc8771150)

[**System Limitations** 16](#_Toc8771151)

[**Possible Improvements** 17](#_Toc8771152)

[**Conclusion** 18](#_Toc8771153)

[**Task 1** 19](#_Toc8771154)

[**Player Class** 19](#_Toc8771155)

[**Guest Class** 21](#_Toc8771156)

[**User Class** 22](#_Toc8771157)

[**Language Class** 25](#_Toc8771158)

[**Library Class** 27](#_Toc8771159)

[**Welcome\_Form Class** 43](#_Toc8771160)

[**Game Class** 58](#_Toc8771161)

[**Game\_Settings\_Form Class** 81](#_Toc8771162)

[**Register\_Form Class** 86](#_Toc8771163)

[**Statistics\_Form Class** 90](#_Toc8771164)

[**Log\_In Class** 110](#_Toc8771165)

[**Modify\_Word\_Sets Class** 119](#_Toc8771166)

[**Task 2** 133](#_Toc8771167)

[**White Box Testing** 133](#_Toc8771168)

[**Black Box Testing** 230](#_Toc8771169)

[**Integration Testing** 286](#_Toc8771170)

[**Test Summary** 311](#_Toc8771171)

[**Conclusion** 312](#_Toc8771172)

# **Introduction**

Memory Games aids the development of individuals with reference to their ability to think rationally and improving their memory. Nowadays one must be able to translate between Languages in order to interact with new people.

“**Language Trainer**” is a game developed with the main scope to overcome these issues.

A detailed class diagram of the system illustrates the properties and functions which make up the system, followed by an explanation of the Diagram.

Testing is an important phase in development, as it tests the rigidness of the system making it robust and ready for deployment.

All of the different classes and functions which make up the system are documented thoroughly, for a complete understanding of the system, and to aid future development to better understand the system.

# **Task 3 – Class Diagram**

## **Introduction**

To further illustrate the structure of the system, a class diagram is created which follows the standards of UML - Unified Modelling Language. A Class Diagram displays the system’s classes, fields, methods/functions and also the relationship between classes and objects.

A Use Case Diagram is also constructed which explains the flow of the User Interface for the User.

The List of Assumptions for the system and justifications are identified, followed by the List of additional features of the system. The Limitations of the system are also identified, together with improvements on how to overcome the limitations and improve the system.

## **Class Diagram**

Figure 3.1 illustrated the System’s Class Diagram

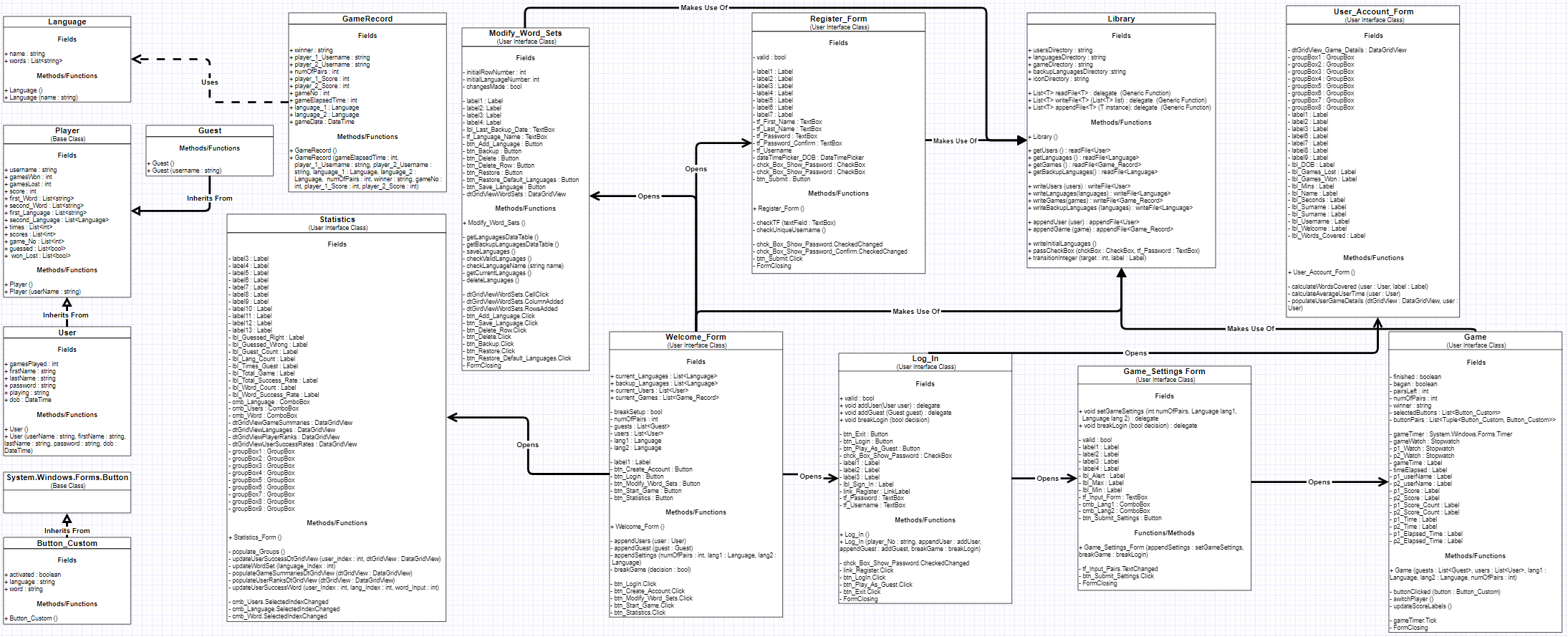


Figure 3.1

## **Class Diagram Explanation**

The Welcome\_Form class serves as the Main class for the system. At initialization the class loads the User, Languages and Games data from the Files, by calling the pertinent methods from the Library class. This data is kept in the form of **Lists**, and is **public** and **static**, which means that is can be used by all other classes without creating an instance of **Welcome\_Form**. Data files are accessed only when the Files need to be updated, such as when saving a new registered User to the system.

When starting the Game setup. The Players are required to Log In as either Users or Guests. This requires an instance of the Log In class to be created, and passing off the arguments to the class constructor. The arguments consist of callback methods which allow the User/Guest logged in to be appended to the List of players. Another callback method passed is **breakGame** which allows the players to cancel the Game Setup instead of continuing Login.

Following the Log In, an instance of **Game\_Settings** is created and given arguments from the Welcome\_Form. The arguments given are callback methods which initialize the Game properties such as the Languages and number of Button pairs, or to cancel the Game Setup process.

If the Game Setup proceeds, an instance of **Game** is created from the Welcome\_Form. The Game class is given the selected Languages for the Game, the Lists of playing Guests and Users, and the number of Button Pairs. Once finished, a Game creates a new instance of Game\_Record and appends it to the existing List of Game Records by calling the **appendGame** method from the **Library** class.

If from the Welcome Form the User decides to access his/her account, this can be done by creating an instance of Log\_In without passing any arguments to the class constructor. If a User logs in, the user’s pertinent **User** instance is passed as an argument to the **User\_Account\_Form** which will display the User’s personal and game details.

If the User decides to Modify the existing Word Sets, an instance of **Modify\_Word\_Sets** is created, which uses the current List of Languages to allow the User to modify it. The user may also decide to load the default Languages or Backup Languages to override the current List of Languages. To do this, methods and functions from the Library class are called to update the Languages Files.

A player may register for a User account through the Welcome Form, which will create a new instance of Register\_Form, allowing the player to input his/her details. When registered successfully, the Register class will call methods and functions from the Library class to access the Users file. Instances of **Register\_Form** may be also created via the Log In class when players are registering as User/Guests to play, or when users are logging into their Account.

From the Welcome Form, an instance of the Statistics class may be created, which displays the past Games’ and User statistics, by using the current List of Users, Languages and Games which are accessible from the **Welcome\_Form**.

A brief description of the Complex Data Types:

**Language**

The Language class is responsible for holding the Language’s name, and a List of its’ pertinent words.

Language objects are used throughout the Program to pass Language instances, such as when crating an instance of **Game**, the selected Languages have to be passed through the constructor.

**Button\_Custom**

The Button\_Custom class inherits from the System.Windows.Forms.Button class, and has additional fields such as **word** and **Language**. This allows the buttons to be displayed in the Game Form, and pass their contributed words to a function (**buttonClicked**) when they are clicked.

**Player**

The Player class stores all of the Details that are pertinent to a Player with reference to Game Details. The Player class serves as a Base class for the **Guest** and **User** classes.

**Guest**

The Guest class inherits from the **Player** class, and objects are derived from this class when Players log in as Guests to play a Game.

**User**

The User class inherits from the **Player** class, but has additional fields that are pertinent to a registered User, such as personal details including name and surname. User instances are saved to a File constantly, which drives home the fact that a User Account is never deleted, allowing a User to log in multiple times as a Player, and Log into his/her account to view his/her personal and Game Details.

**Game\_Record**

The Game\_Record class is responsible for holding the Details of a Game after it has finished. The secondary constructor allows the fields of the class to be initialized directly from the arguments. The main use for this is when a Game has finished, all of the details of that game are passed onto the class constructor when creating an object of Game\_Record.

Objects of this class are used to be serialized to the Game Records File, and displayed for Game Statistics.

## **Use Case Diagram**

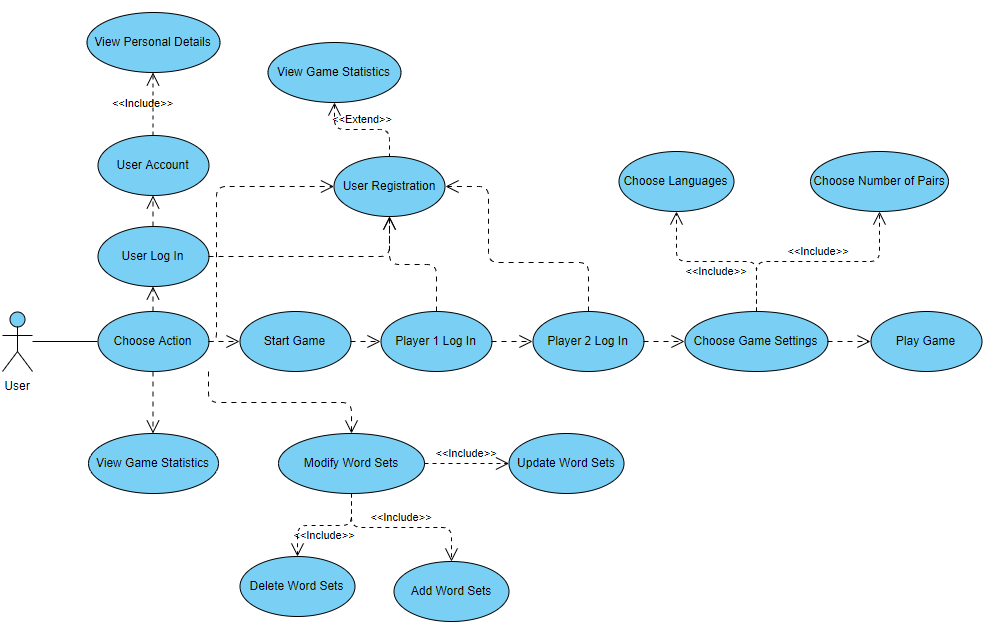


Figure 3.2

The Use Class Diagram (fig 3.2) illustrates the User’s options throughout the Application; how the User uses the application.

The User’s main options are to:

* Log into their Personal Account
* Start the Game Setup Process
* View the Game Statistics
* Modify the Word Sets
* Create a User Account

## **Assumptions**

* Each User must have a unique username, to uniquely identify each other.
* Every player has the ability to view Game and User game statistics, and Modify Word Sets.
* A User must provide his username and password to log into his account and view the personal and Game details.
* Each language is to have the same amount of pertinent words, relating to words from other languages. Each language may have two up to 100 pertinent words
* A player must login as a User to save the Game details or register as a guest to play a game
* The game languages must be different, with the scope of learning and memorizing new and different words
* Guests are given priority to start the first game turn over Users, since they are Guests
* If the Players end the game with the same score, the winner is chosen by the shortest amount of time spent over turns (milliseconds)
* When a Player guesses a pair of buttons, the button’s assigned words are displayed for a short amount of time, then hid again
* If a player guesses a pair of words, s/he is given another turn to guess as a reward for correctly guessing a pair of buttons.
* A User may not Login twice consecutively as both player 1 and player 2. Player 1 and 2 have to be different Users or Guests.

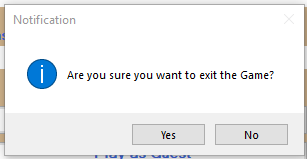
## **Required Features**

* The game accounts for two players
* The players may choose their preferred language (word set).
* The game is able to create a game with 52 (26 pairs) buttons. Each pair has two words, having same meaning but different languages
* The program keeps track of whose turn it is. If a player guesses two buttons that match, he/she is given another turn.
* The program keeps track of the players’ score and buttons’ matches.
* The game scores, winner and languages used are saved to a file system.
* The user’s personal and game details, and success rates with particular words are saved to a file system and displayed within the Application.
* The players have the ability to add, modify and delete word sets.
* The program takes care of storing and displaying word sets and their translations correctly.
* The program creates the game UI, including mapping the buttons and initializing their word values based upon the word sets.
* After a pair of buttons that are recognized as a pair are guessed, they are removed from the UI.

Note: By the term ‘Word Sets’, it is understood as the list of pairs of words and their translations for the Languages. Each Language has a Word Set.

## **Additional Features**

* A Player may register as a User, or play a Guest. The Guest details and information are deleted once the game ends, as they are anonymous. If a Player and a Guest take part in a Game, the Guest takes the first turn, as they are the ‘Guest’.
* Registered players are able to log into their account, where they can see their personal and game details.
* Additional User details are displayed within the User account portal, such as the Total words encountered, games lot and won, and every game’s detail the user took part in.
* Registered users have a unique username. Upon registration the username has to be unique and no prior User has the same.
* Upon registration, it is checked that the User’s passwords match, and 8 or more characters long. The Birth Date of the User is also checked that it is not the current day or a day within the future.
* Message Boxes are used throughout the application to verify Player decisions, such as when a Player is exiting the game setup. The program then executes instructions based upon the User’s decisions.



* A statistics form is created and accessible within the Main Menu form. The Statistics form features:
* The total Games
* The number of existing Languages
* The number of Words per Language
* The player ranks
* A list of existing languages
* The number of total guest entries (A player registered to play as a Guest)
* A user selection list, which allows to display game details of specific a specific User
* A list of each pair of words guessed by the selected user
* The amount of times the User guessed, guessed right, guessed wrong and the success rate of guessing particular word
* The total User success rate with guessing words
* A list of each Game and its’ details is also available inside the Statistics form
* A separate form is dedicated to Modify the Word Sets
* The player has the ability of adding a new language to the existing list of languages. The language’s name is checked that it is unique from other language names, and is not empty
* The Player can also backup a set of languages / word sets, and restore them (replace the existing word sets with the ones backed up)
* The Player has the ability to delete a Language
* If the Player selects a word within the Word Sets list / makes a change to the word sets list and attempts to close the Word Set Modification form, the program outputs a Message Box which reminds the Player if s/he wants to save the changes made to the Word Sets.
* Is the Player selects to save the changes made to the Word Sets, the program first checks that there are no empty entries, as that would leave certain word translations empty
* The number of languages cannot go to lower than 2, as well as the number of words per language
* Upon entering the Password when registering as a User, a checkbox is available if the User would like to view the entered password, since it initially hidden by replacing entered characters with ‘\*’
* Players have the ability to terminate the Game setup or stop the game. If so the game details will not be saved
* Players have the choice to choose between 2 up to the maximum amount of words per language as the number of pairs to be included in the game. The program then generates the UI accordingly
* Players have the choice to choose their preferred languages for the game from the list of existing languages. Players cannot choose the same language to be included in the game, as that would not make sense
* If a Player decides to exit the game of game setup, the program would skip the whole game process using a GoTo statement
* The backup languages are stored in a separate file
* The Players have the ability to restore the default word sets within the Modify word sets form
* The default word sets are stored in a separate file
* The game details and User details are stored individually in separate files
* The game is carried out on a separate Form, which displays the grid of buttons that each relate to a specific word
* The list of words being used in the game is randomized, so that the players do not always play with the same list of words and their translations
* The Game form displays the usernames of the players playing, the player’s scores and their respective elapsed time.
* The game form also displays the total elapsed time, and at the end of the game displays the winner, and prompts the players to return to the home screen
* If the game ends with both players having the same scores, the winner is determined by the shortest total time took in turns
* The size of the buttons, and number of buttons per row depend upon the total number of buttons to be displayed within the Game form. The smaller the number of buttons, the larger the button’s size and the smaller the number of buttons per row
* Players have the ability to play games, modify word sets, view statistics and log into their User accounts from the Main Form, which is always displayed after another form closes, such as the Game Form

## **System Limitations**

* The number of Words Per Language (Word Set Size) cannot go under two words, or higher than 100 words.
* The number of existing Languages cannot go underneath 2.
* Only 1 Backup is allowed for the Word Sets. If one wants to Back up the current List of Languages, the previous Backup must me overwritten.
* Only two (2) Languages with five (5) words each are available as Default Languages. The User does not have any flexibility when overriding the Current Languages with the Default Languages.
* Users do not have a Backup Password or any other measures they may use to Log Into their account in case they forget the Primary Password.
* The security of the system files, such as the Users file is vulnerable if they are acquired by unauthorised Users as they are not encrypted, and the Passwords are not Hashed thus can be seen clearly.
* Users have no option to Backup/Delete their Account.
* Users cannot modify/change their details such as their Username and Password.
* The Application does not adapt to different resolutions, leading to problems when being faced with small resolutions as the content does not fit properly.
* A Game can only consist of two (2) Languages at a time.
* The Game may only be played on the same Machine. Players cannot play from different devices against each other.
* Two players can only play at a time.
* The Guest Game details are not stored after each Game. Only User details are updated and stored.

## **Possible Improvements**

* Allowing Users to change their Account Details such as their Username and Password.
* Encrypting the System Files with a Password, as to avoid Unauthorised Users.
* Hashing the Users’ Passwords, as to make them unreadable in case the System is breached and the files are accessed by unauthorised Users.
* Increasing the size of the Default Languages’ List including the Number of Languages and Words per Language.
* Allowing the Users to specify which Languages, and the number of words per Language they wish to retrieve when replacing the Current Languages with the Default Languages from the Modify Word Set Form.
* Allowing unlimited number of words per Language.
* Allowing multiple backups of Word Sets.
* Adding more details for each User, which can serve as alternatives when the User forgets his/her password.
* Adapting the size of the Windows Forms to different resolution.
* Allow filtering of Game Statistics i.e. selecting Games that were played by Guests only.
* Allowing more than two (2) players to play a Game consecutively.

## **Conclusion**

Together with the requirements, a variety of extra features was added to the system for better User satisfaction and stretch the limits of the system. Despite this, the system itself still has its’ own limits, which can be overcome by implementing some of the suggested improvements.

Assumptions were took prior to the Development of the system, such as having a Unique username for each User and not allowing the same Language to be used for a Game.

The Class Diagram illustrates the layout of the System, together with the Use Case diagram which provides a demonstration of the User’s navigation throughout the Window Forms.

# **Task 1**

## **Player Class**

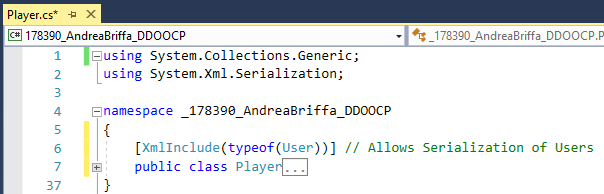


Figure 1.1

Player is a public class which serves as a base class for the User and Guest Classes. The “**XmlInclude**” statement allows the **User** class to be serialized whilst having **Player** as a Base Class

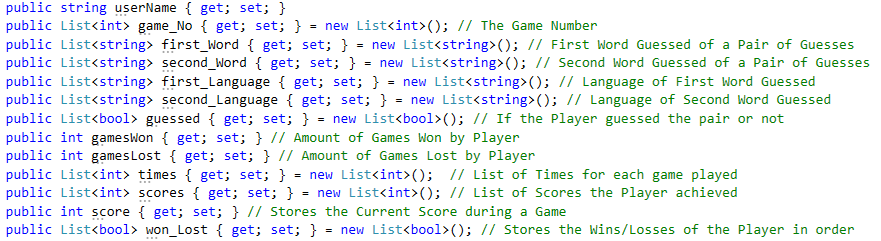


Figure 1.2

In figure 1.2, all of the fields which make up the Player class are displayed, each having a pertinent comment which states their scope.

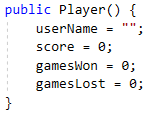


Figure 1.3

An empty constructor is available for the Player class. This caters for creating Lists with Player instances, which initially does not contain any Player information.

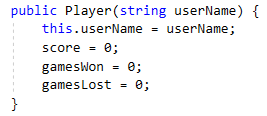


Figure 1.4

A secondary constructor is available for the Player class, which is used for initializing Players and directly assigning them a Username.

## **Guest Class**

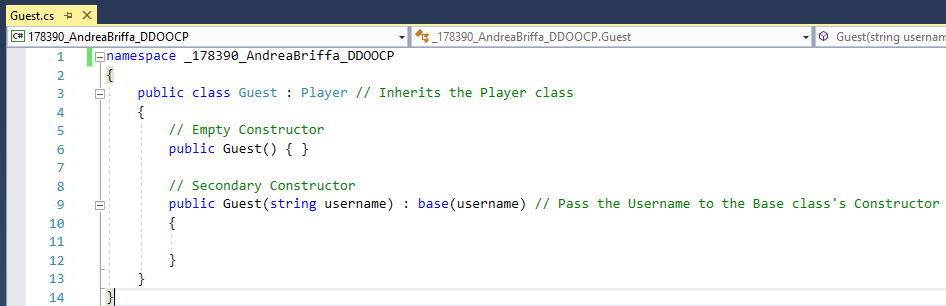


Figure 2.1

The **Guest** Class inherits all of the attributes from the **Player** class. Two constructors are available for the **Guest** class. The Primary constructor allows instances of Guests to be initialized without assigning any initial attributes. The Secondary constructor allows Guests to be initialized with a given username, and passes the Username to the base class (**Player**) using a base constructor.

Note: After a Game is finished, the Guest’s details and his/her Game details are not saved. In order for his/her details to be kept, a Player must register as a User.

## **User Class**

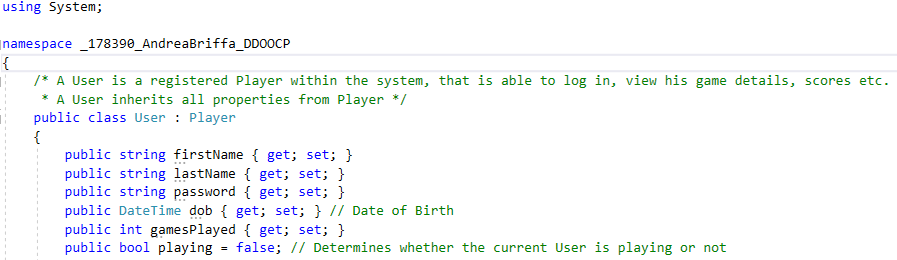


Figure 3.1

The **User** class serves as a class for Players who have registered as Users, meaning they have an account. Users can login for any game any have their details serialized to a File, which they can view by logging into their account.

The **User** class inherits all of the attributes from the base class (**Player**).

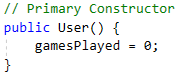


Figure 3.2

The primary constructor for the **User** class serves as an empty constructor, for creating instances of Users without assigning any attributes.

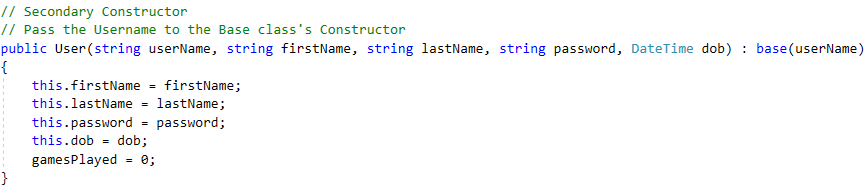


Figure 3.3

The Secondary constructor for the **User** class allows User instances to be initialized directly with the User details, whilst also passing the User name to the Base class (**Player**).

**GameRecord Class**

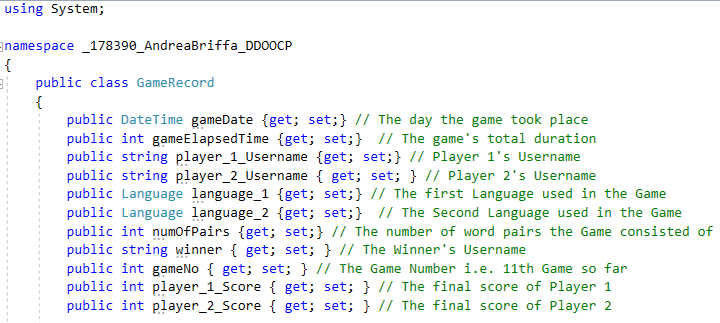


Figure 4.1

The **GameRecord** class keeps track of finished Game details. Each time a Game is finished, a new instance of **GameRecord** is created, and initialized with all of the Game details using the secondary constructor.



Figure 4.2

The primary Constructor allows instances of **GameRecord** to be created without passing any arguments. This is useful for creating Lists of Game Records which initially do not contain any attributes.

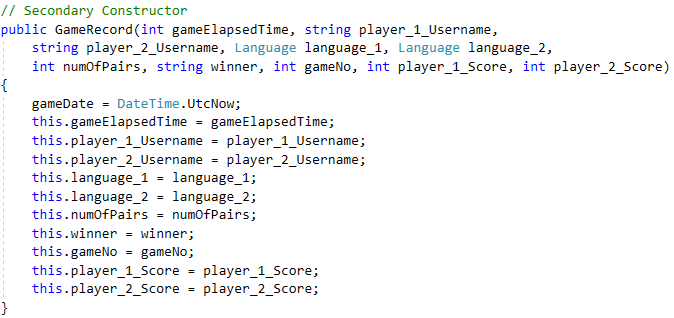


Figure 4.3

The secondary constructor of the Class allows instances of **GameRecord** to be initialized with all the details upon initialization. This comes in use when serializing a new instance of **GameRecord** (Passing in all of the arguments i.e. Game Details) to a File of Game Records, after a Game has finished.

## **Language Class**

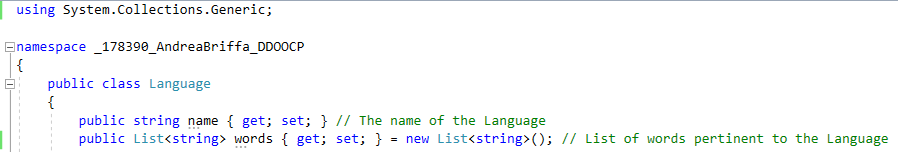


Figure 5.1

The **Language** class serves as a Template for Languages. Each Language has a name, and a Word Set, which is made up of a List of words pertinent to that Language.



Figure 5.2

The primary constructor allows Language instances to be created without passing in any initial details about the Language. This comes into use when creating a List of Languages which initially does not contain any information.

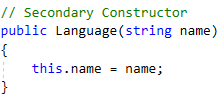


Figure 5.3

The secondary Constructor allows a **Language** instance to be initialized with its’ name initially. This is useful when creating new Languages which all require a Unique name.

Note: The **Language** class is also used to create **Language** instances which serve as **Backup** Languages.

**Program Class**

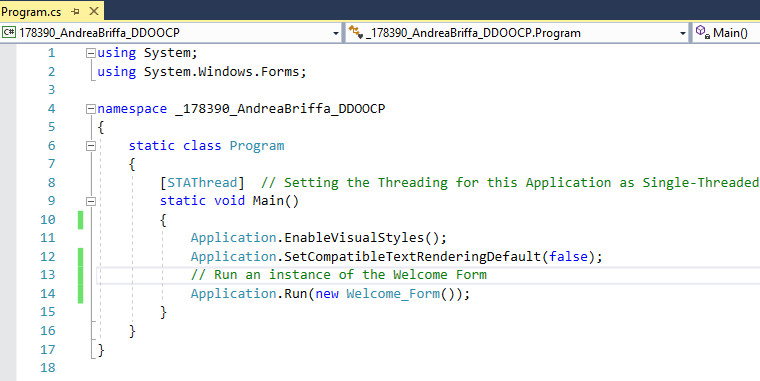


Figure 6.1

The **Program** Class serves as a “Kickstarter” for the entire application. When the application is run, it executes what is within the **Program** Class. Once executed, the **Main** method inside the **Program** class will run a new instance of the **Welcome\_Form** class, which serves as the “*Main Menu*” for the whole application.

## **Library Class**

Brief Description: The Library class serves as a repository which offers functions and methods that are used by other classes throughout the program.

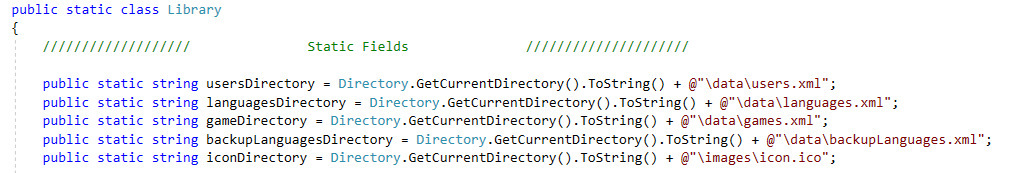


Figure 7.1

The static fields are fixed values which are used by other functions throughout the program. The benefit of using the same field to store a particular value is that if in this scenario, a file’s directory changes, the field’s value can be changed, and it will apply to all of those methods/functions which make use of it.

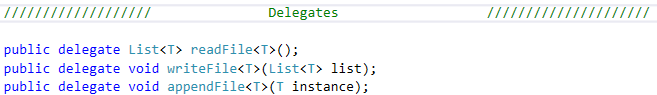


Figure 7.2

The delegates are used as templates, which from them static functions are created. A Generic aspect is used, for the reason that **readFile** may apply to reading a list of Users, Game Records or Languages.

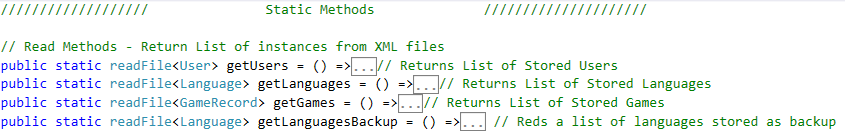


Figure 7.3

Four (4) static functions were derived from the **readFile** delegate, which has the scope of reading a List of a specific data type from a .xml File, and returning it.

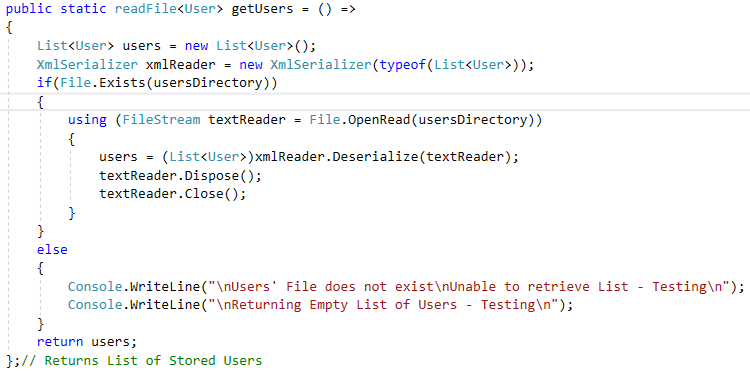


Figure 7.4

To retrieve a List of a specific data type from an XML file, the same process if used throughout the four **readFile** functions (fig 7.4, fig 7.7, fig 7.10, fig 7.13).

In figure 7.4, a List of Users is created, which is going to be returned at the end of the function. An instance of **XmlSerializer** of type “List<User>” is created for DeSerializing the List of Objects read from the **Xml** File to the **List** of Users.

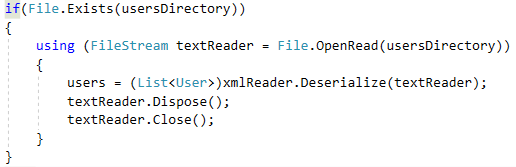


Figure 7.5

If the File within the specified directory exists, a **FileStream** is created with the scope of retrieving the content within that file. The XmlSerializer is used to **DeSerialize** the **FileStream** containing the File data into a **List** of Users. The FileStream is then disposed of and closed, releasing all of the resources it used, such as the File within the specified directory.

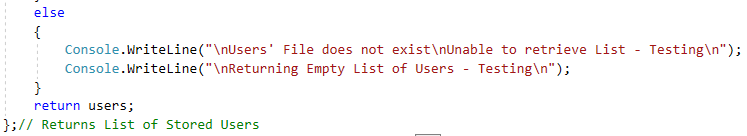


Figure 7.6

If the File does not exist, a message is written to the console, and an empty List of Users is returned.

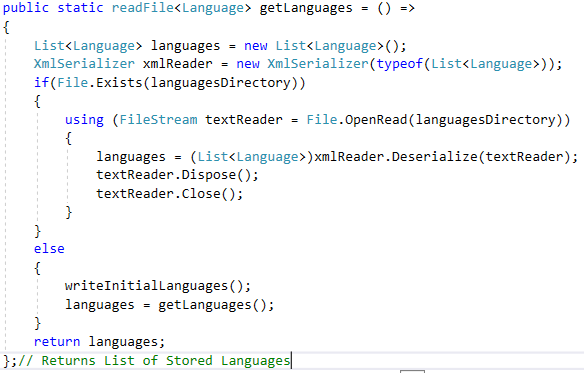


Figure 7.7

The function in figure 7.7 is used to return a List of Languages from the Languages File. A List of Languages is created which will store the List of Languages returned from the file, and then be returned at the End of the Function. An instance of **XmlSerializer** of type “List<Language>” is created for DeSerializing the List of Objects read from the **Xml** File to the **List** of Languages.

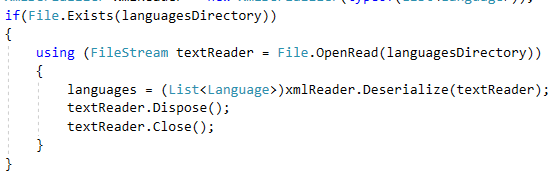


Figure 7.8

If the Languages File exists within the specified Directory, a **FileStream** is used to retrieve the Data from the Languages File, which is then DeSerialized and assigned to the Languages List.

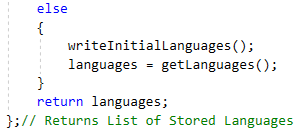


Figure 7.9

If the Languages File does not exist, meaning there are no saved Languages or the file was Unreachable, a new File is created and the initial (default) List of Languages are written to the file.

The function then retrieves contents of the Languages file by recursively calling itself, which in turn retrieves the List of Languages from the Languages File, ant then returns the List of Languages.

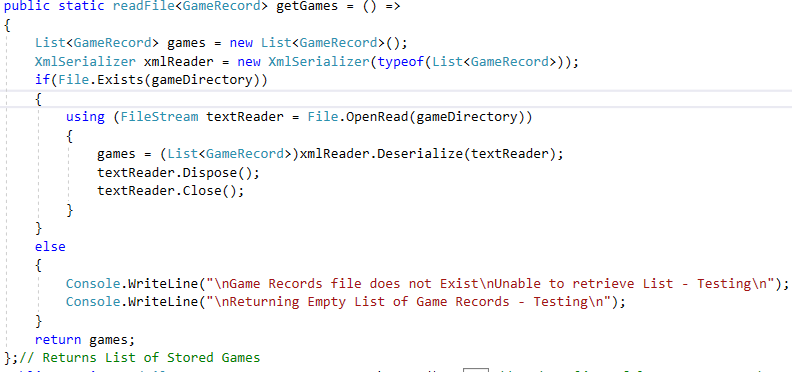


Figure 7.10

The function in Figure 7.10 reads the list of past **Game Records** from the **Games File** within the specified Directory. A List of GameRecords is created which will store the List of **Game Records** returned from the file, and then be returned at the End of the Function. An instance of **XmlSerializer** of type “List<Game>” is created for DeSerializing the List of Objects read from the **Xml** File to the **List** of Games.

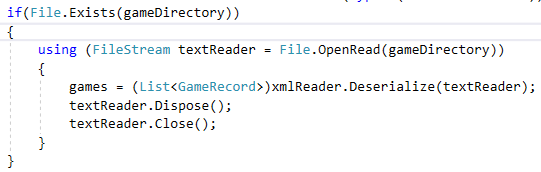


Figure 7.11

If the **Game Records** File exists within the specified Directory, a **FileStream** is used to retrieve the Data from the **Game Records** File, which is then DeSerialized and assigned to the **GameRecords** List.

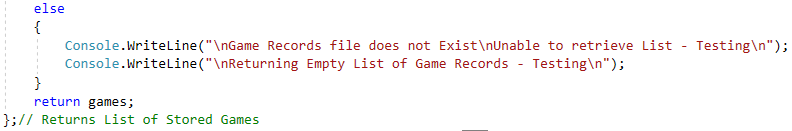


Figure 7.12

If the File does not exist within the specified Directory, a message is written to the console, and an empty List of Game Records is returned.

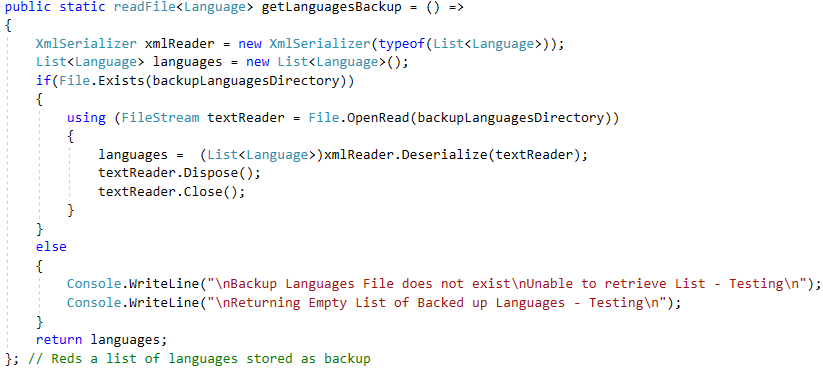


Figure 7.13

The function in Figure 7.13 reads the list of **Backup Languages** from the **Backup Languages File** within the specified Directory. A List of Languages is created which will store the List of **Backup Languages** returned from the file, and then be returned at the End of the Function. An instance of **XmlSerializer** of type “List<Language>” is created for DeSerializing the List of Objects read from the **Xml** File to the **List** of Languages.

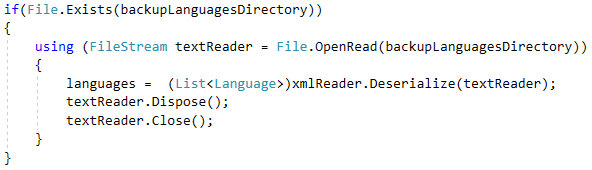


Figure 7.14

If the **Backup Languages** File exists within the specified Directory, a **FileStream** is used to retrieve the Data from the **Backup Languages** File, which is then DeSerialized and assigned to the **Languages** List.

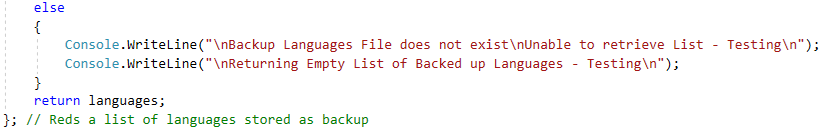


Figure 7.15

If the File does not exist within the specified Directory, a message is written to the console, and an empty List of Languages is returned.

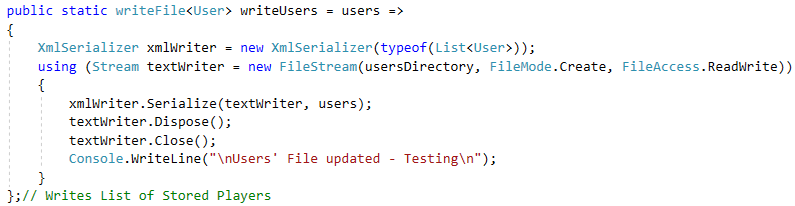


Figure 7.16

To write a List of a specific data type to an **XML** file, the same process if used throughout the four **writeFile** functions (fig 7.16, fig 7.17, fig 7.18, fig 7.19).

In figure 7.16 an **XMLSerializer** instance is created to serialize a List of Users to the Users File within the specified Directory.

A **Stream** is created to hold the List of Users passed as an argument to be serialized by the **XMLSerializer** to the **Users** File within the specified Directory.

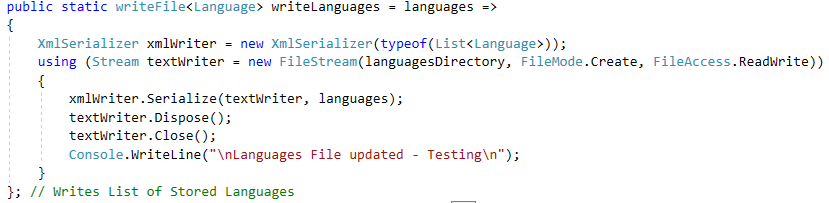


Figure 7.17

In figure 7.17 an **XMLSerializer** instance is created to serialize a List of Languages to the Languages File within the specified Directory.

A **Stream** is created to hold the List of Languages passed as an argument to be serialized by the **XMLSerializer** to the **Languages** File within the specified Directory.

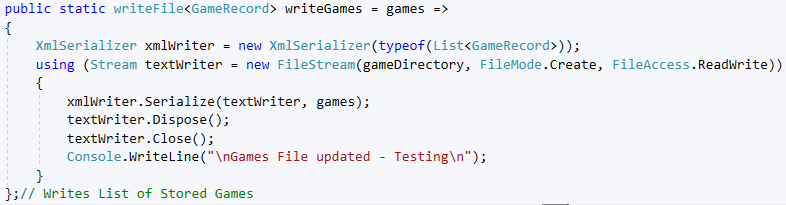


Figure 7.18

In figure 7.18 an **XMLSerializer** instance is created to serialize a List of Game Records to the Game Records File within the specified Directory.

A **Stream** is created to hold the List of Game Records passed as an argument to be serialized by the **XMLSerializer** to the **Game Records** File within the specified Directory.

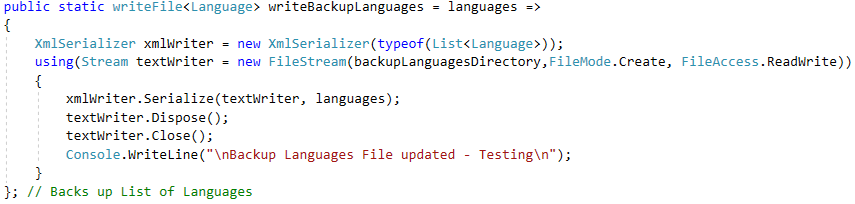


Figure 7.19

In figure 7.19 an **XMLSerializer** instance is created to serialize a List of Languages to the Backup Languages File within the specified Directory.

A **Stream** is created to hold the List of Languages passed as an argument to be serialized by the **XMLSerializer** to the **Backup Languages** File within the specified Directory.

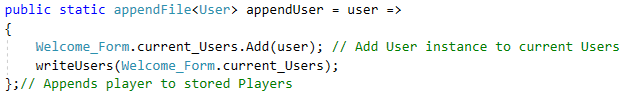


Figure 7.20

The scope of the functions in figure 7.20 and 7.21 are to add an instance of a Class (passed as an argument) to an existing List of the pertinent Class. The functions then write the updated List to their pertinent File.

In figure 7.20, a User instance is added to the existing List of Users, which is then saved to the **Users** File within the specified Directory.

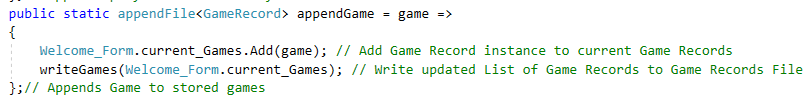


Figure 7.21

In figure 7.21, a Game Record instance is added to the existing List of Game Records, which is then saved to the **Game Records** File within the specified Directory.

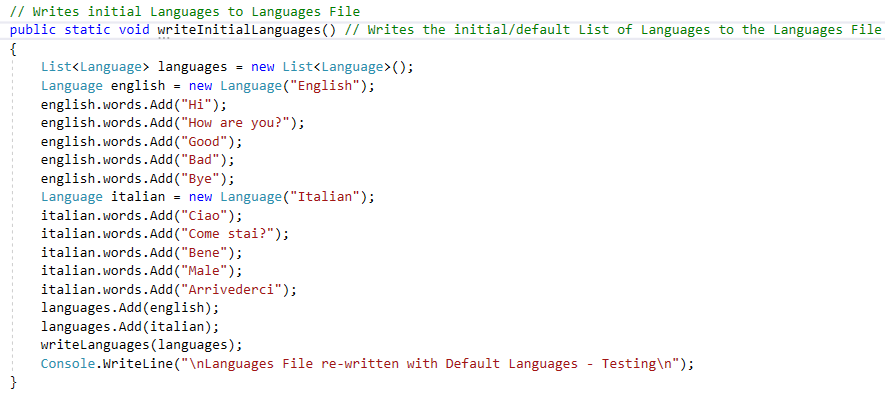


Figure 7.22

The function in figure 7.22 writes the initial Languages to the Languages File.



Figure 7.23

A List of Languages is created, and a set of pertinent set of words was added to each Language which were also given different names.



Figure 7.24

The List of default Languages is then written to the Languages File, overwriting it.

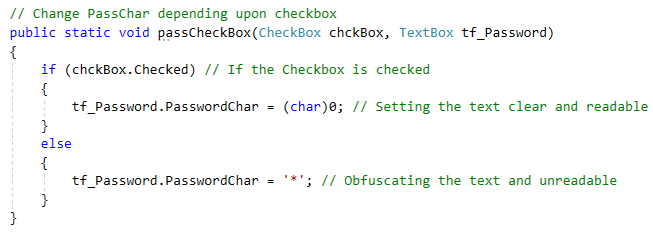


Figure 7.25

In function 7.25, two arguments are passed through the parameters, which are the **Checkbox** that is being checked/unchecked, and the **TextBox** that is going to be affected by the state of the **Checkbox**. The textbox passed as an argument serves as a Password Textbox.

If the Checkbox is checked, the text inside the TextBox is shown clearly by not obfuscating the Text inside the TextBox making it readable.

If the Checkbox is not checked, the text inside the TextBox is obfuscated, making the password unclear/unreadable.

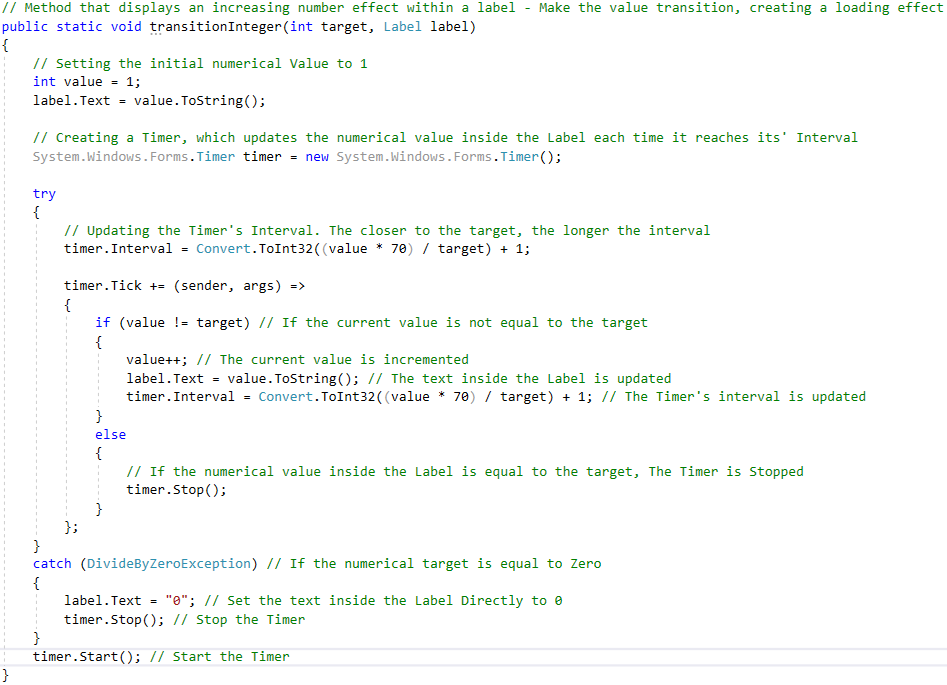


Figure 7.26

The function in figure 7.76, receives a numerical value, which is the value to be placed inside the Label. The scope of this function is to transition the value inside the Label from 0 to the numerical value.

The function operates using a Timer, using the Interval of the Timer to Update the Text inside the Label. The closer the text inside the Label gets to the numerical value target, the longer the Timer’s interval, slowing the incrementation of the Value inside the Label, creating a transition. The **value** variable stores the current value which is being transitioned.

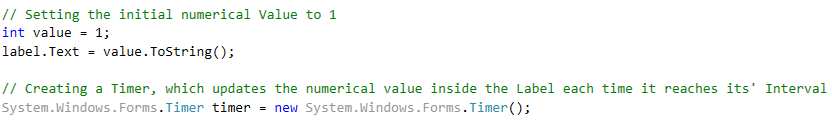


Figure 7.27

The initial text inside the Label was set to 1, before transitioning the value inside the Label to the target Numerical Value. The Timer is also created.

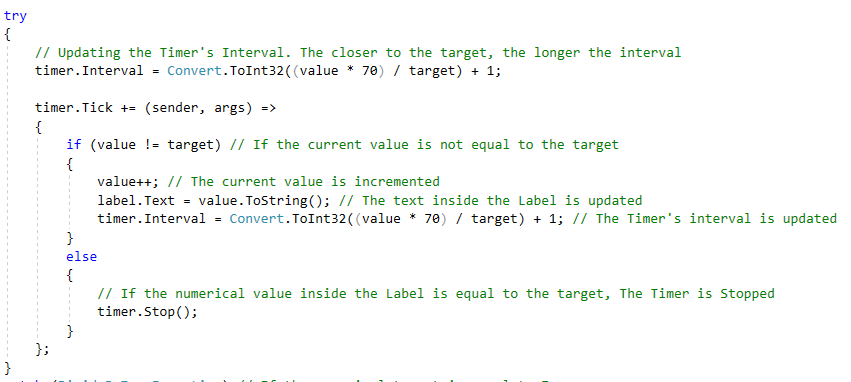


Figure 7.28

The interval of the Timer is set, and the closer the **value** gets to the target numerical value, the longer the interval.

Each Time the Timer ticks, the following function (fig 7.29) gets executed.

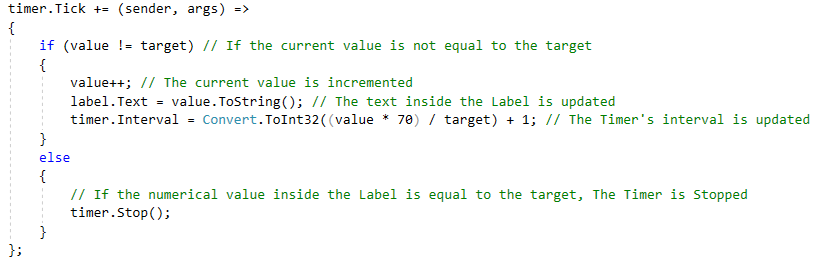


Figure 7.29

If the current Value is equal to the target Numerical value, the Timer is stopped as there is no need to update the text inside the Label.

If the current Value is not equal to the target Numerical Value, the value inside the Label is incremented. The Timer’s interval is also updated to change the transition speed of the numerical value inside the Label.

When updating the Timer’s interval, an **exception** may be thrown when updating the Timer’s interval if the target numerical value is **0**.

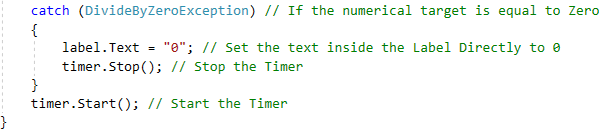


Figure 7.30

If an Exception is thrown when the Timer’s interval is updated, the text inside the Label is set directly to **0**, as the exception was thrown because the target Numerical value was **0**. The Timer is also stopped as there is no need to update the Text inside the Label.

After the Timer’s interval is set, the Timer is started, starting the Numerical Value Transition inside the Label.

## **Welcome\_Form Class**

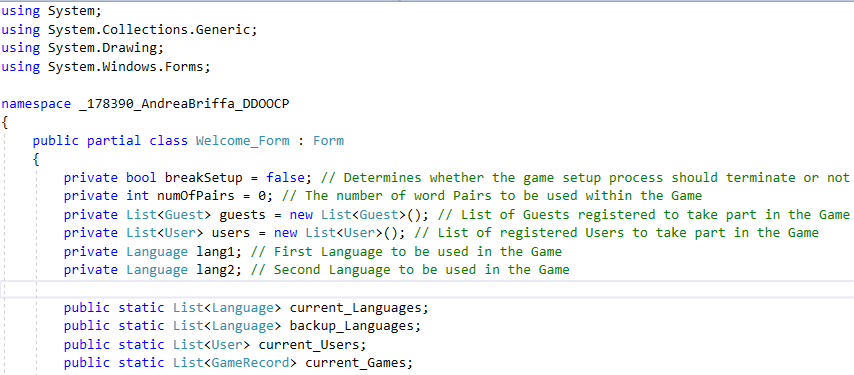


Figure 8.1

The Welcome Form class serves as a “**Main Menu**” for the whole application. Within the Welcome Form, the current Languages, Users, Game Records and Backup Languages are kept, and are made static, thus made constant and available throughout all of the classes.

The breakSetup field is used to determine whether the Game setup should continue or not. The numOfPairs field stores the number of pairs to be carried out in the game.

The List of Guests and Users stores the Users and Guests that have registered as Players to take part in the Game.

The two language instances store the two Languages that are the selected Languages of the Game.

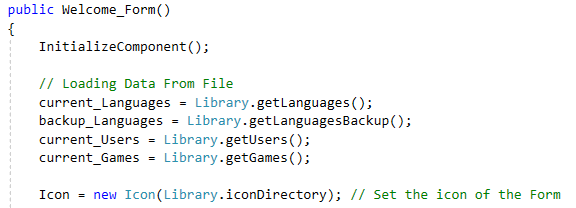


Figure 8.2

At the beginning of the Class constructor, the Data from the Files is loaded and stored inside the static variables. The Form’s Icon was also set to the Icon within the specified directory.

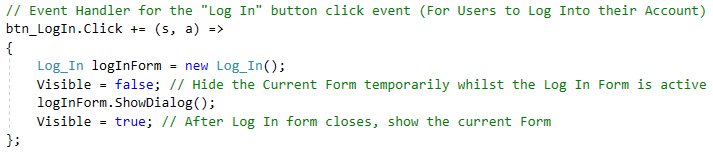


Figure 8.3

Once the Player decides to Log into his User Account by pressing the **Log In** button, the event handler in figure 8.3 is triggered.

A new instance of Log In form is created. The Welcome Form is set Invisible, and the Log In form is displayed. Once the Log In Form is closed, the Welcome Form is set Visible again.

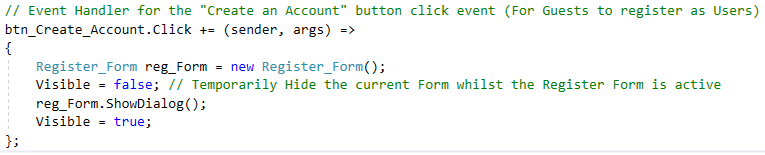


Figure 8.4

If the Player clicks the **Create an Account** button to register as a User, the event handler in figure 8.4 is triggered.

A new instance of Register form is created. The Welcome Form is set Invisible, and the Register form is displayed. Once the Register Form is closed, the Welcome Form is set Visible again.

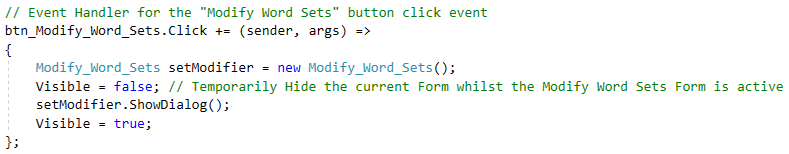


Figure 8.5

If the Player clicks the **Modify Word Sets** button to modify the existing Languages’ Word Sets, the event handler in figure 8.5 is triggered.

A new instance of **Modify Word Sets** is created. The Welcome Form is set Invisible, and the **Modify Word Sets** form is displayed. Once the **Modify Word Sets** Form is closed, the Welcome Form is set Visible again.

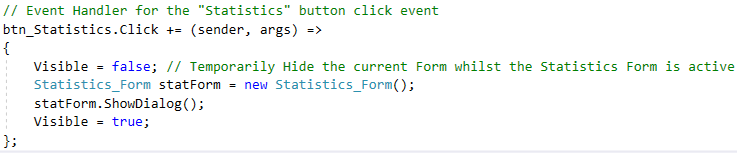


Figure 8.6

If the Player clicks the **Statistics** button to register as a User, the event handler in figure 8.6 is triggered.

A new instance of Statistics is created. The Welcome Form is set Invisible, and the Statistics form is displayed. Once the Statistics Form is closed, the Welcome Form is set Visible again.

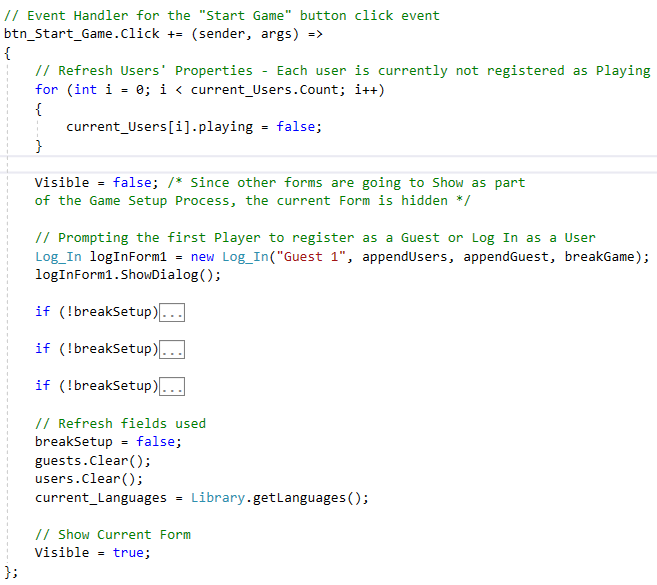


Figure 8.7

If the Player clicks the **Start New Game** button to Start the Game Setup, the event handler in figure 8.7 is triggered.

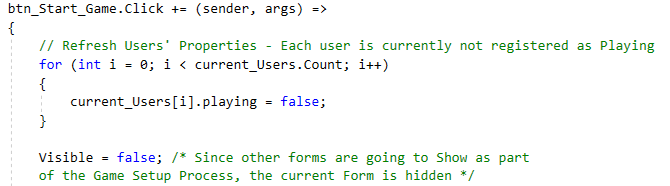


Figure 8.8

Every user’s **playing** attribute was set to false. This means that initially every User is not Playing.

Before Starting the Game Setup process, the Welcome Form is set Visible

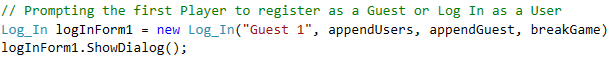


Figure 8.9

An instance of the **Log In** Form for the Player to login as a User/Guest was created. The Log In Form is then Displayed. The functions that:

* Append the Player to the playing Guests’ List
* Append the Player to the playing Users’ List
* Break the Game Setup – By setting breakSetup to True

were passed through the **Log In**’s constructor.

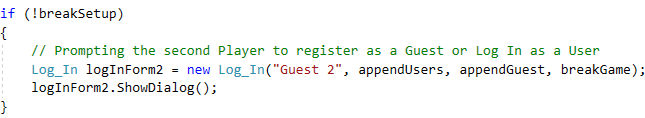


Figure 8.10

If Player 1 did not break the Game Setup, a Log In Form is created and displayed for Player 2 to Log In as a User or as a Guest.

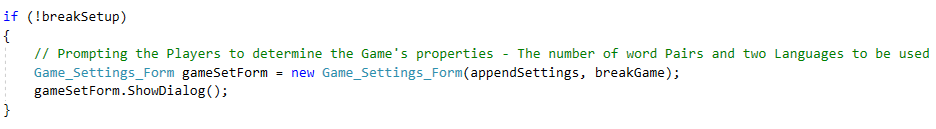


Figure 8.11

If the players did not break the Game Setup, an Instance of the Game Settings Form was created and displayed. This Form is responsible for establishing the Game Settings, including the Number of Word Pairs and the selected Languages. The following functions were passed through the class constructor:

* appendSettings – Sets the Game Settings; The number of Word Pairs and selected Languages
* breakGame – Breaking the Game Setup; Setting **breakSetup** to **True**

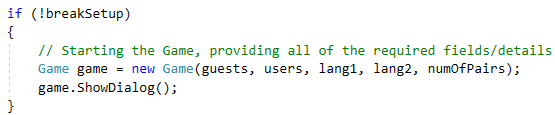


Figure 8.12

If the players did not break the Game Setup, an instance of Game is created, and the:

* List of Playing guests
* List of Playing users
* The two (2) Languages selected
* The number of Word Pairs

are passed through the class constructor. The Game Form is then displayed.

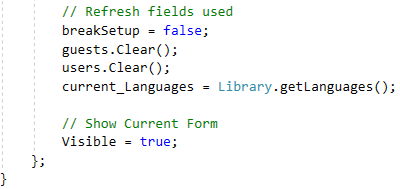


Figure 8.13

After finishing the Game Setup process, the **breakSetup** is set to **false**, and the List of playing Users and Guests are cleared, setting them to their **default** state before starting another Game.

The list of the current Languages was reinitialized with the saved Languages, as it has been used to supply buttons with words from the Languages’ Word Sets, and when assigning the words to the buttons randomly, the words were removed from Language’s word set to avoid assigning the same word to another button repeatedly.

Since the Game process has terminated/finished, the players were guided back to the **Welcome Form**, by making the **Welcome Form** visible again.

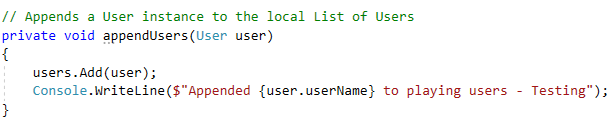


Figure 8.14

In the **appendUser** method, the User that is passed as an argument is appended to the Users playing List.

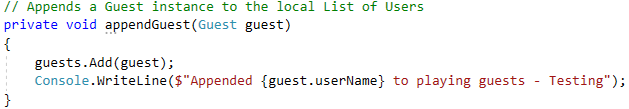


Figure 8.15

In the **appendGuest** method, the Guest that is passed as an argument is appended to the Guests playing List.

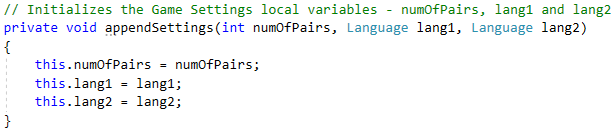


Figure 8.16

In the **appendSettings** method, the number of **word Pairs** that the players decided to include in the Game, and the selected Game **Languages** are initialized.

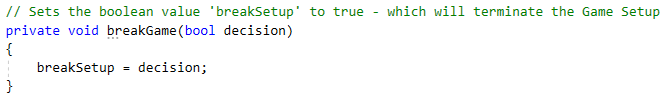


Figure 8.17

The scope of the **breakGame** method is to set the **breakSetup** field to **True**, which will cancel out all of the remaining Game Setup processes throughout the Game Setup.

**User\_Account\_Form Class**

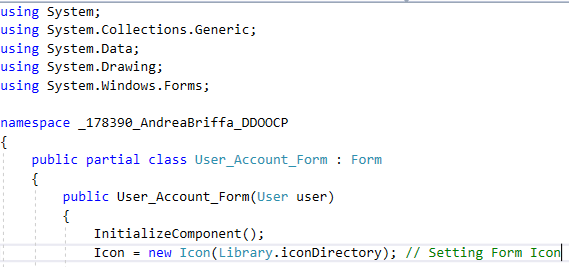


Figure 9.1

The User Account Form Class is responsible for displaying the User’s personal and game details inside the User Account Form. The User Account Form receives the User instance through the class constructor. To access his/her Account Portal, a User must Login using his/her credentials through the Log In Form.

Upon initialization the Form’s Icon is set to the Icon within the specified Directory.

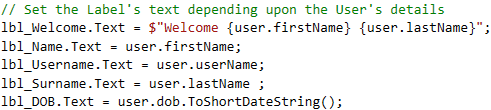


Figure 9.2

The User’s Details within the Account Form are set correspondingly to the User that Logged In.

The User’s Date of Birth is Displayed in a short Date representation, not including the Time.

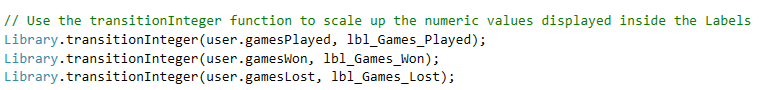


Figure 9.3

To display the Number of:

* Games Played
* Games Won
* Games Lost

the User carried out, the transitionInteger method present in the Library Class is used, to transition the numerical Values inside their respective Labels.

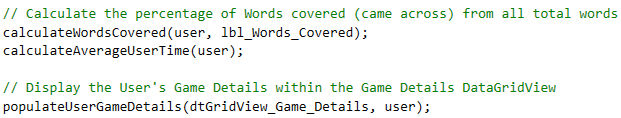


Figure 9.4

The User’s percentage of words covered is calculated using the **calculateWordsCovered (User user, Label label)** method. The User’s average Game time is displayed using the **calculateAverageUserTime (User user)** method.

The User’s Game details which consist of the:

* Game Number
* Personal Time Elapsed
* Score in the Game
* Whether the Game was won or lost

Were displayed inside the User Game Details DataGridView (fig 9.5).

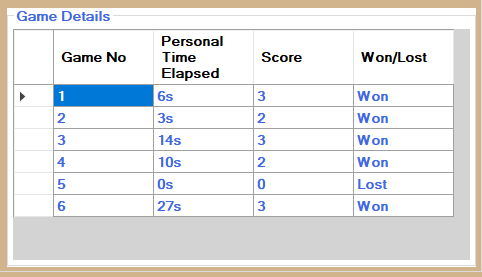


Figure 9.5

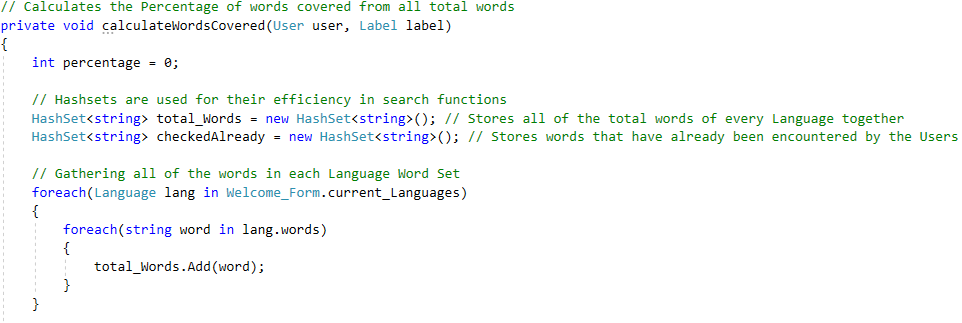


Figure 9.6

The scope of the calculateWordsCovered method is to display the percentage of Words covered by the User from the all of the existing words. The **percentage** variable will first store the number of different existing words guessed by the User, and then used to calculate the percentage of total words covered. Hashsets are used throughout this method due to their efficiency in search/retrieval functions.

Initially, all of each Language’s words are added to a Hashset which stores all of the existing words.

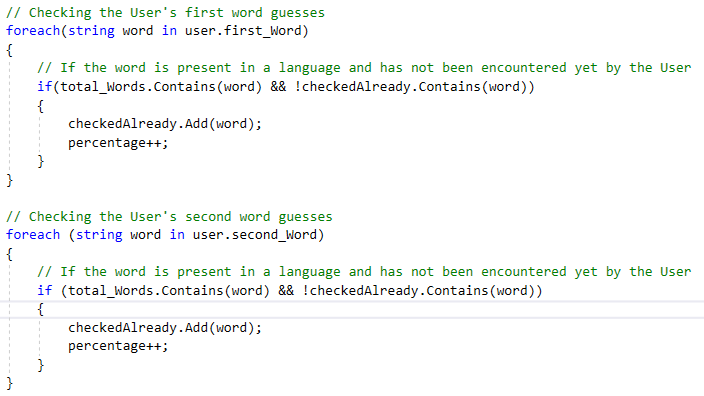


Figure 9.7

For each word that the User tried to Guess, either the first or second guess, it is checked whether the guessed word exists and hasn’t already been checked.

If the word hasn’t been checked yet and exists in the List of Current Languages’ total words, the word is added to a List of checked words, and the **percentage** variable is incremented.

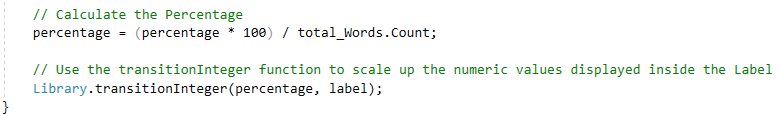


Figure 9.8

To calculate the percentage, the number of different existing words guessed by the User is divided by the total number of different existing words, and then multiplied by 100.

The percentage is then transitioned inside the label passed as an argument, which is responsible for holding the percentage of Words guessed by the User.

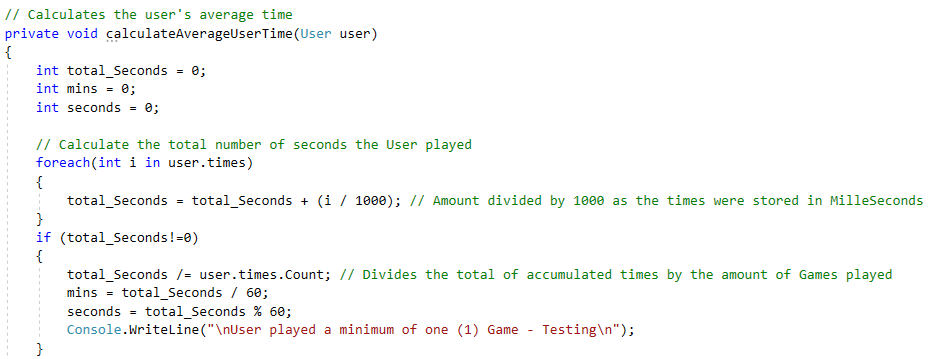


Figure 9.9

The calculateAverageUserTime method calculates the User’s average Game time in seconds.

First the total time of the User’s Game times is gathered in seconds.

If the User’s total Game time is not equal to 0, the average number of seconds for a Game is calculated, which is then divided into minutes and seconds.

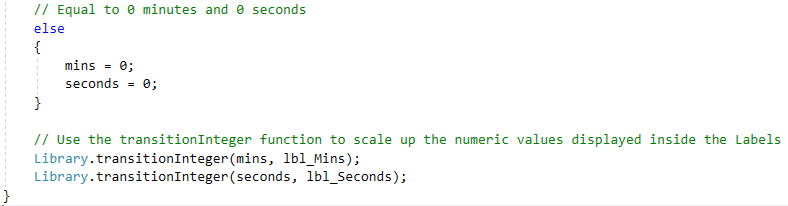


Figure 9.10

Else if the User’s total Game time is equal to 0, the User’s average minutes and seconds are set to **0**.

Once the User’s average time is calculated in minutes and seconds, the minutes and seconds are transitioned within their respective Labels.

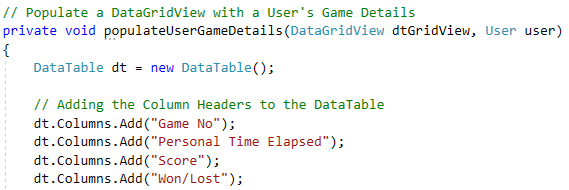


Figure 9.11

The populateUserGameDetails method populates the User’s Game Details DataGridView with the User’s Game Details.

Initially a DataTable is created, and the Game Details are added as columns.

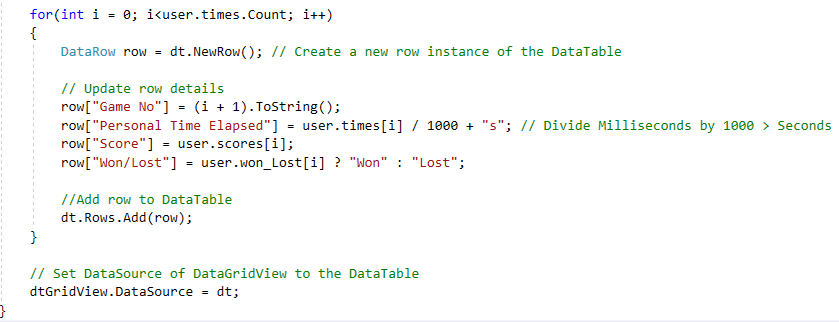


Figure 9.12

For each Game that the User played, a row is created containing the details of that particular Game, and added to the DataTable.

Once finished, the DataTable is set as the User Game Details DataGridView’s DataSource, displaying the User’s Game Details.

## **Game Class**

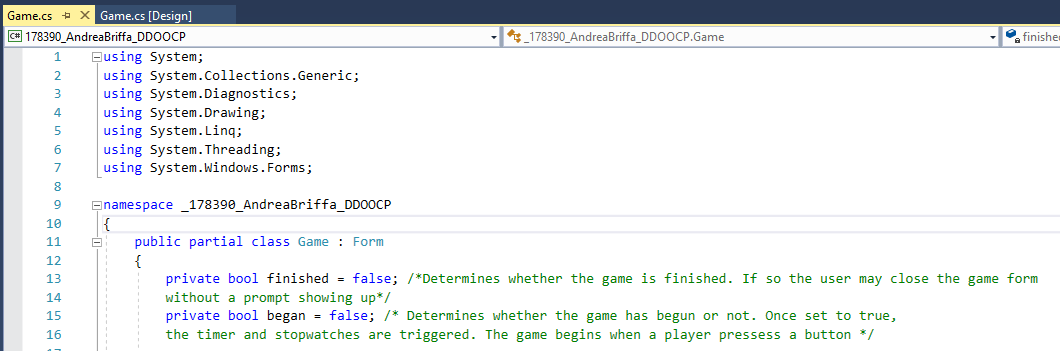


Figure 10.1

The Game class is responsible for handling the Game Form in which the Players take part in the Memory Game (fig 10.2).

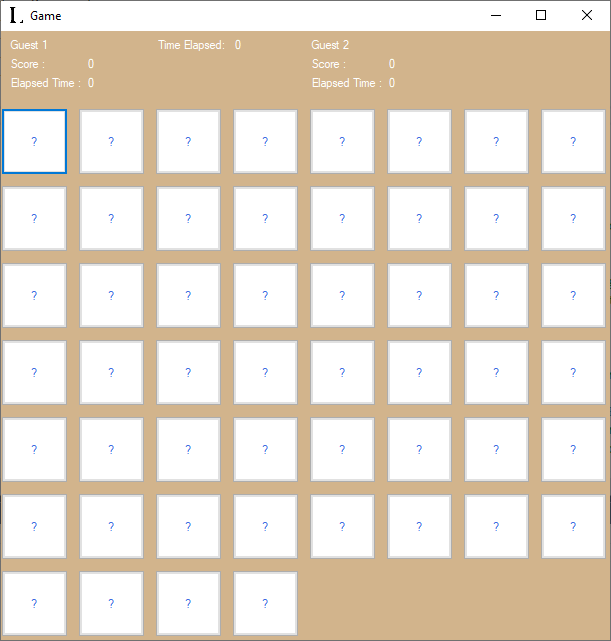


Figure 10.2

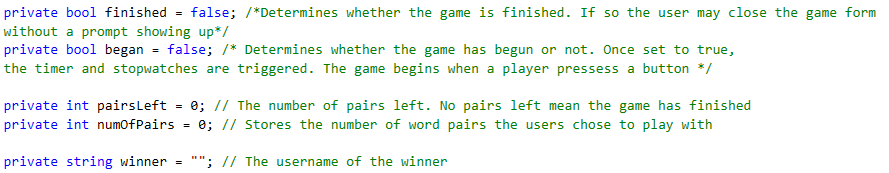


Figure 10.3

The **finished** field determines whether the Game has been finished (no word pairs left to Guess) or not.

The **began** field determines whether the Game has begun or not.

The **pairsLeft** field keeps track of the remaining number of word Pairs left to guess within the Game.

The **numOfPairs** field stores the number of word Pairs that the Users chose to have in the Game.

The **winner** field stores the username of the winning Player.

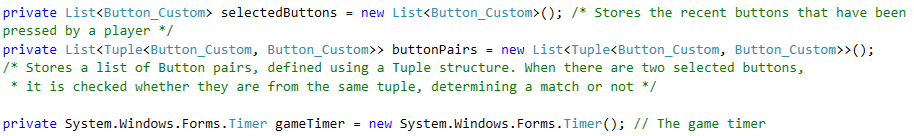


Figure 10.4

The **selectedButtons** list stores references to the buttons that the current Player selected. The List of **buttonPairs** stores a List of the button pairs, which contain matching words.

The **gameTimer** is used to update the game and player time elapsed labels, each time it ticks.

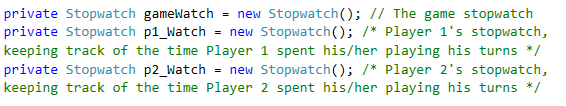


Figure 10.5

An individual **Stopwatch** is used to keep track of the Game’s elapsed time, and each Player’s individual elapsed time in milliseconds.



Figure 10.6

A label was created for every element inside the Game, such as the Player’s details.

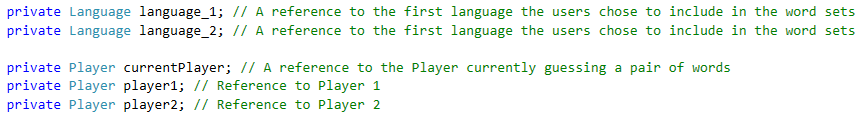


Figure 10.7

Two Language instances are created, with the scope of keeping a reference to the first and second Languages selected for the Game.

The player instances keep references to the first and second player, and also the current player.

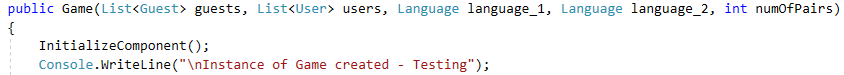


Figure 10.8

The constructor of the Game class receives the

* List of Playing Guests
* List of Playing Users
* Languages selected for the Game
* Number of Word pairs selected for the Game

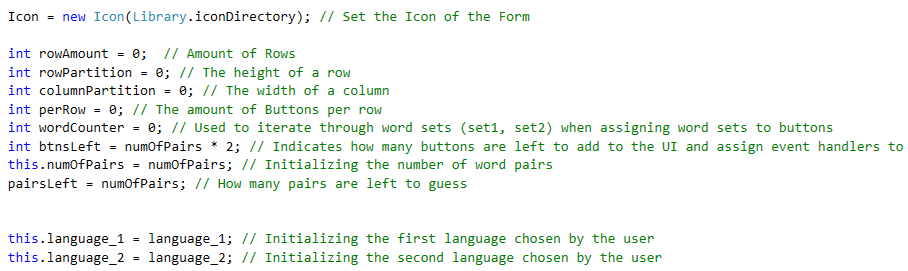


Figure 10.9

The Icon of the Game Form is set to the Icon in the specified Directory.

The fields holding the numOfPairs, pairsLeft and Languages selected for the Game are initialized.

A set of variables is also created:

* **rowAmount** – The number of Rows containing Buttons
* **rowPartition** – The height of a row
* **columnPartition** – The width of each Column containing Buttons
* **perRow** – The number of Buttons per Row / The amount of Columns
* **btnsLeft** – Keeps track of the remaining Buttons to be assigned to the Game Form when setting the Buttons’ properties
* **wordCounter** – Serves as a counter when iterating through the Word Sets (Used Later)

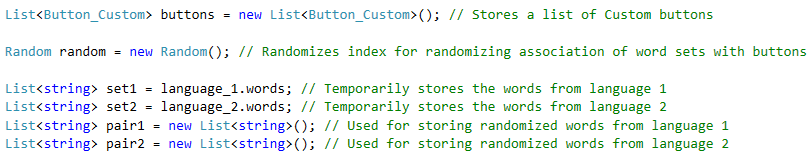


Figure 10.10

The **buttons** List stores a List of the Buttons that are placed inside the Game.

The **Random** class instance is used further throughout to randomize the Word Pairs.

**set1** and **set2** store the selected Languages’ word sets.

**Pair1** and **Pair2** store the selected Languages’ word sets, but randomized.

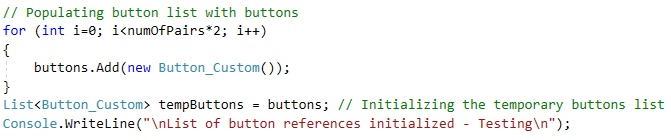


Figure 10.11

The list of Buttons is initialized with **custom\_Button** (fig 10.12) instances, which inherit from the Button class, but also have 3 extra attributes:

* **Word** – The word associated with the button
* **Language** – The language of the word associated with the button
* **Activated** – Determines whether the Button has already been pressed or not

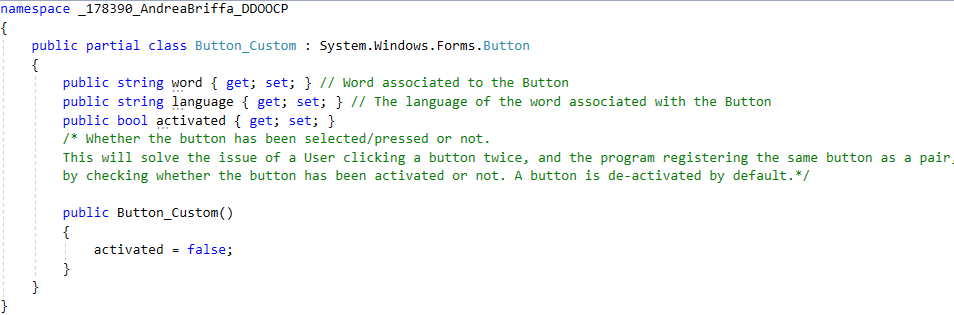


Figure 10.12

A separate List is created to temporarily store the List of Word Buttons, which is used further on to randomize the position of the buttons.

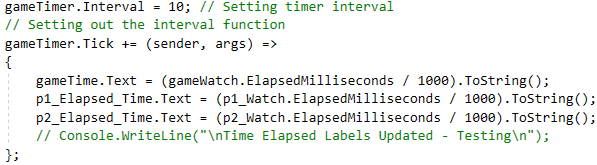


Figure 10.13

The game Timer’s tick event is set to update the Game and the Players’ time elapsed Labels. The interval of the Timer is set to 10 Milliseconds to avoid latency.

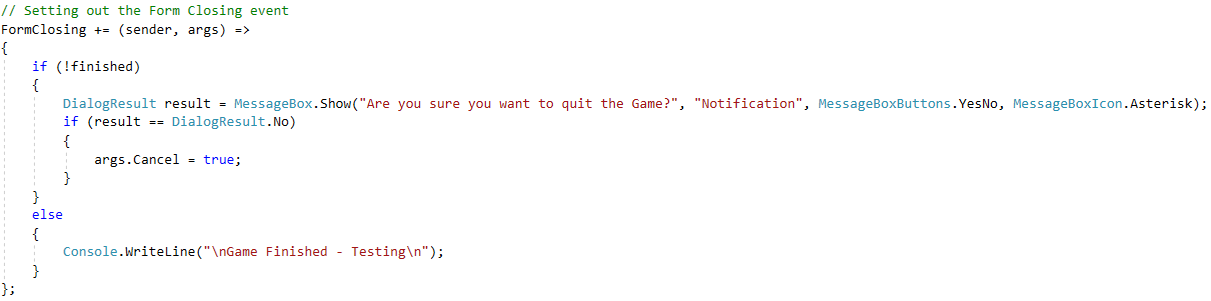


Figure 10.14

When the Game Form is closing, if the Game hasn’t finished yet, the Players are asked to verify if they wish to terminate the Game. If the Game has already finished, no verification is required from the Players.

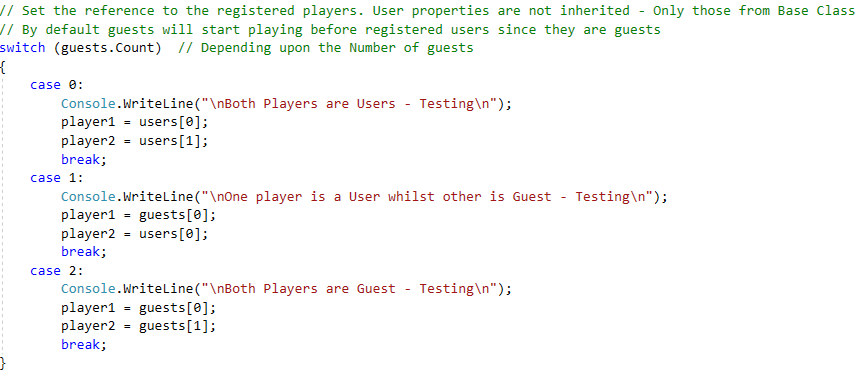


Figure 10.15

Depending upon the Number of Users and Guests logged, the player references were set to the playing Guests/Users.

Since both Guests and Users inherit from the Player class, the player references can refer to Guest and User instances, skipping out the User/Guest related attributes.



Figure 10.16

The current Player which is set to start off the game is set to Player 1.

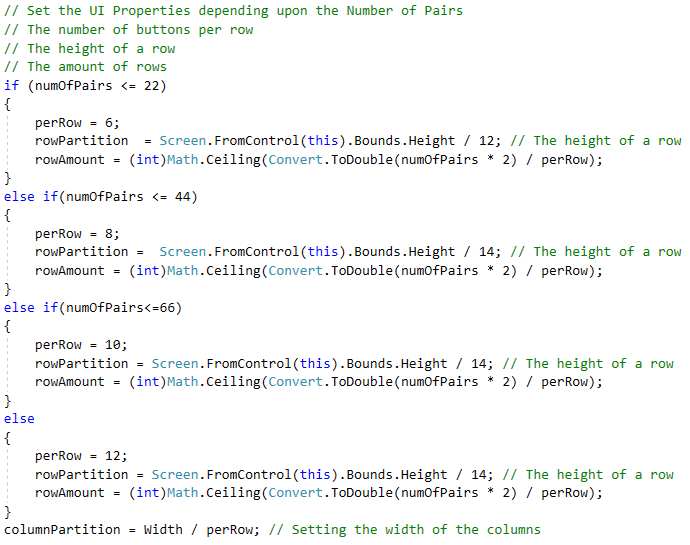


Figure 10.17

Depending upon the Number of word Pairs, the **number of Buttons per Row**, the **height of each row** and the **number of Rows** is set, to scale up to the number of Buttons.

The **number of Rows** is set to fit the buttons inside the Game Form, even if empty space is left inside the last row.

The width of each column is set depending upon the Width of the Form and the number of columns.

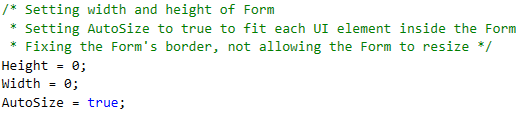


Figure 10.18

The **Height** and **Width** of the Form are set to 0, and **AutoSize** is set to **True**, scaling up the size of the Form to fit all of the elements.

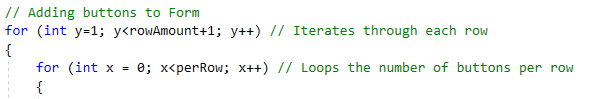


Figure 10.19

A loop will iterate through each row of the Form, and an inner loop will iterate through each Column inside the Form. These combined loops will place the Word Buttons inside the Form in a Grid template.

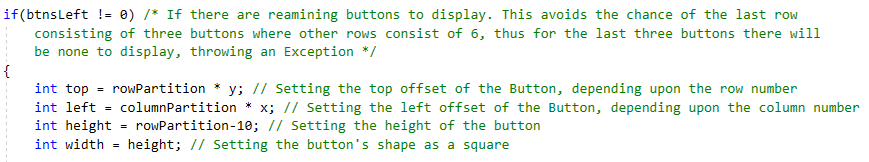


Figure 10.20

If there are remaining Buttons to display within the Game Form, the process of assigning a Button to the Form is started.

Firstly, position of the Button is calculated, followed by calculating the Height and Width of the Button.

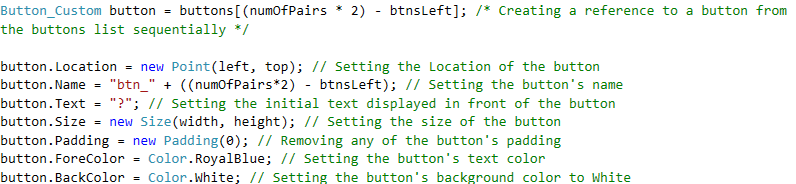


Figure 10.21

A reference to the current Button being added to the Game Form is created.

The properties of the Button are then initialized, such as the Location, the Size and the colours.

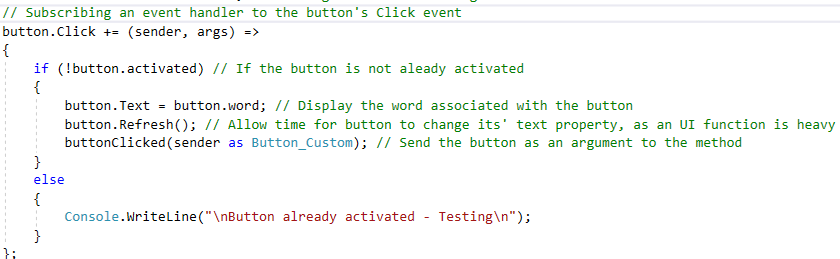


Figure 10.22

An event handler is then created for the Button.

If the button hasn’t been already activated(selected) by a Player, the text of the button is set the value of its’ associated word. The button is refreshed, as time is took to set the Text of the Button, setting a heavy load on the UI Thread.

The selected button is sent as an argument to the **buttonClicked** method, which handles the clicked buttons and checks of the buttons’ words match.

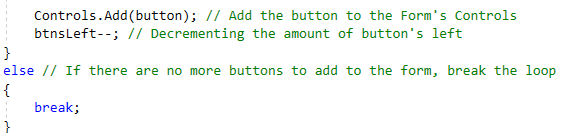


Figure 10.23

After assigning the Event Handler for the button’s **Click** event, the button is added to the Form’s Controls, displaying it.

The remaining number of buttons to add to the Game Form is decremented.

If no remaining buttons are left to assign to the Game Form, the Loop is broken.

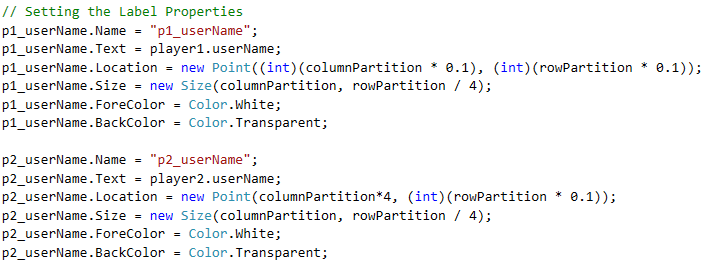


Figure 10.24

The properties of the players’ username Labels are initialized.

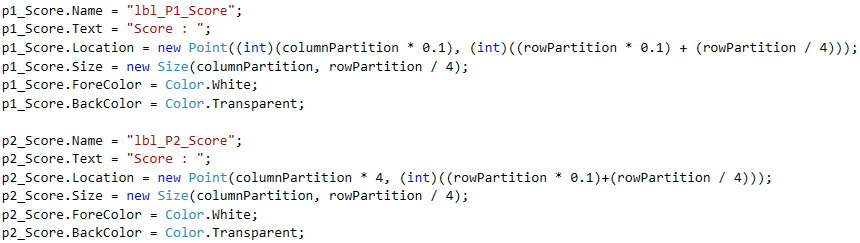


Figure 10.25

The properties of the players’ score Identifier Labels are initialized.

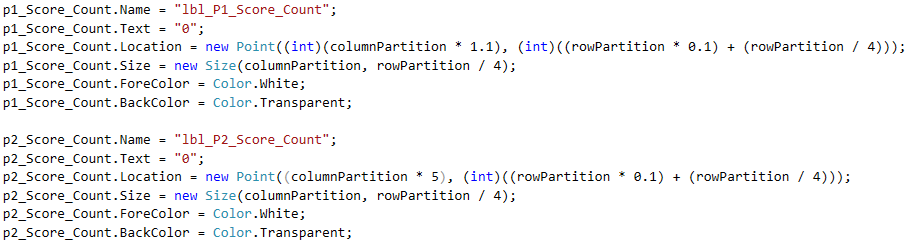


Figure 10.26

The properties of the players’ score Count Labels are initialized.

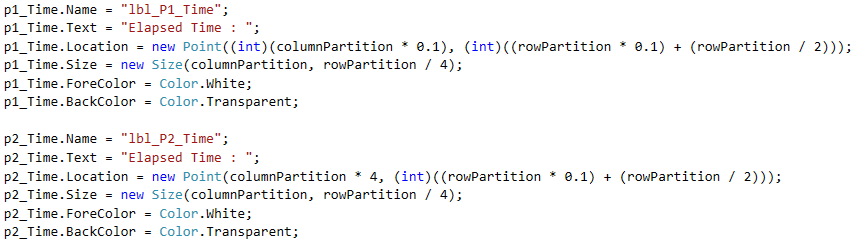


Figure 10.27

The properties of the players’ Time Elapsed Identifier Labels are initialized.

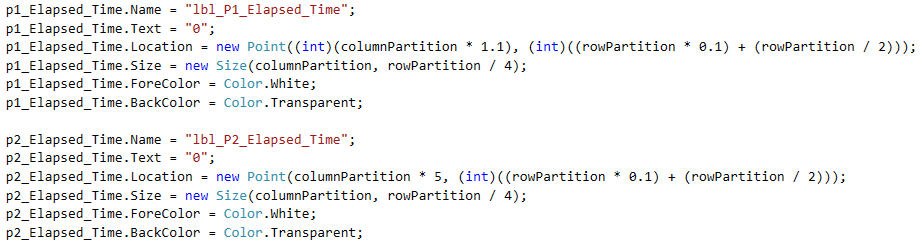


Figure 10.28

The properties of the players’ Time Elapsed Labels are initialized.

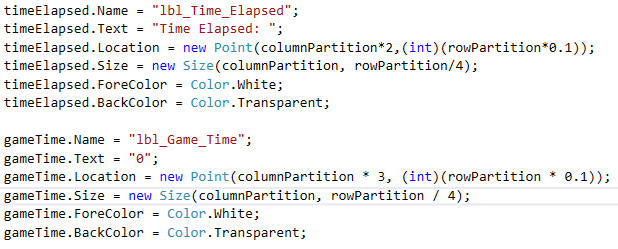


Figure 10.29

The properties of the Game’s Time Elapsed Identifier and Time Elapsed Labels are initialized.

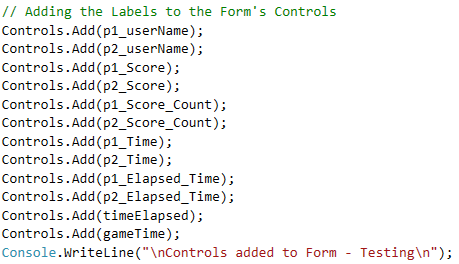


Figure 10.30

Following the initialization of the Labels, the Labels are added to the Game Form’s controls, adding them to the Form’s User Interface.

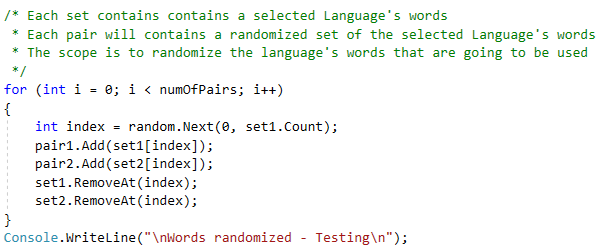


Figure 10.31

The selected language’s word sets are randomized. The process is done by randomly selecting an index which corresponds to a Word Pair. The word pair at the specified index is appended to the randomized word pairs lists (**pair1** , **pair2**). The word pair at the specified index is also removed from the word sets.

The process is repeated until the number of randomized word pairs reach the Player’s selected amount of Word Pairs.

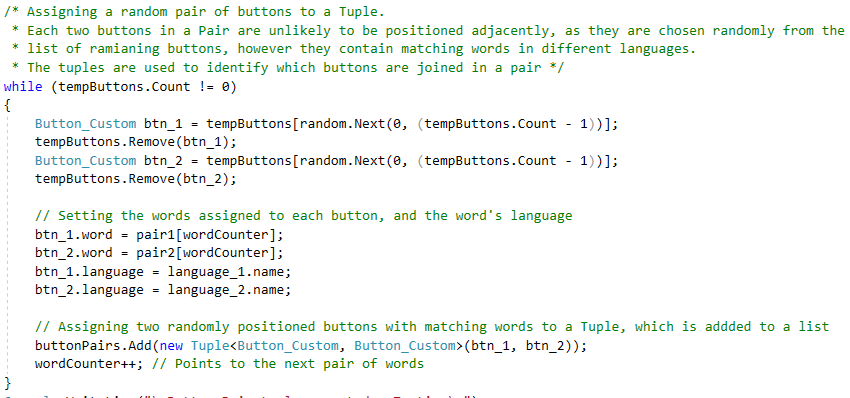


Figure 10.32

For each button that is left to be assigned a word, the process of initializing the button with a Word and pertinent Language is carried out.

Two button references are created, and removed from the List of buttons remaining to get initialized with a pertinent word and language.

The two button references are assigned a word and pertinent language each, each having a word with the same meaning in a different Language.

After a new Tuple is creating, storing the references of the two buttons, which combine as a Pair.

The **wordCounter** is then incremented to point to the next pair of words.

This altogether ends the class constructor.

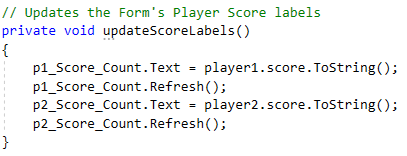


Figure 10.33

The updateScoreLabels method updates the score value inside the players’ score labels.

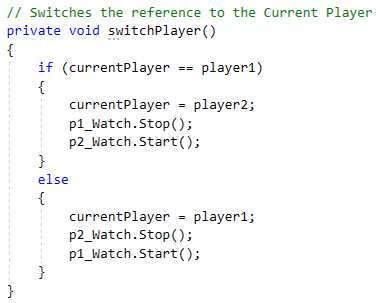


Figure 10.34

The **switchPlayer** method switches the current player reference. The current Player’s **stopwatch** is paused, the other player’s **stopwatch** is continued.

The other Player is set as the **current Player**.

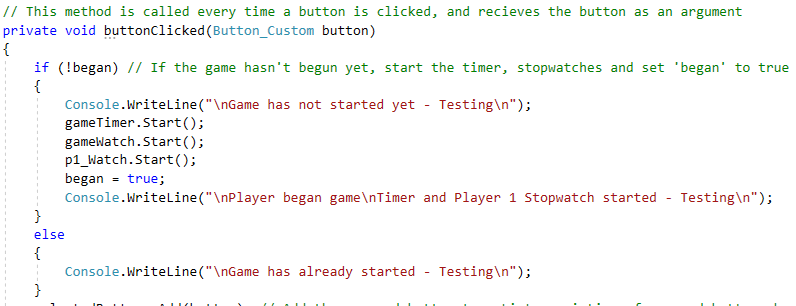


Figure 10.35

The **buttonClicked** method is executed each Time a Game Button is clicked. The method received the selected button as an argument.

If the Game has not started yet, the game Timer and Stopwatch, and player 1’s stopwatch are started.



Figure 10.36

The button clicked by the Player was added to the List of selected buttons. The button’s activated state was set to **True**, avoiding the button being registered as a clicked button consecutively.

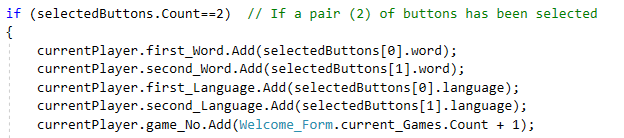


Figure 10.37

If the number of selected buttons is equal to **2**, the details of the guessed word pair are added to the current Player’s details. These details include:

* The first word Guessed
* The second word Guessed
* The first word’s language
* The second word’s languages
* The game number

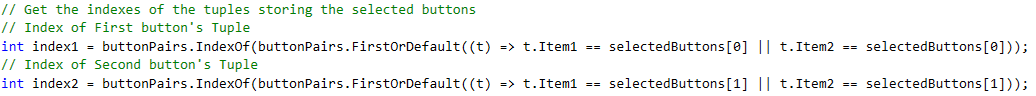


Figure 10.38

The indexes of the Tuples that contain the references to the selected buttons are retrieved. If both indexes match, that means that the buttons are from the same Tuple.

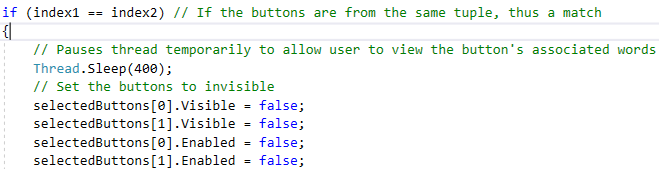


Figure 10.38

If the buttons are from the same tuple, meaning they have matching words, the buttons are disabled and set invisible.

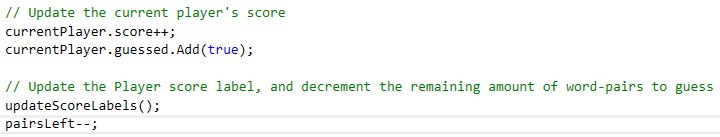


Figure 10.39

The current Player’s score is incremented, and that s/he guessed the pair of words.

The Players’ score Labels are updated to display the updated scores.

The **pairsLeft** value is decremented, indicating that the number of Word pairs left to be guessed has decreased.

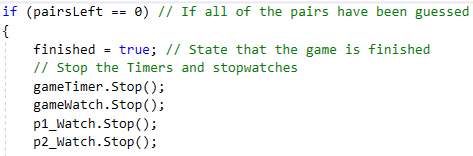


Figure 10.40

If no remaining Word Pairs are left, the Game Timer and all Stopwatches are stopped, and **finished** is set to **True**, indicating the Game has finished.

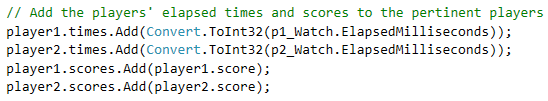


Figure 10.41

The game times and scores were assigned to the respective players.

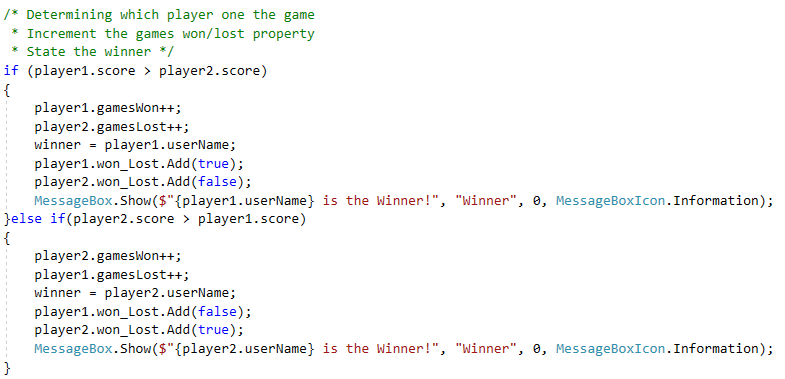


Figure 10.42

The winner of the Game is determined by the Score. Depending upon the Score, the player details are updated.

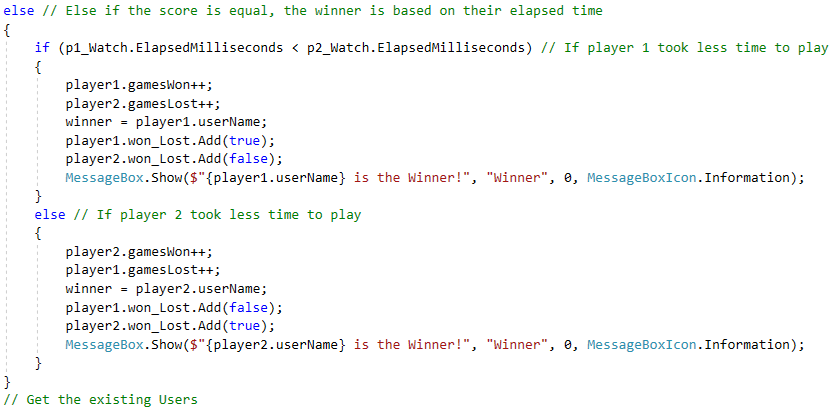


Figure 10.43

If both players have equal scores, the winner is determined by the amount of personal elapsed milliseconds. The player with the least elapsed time is the Winner.

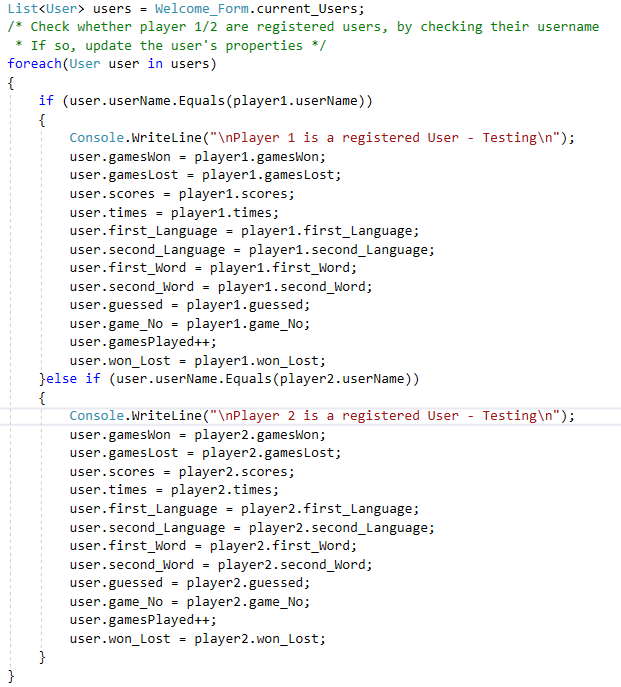


Figure 10.44

A List of existing users is created. Each existing user is checked if its’ username matches player 1’s or player 2’s username. If so, the details of that player will be added to the User’s.

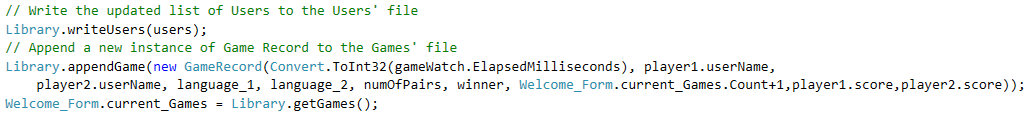


Figure 10.45

The list of Users is then written to the Users File, re-writing it, and the current Game Details are passed onto a new instance of **Game Record**, which is appended to the Game Records File.

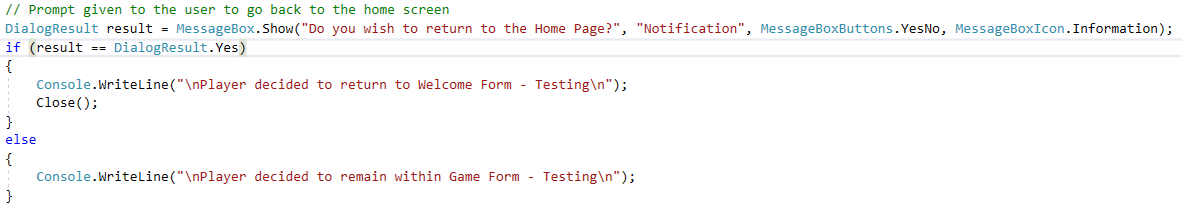


Figure 10.46

The Players are then prompted if they wish to return to the Welcome Form, rather than staying inside the Game Form. If the Players verify to do so, the Game Form is closed. If not, the Players remain on the Game Form.

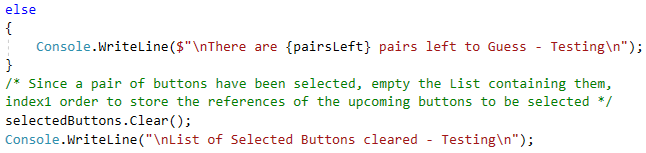


Figure 10.47

After checking whether the selected buttons are from the same Pair or not, the List of Selected Buttons is cleared, to cater for the upcoming selected Buttons.

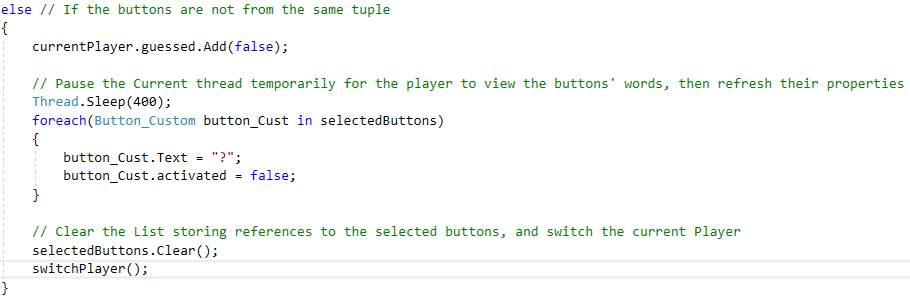


Figure 10.48

If the selected buttons are not from the same Tuple (pair), it is remarked that the current player did not guess the pair of Words. The Players are allowed to view the words associated with the selected buttons for a small period of time, which then are hidden and replaced with “**?**”.

The List of selected Buttons is cleared to cater for the upcoming List of selected Buttons, and the current player is also switched, since the player did not guess the pair of words.

## **Game\_Settings\_Form Class**

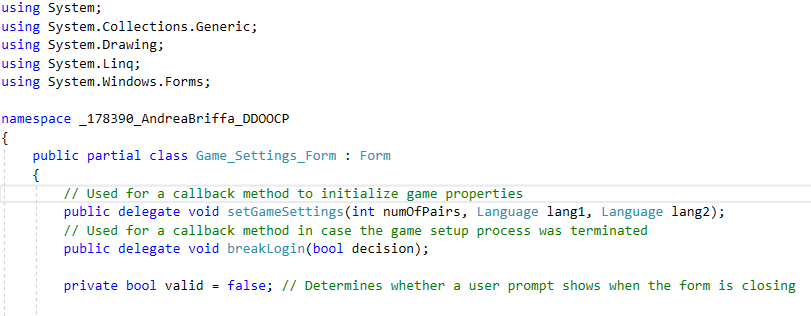


Figure 11.1

The **Game Settings Form** is responsible of giving the ability to the Players to choose the number of Word Pairs and which two Languages they would like to take part in the Game.

To Game Settings Form is opened from the Welcome Form, and passes back the Game settings through the use of **callback methods**, using **delegates**. From the Welcome Form, the Game setup process may also be terminated, if the Players decide so.

The valid field determines whether the Players have submitted their Game Settings preferences correctly, or not. This will affect the **FormClosing** event, as if the Players haven’t submitted their preferences and the Form is closing, that means that they are attempting to terminate the Game Setup.

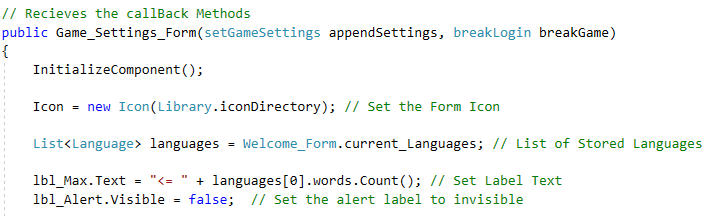


Figure 11.2

Within the constructor, the callback methods are passed. One method passes the Game Settings to the Welcome Form, and another method breaks the Game Setup.

A List of current languages is created, which will be used later to populate the Languages Combo Boxes.

The Label which indicates the maximum amount of word pairs (fig 11.3) is updated correspondingly to the number of Words in each Language’s Word Set.



Figure 11.3

By default, the number of Word Pairs for the Game is set to **2**, which is a valid number, thus the default visible state of the alert Label (fig 11.4) is **False**.



Figure 11.4

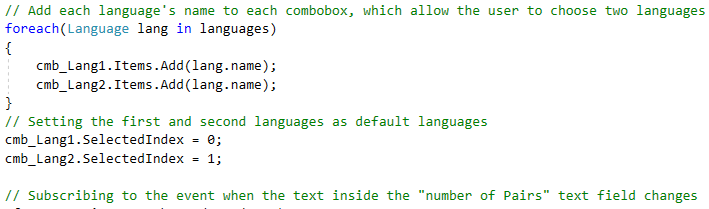


Figure 11.5

Two combo boxes (fig 11.6) are used to allow the players to chose their combination of Languages for the game according to their preferences.

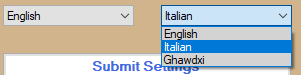


Figure 11.6

Each Language’s name is added to the **drop-down list** of each **ComboBox**. The default selected indexes of the Combo Boxes are set differently, as the same Languages cannot be chosen for the Game.

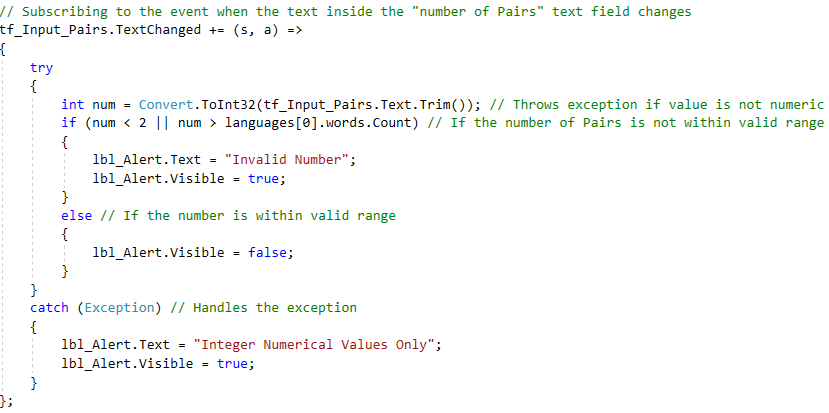


Figure 11.7

When the text value inside the number of Pairs textbox is changed, it is validated to see whether the text value is a valid number of Pairs input or not.

Firstly, the text value of the textbox is converted to an Integer value, which will cause an Exception if it is not in proper Integer format. The Exception will be handled, and the A**lert Label** will display.

If the parsing to an Integer does not throw any Exceptions, the value is if checked whether it is within valid range. If the number is not within valid range accotding to the maximum number of pairs allowed, the Alert Label will be set Visible.

If the value is within valid range, the Alert Label is set Invisible.

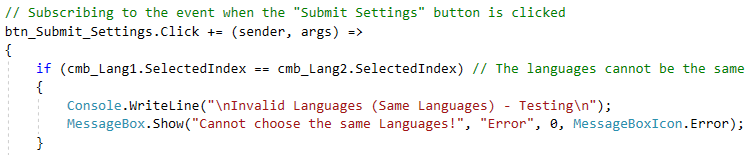


Figure 11.8

If the Submit Settings Button is clicked, the number of Selected Pairs and selected Languages are validated.

If the same Languages are selected, the Game Settings are not Valid.

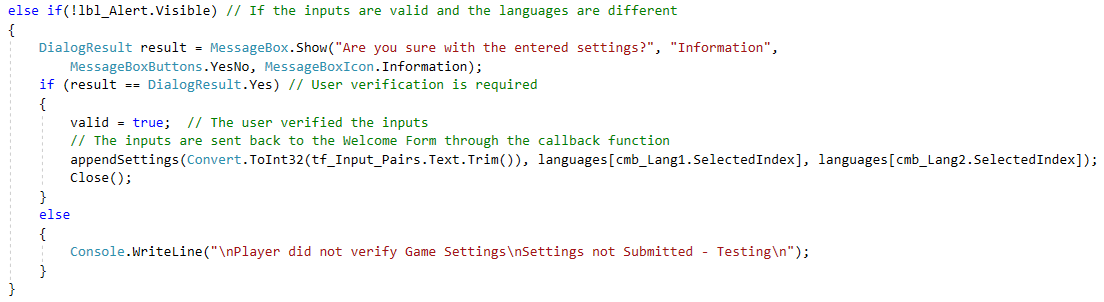


Figure 11.9

If the selected Languages are not the same, and if the Alert Label is not visible, the Players are asked to verify the selected Game Settings before submitting them to the **Welcome Form**.

If the alert Label is Visible, the Game Settings are not submitted as the selected number of Word Pairs is invalid.

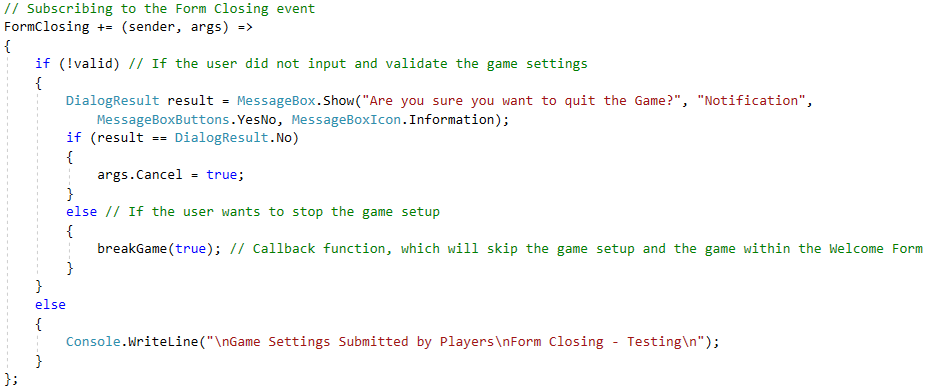


Figure 11.10

The Form closing event may occur if the Players are attempting to cancel the Game Setup, or the Players have submitted their Game Settings.

If the **valid** field is **false**, it means that the Game Settings have not been submitted, meaning that the Players are attempting to Cancel the Game Setup. If so, Player verification is required.

If the **valid** field is **true**, the Form is Closed as the Players have successfully submitted the Game Settings.

## **Register\_Form Class**

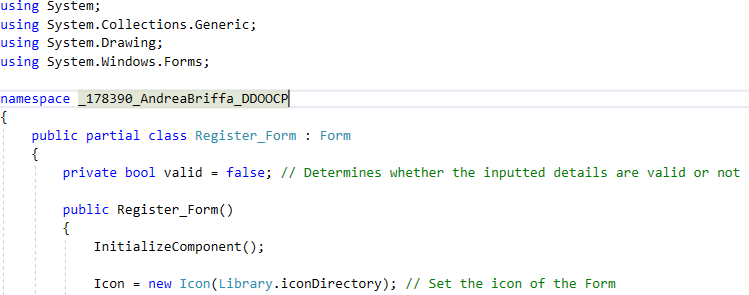


Figure 12.1

The Register Form is responsible for Players to Register as Users.

The **valid** field determines whether the Player has registered Successfully as a User or not. This affects the **Form Closing** event, and if the Player hasn’t registered successfully as a User, verification is required to Close the Registration Form.

The Form’s Icon is set to the **Icon** within the specified Directory.

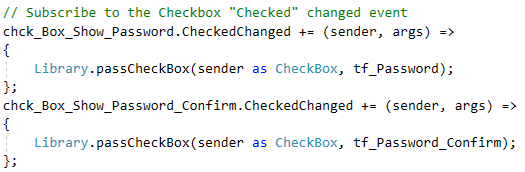


Figure 12.2

If the “**Show Password**” checkboxes are **checked/unchecked**, an Event Handler is subscribed to the Event. The Event Handler will pass the Checkbox as an argument to a static method, which will **obfuscate/show** clearly the text inside the **Password Textbox**.

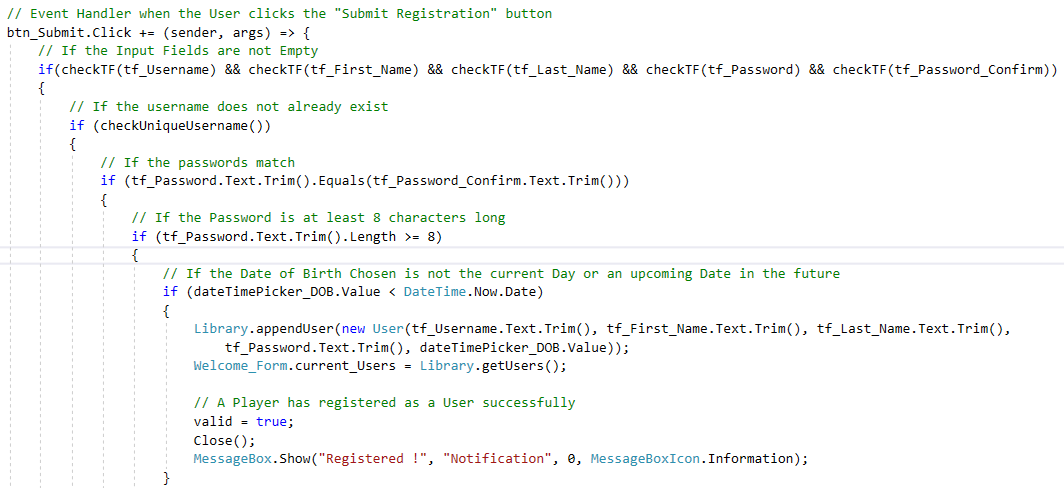


Figure 12.3

When the Submit Button is clicked to submit the Player’s inputs, an Event Handler is subscribed to validate all of the inputs before registering the Player as a User.

The validation phases are:

* Checking that all Inputs contain valid Text
* Checking if the Username is unique
* Checking if the Passwords Match
* Checking that the Password’s Character Length is **8** or higher
* Checking if the Date of Birth is in the Past

If all of the inputs are valid, a new User instance is appended to the existing Users File with the inputted details. The current List of Users is then updated.

**valid** is set to true since the Player has successfully registered as a User, and the Registration Form is closed.

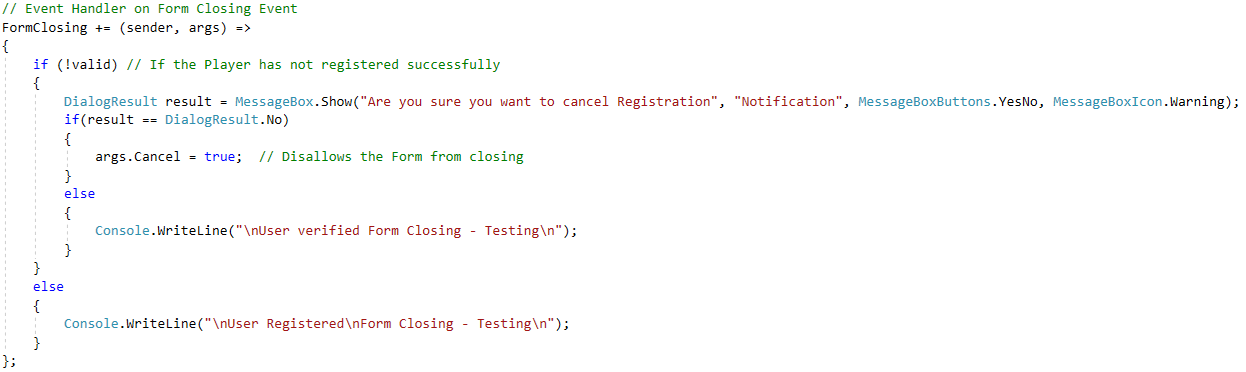


Figure 12.4

Upon the Form Closing Event, if the Player did not successfully register as a User, Verification is required to close the Registration Form, as the inputted details would be Lost.

If the Player has successfully registered as a User, the Form Closing Event is not cancelled.

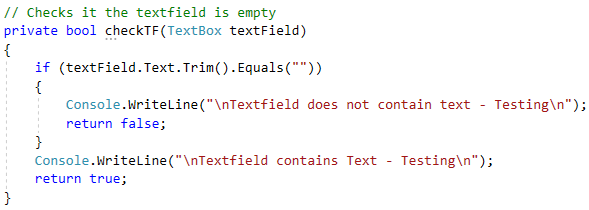


Figure 12.5

The **checkTF** function checks if the **text value** inside the TextBox passed as an argument is empty or not. **Whitespace/Spaces** are not counted as valid text.

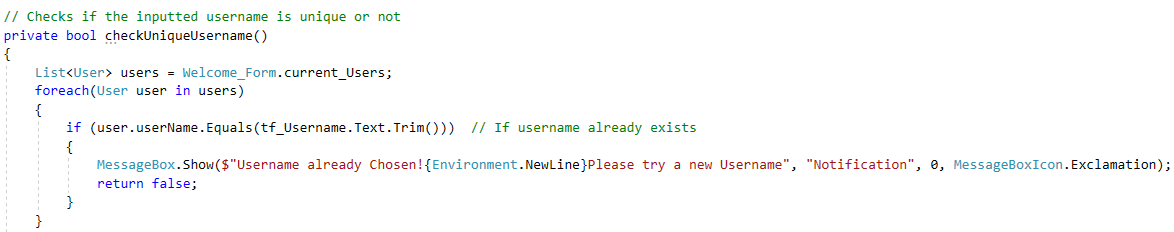


Figure 12.6

The checkUniqueUsername function checks whether the inputted username already exists or not.

The username is compared to each existing User’s username, and if it matches, **false** is returned signifying that the Username is not Unique.

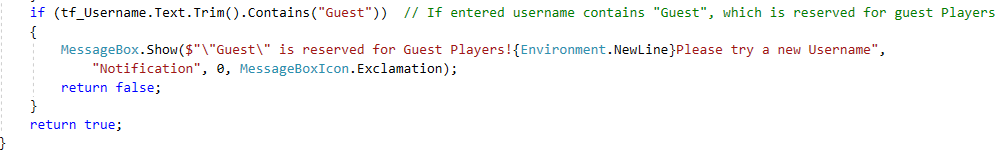


Figure 12.7

If the username is does not exist, it is checked whether the substring “Guest” is present inside the inputted Username. If so, **false** is returned, as the “**Guest**” keyword is reserved for Guest players.

If the Username is unique, and does not contain the substring “**Guest**”, **true** is returned meaning that the Username is valid.

## **Statistics\_Form Class**

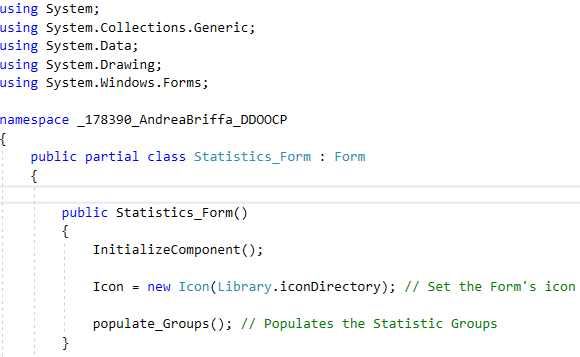


Figure 13.1

The Statistics Form is responsible for displaying the Users and Game Statistics (fig 13.2).



Figure 13.2

The Icon of the Form is set to the Icon within the specified Directory.

The **populateGroups** method is called, which populates statistics Groups within the Statistics Form.

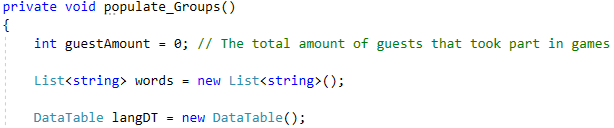


Figure 13.3

The **guestAmount** stores the total number of Guests that have took part in Games.

The **words** List stores the List of words that are pertinent to the selected Language (fig 13.4).

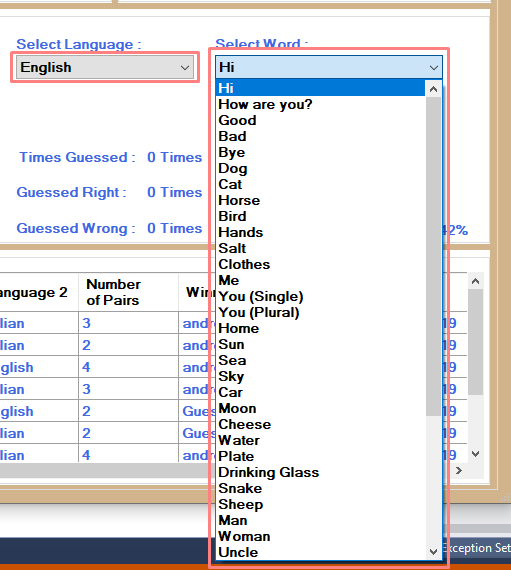


Figure 13.4

The **langDT** DataTable is used to display the Languages’ High Scores (fig 13.5).



Figure 13.5

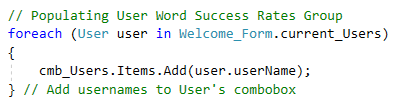


Figure 13.6

For each existing User, the User’s username is added to the User ComboBox. This allows to shop the Word Success Rates of a particular User, by choosing the User from the User combobox (fig 13.7).

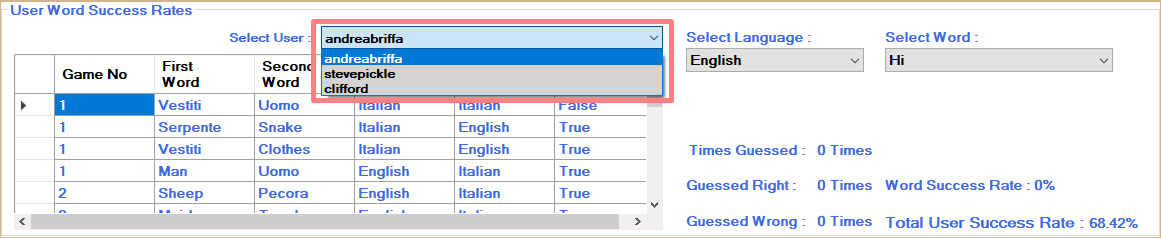


Figure 13.7

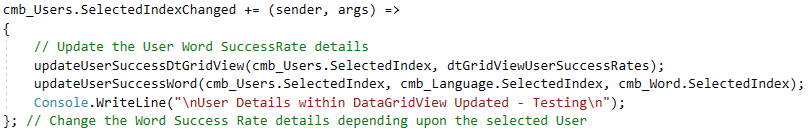


Figure 13.8

When the selected User within the Users combobox is changed, the Word Success Rates are updated to correspond to the selected User.

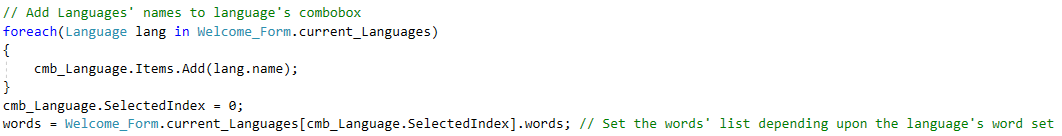


Figure 13.9

Each Language’s name is added to the Language combobox. The selected Language is set to the first Language within the Languages ComboBox.

The **words** List is then filled with the selected Language’s pertinent Word Set.

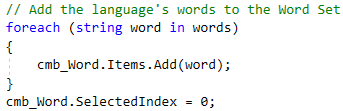


Figure 13.10

For each Word that is within the Word Set of the selected Language, the Word is added to the Words combobox (fig 13.11). The selected Word is then set to the first word.

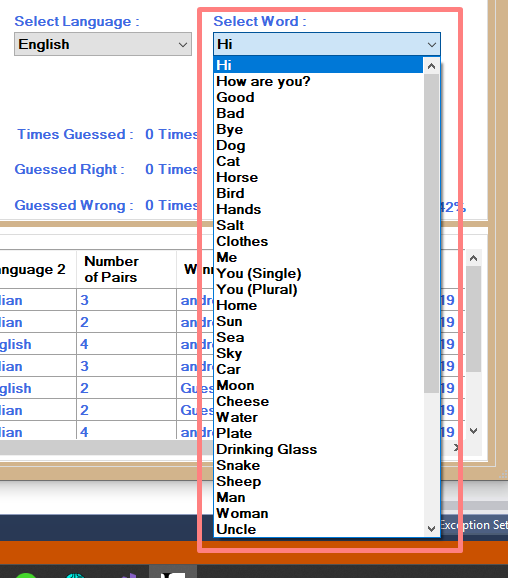


Figure 13.11

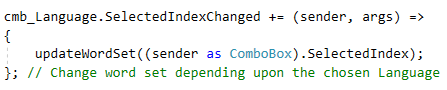


Figure 13.12

When the selected Language is changed within the Language’s ComboBox, the displayed Word Sets are updated (fig 13.13).

The Word Set ComboBox is updated depending upon the **selected index** of the **Languages ComboBox**.

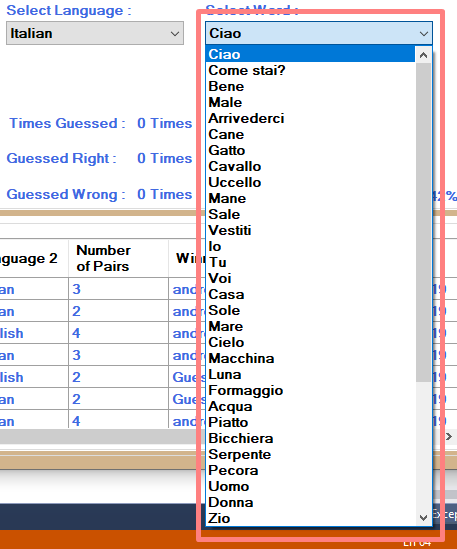


Figure 13.13

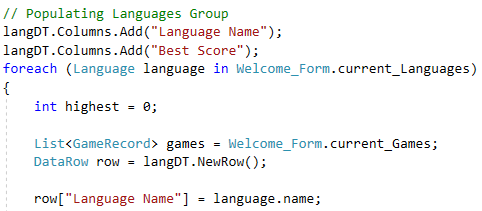


Figure 13.14

The next step is to populate the Languages’ High Score table. The Column Headers are added to the Table. A loop is carried out for each existing Language.

The **highest** variable will store the highest score ever reached with the Language currently being assessed.

The List of Game Records was retrieved. A row is created, which will consist of the **Language’s name** and the **Highest score** reached in that Language.

The **Language’s name** was set to the Language of the current Language being assessed.

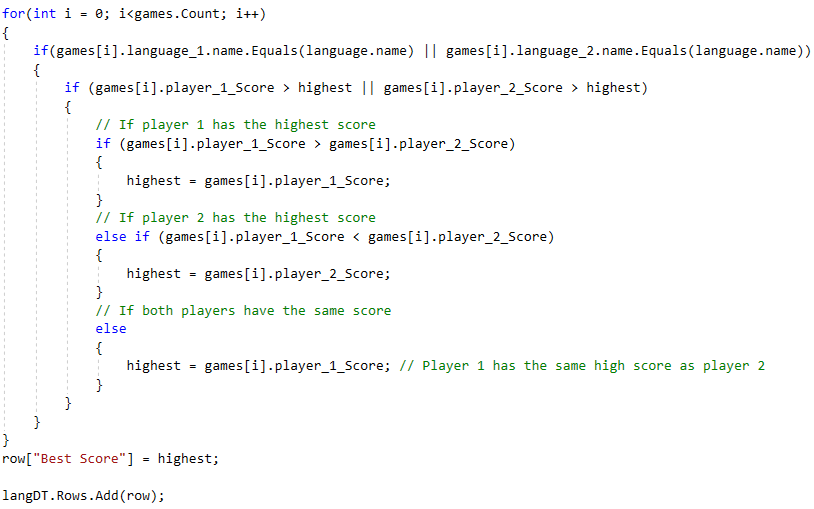


Figure 13.15

Each Game record consists of the **Languages** and **scores reached** in the Game.

For each Game Record, it is checked whether the current Language being assessed was played in the Game.

If the Language was played in the Game, it is checked whether the **scores reached in that Game** are higher than the current **highest** score reached in the Language currently being assessed.

If so, the highest score reached in the game is set as the **highest** score currently reached in the Language currently being assessed.

After checking all of the Game Records and finding the **highest** score reached for the Language currently being assessed, the highest score is added to the **row**.

The row is then added to the **Languages’ High Score** table.

After finding the High score for each Language, the Languages’ High Score table is populated (fig 13.16).



Figure 13.16

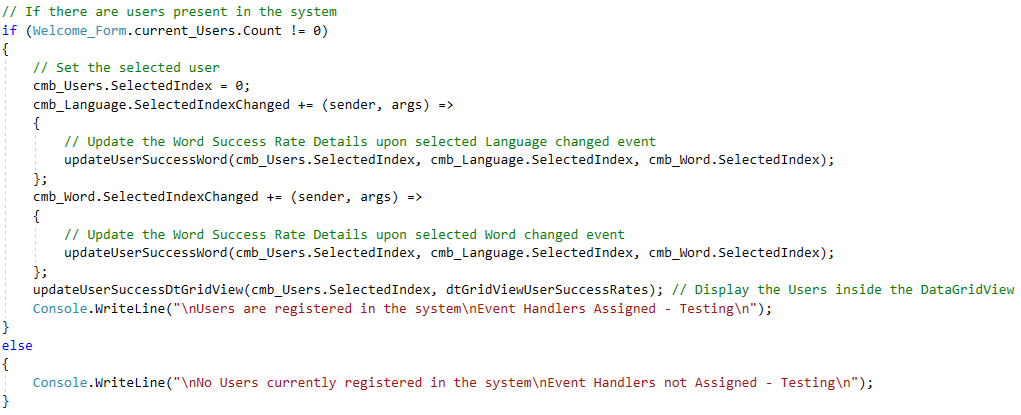


Figure 13.17

If there are Users that are currently registered, the selected Index of the User ComboBox is set to the first User.

The reason checking if there are existing Users is that without any Users, when setting the selected User Index, an Exception would be thrown as there are no existing Users.

As well, if there are current Users registered, event handlers are assigned to when the selected Language or Word is changed. The event handlers update the User’s Word Success Rates to correspond to the selected **Language** and **Word** (fig 13.18).

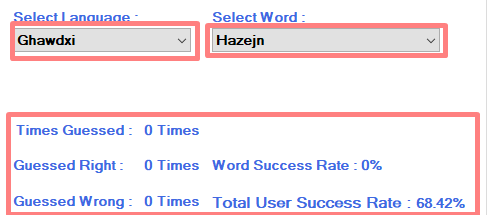


Figure 13.18

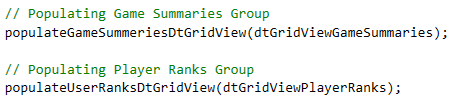


Figure 13.19

The **Game Summaries DataGridView** and the **User Ranks DataGridView** were populated, by calling the respective methods to do so.

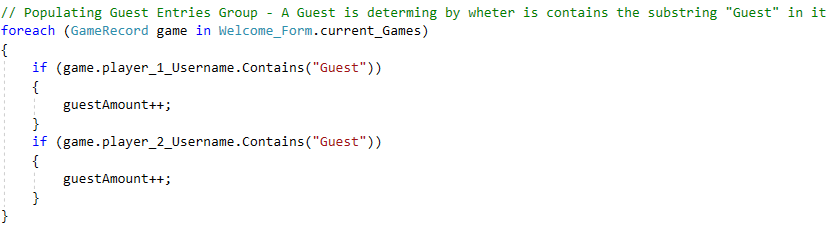


Figure 13.20

The number of Guests is calculated by checking if the players in each Game played so far had the substring “**Guest**” as the username of the first or second player.

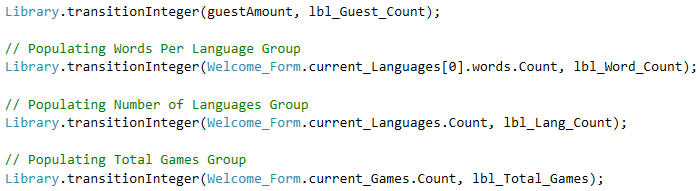


Figure 13.21

The remaining of the Statistics information was displayed (fig 13.21):

* The total Number of Guests that took part in Games (fig 13.22)
* The number of Words Per Language (fig 13.23)
* The Number of existing Languages (fig 13.24)
* The Number of Games played (fig 13.25)

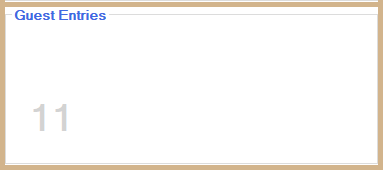


Figure 13.22

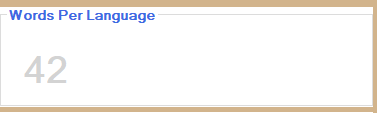


Figure 13.23

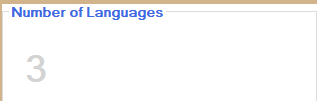


Figure 13.24

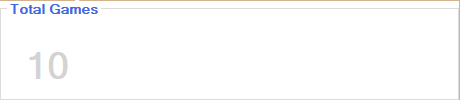


Figure 13.25

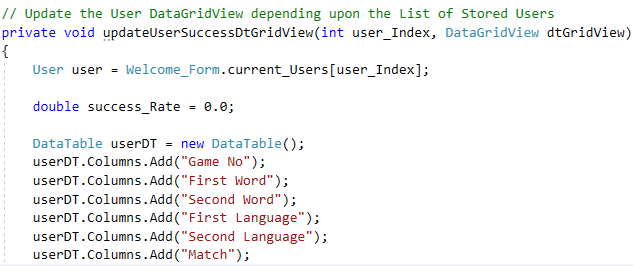


Figure 13.26

The **updateUserSuccessDtGridView** method populates the DataGridView which display’s all of the selected User’s Word Guesses (fig 13.27).

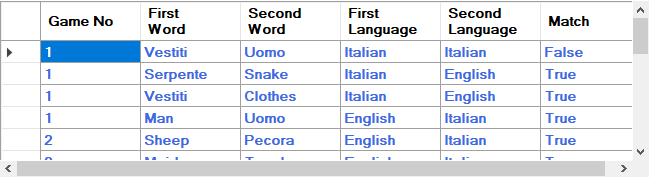


Figure 13.27

A User instance is created, setting reference to the selected User. The **success\_Rate** variable stores the percentage of the User’s success rate with Guessing Words.

A DataTable instance was created to display all of the User’s guesses (fig 13.27).

The **column headers** were then added to the **DataTable** (fig 13.26).

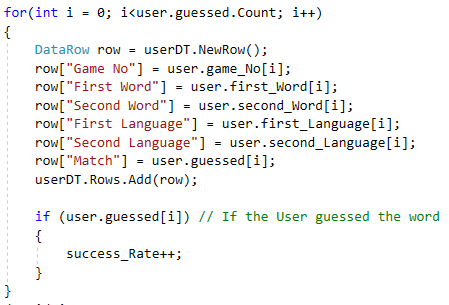


Figure 13.28

A row is created which will store the details about a particular guess. The details for the Words Guessed, the Language of the Words Guessed, if the User Guessed the word pair or not and in which Game the User made the Guess.

If the User had guessed the Pair, the success\_Rate is incremented which currently stores the number of correct guesses the User made.

After adding the details to the Row, the row is appended to the DataTable.

After adding all the rows to the DataTable, the DataSource of the DataGridView is set to the DataTable, displaying all of the User’s Guesses (fig 13.29).



Figure 13.29

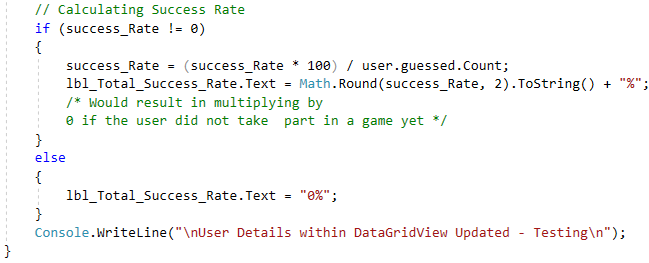


Figure 13.30

If the User did not guess any word pairs, the User’s success rate is set directly to **0**. If the User did guess any word pairs, the success rate of the User is calculated, and displayed in the pertinent **Label**.

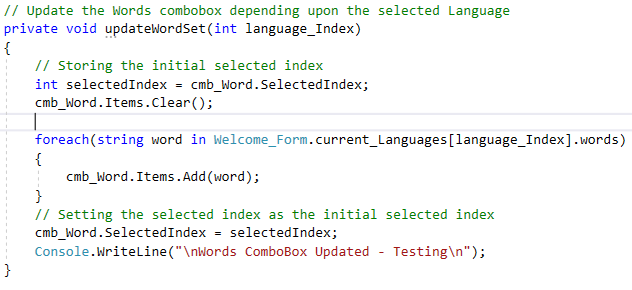


Figure 13.31

The updateWordSet method updates the Word Combo Box (fig 13.13) corresponding to the selected Language. The **current selected index** of the Words ComboBox is stored.

The Words ComboBox is emptied and re-filled with the **selected Language’s Word Set**. The selected Index of the Words ComboBox is set to the previous one, to point to the **same word selected before**, but in a different Language.

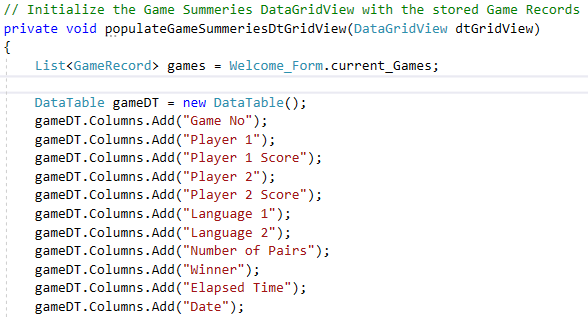


Figure 13.32

The **populateGameSummariesDtGridView** method populates the Game Summaries DataGridView (fig 13.33).

A DataTable is created to be filled with Game Records information. The **Column Headers** were added to the DataTable.



Figure 13.33

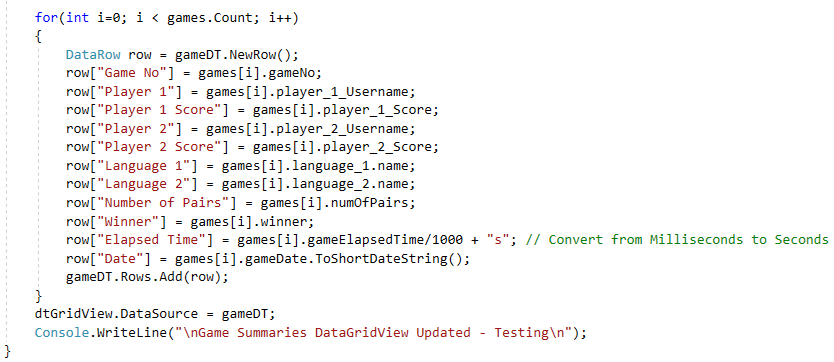


Figure 13.34

For each Game played previously, the Details are added to a row, which is then added to the DataTable.

The DataTable is then set as the **DataSource** for the **Game Summaries DataGridView**.

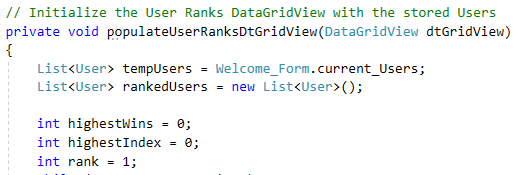


Figure 13.35

The **populateUserRanksDtGridView** method populates the User Ranks table (fig 13.36). The **tempUsers** list temporarily stores the List of existing Users. The **rankedUsers** stores a List of the existing Users, in descending order or Rank, from the best to the Last.

The **highestWins** variable stores the highest number of Wins by a User so far. The **highestIndex** variable stores the Index of the User with the highest number of Wins so far.

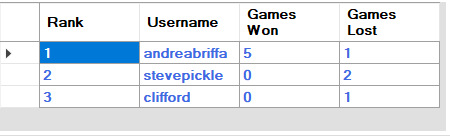


Figure 13.36

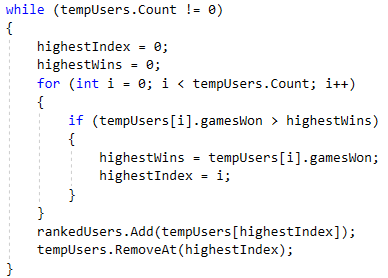


Figure 13.37

To place the Users in the **rankedUsers** List, a loop is carried out, which at every iteration finds out the User with the highest Number of wins within the **tempUsers** List.

The User with the Highest number of wins is added to the **rankedUsers** List, and removed from the **tempUsers** List. The loop iterates until the **tempUsers** List is empty.

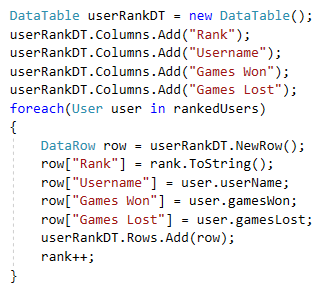


Figure 13.38

A DataTable was created, which will display the User Ranks information. The Column Headers were added to the DataTable.

For each User within the **rankedUsers** List, the User’s details are added to a Row, and the Row is added to the **DataTable**. The rank value is then incremented.

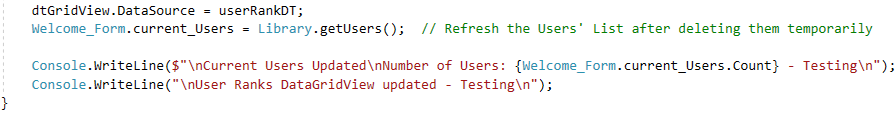


Figure 13.39

The DataTable is then set as the DataSource of the DataGridView, displaying the User Ranks details. (fig 13.36)

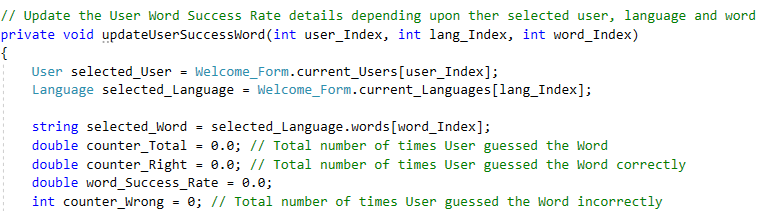


Figure 14.40

The updateUserSuccessWord method updates the User’s SuccessWord details, depending upon the User, Language and Word selected, which are passed as arguments.

The **counter\_Total** variable keeps track of the number of times the User guessed the selected Word successfully. The **counter\_Right** and **counter\_Wrong** variables store the amount of times the User guessed the word correctly and incorrectly respectively.



Figure 14.41

All of the User’s first and second words were checked, to gather the number of times the User attempted to Guess the Word.

If the User guessed the Word correctly, the **counter\_Right** value is incremented, and if the User did not guess the Word correctly, the **counter\_Wrong** value is incremented.



Figure 14.42

The last part of the method is to display the information to the respective Labels.

If the User had not guessed any word pairs which included the selected Word, the Success Rate of the User is set directly to **0%**.

If the User had guessed pairs which consisted of the selected Word, the Success Rate of the User with the selected Word is calculated by dividing the total number of times the User guessed the particular Word by the number of times the User guessed the selected Word correctly.

## **Log\_In Class**

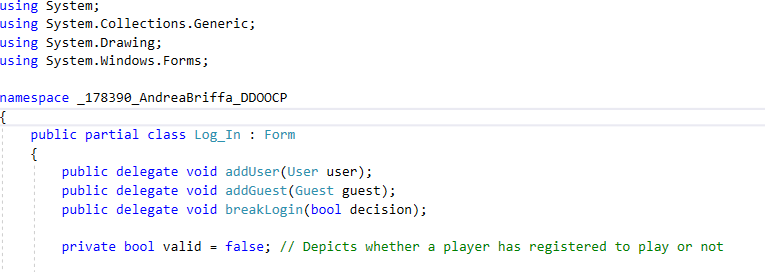


Figure 15.1

The Log In Form serves as a way for a User to Log into the User Account Portal, or Log In as a User when playing a Game.

The delegates are used for callback methods, which pass the details onto the Welcome Form when a User logged in as a Player.

The valid variable determines whether the User or Guest has logged in or not. This affects the Form Closing event, as if the Log In form is closing and no User or Guest has logged in means that the User/Guest is attempting to close the Account Portal login/Game Setup.

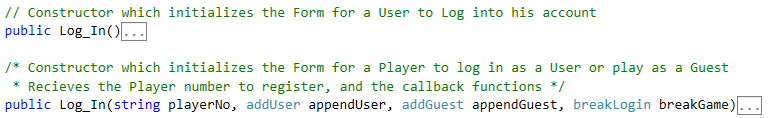


Figure 15.2

The Log In Class contains two Constructors, one responsible for Setting the Form for Users to Log Into their Account, and another for allowing Players to Log In as Users or Guests to take part in a Game.

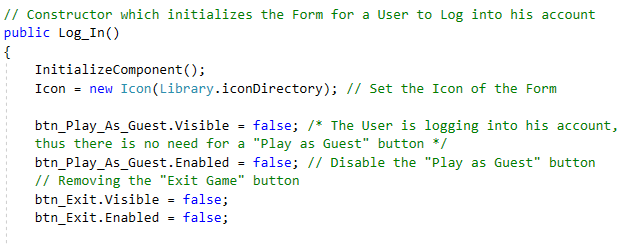


Figure 15.3

The First Constructor (Empty Constructor) is responsible for Setting Up the Log In Form for Users to Log Into their Accounts, viewing their personal Details.

The Form Icon is set to the Icon within the specified directory.

The unnecessary buttons are removed from the User Interface, as the scope of this Form is for Users to Log into their account.



Figure 15.4

When the Password Checkbox is selected/deselected and event handler is triggered which obfuscates/shows clearly the text inside the password TextBox passed as arguments to the **passCheckBox** method.

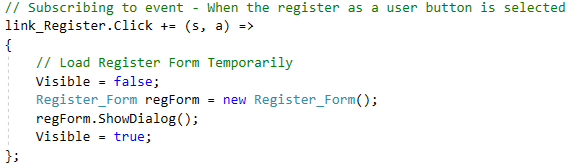


Figure 15.5

When the Register Link is clicked, an instance of Register\_Form is created and displayed for Players to Register as Users. The Log In form is temporarily hidden whilst the Register Form is Visible.

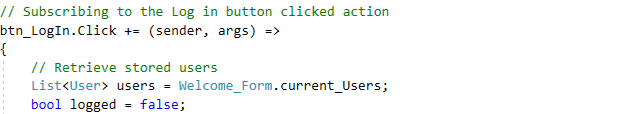


Figure 15.6

When the Log In button is clicked, an Event Handler is executed which checks it the inputs match a User’s.

The **logged** variable determines whether a User has logged in or not.

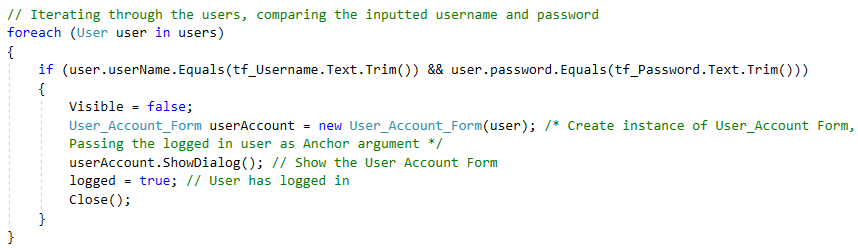


Figure 15.7

For each existing User, it is checked whether the inputted details match. If so, the User’s account Form is created and displayed. The Log In Form is then closed.

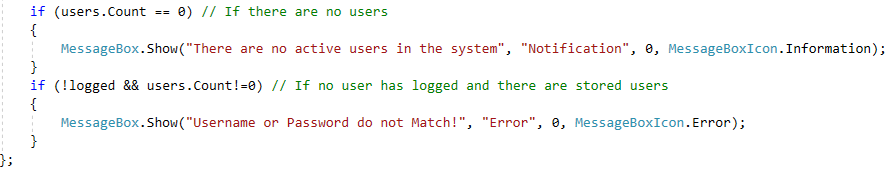


Figure 15.8

If there are no Users registered or the inputs do not Match any User’s details, a corresponding **Alert** message is displayed to the Player attempting to **Log In**.

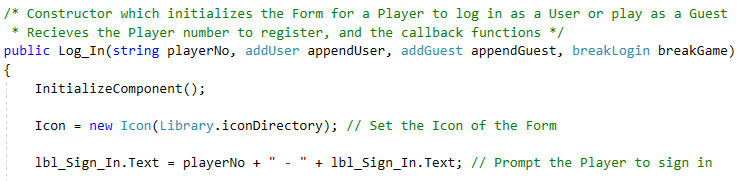


Figure 15.9

The secondary constructor is responsible for laying out the Log In Form for the Player to Log In as a User to Play the Game.

The constructor takes in multiple arguments, including the player number which identifies if Player 1 or Player 2 is logging in. **appendUser**, **appendGuest** and **breakGame** are callback methods, which pass information back to the **Welcome Form**.

The Form’s Icon is set to the Icon within the specified Directory. The Sign In Label is updated corresponding to the Player Number (fig 15.10).



Figure 15.10



Figure 15.11

When the **Play as Guest** button is clicked, an event handler is triggered which asks the Player for verification in order to play as a Guest. If the Player registers as a Guest, a new Guest instance is created which passes it to the List of Playing Guests in the **Welcome Form**. The Log In Form is then closed.

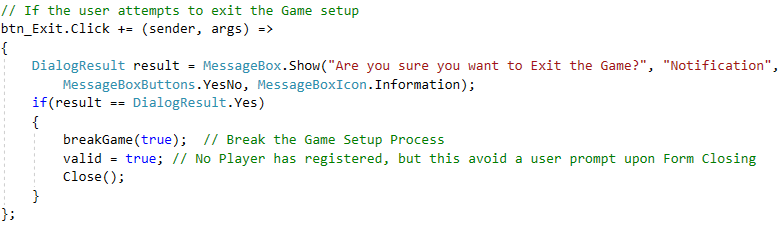


Figure 15.12

When the **Exit Game** button is clicked, Player verification is required as that would mean terminating the Game Setup Process. If the Player verifies to do so, the Log In Form is closed, and the Game Setup process is terminated.

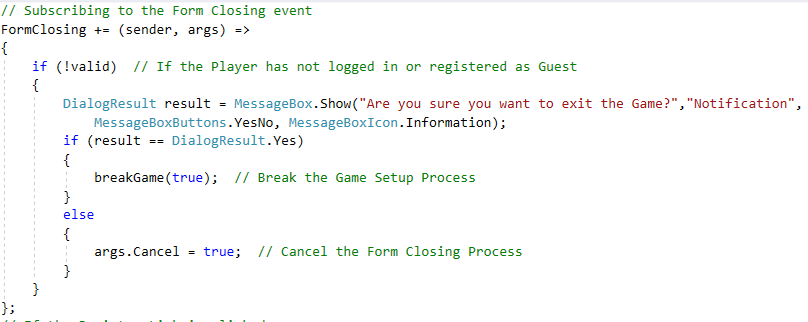


Figure 15.13

When the Log In form is closing, if valid is **false**, meaning the Player did not Log In as a User or Guest, verification is required as the Player would be cancelling the Game Setup Process.

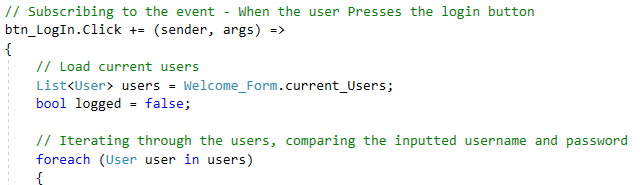


Figure 15.14

The **Log In** button is clicked for the Player to Login as a User. When the button is clicked, the username and password are checked whether they match an existing User’s details.

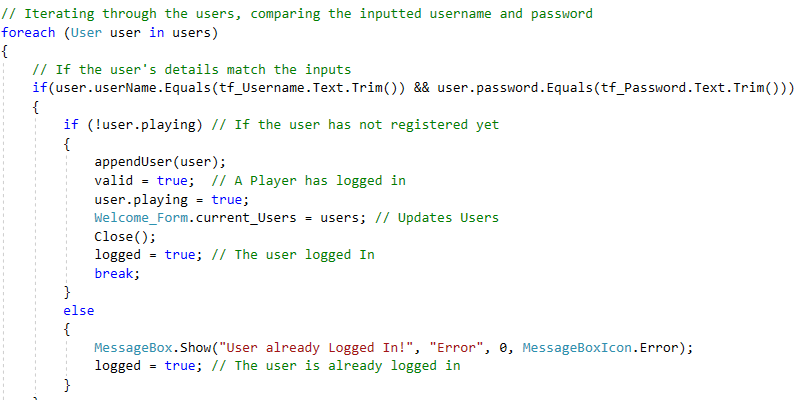


Figure 15.15

If the inputted username and password match an existing User’s details, it is checked whether the User has already registered to play or not. If the User is not currently registered as **playing**, the User is appended to the List of Playing Users. The User’s playing state is then set to **playing**. The loop is then broken, and the Log In Form is closed. Whether the User has already been registered as **playing** or not, **logged** is set to **true**, as the username and password still matched.

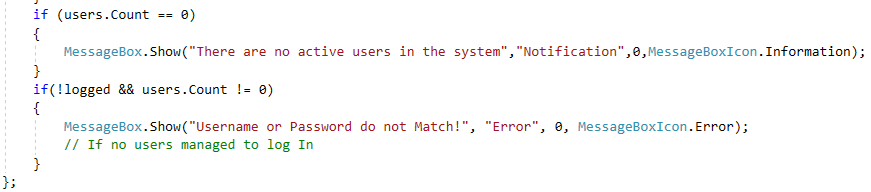


Figure 15.16

If there are no existing Users or the inputs did not match, a corresponding alert message is displayed to the Player attempting to Log In.

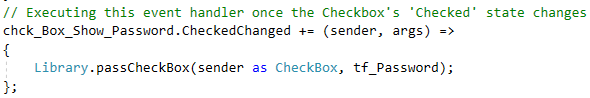


Figure 15.4

When the Password Checkbox is selected/deselected and event handler is triggered which obfuscates/shows clearly the text inside the password TextBox passed as arguments to the **passCheckBox** method.

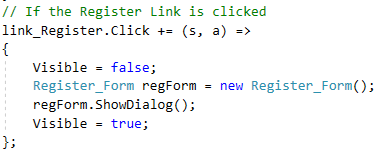


Figure 15.5

When the Register Link is clicked, an instance of Register\_Form is created and displayed for Players to Register as Users. The Log In form is temporarily hidden whilst the Register Form is Visible.

## **Modify\_Word\_Sets Class**

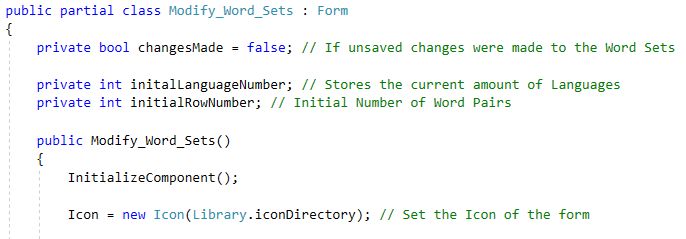


Figure 16.1

The Modify Word Set Form is responsible for allowing the Users to modify the existing Word Sets (fig 16.2).

The **changesMade** variable keeps track whether changes have been made to the Word Sets or not. The Icon of the Form was set to the Icon within the specified Directory.

The **initialLanguageNumber** and **initialRowNumber** variables store the initial amount of Languages and Rows (word pairs) respectively.



Figure 16.2

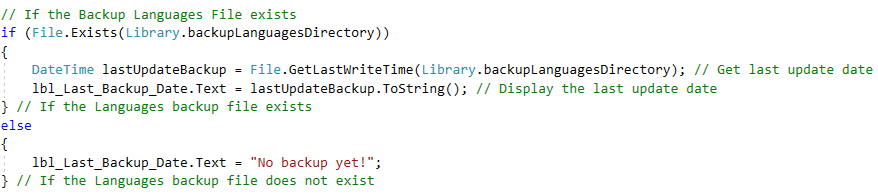


Figure 16.3

If the Backup Languages file exists, the Label which states the last Backup date is updated. If the Backup Languages file does not exist, the Backup Label states that no Backup has occurred to Date.

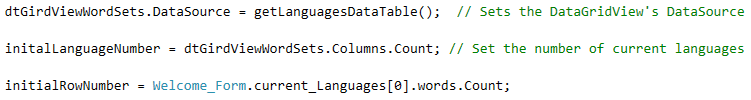


Figure 16.4

The Word Sets DataGridView is populated by updating its’ DataSource, calling the **getLanguagesDataTable** function.

The **initialLanguageNumber** and **initialRowNumber** (number of word pairs) variables are initialized respectively.

As shown in figure 16.5, the **initialLanguageNumber** and **initialRowNumber** variables avoid triggering **changesMade**. Else **changesMade** would be triggered when the DataGridView is being populated initially (on **Form Loading**), which does not count as **changesMade**.

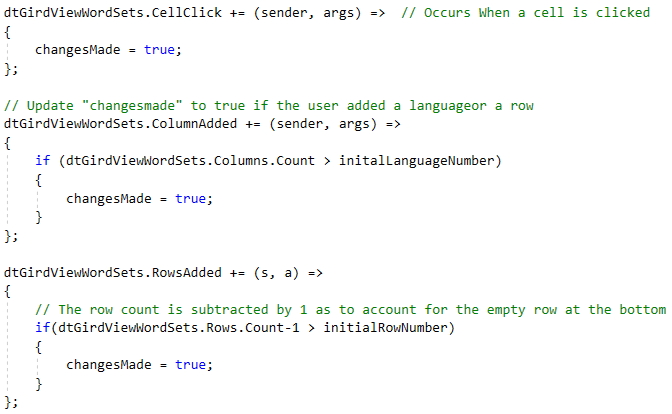


Figure 16.5

If a Cell is clicked, **changesMade** is set to **True**.

If a row is added and the number of total rows exceeds the Number of initial Words per Language, **changesMade** is set to **True**.

If a Language (Column) is added, and the total number of Languages exceed the initial number of Languages, **changesMade** is set to **True**.

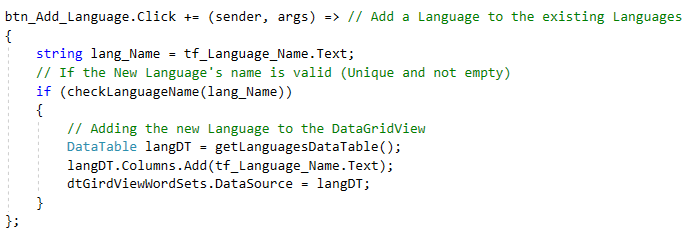


Figure 16.6

If the **Add Language** button is clicked, the new Language’s name typed within the Language name TextBox (fig 16.7) is first checked. If the inputted Language name is valid, the new Language is added to the DataGridView.

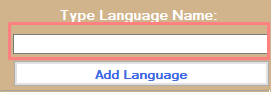


Figure 16.7

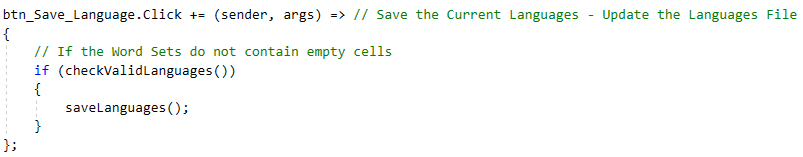


Figure 16.8

When the User presses the **Save Languages** button, if the Languages and their Word Sets are valid (contain no empty cells), the Languages are saved by calling the **saveLanguages** method.

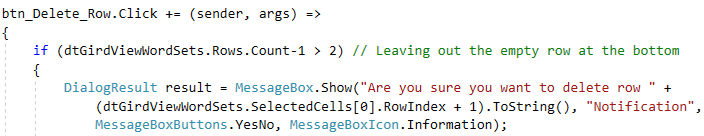


Figure 16.9

When the User selects to **Delete** the selected Row, if the number of Remaining rows that contain Word Pairs is **higher than 2**, a Message prompts the User to verify the Deletion of the Highlighted row.

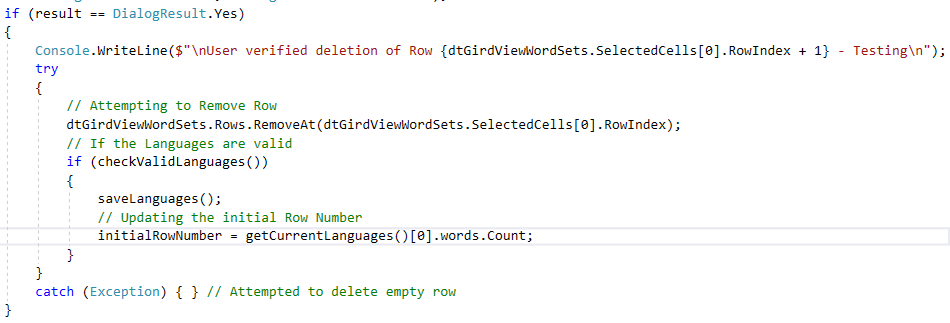


Figure 16.10

If the User verified to delete the Row, the Row is removed from the DataGridView. If the Word Sets within the DataGridView are currently valid, the Languages File is updated and so is the initial number of Rows.

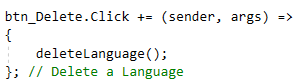


Figure 16.11

If the **Delete Language** button is pressed, the **deleteLanguage** method is called.

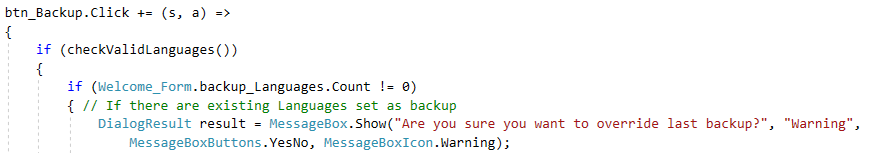


Figure 16.12

When the **Backup Languages** button is clicked, it is checked whether the current Languages present in the DataGridView are valid. If so, if there are current Languages Backed up, the User is prompted to verify if s/he wants to override the current Languages Backup.

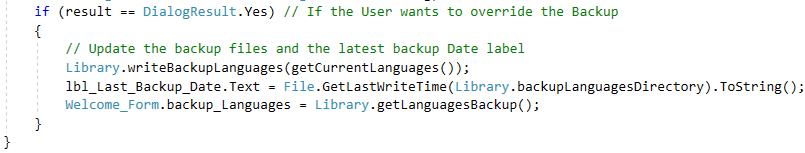


Figure 16.13

If the User verifies to override the Last Backup, the Backup Languages File is overwritten with the current Languages. The last Backup Date label is updated. The current List of Backup Languages is also updated.

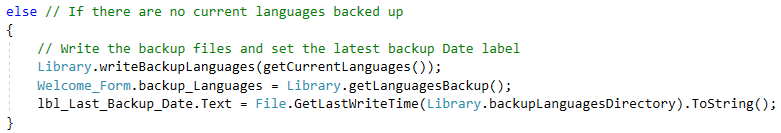


Figure 16.14

If there are no Languages currently backed up User verification is not required. The Backup Languages File is overwritten with the current Languages. The last Backup Date label is updated. The current List of Backup Languages is also updated.

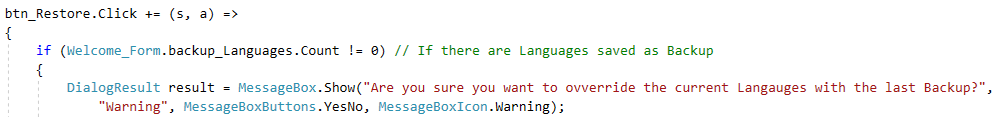


Figure 16.15

If the Restore Backup Languages button is clicked, if there are Languages currently backed up, the User is prompted to verify the replacement of the current Languages with the Backed-up Languages.

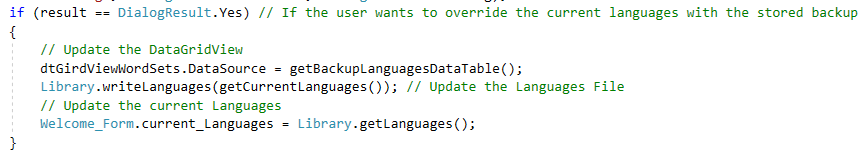


Figure 16.16

If the User verified to replace the Current Languages with those Backed up, the **current List of Languages**, the **WordSetDataGridView** and **Languages File** are updated.

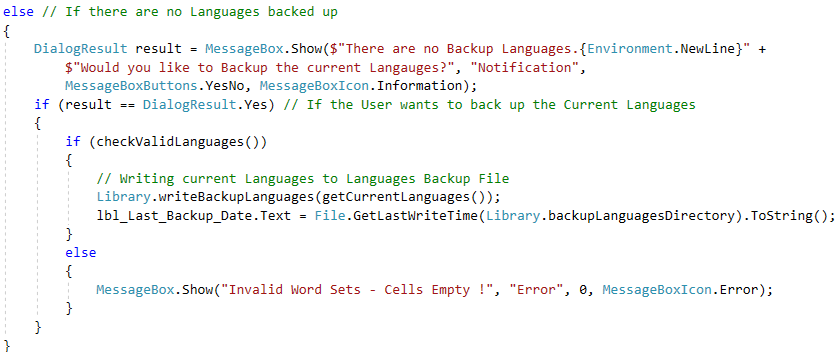


Figure 16.7

If there are no Languages saved as Backup, the User is prompted if s/he wishes to save the current Languages as Backup. If the User verifies to do so, and if the current Languages are valid, The Backup Languages File is updated with the current Languages, and the Last Backup Date Label is updated.

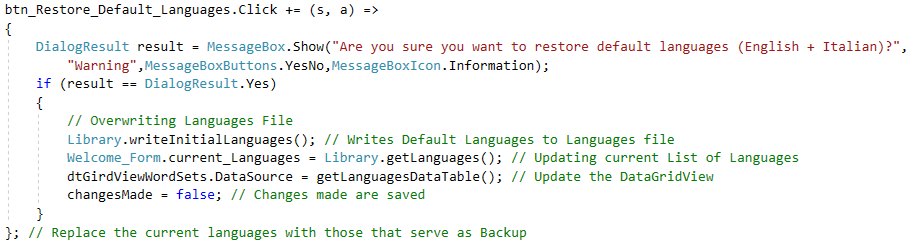


Figure 16.18

If the User selects the **Restore Default Languages** button to replace the current Languages with the Default Languages, User verification is required to do so. If the User verifies to do so, the Languages File and current Languages are updated corresponding to the **default Languages**. The DataGridView is also updated, and **changesMade** is set to **false**.



Figure 16.19

When the **Modify Word Set Form** is closing, it is checked whether there are any **changesMade**. If so, the User is prompted is s/he wished to save the changes made.

If the User agrees to do so, and if the Languages are valid, the current Languages are saved to the Languages File. If the Languages are not valid, the Form closing event is cancelled.

The Form Closing event is also cancelled if the User decided to cancel the Form Closing event. If changesMade was False initially, the Form Closing event would pursue.

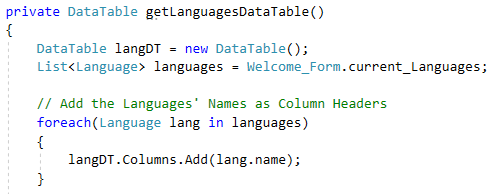


Figure 16.20

The **getLanguagesDataTable** function returns the DataTable of the current List of Languages, to be set as the **DataSource** of the **DataGridView**. For each Language, the Language’s name is added as a Column Header.

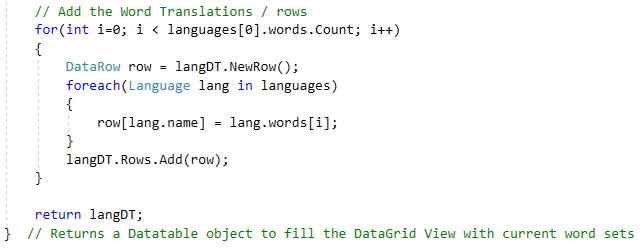


Figure 16.21

After adding the **Column Headers**, for each word inside a Language, a row is created containing the same word in different languages. Each **row** is added to the **DataTable**.

The DataTable is then returned.

The same procedure is followed for the **getBackupLanguagesDataTable**, which returns the DataTable of the Backup Languages, with the exception of loading the Backup Languages instead of the Current Languages.

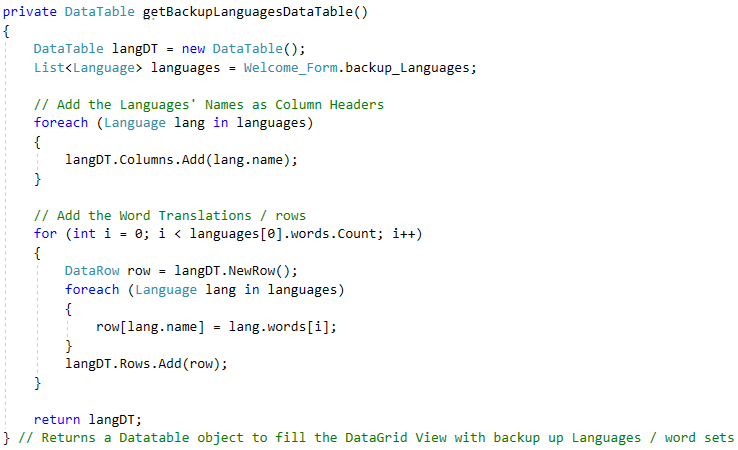


Figure 16.22

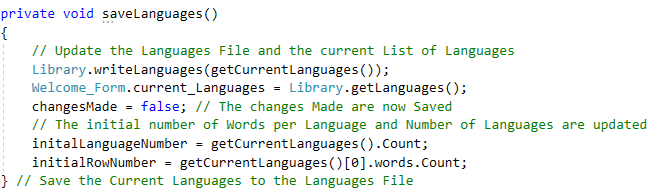


Figure 16.23

The **saveLanguages** method saves the Languages currently present within the DataGridView to the **Languages File**. The current Languages are also updated. **changesMade** is set to **False**, since all changes made are now saved.

The initial Language number and initial Row number values are also updated.

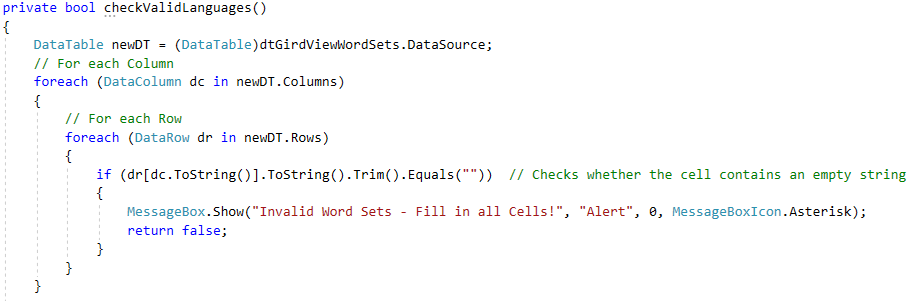


Figure 16.24

The checkValidLanguages function checks whether the Word Sets inside the DataGridView are valid or not.

The first validation occurs in checking every cell of the DataGridView, ensuring that no cell is empty. This is carried out by for each column, and each row, each Cell’s text is checked if it is equal to no text ( “” ). If an empty Cell is found, **false** is returned.

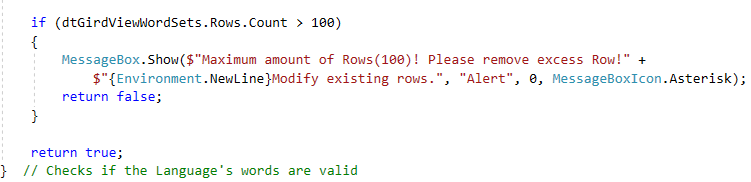


Figure 16.25

If all the cells are valid, the number of Rows is checked if it exceeds 100. If so, **false** is returned.

If all the cells within the DataGridView are valid, and the number of Rows does not exceed 100, **true** is returned, signifying that the Languages inside the DataGridView are valid.

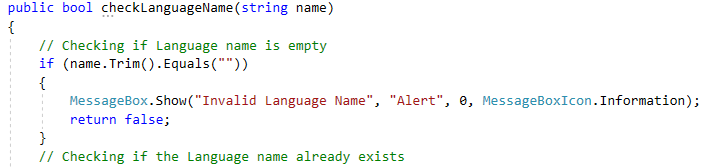


Figure 16.26

The checkLanguageName function determines whether the Language’s name passed as an argument is valid for a new Language’s name or not.

If the name does not contain any valid characters, **false** is returned.

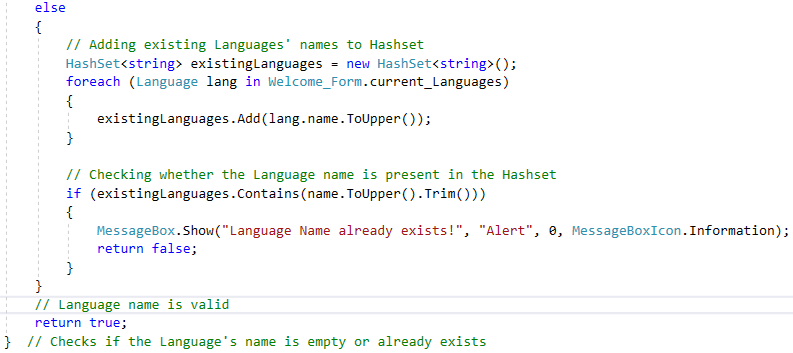


Figure 16.27

If the name consists of valid characters, it is checked whether the Language’s name already exists. This is done by creating a Hashset, which stores all of the existing Languages’ names. The new Language’s name is then compared against the Hashset, checking if the Hashset contains the new Language’s name.

If the Hashset returns **false**, the Language Name is **valid**, thus **true** is returned.

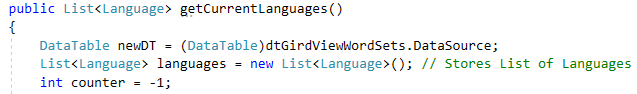


Figure 16.28

The **getCurrentLanguages** method is responsible for returning the List of Languages that are present inside the Word Sets DataGridView.

A new DataTable is created, holding the DataTable of the DataGridView.

A Languages List is created to store the Languages derived from the DataGridView.

The **counter** variable will aid the iteration when adding the Words to their pertinent Language.

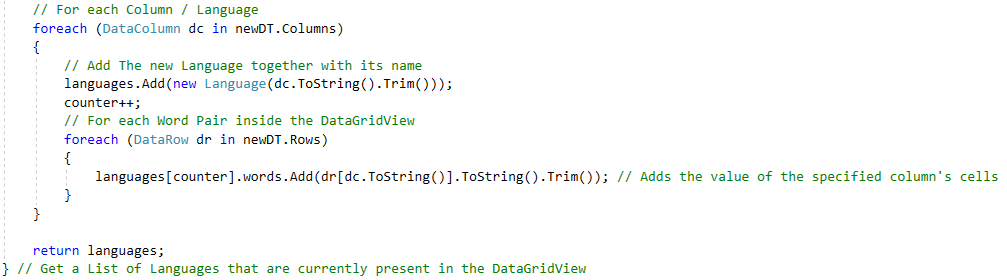


Figure 16.29

For each Language within the DataTable, the Language is added to the **Languages List**. For each Language, each pertinent word is added to the Language’s pertinent words.

After adding all of the Languages and pertinent Word Sets, the List of Languages is returned.

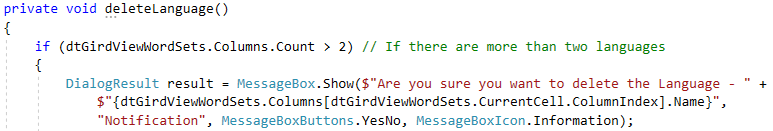


Figure 16.30

The deleteLanguage method is responsible for deleting the Selected Language. Deletion of a Language is only permitted when the current amount of Languages is three (3) or higher. If so, the User is prompted to verify the Deletion of the Language.

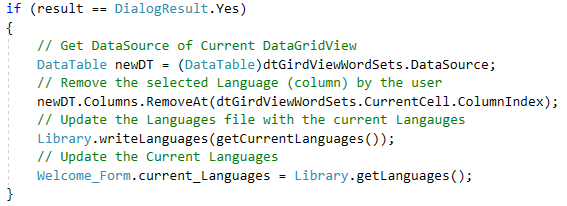


Figure 16.31

If the User did verify the deletion of the Language, the Language is removed from the DataGridView. The Languages File is updated, and so is the Current List of Languages.

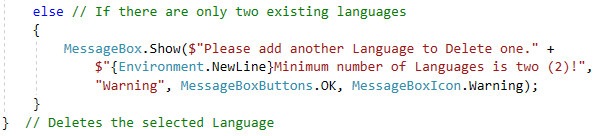


Figure 16.32

If there are only two (2) current existing Languages, the User is given a message indicating an error.

# **Task 2**

## **White Box Testing**

**Program Class** (Starts Application)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1 | Main() | Application Starts | User runs application | Program continues execution | Program continues execution | Pass | Null |  |
| 2 | Main() | Application Terminates | User closes **Welcome\_Form** Form | Program is terminated | The program is terminated | Pass | Null |  |
| 3 | Main() | **Welcome\_Form** instance created | **Welcome\_Form** instance created | **Welcome\_Form** constructor writes Console statement | **Welcome\_Form** constructor wrote Console statement | Pass | Null |  |

**Welcome\_Form Class** (Used as the Main Menu which leads to doing various types of operations. Validation is more likely to occur in other classes)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1 | Welcome\_Form() > VisibleChanged Event | Form set Visible indicator | Form constructor executed | Form becomes visible | Form is visible | Pass | Null |  |
| 2 | Welcome\_Form() | Set Form Icon | Form Constructor run | The Form’s icon is updated | The Form’s icon was updated | Pass | Null |  |
| 3.1 | Welcome\_Form() > **btn\_LogIn**.Click Event | Creating a new instance of **Log\_In** class | User clicks **Log\_In** button | New instance of **Log\_In** class is created | A new instance of **Log\_In** class was created | Pass | Null |  |
| 3.2 | Welcome\_Form() > **btn\_LogIn**.Click Event | Setting the **Welcome\_Form** invisible | User clicks **Log\_In** button | **Welcome\_Form** becomes invisible | **Welcome\_Form** is invisible | Pass | Null |  |
| 3.3 | Welcome\_Form() > **btn\_LogIn**.Click Event | Setting the **Welcome\_Form** visible again | User closes **Log\_In** instance | **Welcome\_Form** becomes visible | **Welcome\_Form** is visible | Pass | Null |  |
| 4.1 | Welcome\_Form() > **btn\_Create\_**  **Account.Click** Event | Creating a new instance of **Register\_Form** class | User clicks **Register** button | New instance of **Register\_Form** class is created | A new instance of **Register\_Form** class was created | Pass | Null |  |
| 4.2 | Welcome\_Form() > **btn\_Create\_**  **Account.Click** Event | Setting the **Welcome\_Form** invisible | User clicks **Register** button | **Welcome\_Form** becomes invisible | **Welcome\_Form** is invisible | Pass | Null |  |
| 4.3.1 | Welcome\_Form() > **btn\_Create\_**  **Account.Click** Event | Setting the **Welcome\_Form** visible again | User closes the **Register\_Form** instance | **Welcome\_Form** becomes visible | **Welcome\_Form** remained invisible | Fail | The Welcome Form wasn’t visible after the Register Form Closed | *// No Output* |
| 4.3.2 | Welcome\_Form() > **btn\_Create\_**  **Account.Click** Event | Setting the **Welcome\_Form** visible again | User closes the **Register\_Form** instance | **Welcome\_Form** becomes visible | **Welcome\_Form** remained invisible | Pass | A statement was required to change Welcome Form’s Visibility to **true** |  |
| 5.1 | Welcome\_Form() > **btn\_Modify\_**  **Word\_Sets.Click** Event | Creating a new instance of **Modify\_Word**  **\_Sets class** | User clicks **Modify\_Word**  **\_Sets** button | New instance of **Modify\_Word**  **Sets** class is created | A new instance of **Modify\_Word**  **\_Sets** is created | Pass | Null |  |
| 5.2 | Welcome\_Form() > **btn\_Modify\_**  **Word\_Sets.Click** Event | Setting the **Welcome\_Form** invisible | User clicks **Modify\_Word**  **\_Sets** button | **Welcome\_Form** becomes invisible | **Welcome\_Form** is invisible | Pass | Null |  |
| 5.3 | Welcome\_Form() > **btn\_Modify\_**  **Word\_Sets.Click** Event | Setting the **Welcome\_Form** visible again | User closes the **Modify\_Word**  **\_Sets** instance | **Welcome\_Form** becomes visible | **Welcome\_Form** is visible | Pass | Null |  |
| 6.1 | Welcome\_Form() > **btn\_Statistics**  **.Click** Event | Creating new instance of **Statistics** class | User clicks **Statistics** button | New instance of **Statistics\_Form** is created | A new instance of **Statistics\_Form** is created | Pass | Null |  |
| 6.2 | Welcome\_Form() > **btn\_Statistics**  **.Click** Event | Setting the  **Welcome\_Form** invisible | User clicks **Statistics** button | **Welcome\_Form** becomes invisible | **Welcome\_Form** is invisible | Pass | Null |  |
| 6.3 | Welcome\_Form() > **btn\_Statistics**  **.Click** Event | Setting the **Welcome\_Form** visible again | User closes the **Statistics\_Form** instance | **Welcome\_Form** becomes visible | **Welcome\_Form** is visible | Pass | Null |  |
| 7.1 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** Event | Create List of Users, and initialize with list of existing Users | User clicks **Start\_Game** button | List of existing Users is created | A List of the existing users was created | Pass | Null |  |
| 7.2 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** Event | Set the **playing** property to **false** for all registered Users | User clicks **Start\_Game** button | All Users’ **playing** property set to **False** | The **playing** property was set to **False** for all Users | Pass | Null |  |
| 7.3 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** Event | Setting the **Welcome\_Form** invisible | User clicks **Start\_Game** button | **Welcome\_Form** will become invisible | The **Welcome\_Form** is invisible | Pass | Null |  |
| 7.4 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** Event | **Log\_In** instance created for Player 1’s login, provided with corresponding arguments | User clicks **Start\_Game** button | **Log\_In** instance created | **Log\_In** instance is created | Pass | Null |  |
| 7.5 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** Event | Game setup cancelled from player1’s **Log\_In** form | Player1 exits game Setup | **breakSetup** property is set to true | The **breakSetup** property was set to true | Pass | Null |  |
| 7.6 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break Game setup following **Player1** login | Player1 does not terminate game setup - **breakSetup** = **false** | Game setup proceeds and does not terminate | Game setup proceeded and did not terminate | Pass | Null |  |
| 7.7 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break Game setup following **Player1** login | Player1 terminates game setup - **breakSetup** = **true** | Game setup terminates. The flow  of executes continues at the **GoTo** destination | Game setup is terminated | Pass | Null |  |
| 7.8 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | **Log\_In** instance created for Player 2’s login, provided with corresponding arguments | User clicks **Start\_Game** button | **Log\_In** instance created | **Log\_In** instance is created | Pass | Null |  |
| 7.9 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break Game setup following **Player2** login | Player2 does not terminate game setup - **breakSetup** = **false** | Game setup  proceeds and does not terminate | Game setup proceeded and did not terminate | Pass | Null |  |
| 7.10 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break Game setup following **Player2** login | Player2 terminates game setup - **breakSetup** = **true** | Game setup terminates. The flow  of executes continues at the **GoTo** destination | Game setup is terminated | Pass | Null |  |
| 7.11 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | **Game\_Settings\_Form** instance created, given appropriate arguments | User clicks **Start\_Game** button | **Game\_Settings\_Form** instance created | An instance of **Game\_Settings\_Form** was created | Pass | Null |  |
| 7.12 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break game setup following **Game\_Settings** form closed | Players did not terminate game Setup – **breakSetup** = **false** | Game setup proceeds and does not terminate | Game setup proceeded and did not terminate | Pass | Null |  |
| 7.13 | Welcome\_Form() > **btn\_Start\_Game**  **.Click** | Break game setup following **Game\_Settings** form closed | Players did not terminate game Setup – **breakSetup** = **true** | Game setup terminates. The flow of executes continues at the **GoTo** destination | Game setup is terminated | Pass | Null |  |
| 7.14 | Welcome\_Form () > **btn\_Start\_Game**  **.Click** | Create new instance of **Game** class, giving current arguments | User clicks **Start\_Game** button | New instance of **Game** created | A new instance of **Game** was created | Pass | Null |  |
| 7.15 | Welcome\_Form () > **btn\_Start\_Game**  **.Click** | Empty Guest Players List | User clicks **Start\_Game** button | List of Guest players empty | The Guest Players’ List was cleared (empty) | Pass | Null |  |
| 7.16 | Welcome\_Form () > **btn\_Start\_Game**  **.Click** | Empty Users Playing List | User clicks **Start\_Game** button | List of Users Playing empty | The Users Playing List was cleared (empty) | Pass | Null |  |
| 7.17 | Welcome\_Form () > **btn\_Start\_Game**  **.Click** | **Welcome\_Form** set to visible again | User clicks **Start\_Game** button - Game setup is finished | **Welcome\_Form** visible | The **Welcome\_Form** was set to visible | Pass | Null |  |
| 8.1 | appendUser (User) | Append User passed as argument to existing logged (playing) Users | **Input:**  **User reference** | User passed as argument appended to existing logged Users | The user was appended to the existing logged Users | Pass | Null |  |
| 9.1.1 | appendGuest (Guest) | Append Guest passed as argument to playing guests | **Input:**  **Guest reference** | Guest added to List of Guests logged in to play | The guest passed as an argument was not added Successfully | **Fail** | The Guest was being initialized again |  |
| 9.1.2 | appendGuest (Guest) | Append Guest passed as argument to playing guests | **Input:**  **Guest reference** | Guest added to List of Guests logged in to play | The guest was added to the List of Guests playing | Pass | The Guest passed as an argument was not initialized again |  |
| 10.1 | appendSettings (int, Language, Language) | Setting the number of button pairs the game is to consist of | ***Input*:**  **7 / Language reference / Language reference** | The **numOfPairs** property is initialized | The **numOfPairs** property was initialized | Pass | Null |  |
| 10.2 | appendSettings (int, Language, Language) | Initializing the First language to be used in the game according to the first Language passed as argument | ***Input*:**  **7 / Language reference / Language reference** | The **lang1** property is initialized | The **lang1** property was initialized | Pass | Null |  |
| 10.3 | appendSettings (int, Language, Language) | Initializing the Second language to be used in the game according to the second Language passed as argument | ***Input*:**  **7 / Language reference / Language reference** | The **lang2** property is initialized | The **lang2**  property was initialized | Pass | Null |  |
| 11.1 | breakGame(bool) | Setting **Boolean** value which later when checked will skip the game setup | ***Input*:**  **true** | The **breakSetup** property is set to true | The **breakSetup** property was set to true | Pass | Null |  |
| 12.1 | Welcome\_Form () > **FormClosed** Event | Form Closed Indicator | **Welcome\_Form** closed | Console Statement indicating **Welcome\_Form** Closed | A Console statement was written, indicating that the **Welcome\_Form** was closed | Pass | Null |  |

**Modify** **Word Set Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1 | Modify\_Word\_Sets () | Updating Form Icon | Class Constructor Executed | Form Icon Updates | The Form Icon was updates | Pass | Null |  |
| 2.1 | Modify\_Word\_Sets () | Checking if Languages Backup File exists in specified directory | File does not exist in specified Directory | Console Message indicating that no the file does not exist | A console Message was printed indicating that the File does not exist | Pass | Null |  |
| 2.2 | Modify\_Word\_Sets () | Initializing Backup Label depending on last languages backup date | Backup File does not exist in specified Directory | Backup Date Label stating there is no Backup yet | The Label stated that there was no backup to date | Pass | Null |  |
| 2.3 | Modify\_Word\_Sets () | Checking if Languages Backup File exists in specified directory | File exists in specified Directory | Console Message indicating that the file exists | A console Message was printed indicating that the File exists | Pass | Null |  |
| 2.4.1 | Modify\_Word\_Sets () | Fetching and storing last backup date | File exists in specified Directory | Successful Retrieval of Last File Write Date | The current Date was retrieved | **Fail** | The Current Date was being Fetched |  |
| 2.4.2 | Modify\_Word\_Sets () | Fetching and storing last backup date | File exists in specified Directory | Successful Retrieval of Last File Write Date | The file was retrieved successfully | Pass | The Last write time of the File was retrieved |  |
| 2.5 | Modify\_Word\_Sets () | Initializing Backup Label depending on last languages backup date | File exists in specified Directory | Console Message indicating that the text of the label has been set as the last backup date | A console Message was printed stating that the label’s text was set to the last backup date | Pass | Null |  |
| 3.1 | Modify\_Word\_Sets () | Setting the Data Source of the DataGridView | Class Constructor Executed | DataGridView’s Data Source initialized | The Data Source of the DataGridView was initialized | Pass | Null |  |
| 3.2 | Modify\_Word\_Sets () | Storing number of current Languages | Class Constructor Executed | The number of current languages stored | The number of Current Languages was stored | Pass | Null |  |
| 4.1 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.Click()** | Record that the User has made changes to the Word Sets | Modify Set Form Loaded | No changes registered | No message was printed indicating that the User made changes | Pass | Null | *No message was printed* |
| 4.2 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.Click()** | Record that the User has made changes to the Word Sets | A cell within the Word Sets DataGridView was selected | Message printed indicating changes made | A message was printed every time a cell was selected | Pass | Null |  |
| 5.1 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **ColumnAdded ()** | Record that the User added a new Language | Modify Set Form Loaded | No changes registered | No message was printed indicating that the User made changes | Pass | Null | *No message was printed* |
| 5.2 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **ColumnAdded ()** | Record that the User added a new Language | User added new Language *(added new Column)* | Message printed indicating changes made | Message was printed indicating changes made when new language was created | Pass | Null |  |
| 5.3 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **ColumnAdded ()** | Record that the User added a new Language | User deleted Language *(removed Column)* | No changes registered | No message was printed indicating that the User made changes | Pass | Null | *No message was printed* |
| 6.1 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **RowsAdded ()** | Record that the User added a new row | Modify Set Form Loaded | No changes registered | Multiple messages printed indicating changes made | **Fail** | The cause of this bug was that the rows being added to the datagridview initially were being registered as User changes |  |
| 6.1.1 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **RowsAdded ()** | Record that the User added a new row | Modify Set Form Loaded | No changes registered | No message propmts indicating changes made | Pass | Null | The bug was solved by checking whether the number of rows is more than the initial number of rows  *No message was printed* |
| 6.2 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **RowsAdded ()** | Record that the User added a new row | User added new row | Message indicating changes made | Message was printed indicating changes made when new Row was added | Pass | Null |  |
| 6.3 | Modify\_Word\_Sets () >  **dtGirdViewWord**  **Sets.**  **RowsAdded ()** | Record that the User added a new row | User deleted existing row | No changes registered | No message propmts indicating changes made | Pass | Null | *No message was printed* |
| 7.1 | Modify\_Word\_Sets () > **btn\_Add\_Language .Click()** | Checking the new Language name | User entered no text as the new Language’s name  ***Input*** : **“”** | Name Invalid as it is empty | Message printed indicating name is invalid | Pass | Null |  |
| 7.2 | Modify\_Word\_Sets () > **btn\_Add\_Language .Click()** | Checking the new Language name | User entered existing Language’s name as new Language’s name  ***Input***: “**English**” | Name Invalid as Name already exists | Message printed indicating name is invalid | Pass | Null |  |
| 7.3 | Modify\_Word\_Sets () > **btn\_Add\_Language .Click()** | Checking the new Language name | User entered existing Language’s name as new Language’s name, but with different letter casing  ***Input***: “E**nGLiSH**” | Name Invalid as Name already exists | Message printed indicating name is invalid | Pass | Null |  |
| 7.4 | Modify\_Word\_Sets () > **btn\_Add\_Language .Click()** | Checking the new Language name | User entered a unique language name as a new Language’s name  ***Input***: “**French**” | Valid Language name, new Language created | A message was printed indicating that the language name is valid | Pass | Null |  |
| 7.5 | Modify\_Word\_Sets () > **btn\_Add\_Language .Click()** | Checking the new Language name | User entered a unique language name as a new Language’s name  ***Input***: “**French**” | New Language added to the Languages DataGridView | A message was printed that the new Language was added to the DataGridView | Pass | Null |  |
| 8.1 | Modify\_Word\_Sets () > **btn\_Save\_Language.Click()** | Testing Whether the Word Sets within the DataGridView are valid | ***Input***:  **Word sets including empty cells** | Message indicating Word Sets are Invalid | A console message was printed indicating that the word sets are invalid | Pass | Null |  |
| 8.2 | Modify\_Word\_Sets () > **btn\_Save\_Language.Click()** | Testing Whether the Word Sets within the DataGridView are valid | ***Input***:  **Word sets with no empty cells** | Message indicating Word Sets are valid | A console message was printed indicating that the word sets are valid | Pass | Null |  |
| 9.1 | Modify\_Word\_Sets () > **btn\_Save\_Language.Click()** | Checking whether the amount of word Sets left is higher than 2 | ***Input:***  **Amount of Word sets left > 2** | Message indicating amount of Word Sets left is higher than 2 | A Console Message indicating that the amount of Word Sets left is higher than 2 was printed | Pass | Null |  |
| 9.2 | Modify\_Word\_Sets () > **btn\_Save\_Language.Click()** | Checking whether the amount of word Sets left is higher than 2 | ***Input:***  **Amount of Word sets left = 2** | Message indicating minimum amount of Word Sets reached | A console message was printed indicating that the minimum amount of Word Sets was reached | Pass | Null |  |
| 9.3 | Modify\_Word\_Sets () > **btn\_Delete\_Row.Click()** | Testing User verification when deleting a row | ***Input*:**  **Row 41 selected. Delete Row button clicked. When prompted to verify deletion of row, *No was selected*** | Message indicating that the User did not verify the deletion of the Row | A console message was printed indicating that the user did not verify the deletion of the row | Pass | Null |  |
| 9.4 | Modify\_Word\_Sets () > **btn\_Delete\_Row.Click()** | Testing User verification when deleting a row | ***Input*:**  **Row 41 selected. Delete Row button clicked. When prompted to verify deletion of row, *Yes was selected*** | Message indicating that the User did verified the deletion of the Row | A console message was printed indicating that the user verified the deletion of the row | Pass | Null |  |
| 9.5 | Modify\_Word\_Sets () > **btn\_Delete\_Row.Click()** | Deleting selected row from DataGridView | ***Input*:**  **Row 41 selected. Delete Row button clicked. When prompted to verify deletion of row, *Yes was selected*** | Selected Row Deleted | The selected row was deleted | Pass | Null |  |
| 9.6 | Modify\_Word\_Sets () > **btn\_Delete\_Row.Click()** | Deleting selected row from DataGridView | ***Input*:**  **Row 41 selected. Delete Row button clicked. When prompted to verify deletion of row, *Yes was selected*** | Current Languages saved to file | The current Languages were saved | Pass | Null |  |
| 9.7 | Modify\_Word\_Sets () > **btn\_Delete\_Row.Click()** | Updating the initial number of rows | ***Input*:**  **Row 41 selected. Delete Row button clicked. When prompted to verify deletion of row, *Yes was selected*** | Variable storing the number of Rows updated | The variable storing the number of Rows was updated | Pass | Null |  |
| 10.1 | Modify\_Word\_Sets () > **btn\_Delete.Click()** | Verifying that the process of Deleting a Language started | ***Input*:**  **Button Delete Language Clicked** | Message indicating that the process of deleting a language commenced | A Message indicating that the process of deleting a language commenced was printed | Pass | Null |  |
| 11.1 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking whether Word Sets are valid for backup | ***Input*:**  **Word Sets Containing empty cells** | Invalid Languages for Backup | The Languages were invalid for Backup | Pass | Null |  |
| 11.2 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking whether Word Sets are valid for backup | ***Input*:**  **Word Sets containing no empty Cells** | Languages valid for Backup | The Languages were Invalid for Backup | Pass | Null |  |
| 11.3 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking whether the User has already backed up a list of Languages | ***Input*:**  **User has backed up Languages already. Backup File exists** | Message indicating that backup has already took place before | A Message was printed indicating that backup has already took place before | Pass | Null |  |
| 11.4 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | User verifying Backup override | ***Input*:**  **User Verified: Yes** | Message indicating User verified Backup override | A Message was printed indicating that the User verified Backup override | Pass | Null |  |
| 11.5 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking that the Backup file was overridden | ***Input*:**  **User Verified: Yes** | Backup Overridden | The backup was overridden | Pass | Null |  |
| 11.6 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Updating the Label stating the last backup date | ***Input*:**  **User Verified: Yes** | Last Backup Date Label Updated | The Last Backup Date Label has been Updated | Pass | Null |  |
| 11.7 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking whether the User has already backed up a list of Languages | ***Input*:**  **User never backed up Languages. Backup file does not exist** | Message indicating that backup has never took place before | A Message indicating that backup has never took place before | Pass | Null |  |
| 11.8 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Checking that the Backup file was created | ***Input*:**  **User Verified: No** | Backup File created | The backup file was created | Pass | Null |  |
| 11.9 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | Updating the Label stating the last backup date | ***Input*:**  **User Verified: No** | Last Backup Date Label Updated | The Last Backup Date Label has been Updated | Pass | Null |  |
| 11.10 | Modify\_Word\_Sets () > **btn\_Backup.Click()** | User verifying Backup override | ***Input*:**  **User Verified: No** | Message indicating User did not verify Backup override | A Message was printed indicating that the User did not verify Backup override | Pass | Null |  |
| 12.1 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether there are Languages backed up | ***Input*:**  **There are Languages already backed up** | Message indicating that there are already Languages backed up | A message has been printed indicating that there are already Languages backed up | Pass | Null |  |
| 12.2 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Verifying whether User wants to carry out Backup Restoration | ***Input*:**  **User Verified: Yes** | Message indicating User verified backup restoration | A Message was Printed Indicating that the User verified backup restoration | Pass | Null |  |
| 12.3 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether DataGridView has updated | ***Input*:**  **Languages Restored from Backup** | DataGridView updated | The DataGridView has updated | Pass | Null |  |
| 12.4 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether current Languages have been updated | ***Input*:**  **Current Languages present in DataGridView** | Current Languages stored in file updated | The current Languages stored in File have been updated | Pass | Null |  |
| 12.5 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Verifying whether User wants to carry out Backup Restoration | ***Input*:**  **User Verified: No** | Message indicating User did not verify backup restoration | A message has been printed indicating that the User did not verify backup restoration | Pass | Null |  |
| 12.6 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether there are Languages backed up | ***Input*:**  **There are no Languages currently backed up** | Message indicating There are no Languages Backed up | A message has been printed indicating that there are no Languages backed up | Pass | Null |  |
| 12.7 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Verifying whether User wants to Backup Current Languages | ***Input*:**  **User Verified: Yes** | Message indicating User did verify Backup | A message was printed indicating that the User verified to backup the Current Languages | Pass | Null |  |
| 12.8 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether Word Sets are valid for backup | ***Input*:**  **Word Sets Containing empty cells** | Invalid Languages for Backup | The Languages were invalid for Backup | Pass | Null |  |
| 12.9 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether Word Sets are valid for backup | ***Input*:**  **Word Sets containing no empty Cells** | Languages valid for Backup | The Languages were Invalid for Backup | Pass | Null |  |
| 12.10 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether Languages have been backed Up | ***Input*:**  **Current Languages from DataGridView** | Successful Backup of Languages | Successful Backup of Languages | Pass | Null |  |
| 12.11 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether label stating Last Backup Date has been Updated | ***Input:***  **Last Backup Date** | Message indicating label stating last Backup Date has been updated | The label did not change its’ ***text*** at all | **Fail** | Nothing was being done to do so | *No message was printed* |
| 12.11.1 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Checking whether label stating Last Backup Date has been Updated | ***Input:***  **Last Backup Date** | Message indicating label stating last Backup Date has been updated | The label did not change its’ ***text*** at all | Pass | A statement was written to update the label right after Backup |  |
| 12.12 | Modify\_Word\_Sets () > **btn\_Restore.Click()** | Verifying whether User wants to Backup Current Languages | ***Input*:**  **User Verified: No** | Message indicating User did not verify backup | A message was printed indicating that the User did not verify to backup the Current Languages | Pass | Null |  |
| 13.1 | Modify\_Word\_Sets () > **btn\_Restore\_Default\_**  **Languages.Click()** | User Verification for Restoring Default Languages | ***Input*:**  **User Verified: Yes** | Message indicating User verified Default Language Restoration | A Message was printed indicating that the User verified the Default Languages Restoration | Pass | Null |  |
| 13.2 | Modify\_Word\_Sets () > **btn\_Restore\_Default\_**  **Languages.Click()** | Default Languages written to Languages File | Input:  Default Languages | Default Languages written to Languages File | The Default Languages were written to the file | Pass | Null |  |
| 13.3 | Modify\_Word\_Sets () > **btn\_Restore\_Default\_**  **Languages.Click()** | DataSource initialized with default Languages | Input:  Default Languages | DataSource set to Default Languages | The DataSource was set to the Default Languages | Pass | Null |  |
| 13.4 | Modify\_Word\_Sets () > **btn\_Restore\_Default\_**  **Languages.Click()** | Changing the ***changesMade*** field to false | ***Input*:**  **False** | changesMade Property set to **false** | The changesMade property was set to **false** | Pass | Null |  |
| 13.5 | Modify\_Word\_Sets () > **btn\_Restore\_Default\_**  **Languages.Click()** | User Verification for Restoring Default Languages | ***Input*:**  **User Verified: No** | Message indicating User did not verify Default Language Restoration | A message was printed indicating that the User did not verify Default Language restoration | Pass | Null |  |
| 14.1 | Modify\_Word\_Sets () >  **FormClosing** | Checking if Changes were Made | ***Input*:**  **changesMade = true** | Result comes positive | The result came out positive | Pass | Null |  |
| 14.2 | Modify\_Word\_Sets () >  **FormClosing** | User verification for saving Changes Made | ***Input*:**  **User Verified: Yes** | Message stating User verified to save changes made | A message was printed indicating that the User verified to save the changes made | Pass | Null |  |
| 14.3 | Modify\_Word\_Sets () >  **FormClosing** | Checking if word sets are valid | ***Input*:**  **DataSource of DataGridView in correct format** | Valid Word Sets. Word Sets saved | The Word sets were deemed valid, and saved to the Languages file | Pass | Null |  |
| 14.4 | Modify\_Word\_Sets () >  **FormClosing** | Checking if word sets are valid | ***Input*:**  **DataSource of DataGridView including empty Cells** | Invalid Word Sets. Form Closing event cancelled | The Word Sets were not valid, and the Form was cancelled from closing | Pass | Null |  |
| 14.5 | Modify\_Word\_Sets () >  **FormClosing** | User verification for saving Changes Made | ***Input*:**  **User Verified: Cancel** | Message stating User cancelled Form Closing event | A message was printed indicating that the User cancelled the Form Closing event | Pass | Null |  |
| 14.6 | Modify\_Word\_Sets () >  **FormClosing** | User verification for saving Changes Made | ***Input*:**  **User Verified: No** | Message stating User did not verify to save changes made | A message was printed indicating that the User did not verify to save the changes made | Pass | Null |  |
| 14.7 | Modify\_Word\_Sets () >  **FormClosing** | Checking if Changes were Made | ***Input*:**  **changesMade = false** | Result comes negative | The result came out negative | Pass | Null |  |
| 14.8 | Modify\_Word\_Sets () >  **FormClosing** | Checking whether the Form is closing or not | ***Input*:**  **args.Cancel = true** | The Form Closing event cancelled | The Form closing event is cancelled | Pass | Null |  |
| 14.9 | Modify\_Word\_Sets () >  **FormClosing** | Checking whether the Form is closing or not | ***Input*:**  **args.Cancel = false** | The Form Closing event not cancelled | The Form closing event is not cancelled | Pass | Null |  |
| 15.1 | getLanguagesDataTable () | Fetching the Languages from the Languages File | ***Input*:**  **Languages stored in Languages File** | The languages have been fetched successfully | The languages have been fetched successfully | Pass | Null |  |
| 15.2 | getLanguagesDataTable () | Adding Columns to Languages DataTable | ***Input*:**  **List of Stored Languages** | Columns were added successfully | The Columns were added successfully to the DataTable | Pass | Null |  |
| 15.3 | getLanguagesDataTable () | Adding Rows to Languages DataTable. Then returning Datatable | **List of Stored Languages** | Rows were added successfully | The Rows were added successfully to the DataTable. The table is then returned | Pass | Null |  |
| 16.1 | getBackupLanguagesDataTable () | Fetching the Languages from the Languages File | ***Input*:**  **Languages stored in Backup Languages File** | The languages have been fetched successfully | The languages have been fetched successfully | Pass | Null |  |
| 16.2 | getBackupLanguagesDataTable () | Adding Columns to Languages DataTable | ***Input*:**  **List of Stored Backup Languages** | Columns were added successfully | The Columns were added successfully to the backup DataTable | Pass | Null |  |
| 16.3 | getBackupLanguagesDataTable () | Adding Rows to Languages DataTable. Then returning Datatable | **List of Stored Backup Languages** | Rows were added successfully | The Rows were added successfully to the backup DataTable. The table is then returned | Pass | Null |  |
| 17.1 | saveLanguages () | Updating Languages file | ***Input*:**  **Current Languages present in DataGridView** | Languages file updated successfully | The languages File was updated successfully | Pass | Null |  |
| 17.2 | saveLanguages () | Changing the ***changesMade*** field to false | ***Input*:**  **False** | The **changesMade** property is changed | The **changesMade** property was changed | Pass | Null |  |
| 17.3 | saveLanguages () | Updating the initial row and column amount numbers | ***Input*:**  **Current Languages present in DataGridView** | Initial number of rows and column updated | The initial number of columns and rows are updated | Pass | Null |  |
| 18.1 | checkValidLanguages () | Retrieving DataSource from DataGridView | ***Input*:**  **DataSource of Languages DataGridView** | List of Languages Retrieved from File | Languages were successfully retrieved from file | Pass | Null |  |
| 18.2 | checkValidLanguages () | Checking empty cells within DataTable | ***Input*:**  **DataSource of Languages DataGridView including empty Cells** | Error Prompt, indicating empty Cells were found | Empty cells were found | Pass | Null |  |
| 18.3 | checkValidLanguages () | Checking empty cells within DataTable | ***Input*:**  **DataSource of Languages DataGridView with no empty cells** | Message indicating no empty cells were found | No empty word cells were found | Pass | Null |  |
| 18.4 | checkValidLanguages () | Checking number of rows within DataTable | ***Input*:**  **DataSource of Languages DataGridView with 101 rows** | Error prompt indicating number of rows exceeded 100 and is invalid | The row count exceeded 100 and raised an error | Pass | Null |  |
| 18.5 | checkValidLanguages () | Checking number of rows within DataTable | ***Input*:**  **DataSource of Languages DataGridView with 100 rows** | Message indicating that the row count is valid | The row count did not exceed 100 thus valid | Pass | Null |  |
| 18.6 | checkValidLanguages () | Checking number of rows within DataTable | ***Input*:**  **DataSource of Languages DataGridView with 99 rows** | Message indicating that the row count is valid | The row count did not exceed 100 thus valid | Pass | Null |  |
| 19.1 | checkLanguageName  (**string** name) | Check if name is empty | ***Input*:**  **name = “ “** | Invalid Language Name | Empty Language name | Pass | Null |  |
| 19.2 | checkLanguageName  (**string** name) | Check if name is empty | ***Input*:**  **name = ““** | Invalid Language Name | Empty Language name | Pass | Null |  |
| 19.3 | checkLanguageName  (**string** name ) | Check if name is empty | ***Input*:**  **Name = “French”** | Language name is not empty - Valid | The Language name is not empty | Pass | Null |  |
| 19.4 | checkLanguageName  (**string** name ) | Initializing HashSet with existing Language Names | ***Input*:**  **Languages currently stored in Languages File** | Hashset Languages names initialized | The Language names’ Hashset was filled | Pass | Null |  |
| 19.5 | checkLanguageName  (**string** name ) | Checking whether the new Language’s name already exists | ***Input*:**  **Name = “English”** | Language name already exists | The language name already exists | Pass | Null |  |
| 19.6 | checkLanguageName  (**string** name ) | Checking whether the new Language’s name already exists | ***Input*:**  **Name = “ENgLisH”** | Language name already exists | The language name already exists | Pass | Null |  |
| 19.7 | checkLanguageName  (**string** name ) | Checking whether the new Language’s name already exists | ***Input*:**  **Name = “French”** | Language name does not exist - Valid | The language name was unique | Pass | Null |  |
| 19.8 | checkLanguageName  (**string** name) | Checking whether Language name is valid | ***Input*:**  **Name = “French”** | Language name is valid | The Language name was valid | Pass | Null |  |
| 20.1 | getCurrentLanguages () | Retrieving DataSource from DataGridView | ***Input*:**  **DataGridView with Valid Word Sets** | DataSource acquired | The DataSource was acquired successfully | Pass | Null |  |
| 20.2 | getCurrentLanguages () | Creating List of Languages from DataSource | ***Input*:**  **DataGridView with Valid Word Sets** | List of Languages created | The List of Languages was successfully created | Pass | Null |  |
| 21.1 | deleteLanguage () | Checking number of Languages Left | **Input:**  **Two Languages remaining** | Deletion of Language not permitted | The language was not deleted | Pass | Null |  |
| 22.2 | deleteLanguage () | Checking number of Languages Left | **Input:**  **Three Languages remaining** | Deletion of Language permitted | The deletion of the language was allowed | Pass | Null |  |
| 22.3 | deleteLanguage () | User verification for Language deletion | **Input:**  **User Verified: No** | Deletion of Language Cancelled | The deletion of the Language was cancelled, as it was not verified by the User | Pass | Null |  |
| 22.4 | deleteLanguage () | User verification for Language deletion | **Input:**  **User Verified: Yes** | Deletion of Language verified | The deletion of the language was verified by the User | Pass | Null |  |
| 22.5 | deleteLanguage () | Removing Language from DataGridView | **Input:**  **Current Cell Selected within DataGridView** | Language removed from DataGridView | The Language was removed from the DataGridView | Pass | Null |  |
| 22.6 | deleteLanguage () | Updating Languages File | **Input:**  **DataGridView with Valid Word Sets** | File Updated with remaining Languages | The File was updated with the remaining Languages | Pass | Null |  |

**Log\_In Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Log\_In () | Creating User Interface | Constructor executed | No errors | No errors arose | Pass | Null |  |
| 2.1 | Log\_In () > chck\_Box\_  **Show\_Password.**  **CheckChanged** | Changing the State of the Checkbox | Checkbox selected | Checkbox’s state changed | The checkbox changed its’ state | Pass | Null |  |
| 3.1 | Log\_In () > **link\_Register**  **.Click** | Displaying Register Form temporarily | Register Link clicked | Register Form created and displayed | The register Form was created and displayed | Pass | Null |  |
| 3.2 | Log\_In () > **link\_Register**  **.Click** | Setting Log In Form back to focus after Register Form Closed | Register Form Closed | Log In Form visible again | The Log In form was given focus again after Register Form was closed | Pass | Null |  |
| 4.1 | Log\_In () > **btn\_Login**  **.Click** | Fetching Users from Users’ file | ***Input*:**  **Users’ File** | List of Users fetched from File | The list of Users was fetched from the File | Pass | Null |  |
| 4.2.1 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Empty Users’ List** | Message indicating there are no existing Users | No Message Was Printed | **Fail** | No Message statement was written | *// No Message was displayed* |
| 4.2.2 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Empty Users’ List** | Message indicating there are no existing Users | A message was printed indicating that there are no existing Users | Pass | Null |  |
| 4.3 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Users’ List not Empty.**  **Valid Username and Password Entered** | Successful User Log In | The User Logged In successfully | Pass | Null |  |
| 4.4 | Log\_In () > **btn\_Login**  **.Click** | Displaying User Account | ***Input*:**  **Users’ List not Empty.**  **Valid Username and Password Entered** | Logged In User Account displayed | The Logged In User’s account was displayed | Pass | Null |  |
| 4.5 | Log\_In () > **btn\_Login**  **.Click** | Closing Log In Form | ***Input*:**  **Users’ List not Empty.**  **Valid Username and Password Entered** | Log In Form closed | The Log In Form was closed | Pass | Null |  |
| 4.6 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Users’ List not Empty.**  **Invalid Username and Password Entered. No User has logged in** | Unsuccessful User Log In | The User did not Log In successfully | Pass | Null |  |
| 4.7 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Users’ List not Empty.**  **Valid Username but invalid Password Entered. No User has logged in** | Unsuccessful User Log In | The User did not Log In successfully | Pass | Null |  |
| 4.8 | Log\_In () > **btn\_Login**  **.Click** | Checking if User Log In successful | ***Input*:**  **Users’ List not Empty.**  **Invalid Username but valid Password Entered. No User has logged in** | Unsuccessful User Log In | The User did not Log In successfully | Pass | Null |  |
| 5.1 | Log\_In (string, addUser, addGuest, breakLogin) | Creating User Interface | Constructor executed | No errors | No errors arose | Pass | Null |  |
| 5.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_**  **Guest.Click** | User Verification to Play as Guest | ***Input*:**  **User Verified: Yes** | Message indicating User verified to Play as Guest | A message indicating that the user verified to play as guest was printed | Pass | Null |  |
| 5.3 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_**  **Guest.Click** | Appending new Guest to playing Guests’ List | ***Input*:**  **New Guest instance** | New Guest added to playing Guests’ List | A new Guest was appended to the Playing Guests’ List | Pass | Null |  |
| 5.4 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_**  **Guest.Click** | Closing Login Form – Player registered as Guest | ***Input*:**  **Player registered as Guest** | Form Closes and Guest added to playing Guests’ List | The Log In form closes and the guest is appended to the Player’s list | Pass | Null |  |
| 5.5 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_**  **Guest.Click** | User Verification to Play as Guest | ***Input*:**  **User Verified: No** | Message indicating User did not verify to Play as Guest | A message indicating that the user did not verify to play as guest was printed | Pass | Null |  |
| 6.1 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Player verification to exit Game Setup | ***Input*:**  **User Verified: Yes** | Message indicating player verification to exit Game setup | A message was printed indicating that the player verified to exit the Game setup | Pass | Null |  |
| 6.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Exiting Game Setup | ***Input*:**  **User Verified: Yes** | Message indicating game setup exited | Message indicating that the game setup has exited | Pass | Null |  |
| 6.3 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Closing Log In Form | ***Input*:**  **User Verified: Yes** | Message indicating Log In form closed | Message indicating that Log In form has closed | Pass | Null |  |
| 6.4 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Player verification to exit Game Setup | ***Input*:**  **User Verified: No** | Message indicating player did not verify to exit Game setup | Message indicating that the player did not verify to exit Game Setup | Pass | Null |  |
| 7.1 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Fetching List of Existing Users | ***Input*:**  **Login Button clicked** | List of registered Users retrieved | The List of Users was retrieved | Pass | Null |  |
| 7.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining user login result | ***Input*:**  **List of registered Users empty** | Log In unsuccessful | The Log In was unsuccessful | Pass | Null |  |
| 7.3 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining user login result | ***Input*:**  **List of registered Users not empty. Incorrect Password and Username provided** | Log In unsuccessful | The Log In was unsuccessful | Pass | Null |  |
| 7.4 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining user login result | ***Input*:**  **List of registered Users not empty. Correct Password and Incorrect Username provided** | Log In unsuccessful | The Log In was unsuccessful | Pass | Null |  |
| 7.5 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining user login result | ***Input*:**  **List of registered Users not empty. Correct Username and Incorrect Password provided** | Log In unsuccessful | The Log In was unsuccessful | Pass | Null |  |
| 7.6 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining user login result | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided** | Credentials Match | Credentials Match | Pass | Null |  |
| 7.7 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining if user is already playing or not | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided. User already playing** | Log In unsuccessful | The Log In was unsuccessful | Pass | Null |  |
| 7.8 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Determining if user is already playing or not | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided. User is not already playing** | Log In successful | The Log In was successful | Pass | Null |  |
| 7.9 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Appending User to List of Users’ playing | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided. User is not already playing** | User appended to List of playing Users | The User was added to the List of playing Users | Pass | Null |  |
| 7.10 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Updating Users File | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided. User is not already playing** | Users’ File updated | The Users’ file was successfully updated | Pass | Null |  |
| 7.11 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_LogIn.Click** | Closing Log In Form | ***Input*:**  **List of registered Users not empty. Correct Username and Password provided. User is not already playing** | Log In form closed | The Log In Form closed successfully | Pass | Null |  |
| 8.1 | Log\_In (string, addUser, addGuest, breakLogin) > **chckb\_Box\_Show\_**  **Password.CheckChanged** | Changing the State of the Checkbox | ***Input*:**  **Checkbox selected** | Checkbox state changed | The checkbox’s state was changed | Pass | Null |  |
| 9.1 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Checking whether a Player has registered or Not | ***Input*:**  **Guest has logged in** | Form Closes | The Form Closed | Pass | Null |  |
| 9.2 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Checking whether a Player has registered or Not | **Input:**  **User has logged in** | Form Closes | The Form Closed | Pass | Null |  |
| 9.3 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Checking whether a Player has registered or Not | ***Input*:**  **No Guest or User has logged in** | Message indicating Player did not register | A Message indicating that a Player did not register was printed | Pass | Null |  |
| 9.4 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | User verifying Game Setup termination | ***Input*:**  **No Guest or User has logged in. Player Verified: No** | Form closing event cancelled | The Form Closing event was Cancelled | Pass | Null |  |
| 9.5 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | User verifying Game Setup termination | ***Input*:**  **No Guest or User has logged in. Player Verified: Yes** | Form closed | The Form was closed | Pass | Null |  |
| 10.1 | Log\_In (string, addUser, addGuest, breakLogin) > **linkRegsiter.Click** | Creating and Displaying Register Form | **Input:**  **Player opened User Registration Form** | Register Form displayed and given Focus | The register form was displayed and given focus | Pass | Null |  |
| 10.2 | Log\_In (string, addUser, addGuest, breakLogin) > **linkRegsiter.Click** | Closing Register Form | **Input:**  **Register Form Closed** | Register Form closed and Log In form given focus | The Register Form was closed, and the Log In form was given focus | Pass | Null |  |

**Register Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Register\_Form () | Creating User Interface | Constructor executed | User Interface Created | The User Interface was created | Pass | Null |  |
| 2.1 | Register\_Form () > **chck\_Box\_Show\_**  **Password.Checked**  **Changed** | Checkbox’s state changed | Checkbox clicked | Checkbox state changed | The checkbox’s state was changed | Pass | Null |  |
| 3.1 | Register\_Form () > **chck\_Box\_Show\_**  **Password**  **\_Confirm.Checked**  **Changed** | Checkbox’s state changed | Checkbox clicked | Checkbox state changed | The checkbox’s state was changed | Pass | Null |  |
| 4.1 | Register\_Form () > **btn\_Submit.Click** | Checking for empty input fields | ***Input*:**  **All input fields are left empty** | Error arises, inputs are not valid | The inputs were not valid | Pass | Null |  |
| 4.2 | Register\_Form () > **btn\_Submit.Click** | Checking for empty input fields | ***Input*:**  **Random input fields are left empty** | Error arises, inputs are not valid | The inputs were not valid | Pass | Null |  |
| 4.3 | Register\_Form () > **btn\_Submit.Click** | Checking for empty input fields | ***Input*:**  **All input fields are filled** | No error arises, Input fields valid | The input fields were valid | Pass | Null |  |
| 4.4. | Register\_Form () > **btn\_Submit.Click** | Checking if Username is Unique | ***Input*:**  **Existing username entered: “andreabriffa”** | Invalid Username | The username was invalid as it was already existent | Pass | Null |  |
| 4.5 | Register\_Form () > **btn\_Submit.Click** | Checking if Username is Unique | ***Input*:**  **Unique username entered that includes “Guest” in it** | Invalid Username | The username was invalid as it contained the substring “**Guest**” | Pass | Null |  |
| 4.6 | Register\_Form () > **btn\_Submit.Click** | Checking if Username is Unique | ***Input*:**  **Unique username entered, which does not contain “Guest”: newGuy101** |  | The username was valid | Pass | Null |  |
| 4.7 | Register\_Form () > **btn\_Submit.Click** | Checking if Passwords match | ***Input*:**  **Password 1: abcdefsed**  **Password 2: abdefccee** | Invalid Passwords. Passwords do not match | The passwords were invalid as they did not match | Pass | Null |  |
| 4.8 | Register\_Form () > **btn\_Submit.Click** | Checking if Passwords match | ***Input*:**  **Password 1: abcdefgh**  **Password 2: abcdefgh** | Passwords are valid. Passwords match | The passwords matched and are valid | Pass | Null |  |
| 4.9 | Register\_Form () > **btn\_Submit.Click** | Checking Password Length | ***Input*:**  **Password1: secret**  **Password2:**  **secret** | Invalid password – Too short | The password is invalid as it is smaller than 8 characters long | Pass | Null |  |
| 4.10 | Register\_Form () > **btn\_Submit.Click** | Checking Password Length | ***Input*:**  **Password1: length\_8**  **Password2:**  **length\_8** | Valid Password | The password’s length is valid as it is 8 characters long or more | Pass | Null |  |
| 4.11 | Register\_Form () > **btn\_Submit.Click** | Checking Password Length | ***Input*:**  **Password1: mypassword**  **Password2:**  **mypassword** | Valid Password | The password’s length is valid as it is 8 characters long or more | Pass | Null |  |
| 4.12 | Register\_Form () > **btn\_Submit.Click** | Checking Date of Birth | ***Input*:**  **DateTime (Calendar) date not changed (current Date)** | Invalid Date – Current Date | Date is invalid as it is the Current date. Makes no sense for somebody registering on birth | Pass | Null |  |
| 4.13 | Register\_Form () > **btn\_Submit.Click** | Checking Date of Birth | ***Input*:**  **A date in the future was selected** | Invalid Date – Date in the Future | We do not make time Machines. Birth Date has to be in the past | Pass | Null |  |
| 4.14 | Register\_Form () > **btn\_Submit.Click** | Checking Date of Birth | ***Input*:**  **A date in the past was selected** | Valid Date – Date in the Past | The Date selected was in the Past and is valid | Pass | Null |  |
| 4.15 | Register\_Form () > **btn\_Submit.Click** | Verifying inputs are valid | ***Input*:**  **All fields filled.**  **A Unique username. Matching passwords which are 8 characters long.**  **Date in the past selected.**  ***All inputs are valid*** | Inputs valid | All inputs are valid | Pass | Null |  |
| 4.16 | Register\_Form () > **btn\_Submit.Click** | Appending registered User to existing Users | ***Input*:**  ***All inputs are valid*** | Appending registered User to existing Users | The User was successfully added to the existing Users’ file | Pass | Null |  |
| 4.17 | Register\_Form () > **btn\_Submit.Click** | Closing Register Form | ***Input*:**  ***All inputs are valid*** | Register Form Closed | The Register Form closed | Pass | Null |  |
| 5.1 | Register\_Form () > **FormClosing** | Checking whether registration has occurred or not | ***Input*:**  **valid: *true*** | User Registered – Register Form closed | A User had Registered, thus the Registration Form closed | Pass | Null |  |
| 5.2 | Register\_Form () > **FormClosing** | Checking whether registration has occurred or not | ***Input*:**  **valid: *false*** | A User did not register yet, thus the Form Closing event was stopped temporarily for User verification | The Register Form did not close, and User Verification followed | Pass | Null |  |
| 5.3 | Register\_Form () > **FormClosing** | User verifying Form Closing event | ***Input*:**  **User Verified: *Yes*** | Register Form closed as User verified to do so | The Register Form closed upon User verification | Pass | Null |  |
| 5.4 | Register\_Form () > **FormClosing** | User verifying Form Closing event | ***Input*:**  **User Verified: *No*** | Register Form closing event is cancelled | Register Form closing event cancelled | Pass | Null |  |
| 6.1 | Register\_Form () > ***checkTF (TextBox)*** | Checking If textbox contains text | ***Input*:**  **Text: “ “** | Invalid Text input – No valid text | The text input was not valid | Pass | Null |  |
| 6.2 | Register\_Form () > ***checkTF (TextBox)*** | Checking If textbox contains text | ***Input*:**  **Text: ““** | Invalid Text – No text at all | The text input was not valid | Pass | Null |  |
| 6.3 | Register\_Form () > ***checkTF (TextBox)*** | Checking If textbox contains text | ***Input*:**  **Text: “troll “** | Valid text – Valid input of characters | The text input is valid | Pass | Null |  |
| 7.1 | Register\_Form () > **checkUniqueUsername** | Retrieving List of existing users | ***Input*:**  **None** | List of existing Users retrieved | The List of Users was successfully retrieved | Pass | Null |  |
| 7.2 | Register\_Form () > **checkUniqueUsername** | Checking if Username exists | ***Input*:**  **Username that already exists: *andreabriffa*** | Invalid Username – Username already exists | The Username was invalid as it already exists | Pass | Null |  |
| 7.3 | Register\_Form () > **checkUniqueUsername** | Checking if Username exists | ***Input*:**  **Unique username: *newGuy101*** | Valid Username – Username is unique and does not contain the substring “**Guest**” | The Username was valid as its’ unique and does not have the substring “**Guest**” | Pass | Null |  |
| 7.4 | Register\_Form () > **checkUniqueUsername** | Checking if username contains the substring “**Guest**” | ***Input*:**  **Username: *andreaGuest101*** | Invalid Username – Username contains substring “**Guest**” | The username was invalid as it had the substring “**Guest**” | Pass | Null |  |
| 7.5 | Register\_Form () > **checkUniqueUsername** | Checking if username contains the substring “**Guest**” | ***Input*:**  **Username: *andreabriffa101*** | Username does not contain substring “**Guest**” – valid Username | The username is valid as it does not have the substring “**Guest**” | Pass | Null |  |
| 7.6 | Register\_Form () > **checkUniqueUsername** | Checking if Username is valid | ***Input*:**  **Username that is unique and does not contain “Guest”: *andreabriffa101*** | Valid Username – Username is unique and does not contain the substring “**Guest**” | The Username was valid as its’ unique and does not have the substring “**Guest**” | Pass | Null |  |

**Game Settings Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Game \_Settings\_Form (setGameSettings, breakLogin) | Retrieving List of Languages from Languages File | Languages file existent and filled with a List of Languages | List of Languages fetched | The list of languages was fetched from the file | Pass | Null |  |
| 1.2 | Game \_Settings\_Form (setGameSettings, breakLogin) | Creating the User Interface of the Form | Class constructor executed | UI created | The UI was created successfully | Pass | Null |  |
| 2.1 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Verifying text changed event triggered when text inside textbox is changed | Text inside textbox changed | Text changed event triggered | The text changed event was triggered | Pass | Null |  |
| 2.2 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*hey***” | Text is not in integer format | Error given indicating value entered is not valid | Pass | Null |  |
| 2.3 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “2*he43y***” | Text is not in integer format | Error given indicating value entered is not valid | Pass | Null |  |
| 2.4 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*3.53***” | Text is not in integer format | Error given indicating value entered is not valid | Pass | Null |  |
| 2.5 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*.54***” | Text is not in integer format | Error given indicating value entered is not valid | Pass | Null |  |
| 2.6 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*0***” | Text is in numerical format | No error given. Text value is in valid integer format | Pass | Null |  |
| 2.7 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*1***” | Text is in numerical format | No error given. Text value is in valid integer format | Pass | Null |  |
| 2.8 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*34*”** | Text is in numerical format | No error given. Text value is in valid integer format | Pass | Null |  |
| 2.9 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if value inside number of Pairs textbox is in integer format or not | ***Input*:**  **Text: “*-32*”** | Text is in numerical format | No error given. Text value is in valid integer format | Pass | Null |  |
| 2.10 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *-342*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.11 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *-1*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.12 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *0*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.13 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *1*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.14 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *2*** | Number within valid range | The number is within valid range | Pass | Null |  |
| 2.15 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *21*** | Number within valid range | The number is within valid range | Pass | Null |  |
| 2.16 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *41*** | Number within valid range | The number is within valid range | Pass | Null |  |
| 2.17 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *42*** | Number within valid range | The number is within valid range | Pass | Null |  |
| 2.18 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *43*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.19 | Game \_Settings\_Form (setGameSettings, breakLogin) > **tf\_Input\_Pairs.TextChanged** | Checking if Integer value is between valid range | ***Input*:**  **Min value: 2**  **Max Value: 42**  **Input Value: *342*** | Number not within valid range | The number is not within valid range | Pass | Null |  |
| 2.20 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Checking if selected Languages are different from each other | ***Input*:**  **Language 1: *English***  **Language 2:**  ***English*** | Invalid Languages – (Same Languages Selected) | The languages inputted were invalid as they were the same | Pass | Null |  |
| 2.21 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Checking if selected Languages are different from each other | ***Input*:**  **Language 1: *Italian***  **Language 2:**  ***English*** | Valid Languages – Different Languages selected | The languages inputted are valid as they are different | Pass | Null |  |
| 2.22 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Checking if the Label alert is Visible | ***Input*:**  **Alert Label Visible State: *true*** | Invalid Number of Pairs Identified | Number of Pairs entered is invalid | Pass | Null |  |
| 2.23 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Checking if the Label alert is Visible | ***Input*:**  **Alert Label Visible State: *false*** | Valid Number of Pairs Identified | Number of Pairs entered is valid | Pass | Null |  |
| 2.24 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | User verifying Game Settings | ***Input*:**  **Valid inputs were given**  **User Verified: *No*** | Game Settings are not going to be submitted | User did not verify Game Settings | Pass | Null |  |
| 2.25 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | User verifying Game Settings | ***Input*:**  **Valid inputs were given**  **User Verified: *Yes*** | Game Settings are going to be Submitted | User verified Game Settings | Pass | Null |  |
| 2.26 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Updating Game Settings | ***Input*:**  **Valid inputs were given**  **Int: *10***  **Language:**  ***English***  **Language:**  ***Italian*** | Game Settings Submitted | The game settings were satisfied and submitted | Pass | Null |  |
| 2.27 | Game \_Settings\_Form (setGameSettings, breakLogin) > **btn\_Submit\_Settings.Click** | Closing Game Settings Form | ***Input*:**  ***No Input Given*** | Game Settings Form Closed | The Game Settings Form was closed | Pass | Null |  |
| 2.28 | Game \_Settings\_Form (setGameSettings, breakLogin) > **FormClosing** | Testing if Game Settings have been chosen correctly | ***Input*:**  **valid = *true*** | Game settings were selected correctly | The Form Closing event continues | Pass | Null |  |
| 2.29 | Game \_Settings\_Form (setGameSettings, breakLogin) > **FormClosing** | Testing if Game Settings have been chosen correctly | ***Input*:**  **valid = *false*** | Game settings were not submitted | User verification required for pursuing Form Closing event | Pass | Null |  |
| 2.30 | Game \_Settings\_Form (setGameSettings, breakLogin) > **FormClosing** | User verification regarding closing Game Setup | ***Input*:**  **valid = *false***  **User Verification: false** | Game setup termination cancelled. Game Settings Form closing event cancelled | The Game setup was not verified and the Game Settings Form’s closing event is cancelled | Pass | Null |  |
| 2.31 | Game \_Settings\_Form (setGameSettings, breakLogin) > **FormClosing** | User verification regarding closing Game Setup | ***Input*:**  **valid = *false***  **User Verification: true** | Game setup termination verified. Game Settings Form closing event verified | The Game setup was verified and Game Settings Form closing | Pass | Null |  |

**User\_Account\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | User\_Account\_Form (User) | Loading and Displaying User Account Form | ***Input*:**  **Existing User instance** | User Form Loaded and Display User details appropriately | The User Form loaded and displayed the User details appropriately | Pass | Null |  |
| 1.2 | User\_Account\_Form (User) | Loading and Displaying User Account Form | ***Input*:**  **New User instance** | User Form Loaded and Display User details appropriately | The User Form loaded and displayed the User details appropriately | Pass | Null |  |
| 2.1 | User\_Account\_Form (User) > **calculateWordsCovered (User, Label)** | Storing Total existing Words inside Hashset | ***Input*:**  **List of existing Languages** | Total words stored in Hashset | All of the words were stored inside the Hashset | Pass | Null |  |
| 2.2 | User\_Account\_Form (User) > **calculateWordsCovered (User, Label)** | Storing total words Guessed by User inside Hashset | ***Input*:**  **List of first and second words guessed by User** | Total words guessed by User stored in Hashset | All of the words that were guessed by the User were stored inside the Hashset | Pass | Null |  |
| 2.3 | User\_Account\_Form (User) > **calculateWordsCovered (User, Label)** | Calculating percentage of Total words covered by User | ***Input*:**  **Amount of words Guessed by User: 20**  **Total amount of existing words: 50** | Percentage calculated correctly: **40%** | The percentage was calculated correctly: **40%** | Pass | Null |  |
| 2.4 | User\_Account\_Form (User) > **calculateWordsCovered (User, Label)** | Calculating percentage of Total words covered by User | ***Input*:**  **Amount of words Guessed by User: 20**  **Total amount of existing words: 200** | Percentage calculated correctly: **40%** | The percentage was calculated correctly: 1**0%** | Pass | Null |  |
| 2.5 | User\_Account\_Form (User) > **calculateWordsCovered (User, Label)** | Displaying percentage of words that the User covered | ***Input*:**  **User: *existing* *User***  **Label: *existing* *Label*** | The user’s percentage of words covered is displayed within the Label provided inside the arguments | The user’s percentage of words covered was displayed inside the Label provided inside the arguments | Pass | Null |  |
| 3.1 | User\_Account\_Form (User) > **calculateAverage**  **UserTime (User)** | Gathering User’s total elapsed time | ***Input*:**  **User’s list of times for each Game** | User’s total time gathered | The User’s total time was gathered | Pass | Null |  |
| 3.2 | User\_Account\_Form (User) > **calculateAverage**  **UserTime (User)** | Determining whether a User took part in a Game or not | ***Input*:**  **Empty List of Game Times** | User’s total minutes and seconds played set to ‘**0**’ | User’s total minutes and seconds played were set to ‘**0**’ | Pass | Null |  |
| 3.3 | User\_Account\_Form (User) > **calculateAverage**  **UserTime (User)** | Determining whether a User took part in a Game or not | ***Input*:**  **List of Game Times which is not empty** | User’s total minutes and seconds played were calculated using the total time of seconds played | The user’s total minutes and seconds played were calculated using the total time of seconds played | Pass | Null |  |
| 3.4 | User\_Account\_Form (User) > **calculateAverage**  **UserTime (User)** | Displaying the User’s average time in minutes and seconds | ***Input*:**  **List of Game Times which is not empty** | User’s average time was displayed in minutes and seconds | The User’s average time was displayed in minutes and seconds | Pass | Null |  |
| 4.1 | User\_Account\_Form (User) > **pupulateUserGame**  **Details (DataGridView, User)** | Adding columns to Datatable | ***Input*:**  **New DataColumns** | Columns added to DataTable | The Columns were added to the DataTable | Pass | Null |  |
| 4.2 | User\_Account\_Form (User) > **pupulateUserGame**  **Details (DataGridView, User)** | Adding Rows to DataTable | ***Input*:**  **User with no prior games Played** | No rows added to GameDetails DataTable | No rows were added to the Game Details DataTable | Pass | Null |  |
| 4.3 | User\_Account\_Form (User) > **pupulateUserGame**  **Details (DataGridView, User)** | Adding Rows to DataTable | ***Input*:**  **User with prior games Played** | Rows were added to the GameDetails DataTable | The Rows were added to the GameDetails DataTable | Pass | Null |  |
| 4.4 | User\_Account\_Form (User) > **pupulateUserGame**  **Details (DataGridView, User)** | Updating DataGridView’s DataSource | ***Input*:**  **DataTable with created Columns and Rows** | DataGridView’s DataSource updated | The DataGridView’s DataSource was updated | Pass | Null |  |

**Game Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Game (List<Guest>, List<User>, Language, Language, int) | Initializing List of Buttons | ***Input*:**  **New List of Button Instances** | List of Button references created | The List of button references was created | Pass | Null |  |
| 1.2 | Game (List<Guest>, List<User>, Language, Language, int) | Setting the Users/Guests registered as Players | ***Input*:**  **Two Users registered** | Two Players were created from the User instances | Two Players were created from the User instances | Pass | Null |  |
| 1.3 | Game (List<Guest>, List<User>, Language, Language, int) | Setting the Users/Guests registered as Players | ***Input*:**  **Two Guests registered** | Two Players were created from the Guest instances | Two Players were created from the Guest instances | Pass | Null |  |
| 1.4 | Game (List<Guest>, List<User>, Language, Language, int) | Setting the Users/Guests registered as Players | ***Input*:**  **Both User and Guest Registered** | Two Players were created from the User and Guest instances | Two Players were created from the User and Guest instances | Pass | Null |  |
| 1.5 | Game (List<Guest>, List<User>, Language, Language, int) | Randomizing the word sets before assigning them to the buttons | ***Input*:**  **Existing word sets** | Word sets were randomized | The word sets were randomized | Pass | Null |  |
| 1.6 | Game (List<Guest>, List<User>, Language, Language, int) | Creating button pair tuples | ***Input*:**  **List of Buttons** | Button Pair tuples created | The button pair tuples were created | Pass | Null |  |
| 1.7 | Game (List<Guest>, List<User>, Language, Language, int) | Verifying Game User Interface created | ***Input*:**  **No Inputs** | The Game User Interface created | The Game User Interface was created | Pass | Null |  |
| 2.1 | Game (List<Guest>, List<User>, Language, Language, int) > **switchPlayer ()** | Switching Current Player | ***Input*:**  **Current Player: Player1** | Switching from Player 1 to Player 2 |  | Pass | Null |  |
| 2.2 | Game (List<Guest>, List<User>, Language, Language, int) > **switchPlayer ()** | Switching the Current Player | ***Input*:**  **Current Player: Player1** | Current Player set to Player 2. Player 1 Stopwatch stopped. Player 2 Stopwatch continued | The current Player was set to Player 2. Player 1’s stopwatch was stopped, and Player 2’s stopwatch was continued | Pass | Null |  |
| 2.3 | Game (List<Guest>, List<User>, Language, Language, int) > **switchPlayer ()** | Switching Current Player | ***Input*:**  **Current Player: Player2** | Switching from Player 2 to Player 1 |  | Pass | Null |  |
| 2.4 | Game (List<Guest>, List<User>, Language, Language, int) > **switchPlayer ()** | Switching the Current Player | ***Input*:**  **Current Player: Player2** | Current Player set to Player 1. Player 2 Stopwatch stopped. Player 1 Stopwatch continued | The current Player was set to Player 1. Player 1’s stopwatch was stopped, and Player 1’s stopwatch was continued | Pass | Null |  |
| 3.1 | Game (List<Guest>, List<User>, Language, Language, int) > **updateScoreLabels ()** | Updating the Score Labels | ***Input*:**  **player1.score**  **player2.score** | Score Labels updated | The score labels were updated | Pass | Null |  |
| 4.1 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the Game has started or not | ***Input*:**  **began: *false*** | Starting Game Timer and Stopwatches | The Game timer and stopwatches | Pass | Null |  |
| 4.2 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Starting the Timers and Stopwatches | ***Input*:**  **began: *false*** | Game Timer and Stopwatches started | The Game Timer and Stopwatches were stopped | Pass | Null |  |
| 4.3 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the Game has started or not | ***Input*:**  **began: *true*** | Game Already Started | The Game has already started | Pass | Null |  |
| 4.4 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Adding selected button to List of Selected Buttons | ***Input*:**  **Custom\_Button instance** | Selected Button added to List of Selected Buttons | The Selected Button was added to the List of Selected Buttons | Pass | Null |  |
| 4.5 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the current Player has selected two different buttons consecutively | ***Input*:**  **2 Buttons Selected** | Updating Player Details | The Player details where updated | Pass | Null |  |
| 4.6 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the buttons are from the same tuple | ***Input*:**  **2 Buttons Selected, which are from the same Tuple** | Game temporarily paused. Selected buttons are not visible anymore | The Game was temporarily paused, and the Selected buttons are not visible anymore | Pass | Null |  |
| 4.7 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether there are pairs left to Guess | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **No button Pairs Left** | Game Finished. Game timer and stopwatches stopped. | The Game was Finished. The Game timer and stopwatches were stopped. | Pass | Null |  |
| 4.8 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Determining Game Winner | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **No button Pairs Left.**  **Player 1 Score:*2***  **Player 2 Score:*6*** | Player 2 is the Winner | The winner was player 2 | Pass | Null |  |
| 4.9 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Determining Game Winner | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **No button Pairs Left.**  **Player 1 Score:*9***  **Player 2 Score:*4*** | Player 1 is the Winner | The winner was player 1 | Pass | Null |  |
| 4.10 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Determining Game Winner | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **No button Pairs Left.**  **Player 1 Score:*5***  ***Time: 10607 Ms***  **Player 2 Score:*10***  ***Time: 9865 Ms*** | Player 2 is the Winner | The winner was player 2 – Less Time Spent Playing | Pass | Null |  |
| 4.11 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Determining Game Winner | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **No button Pairs Left.**  **Player 1 Score:*5***  ***Time: 7764 Ms***  **Player 2 Score:*10***  ***Time: 9664Ms*** | Player 1 is the Winner | The winner was player 1 – Less Time Spent Playing | Pass | Null |  |
| 4.12 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Fetching List of Users | ***Input*:**  **No input** | List of existing Users fetched | The List of existing Users was fetched | Pass | Null |  |
| 4.13 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether player has the same name as a registered User | ***Input*:**  **Game finished**  **Player1 username: Guest1**  **Player2 username: Guest2**  **Registered Username: johndoe** | No match | No match was found | Pass | Null |  |
| 4.14 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether player has the same name as a registered User | ***Input*:**  **Game finished**  **Player1 username: andreabriffa**  **Player2 username: Guest1**  **Registered Username: andreabriffa** | Match – Player 2’s details passed onto User’s details | A match was found between player 2 and a User. Player 2’s Properties were added to the User’s | Pass | Null |  |
| 4.15 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether player has the same name as a registered User | ***Input*:**  **Game finished**  **Player1 username: andreabriffa**  **Player2 username: stevepickle**  **Registered Username: stevepickle** | Both Players Match – Player properties passed on to Users’ | A match was found between both players and two Users. Player 1’s and 2’s Properties were added to the Users’ | Pass | Null |  |
| 4.16 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Saving User and Game records to the Users’ and Games’ File | ***Input*:**  **List<Users>**  **Game Properties** | Users’ and Games’ files updates | The Users’ and Games’ file were updated | Pass | Null |  |
| 4.17 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Directing Players to Welcome Form. Seeking Players’ approval | ***Input*:**  **Game Finished.**  **User Verification: No** | Players remained on the Game Form | The Game Form was not closed | Pass | Null |  |
| 4.18 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Directing Players to Welcome Form. Seeking Players’ approval | ***Input*:**  **Game Finished.**  **User Verification: Yes** | Players guided back to Welcome Form | The Game Form was closed and the Players were set back to the Welcome Form | Pass | Null |  |
| 4.19 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether there are pairs left to Guess | ***Input*:**  **2 Buttons Selected, which are from the same Tuple.**  **More than 0 button Pairs Left** | Game does not finish | The Game continued playing | Pass | Null |  |
| 4.20 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the buttons are from the same tuple | ***Input*:**  **2 Buttons Selected, which are not from the same Tuple** | Game temporarily paused. Selected buttons hidden again | The Game was temporarily paused. The selected buttons were hidden again | Pass | Null |  |
| 4.21 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Switching the current Player | ***Input*:**  **2 Buttons selected by Player not from same Tuple** | Current Player switched | The current Player was switched | Pass | Null |  |
| 4.22 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Checking whether the current Player has selected two different buttons consecutively | ***Input*:**  **Only 1 Button Selected** | No action | No action was took | Pass | Null |  |
| 4.23 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Selecting the same button twice consecutively | ***Input*:**  **Same button pressed twice** | Second press does not register as a second button selected | The Second press on the same button dig not register as a second button | Pass | Null |  |
| 4.23 | Game (List<Guest>, List<User>, Language, Language, int) > **buttonClicked (Button\_Custom)** | Executing timer interval method | ***Input*:**  **Time ticked** | Time elapsed Labels updated | The time elapsed labels were updated | Pass | Null |  |

**Statistics\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Statistics\_Form () | Creating the  User Interface | **Class Constructor Executed** | User Interface Initialized | The User interface was created | Pass | Null |  |
| 2.1 | Populate\_Groups () | Assigning Event Handlers to ComboBoxes’ SelectedIndexChanged event | ***Input*:**  **No Users are Registered in the System** | Event Handlers not Assigned | The Event Handlers were not Assigned | Pass | Null |  |
| 2.2 | Populate\_Groups () | Assigning Event Handlers to ComboBoxes’ SelectedIndexChanged event | ***Input*:**  **Users are Registered in the System** | Event Handlers Assigned | The event Handlers were not Assigned | Pass | Null |  |
| 2.3 | Populate\_Groups () | Finalising Form Statistics | ***Input*:**  **No input** | Form Statistics Updated Successfully | The Form’s statistics were updated successfully | Pass | Null |  |
| 2.4 | Populate\_Groups () | Updating User Details within DataGridView | ***Input*:**  **Selected Index of User combobox changed** | The User details within the DataGridView updated | The User details within the DataGridView were updated | Pass | Null |  |
| 3.1 | updateUserSuccess  DataGridView (int, DataGridView) | Fetching Selected User from List of existing Users | ***Input*:**  **Int: userIndex**  **DataGridView** | Selected User retrieved by Index | The selected User was retrieved | Pass | Null |  |
| 3.2 | updateUserSuccess  DataGridView (int, DataGridView) | Updating the User Details within the DataGridView | ***Input*:**  **Int: userIndex**  **DataGridView** | User Details within DataGridView Updated | The User details within the DataGridView were updated | Pass | Null |  |
| 4.1 | updateWordSet (int) | Updating the Words Combobox | ***Input*:**  **Int: Index of Selected Langauge from Languages Combo Box** | Words ComboBox updated | The words ComboBox was updated | Pass | Null |  |
| 5.1 | populateGame  Summaries  DataGridView () | Retrieving List of Games | ***Input*:**  **No input** | List of Games fetched from Game Records File | The List of Games was fetched from the Game Records File | Pass | Null |  |
| 5.2 | populateGame  Summaries  DataGridView () | Updating the Game Summaries DataGridView | ***Input*:**  **GameSummaries**  **DataGridView** | Game Summaries DataGridView Updated | The Game Summaries DataGridView was updated | Pass | Null |  |
| 6.1 | populateUser  RanksDtGridView () | Retrieving List of Users from Users File | ***Input*:**  **No input** | List of Current Users Retrieved | The List of Current Users was retrieved | Pass | Null |  |
| 6.2 | populateUser  RanksDtGridView () | Updating User Ranks DataGridView | ***Input*:**  **User Ranks**  **DataGridView** | User Ranks DataGridView Updated | The User Ranks DataGridView was updated | Pass | Null |  |
| 7.1 | updateUser  SuccessWord (int, int, int) | Fetching List of Existing Users and Languages from Files | Input:  Int: selected User Index  Int: Language selected Index  Int: word selected Index | List of Languages and Users fetched from Files | The List of Languages and Users were fetched from their respective Files | Pass | Null |  |
| 7.2 | updateUser  SuccessWord (int, int, int) | Verifying User Success Word Rate Details updated | Input:  No input | User Success Word Rates Details Updated | The User Success Word Rates Details were updated | Pass | Null |  |

**Library Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | readFile<User> getUsers () | Checking if User File exists | ***Input*:**  **User File exists: *false*** | Unable to fetch List of Users | The File was not found. Unable to fetch list of Users | Pass | Null |  |
| 1.2 | readFile<User> getUsers () | Checking if User File exists | ***Input*:**  **User File exists: *true*** | Users File found | The User File was found | Pass | Null |  |
| 1.3 | readFile<User> getUsers () | Retrieving List of existing Users | ***Input*:**  **User File exists: *true*** | List of Users fetched | The list of Users was fetched successfully | Pass | Null |  |
| 2.1 | readFile<Language> getLanguages () | Checking if Languages File exists | ***Input*:**  **Languages File exists: *false*** | Unable to fetch List of Languages | The File was not found. Unable to fetch list of Languages | Pass | Null |  |
| 2.2 | readFile<Language> getLanguages () | Checking if Languages File exists | ***Input*:**  **Languages File exists: *true*** | Languages File found | The Languages File was found | Pass | Null |  |
| 2.3 | readFile<Language> getLanguages () | Retrieving List of existing Languages | ***Input*:**  **Languages File exists: *true*** | List of Languages fetched | The list of Languages was fetched successfully | Pass | Null |  |
| 3.1 | readFile< GameRecord > getGames () | Checking if Game Records File exists | ***Input*:**  **Game Records File exists: *false*** | Unable to fetch List of Game Records | The File was not found. Unable to fetch list of Games Records | Pass | Null |  |
| 3.2 | readFile< GameRecord > getGames () | Checking if Game Records File exists | ***Input*:**  **Game Records File exists: *true*** | Game Records File found | The Game Records File was found | Pass | Null |  |
| 3.3 | readFile< GameRecord > getGames () | Retrieving List of existing Game Records | ***Input*:**  **Game Records File exists: *true*** | List of Game Records fetched | The list of Game Records was fetched successfully | Pass | Null |  |
| 4.1 | readFile<Language> getLanguages  Backup () | Checking if Backup Languages File exists | ***Input*:**  **Backup Languages File exists: *false*** | Unable to fetch List of Backup Languages | The File was not found. Unable to fetch list of Backup Languages | Pass | Null |  |
| 4.2 | readFile<Language> getLanguages  Backup () | Checking if Backup Languages File exists | ***Input*:**  **Backup Languages File exists: *true*** | Backup Languages File found | The Backup Languages File was found | Pass | Null |  |
| 4.3 | readFile<Language> getLanguages  Backup () | Retrieving List of existing Backup Languages | ***Input*:**  **Backup Languages File exists: *true*** | List of Backup Languages fetched | The list of Backup Languages was fetched successfully | Pass | Null |  |
| 5.1 | writeFile<User> writeUsers (List<User>) | Write List of Users to Users File | ***Input*:**  **List<User>** | List of Users written to Users file | The List of Users was written to the Users File | Pass | Null |  |
| 6.1 | writeFile<Language> writeLanguages (List< Language >) | Write List of Languages to Languages File | ***Input*:**  **List<** **Language >** | List of Languages written to Languages file | The List of Languages was written to the Languages File | Pass | Null |  |
| 7.1 | writeFile<GameRecord> writeGames (List< GameRecord >) | Write List of Game Records to Game Records File | ***Input*:**  **List<** **GameRecord >** | List of Game Records written to Game Records file | The List of Game Records was written to the Game Records File | Pass | Null |  |
| 8.1 | writeFile<Language> writeBackupLanguages (List< Language >) | Write List of Languages to Backup Languages File | ***Input*:**  **List<** **Language >** | List of Languages written to Backup Languages file | The List of Languages was written to the Backup Languages File | Pass | Null |  |
| 9.1 | appendFile<User> appendUser (user) | Append User instance to Users File | ***Input*:**  **User instance** | User instance added to Users File | The User Instance was added to the Users File | Pass | Null |  |
| 10.1 | appendFile< GameRecord > appendGame (GameRecord) | Append Game Record instance to Game Records File | ***Input*:**  **GameRecord instance** | Game Record instance added to Game Records File | The Game Record Instance was added to the Game Records File | Pass | Null |  |
| 11.1 | writeInitialLanguages () | Overwriting the current Languages with the Default Languages | ***Input*:**  **List<Language**> | Current Languages overridden with default Languages | The current Languages were overridden with the Current Languages | Pass | Null |  |
| 12.1 | passCheckBox (CheckBox, TextBox) | Obfuscating/  Making the Text inside a Textbox clear | **Input:**  **CheckBox checked state: *checked*** | Password Text inside Textbox visible clearly | The password text inside the Textbox was made clear | Pass | Null |  |
| 12.2 | passCheckBox (CheckBox, TextBox) | Obfuscating/  Making the Text inside a Textbox clear | **Input:**  **CheckBox checked state: un*checked*** | Password Text inside Textbox obfuscated | The password text inside the Textbox was obfuscated | Pass | Null |  |
| 13.1 | transitionInteger (int, Label) | Setting Time Interval Function | Input:  Value: 0 | Label’s text set to ‘**0**’ | The Label’s text was set to ‘**0**’ | Pass | Null |  |
| 13.2 | transitionInteger (int, Label) | Setting Time Interval Function | Input:  Value: 16 | Label’s text transitioned to 16 | The Label’s text transitioned to **16** | Pass | Null |  |

## **Black Box Testing**

**Welcome\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Welcome\_Form () | Displaying the Welcome Form | ***Input:***  **Application Run** | The Welcome Form is Displayed | The Welcome Form is Displayed | Pass | Null |  |
| 6.2 | Welcome\_Form () > **VisibleChanged** | Changing the visibility of the Form | ***Input*:**  **User closed separate Form, returning to Welcome Form** | Welcome Form Set Visible | The Welcome Form was set Visible | Pass | Null |  |
| 7.1 | Welcome\_Form () > **btn\_Login.Click** | Creating new instance Login Form and displaying it | ***Input*:**  **User clicked Log In Button** | Login Form displayed | The Login Form was displayed | Pass | Null |  |
| 8.1 | Welcome\_Form () > **btn\_Create\_Account**  **.Click** | Creating new instance of Registration Form and displaying it | ***Input*:**  **User clicked Create New Account Button** | Registration Form displayed | The Registration Form was displayed | Pass | Null |  |
| 9.1 | Welcome\_Form () > **btn\_Modify\_**  **Word\_Sets**  **.Click** | Creating new instance of Modify Word Set Form and displaying it | ***Input*:**  **User clicked Modify Word Set Button** | Modify Word Set Form displayed | The Modify Word Set Form was displayed | Pass | Null |  |
| 10.1 | Welcome\_Form () > **btn\_Statistics**  **.Click** | Creating new instance of Statistics Form and displaying it | ***Input*:**  **User clicked Statistics Button** | Statistics Form displayed | The Statistics Form was displayed | Pass | Null |  |
| 11.1 | Welcome\_Form () > **btn\_Start\_Game**  **.Click** | Displaying the Log In form for the Players | ***Input*:**  **User clicked Start Game Button** | Log In Form displayed | The Log In Form was displayed | Pass | Null |  |

**Modify\_Word\_Sets Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Modify\_Word\_Sets () | Displaying the Modify Word Set Form | Constructor executed | Modify Word Set For Displayed | The Modify Word Set Form was displayed | Pass | Null |  |
| 2.1 | saveLanguages () | Saving Current Languages present in DataGridView | ***Input*:**  **Two (2) Languages, each containing five (5) words** | Languages saved to Languages File | The Languages were saved to the Languages File | Pass | Null |  |
| 2.2 | saveLanguages () | Saving Current Languages present in DataGridView | ***Input*:**  **Three (3) Languages, each containing forty-two (42) words** | Languages saved to Languages File | The Languages were saved to the Languages File | Pass | Null |  |
| 3.1 | deleteLanguage () | Deleting a Language from the DataGridView | ***Predecessor***:  **User Verified to Delete Language: *Yes***  **Number of Current Languages: 4** | Selected Language Deleted | The selected Language was Deleted as the User verified so and the minimum number of Languages (2) wasn’t met | Pass | Null |  |
| 4.1 | getCurrentLanguages () | Returns the List of Languages that are currently present in the DataGridView | ***Predecessor*: DataGridView Word Sets already checked and are valid**  ***Input*:**  **DataGridView contains two (2) Languages each having thirty (30) words** | List of Languages Created | The List of Languages was created successfully | Pass | Null |  |
| 4.2 | getCurrentLanguages () | Returns the List of Languages that are currently present in the DataGridView | ***Predecessor*: DataGridView Word Sets already checked and are valid**  ***Input*:**  **DataGridView contains three (3) Languages each having forty-two (42) words** | List of Languages Created | The List of Languages was created successfully | Pass | Null |  |
| 5.1 | deleteLanguage () | Deleting a Language from the DataGridView | ***Predecessor***:  **User Verified to Delete Language: *Yes***  **Number of Current Languages: *3*** | Selected Language Deleted | The selected Language was Deleted as the User verified so and the minimum number of Languages (2) wasn’t met | Pass | Null |  |
| 5.2 | deleteLanguage () | Deleting a Language from the DataGridView | ***Predecessor***:  **User Verified to Delete Language: *No***  **Number of Current Languages: 4** | Language not deleted | The selected Language was not Deleted as the User did not verify to do so | Pass | Null |  |
| 5.3 | deleteLanguage () | Deleting a Language from the DataGridView | ***Predecessor***:  **Number of Current Languages: 2** | Language not deleted | The selected Language was not Deleted as the minimum number of Languages (2) was reached | Pass | Null |  |
| 6.1 | Modify\_Word\_Sets > **dtGirdViewWordSets**  **.CellClick** | Keeping track that changes were made | ***Input*:**  **User selected Cell within DataGridView** | Changes tracked | The Changes were Tracked | Pass | Null |  |
| 7.1 | Modify\_Word\_Sets > **dtGirdViewWordSets.**  **ColumnAdded** | Keeping track that changes were made | ***Input*:**  **User added new Column within DataGridView** | Changes tracked | The Changes were Tracked | Pass | Null |  |
| 8.1 | Modify\_Word\_Sets > **dtGirdViewWordSets.**  **RowsAdded** | Keeping track that changes were made | ***Input*:**  **User added new Row within DataGridView** | Changes tracked | The Changes were Tracked | Pass | Null |  |
| 9.1 | Modify\_Word\_Sets > **btn\_Add\_Language.Click** | Adding Language to existing Languages | ***Input*:**  **Empty Language Name Entered** | Invalid Language Name | The Language Name was invalid | Pass | Null |  |
| 9.2 | Modify\_Word\_Sets > **btn\_Add\_Language.Click** | Adding Language to existing Languages | ***Input*:**  **Unique and not Empty Language Name Entered: German** | Valid Language Name | The Language Name is valid | Pass | Null |  |
| 10.1 | Modify\_Word\_Sets > **btn\_Save\_Language**  **.Click** | Saving existing Languages | ***Input*:**  **List of Languages with Empty words** | Invalid Languages – Not Saved | The Languages were not valid and weren’t saved | Pass | Null |  |
| 10.2 | Modify\_Word\_Sets > **btn\_Save\_Language**  **.Click** | Saving Existing Languages | ***Input*:**  **List of Languages with Empty words** | Valid Languages – Saved | The Languages were valid and Saved | Pass | Null |  |
| 11.1 | Modify\_Word\_Sets > **btn\_Delete\_Row.Click** | Deleting Row within DataGridView | ***Input*:**  **More than 2 Rows left.**  **User pressed Delete Row Button. User verified deletion of Row** | Selected Row deleted | The selected was Deleted | Pass | Null |  |
| 11.2 | Modify\_Word\_Sets > **btn\_Delete\_Row.Click** | Deleting Row within DataGridView | ***Input*:**  **2 Rows Left.**  **User pressed Delete Row Button.** | Row not Deleted as minimum number of Rows reached | The selected was not Deleted | Pass | Null |  |
| 12.1 | Modify\_Word\_Sets > **btn\_Delete.Click** | Starting Process of Language Deletion | ***Input*:**  **User pressed Delete Language Button** | Process of Language Deletion Started | The Process of Deleting a Language Started | Pass | Null |  |
| 13.1 | Modify\_Word\_Sets > **btn\_Backup.Click** | Backup Up List of Current Languages | ***Input*:**  **List of Languages with empty words** | Invalid Languages – Languages not Backed Up | The Languages were not Backed Up | Pass | Null |  |
| 13.2 | Modify\_Word\_Sets > **btn\_Backup.Click** | Backup Up List of Current Languages | ***Input*:**  **List of Languages with no empty words**  **Backup already exists**  **User verified overriding Languages Backup** | Valid Languages – Languages Backed Up | The Languages were Backed Up | Pass | Null |  |
| 13.3 | Modify\_Word\_Sets > **btn\_Backup.Click** | Backup Up List of Current Languages | ***Input*:**  **List of Languages with no empty words**  **Backup already exists**  **User did not verify overriding Languages Backup** | Languages not backed up – Languages Not Backed Up | The Languages were not Backed Up | Pass | Null |  |
| 13.4 | Modify\_Word\_Sets > **btn\_Backup.Click** | Backup Up List of Current Languages | ***Input*:**  **List of Languages with no empty words**  **Backup does not Exist** | Languages not backed up – Languages Backed Up | The Languages were Backed Up | Pass | Null |  |
| 14.1 | Modify\_Word\_Sets > **btn\_Restore.Click** | Replacing current Languages with Backup Languages | ***Input*:**  **No Languages Backed up** | User prompted to back up current Languages | The User was prompted to back up the current Languages | Pass | Null |  |
| 14.2 | Modify\_Word\_Sets > **btn\_Restore.Click** | Replacing current Languages with Backup Languages | ***Input*:**  **Languages are Backed up** | User Prompted to verify Restoring the Backup Languages | The User was Prompted to verify Restoring the Backup Languages | Pass | Null |  |
| 15.1 | Modify\_Word\_Sets > **btn\_Restore\_Default**  **\_Languages.Click** | Restoring the default Languages | ***Input*:**  **User verified to restore default languages** | Default Languages Restored | The Default Languages were restored | Pass | Null |  |
| 15.2 | Modify\_Word\_Sets > **btn\_Restore\_Default**  **\_Languages.Click** | Restoring the default Languages | ***Input*:**  **User did not verify to restore default Languages** | Default Languages Restored | The Default Languages were not restored | Pass | Null |  |
| 16.1 | Modify\_Word\_Sets > **FormClosing** | Closing Form | ***Input*:**  **changes made = true** | User prompted to Save the Changes Made | The User was prompted to save the changes made | Pass | Null |  |
| 16.2 | Modify\_Word\_Sets > **FormClosing** | Closing Form | ***Input*:**  **changes made = false** | Modify Word Set Form Closed | The Modify Word Set Form was Closed | Pass | Null |  |

**Game Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Game (List<Guest> guests, List<User> users, Language language\_1, Language language\_2, int numOfPairs) | Displaying the Game Form (User Interface) | Number of Buttons to Display: **52** | Game Form created with **52** buttons | Game Form created with **52** buttons | Pass | Null |  |
| 1.2 | Game (List<Guest> guests, List<User> users, Language language\_1, Language language\_2, int numOfPairs) | Displaying the Game Form (User Interface) | Number of Buttons to Display: **10** | Game Form created with **10** buttons | Game Form created with **10** buttons | Pass | Null |  |
| 1.3 | Game (List<Guest> guests, List<User> users, Language language\_1, Language language\_2, int numOfPairs) | Displaying the Game Form (User Interface) | Number of Buttons to Display: **26** | Game Form created with **26** buttons | Game Form created with **26** buttons | Pass | Null |  |
| 1.4 | Game (List<Guest> guests, List<User> users, Language language\_1, Language language\_2, int numOfPairs) | Displaying the Game Form (User Interface) | Number of Buttons to Display: **84** | Game Form created with **84** buttons | Game Form created with **84** buttons | Pass | Null |  |
| 2.1 | buttonClicked (Button\_Custom button) | Handling Button Clicked | ***Input*:**  **One button pressed consecutively** | No action takes place | No action took place | Pass | Null |  |
| 2.2 | buttonClicked (Button\_Custom button) | Handling Button Clicked | ***Input*:**  **Two different Buttons pressed consecutively** | Updating Player Details | The Player details were updated | Pass | Null |  |
| 2.3 | buttonClicked (Button\_Custom button) | Handling Button Clicked | ***Predecessor:***  **Two different Buttons pressed consecutively**  ***Input*:**  **Two buttons are from same Pair** | Buttons removed from UI and scores updated | The buttons were removed from the UI and the Scores were updated | Pass | Null |  |
| 2.4 | buttonClicked (Button\_Custom button) | Handling Button Clicked | ***Predecessor:***  **Two different Buttons pressed consecutively**  ***Input*:**  **Two buttons are not from same Pair** | Buttons’ words were obfuscated again | The Buttons’ words were obfuscated again | Pass | Null |  |
| 2.5 | buttonClicked (Button\_Custom button) | Handling Button Clicked | ***Predecessor:***  **Two different Buttons pressed consecutively and are from same pair**  ***Input*:**  **No more buttons left to Guess** | Game finishes | The Game is terminated | Pass | Null |  |
| 3.1 | switchPlayers () | Switching Players | ***Input*:**  **CurrentPlayer: *player1*** | Current Player Set to Player 2 | The Current Player was set to Player 2 | Pass | Null |  |
| 3.2 | switchPlayers () | Switching Players | ***Input*:**  **CurrentPlayer: *player2*** | Current Player Set to Player 1 | The Current Player was set to Player 1 | Pass | Null |  |
| 4.1 | updateScoreLabels () | Updating the Score Labels | ***Input*:**  **Scores of Player 1 and Player 2** | Score Labels Updated | The Score Labels were updated | Pass | Null |  |
| 5.1 | gameTimer.Tick | Updating time elapsed Labels | ***Input*:**  **Interval: 1 second** | Time Elapsed Labels Updated each time the timer ticks | Every time the Timer ticked, the time elapsed Labels were updated | Pass | Null |  |
| 6.1 | FormClosing | Closing the Game Form | ***Input*:**  **Game Finished = true** | Closing the Game form as the Game has finished | The Game form was closed | Pass | Null |  |
| 6.2 | FormClosing | Closing the Game Form | ***Input*:**  **Game Finished = false** | Player verification required to stop the game | The Player was asked to verify to stop the Game | Pass | Null |  |
| 7.1 | buttonClicked (Button\_Custom button) | Event Handler for when a button is clicked | ***Input*:**  **One Button Pressed** | Word associated with button displayed | The word associated with the Button was displayed | Pass | Null |  |
| 7.2 | buttonClicked (Button\_Custom button) | Event Handler for when a button is clicked | ***Input*:**  **Two Buttons pressed consecutively – Not from the same pair** | The buttons’ words are hidden again. Current Player switched | The words displayed on the buttons were hidden again. The current Player is switched | Pass | Null |  |
| 7.3 | buttonClicked (Button\_Custom button) | Event Handler for when a button is clicked | ***Input*:**  **Two Buttons pressed consecutively – Buttons are from the same pair** | The buttons are removed from the Game Form – Current User receives a point | The buttons were removed from the Game Form – Current User received a point | Pass | Null |  |
| 7.4 | buttonClicked (Button\_Custom button) | Event Handler for when a button is clicked | ***Input*:**  **Two Buttons pressed consecutively – Buttons are from the same pair** | Game finished as all button pairs have been guessed | The game finished as there were no remaining button pairs left. Winner declared | Pass | Null |  |

**User\_Account\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | calculateWords  Covered (User user, Label label) | Calculates the percentage of Words that the User covered | ***Input*:**  **User: *User instance***  **Label: F*orm Label*** | Label initialized with the percentage of words covered by the User | The Label was initialized with the percentage of words covered by the User | Pass | Null |  |
| 2.1 | calculateAverage  UserTime (User user) | Calculates the average time the User takes to play a Game | **Input:**  **User: *User instance*** | The User’s average time was calculated in minutes and seconds, then Updating the Form | The User’s average time was calculated in minutes and seconds, then the Form was updated, displaying the User’s average time | Pass | Null |  |
| 3.1 | populateUser  GameDetails (DataGridView dtGridView, User user) | Populates the User’s Game Details DataGridView | ***Input*:**  **Game Details DataGridView**  **User Instance** | The User’s Game DataGridView was populated using the User’s Game Details | The User’s Game DataGridView was populated using the User’s Game Details | Pass | Null |  |

**Register\_Form**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | checkUniqueUsername () | Checking if Username is valid | ***Input*:**  **Username: “”** | Invalid Username – Contains no text | The Username was invalid as it didn’t contain any text | Pass | Null |  |
| 1.2 | checkUniqueUsername () | Checking if Username is valid | ***Input*:**  **Username: “ ”** | Invalid Username – Contains no text | The Username was invalid as it didn’t contain any text | Pass | Null |  |
| 1.3 | checkUniqueUsername () | Checking if Username is valid | ***Input*:**  **Username: “andGuest”** | Invalid Username – Contains the Substring “**Guest**” | The Username was invalid as it contains the Substring “**Guest**” | Pass | Null |  |
| 1.4 | checkUniqueUsername () | Checking if Username is valid | ***Input*:**  **Username: “andreabriffa”** | Invalid Username – Already Exists | The Username is invalid as it already exists | Pass | Null |  |
| 1.5 | checkUniqueUsername () | Checking if Username is valid | ***Input*:**  **Username: “newGuy102”** | Valid Username | The Username is Unique - Valid | Pass | Null |  |
| 2.1 | checkTF (TextBox textfield) | Checks if the textbox contains any text | ***Input*:**  **Text: “”** | Invalid Text | There was no text | Pass | Null |  |
| 2.2 | checkTF (TextBox textfield) | Checks if the textbox contains any text | ***Input*:**  **Text: “ ”** | Invalid Text | There was no text – only spaces | Pass | Null |  |
| 2.3 | checkTF (TextBox textfield) | Checks if the textbox contains any text | ***Input*:**  **Text: “doggo”** | Valid Text | Textbox Contains Text | Pass | Null |  |
| 3.1 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “abcedrd”**  **confirmPassword: “mdsmkjk”**  **Date of Birth: 12/02/2002** | Invalid Inputs – Empty fields are not accepted | The Inputs were invalid | Pass | Null |  |
| 3.2 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “andreabriffa”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “abcedrd”**  **confirmPassword: “mdsmkjk”**  **Date of Birth: 12/02/2002** | Invalid Inputs – Username already exists | The Inputs were Invalid | Pass | Null |  |
| 3.3 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “andreaGuest”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “abcedrd”**  **confirmPassword: “mdsmkjk”**  **Date of Birth: 12/02/2002** | Invalid Inputs – Username cannot contain the substring “Guest” | The Inputs were Invalid | Pass | Null |  |
| 3.4 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “readyPlayer”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “abcedrd”**  **confirmPassword: “mdsmkjk”**  **Date of Birth: 12/02/2002** | Invalid Inputs – Passwords do not match | The Inputs were Invalid | Pass | Null |  |
| 3.5 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “readyPlayer”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “12344321”**  **confirmPassword: “12344321”**  **Date of Birth: 08/05/2019** | Invalid Inputs – Current Date cannot be selected | The Inputs were Invalid | Pass | Null |  |
| 3.6 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “readyPlayer”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “12344321”**  **confirmPassword: “12344321”**  **Date of Birth: 08/05/2019** | Invalid Inputs – A birth date in the future cannot be selected | The Inputs were Invalid | Pass | Null |  |
| 3.7 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “readyPlayer”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “1234321”**  **confirmPassword: “1234321”**  **Date of Birth: 13/07/2002** | Invalid Inputs – Password too Short (requires to be of minimum 8 characters) | The Inputs were invalid | Pass | Null |  |
| 3.8 | btn\_Submit.Click | Player attempting to register as User | ***Input*:**  **Player presses Register button**  **Username: “readyPlayer”**  **firstName: “Andrea”**  **lastName: “Brif”**  **password: “1234321”**  **confirmPassword: “1234321”**  **Date of Birth: 13/07/2002** | Valid Inputs – No inputs are empty. Username is unique and does not contain “Guest”. Passwords match and satisfy minimum length. The Selected Data is in the past | The Inputs were valid | Pass | Null |  |

**Log\_In Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Log\_In () > chck\_Box\_  **Show\_Password.**  **CheckedChanged** | Changing State of Checkbox | ***Input*:**  **Checkbox state: *checked*** | Text Inside Password Textbox shown clearly | The text inside the Password Textbox was shown clearly | Pass | Null |  |
| 1.2 | Log\_In () > chck\_Box\_  **Show\_Password.**  **CheckedChanged** | Changing State of Checkbox | ***Input*:**  **Checkbox state: Un*checked*** | Text Inside Password Textbox Obfuscated | The text inside the Password Textbox was obfuscated | Pass | Null |  |
| 2.1 | Log\_In () > **link\_Register.**  **Click** | Displaying Register Form | ***Input*:**  **Register Button Clicked** | Register Form displayed | The Register Form was displayed | Pass | Null |  |
| 3.1 | Log\_In () > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “”**  **Password: “”** | Inputs do not Match | The inputs did not match | Pass | Null |  |
| 3.2 | Log\_In () > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “andreabriffa”**  **Password: “passwo”** | Inputs do not Match | The inputs did not match | Pass | Null |  |
| 3.3 | Log\_In () > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “andreabriffa”**  **Password: “password1”** | Inputs matched User’s Details | The inputs did not match | Pass | Null |  |
| 4.1 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “”**  **Password: “”** | Inputs do not Match | The inputs did not match | Pass | Null |  |
| 4.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “andreabriffa”**  **Password: “passwo”** | Inputs do not Match | The inputs did not match | Pass | Null |  |
| 4.3 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Login.Click** | Checking whether the inputs match a User’s details | ***Input*:**  **Username: “andreabriffa”**  **Password: “password1”** | Inputs matched User’s Details | The inputs did not match | Pass | Null |  |
| 5.1 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_Guest.Click** | Player registering as Guest | ***Input*:**  **Player clicked Play as Guest Button.**  **Player did not verify to Play as Guest** | Player not registered as Guest | The Player did not register as a Player | Pass | Null |  |
| 5.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Play\_As\_Guest.Click** | Player registering as Guest | ***Input*:**  **Player clicked Play as Guest Button.**  **Player verified to Play as Guest** | Player registered as Guest | The Player was registered as a Guest | Pass | Null |  |
| 6.1 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Player exiting Game Setup | ***Input*:**  **Player pressed close Form Button.**  **Player did not verify to cancel Game Setup** | Game Setup not cancelled | The Game Setup was not cancelled | Pass | Null |  |
| 6.2 | Log\_In (string, addUser, addGuest, breakLogin) > **btn\_Exit.Click** | Player exiting Game Setup | ***Input*:**  **Player pressed close Form Button.**  **Player verified cancel Game Setup** | Game Setup cancelled | The Game Setup was cancelled | Pass | Null |  |
| 7.1 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Determining whether a player has registered as a Guest/User or not | ***Input*:**  **Player did not Register**  **Player verified to cancel Game setup** | Game setup was cancelled | The Game setup was cancelled | Pass | Null |  |
| 7.2 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Determining whether a player has registered as a Guest/User or not | ***Input*:**  **Player did not Register**  **Player did not verify to cancel Game setup** | Game setup was not cancelled | The Game setup was not cancelled | Pass | Null |  |
| 7.3 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Determining whether a player has registered as a Guest/User or not | ***Input*:**  **Player registered as User** | Form Closed. Game setup continued | The Login Form closed, and the Game Setup continued | Pass | Null |  |
| 7.4 | Log\_In (string, addUser, addGuest, breakLogin) > **FormClosing** | Determining whether a player has registered as a Guest/User or not | ***Input*:**  **Player registered as Guest** | Form Closed. Game setup continued | The Login Form closed, and the Game Setup continued | Pass | Null |  |
| 8.1 | Log\_In (string, addUser, addGuest, breakLogin) > chck\_Box\_  **Show\_Password.**  **CheckedChanged** | Changing State of Checkbox | ***Input*:**  **Checkbox state: *checked*** | Text Inside Password Textbox shown clearly | The text inside the Password Textbox was shown clearly | Pass | Null |  |
| 8.2 | Log\_In (string, addUser, addGuest, breakLogin) > chck\_Box\_  **Show\_Password.**  **CheckedChanged** | Changing State of Checkbox | ***Input*:**  **Checkbox state: Un*checked*** | Text Inside Password Textbox Obfuscated | The text inside the Password Textbox was obfuscated | Pass | Null |  |
| 9.1 | Log\_In (string, addUser, addGuest, breakLogin) > **link\_Register.Click** | Opening Register Form | Input:  Register Form Link clicked | Register Form was displayed | The Register Form was displayed | Pass | Null |  |

**Game\_Settings\_Form**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | Language 1: **English**  Language 2: **English** | Invalid Game Settings – Same Languages chosen | The selected Languages were Invalid | Pass | Null |  |
| 1.2 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | Language 1: **English**  Language 2: **Italian** | Valid Languages Chosen | The selected Languages were valid | Pass | Null |  |
| 1.3 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**1**” | Invalid number of Pairs chosen | The number of pairs was invalid | Pass | Null |  |
| 1.4 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**2**” | Valid Number of Pairs chosen | The number of Pairs chosen was valid | Pass | Null |  |
| 1.5 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**10**” | Valid Number of Pairs chosen | The number of Pairs chosen was valid | Pass | Null |  |
| 1.6 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**43**” | Invalid number of Pairs chosen | The number of pairs was invalid | Pass | Null |  |
| 1.7 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**42**” | Valid Number of Pairs chosen | The number of Pairs chosen was valid | Pass | Null |  |
| 1.8 | Btn\_Submit\_Settings.Click | Players submitting Game Settings | **Predecessor**: Valid Languages Chosen  Number of Pairs: “**hgf**” | Invalid input | The input was in invalid format | Pass | Null |  |
| 2.1 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “”** | Error Label visible – Invalid Numerical Input | The error Label was visible | Pass | Null |  |
| 2.2 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “bvc”** | Error Label visible – Invalid Numerical Input | The error Label was visible | Pass | Null |  |
| 2.3 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “1”** | Error Label visible – Numerical Input within invalid range | The error Label was visible | Pass | Null |  |
| 2.4 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “2”** | Error Label invisible – Numerical Input within valid range | The error Label was invisible indicating no errors | Pass | Null |  |
| 2.5 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “30”** | Error Label invisible – Numerical Input within valid range | The error Label was invisible indicating no errors | Pass | Null |  |
| 2.6 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “42”** | Error Label invisible – Numerical Input within valid range | The error Label was invisible indicating no errors | Pass | Null |  |
| 2.7 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “43”** | Error Label visible – Numerical Input within invalid range | The error Label was visible | Pass | Null |  |
| 2.8 | tf\_Input\_Pairs.TextChanged | Validating Number of Pairs Input | ***Input*:**  **Value: “100”** | Error Label visible – Numerical Input within invalid range | The error Label was visible | Pass | Null |  |
| 3.1 | FormClosing | Closing Game Settings Form | ***Input*:**  **Game Settings submitted** | Form Closed – Game Settings Submitted | The Game Settings Form closed | Pass | Null |  |
| 3.2 | FormClosing | Closing Game Settings Form | ***Input*:**  **Game Settings not submitted** | Player verification Required as Game Settings were not Submitted | The Player was prompted to verify cancelling the Game setup | Pass | Null |  |

**Statistics Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | updateUserSuccess  DtGridView (int user\_Index, DataGridView dtGridView) | Update the User DataGridView section | ***Input***:  *Number of Users: 3*  user\_Index\_Selected: **0**  DataGridView: **UserDataGridView** | User DataGridView updated | The User DataGridView was updated | Pass | Null |  |
| 1.2 | updateUserSuccess  DtGridView (int user\_Index, DataGridView dtGridView) | Update the User DataGridView section | ***Input***:  *Number of Users: 3*  user\_Index\_Selected: **2**  DataGridView: **UserDataGridView** | User DataGridView updated | The User DataGridView was updated | Pass | Null |  |
| 2.1 | updateWordSet (int language\_Index) | Updating the Words Drop Down List | ***Input***:  Number of Languages: **3**  Language\_Index Selected: **0** | Words Drop Down Menu updated | The Words Drop Down Menu was updated | Pass | Null |  |
| 2.2 | updateWordSet (int language\_Index) | Updating the Words Drop Down List | ***Input***:  Number of Languages: **3**  Language\_Index Selected: **2** | Words Drop Down Menu updated | The Words Drop Down Menu was updated | Pass | Null |  |
| 3.1 | populateGameSummaries  DtGridView (DataGridView dtGridView) | Populating the Game Summaries DataGridView | ***Input***:  DataGridView: **GameSummaries**  **DataGridView** | Game Summaries DataGridView populated | The Game summaries DataGridView was populated | Pass | Null |  |
| 4.1 | Populate\_Groups () | Updated the Form statistics | ***Input***:  Class Constructor Executed | The Form statistics were updated | The Form statistics were updated | Pass | Null |  |
| 5.1 | populateUser  RankDtGridView (DataGridView dtGridView) | Updating the User Ranks DataGridView | ***Input***:  DataGridView: **User Ranks DataGridView**  Number of Users: **3** | The User Ranks DataGridView was updated | The User Ranks DataGridView was updated | Pass | Null |  |
| 5.2 | populateUser  RankDtGridView (DataGridView dtGridView) | Updating the User Ranks DataGridView | ***Input***:  DataGridView: **User Ranks DataGridView**  Number of Users: 0 | The User Ranks DataGridView was left empty | The User Ranks DataGridView was left empty | Pass | Null |  |
| 6.1 | updateUserSuccessWord (int user\_Index, int lang\_Index, int word\_Index) | Updating the Users Success Word Section | ***Input***:  User\_Index: **0**  Lang\_Index: **0**  Word\_Index: **0** | User SuccessWord details updated | The User success word details were updated | Pass | Null |  |

## **Integration Testing**

**Welcome\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1.1 | Welcome\_Form () | Retrieving List of saved Languages from File | Languages File does not exist | Default Languages Retrieved –  **2 Languages** | Empty List of Languages retrieved | **Fail** | The Default Languages were not being written to the Languages File in the case the file did not exist/was empty |  |
| 1.1.2 | Welcome\_Form () | Retrieving List of saved Languages from File | Languages File does not exist | Default Languages Retrieved –  **2 Languages** | The Default Languages were retrieved –  **2 Languages** | Pass | Null |  |
| 1.2 | Welcome\_Form () | Retrieving List of saved Languages from File | Languages File exists > Contains 3 Languages | Languages Retrieved –  **3 Languages** | Languages were retrieved –  **3 Languages** | Pass | Null |  |
| 2.1 | Welcome\_Form () | Retrieving List of Backed Up Languages from File | Backup Languages File does not exist | Backed Up Languages Retrieved –  **0 Languages** | Backed Up Languages were retrieved –  **0 Languages** | Pass | Null |  |
| 2.2 | Welcome\_Form () | Retrieving List of Backed Up Languages from File | Backup Languages File exists > Contains 0 Languages | Backed Up Languages Retrieved –  **0 Languages** | Backed Up Languages were retrieved –  **0 Languages** | Pass | Null |  |
| 2.3 | Welcome\_Form () | Retrieving List of Backed Up Languages from File | Backup Languages File exists > Contains 3 Languages | Backed Up Languages Retrieved –  **3 Languages** | The Backed Up Languages were retrieved –  **3 Languages** | Pass | Null |  |
| 3.1 | Welcome\_Form () | Retrieving List of existing Users from Users File | Users File does not exist | 0 Users Retrieved | 0 Users were retrieved | Pass | Null |  |
| 3.2 | Welcome\_Form () | Retrieving List of existing Users from Users File | Users File exists  Number of Saved Users: 0 | 0 Users Retrieved | 0 Users were retrieved | Pass | Null |  |
| 3.3 | Welcome\_Form () | Retrieving List of existing Users from Users File | Users File exists  Number of Saved Users: 3 | 3 Users Retrieved | 3 Users were retrieved | Pass | Null |  |
| 4.1 | Welcome\_Form () | Retrieving List of saved Game Records from Games File | **Games File does not exist** | 0 Game Records Retrieved | 0 Game Records were retrieved | Pass | Null |  |
| 4.2 | Welcome\_Form () | Retrieving List of saved Game Records from Games File | Games File exists > **Contains 0 Game Records** | 0 Game Records Retrieved | 0 Game Records were retrieved | Pass | Null |  |
| 4.3 | Welcome\_Form () | Retrieving List of saved Game Records from Games File | Games File exists > **Contains 1 Game Records** | 1 Game Records Retrieved | 1 Game Records were retrieved | Pass | Null |  |
| 4.4 | Welcome\_Form () | Retrieving List of saved Game Records from Games File | Games File exists > **Contains 10 Game Records** | 10 Game Records Retrieved | 10 Game Records were retrieved | Pass | Null |  |

**Library Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | readFile<Language> getLanguages () | Writing initial Languages in case Languages File does not exist | Languages File does not exist | Initial Languages created in Languages File | The initial Languages were created in the Languages File | Pass | Null |  |
| 1.2 | readFile<Language> getLanguages () | Function called itself (Recursive) – Retrieving List of Saved Languages | Initial List of Languages saved within Languages File | List of Languages retrieved > **2 Languages** | The List of Languages was retrieved > **2 Languages** | Pass | Null |  |
| 2.1 | writeInitialLanguages () | Writing initial Languages to Languages File | Number of initial List of Languages | List of Languages written to Languages File | The Initial List of Languages was written to Languages File | Pass | Null |  |
| 3.1 | appendFile<User> appendUser (User user) | Retrieving List of saved Users from Users File | Number of Saved Users: **3** | List of saved Users retrieved from Users File | The List of Users was retrieved from the File | Pass | Null |  |
| 3.2 | appendFile<User> appendUser (User user) | Saving List of Users to Users File | Number of Users: 4 | List of Users saved to File | The List of Users was saved to the Users File | Pass | Null |  |
| 3.3 | appendFile<User> appendUser (User user) | Retrieving List of saved Users from Users File | Number of Saved Users: **0** | List of saved Users retrieved from Users File | The List of Users was retrieved from the File | Pass | Null |  |
| 3.4 | appendFile<User> appendUser (User user) | Saving List of Users to Users File | Number of Users: **1** | List of Users saved to File | The List of Users was saved to the Users File | Pass | Null |  |
| 3.5 | appendFile<User> appendUser (User user) | Retrieving List of saved Users from Users File | Users File does not exist | Empty List of Users retrieved | An empty List of Users was retrieved | Pass | Null |  |
| 3.6 | appendFile<User> appendUser (User user) | Saving List of Users to Users File | Number of Users: **1** | List of Users saved to File | The List of Users was saved to the Users File | Pass | Null |  |
| 4.1 | appendFile<Game  Record> appendGame | Retrieving List of Game Records from Games File | Games File does not exist | Empty List of Game Records retrieved | An empty List of Game Records was retrieved | Pass | Null |  |
| 4.2 | appendFile<Game  Record> appendGame | Saving List of Game Records to Games File | Number of Game Records: 1 | List of Game Records saved to File | The list of Game Records was saved to the Games File | Pass | Null |  |
| 4.3 | appendFile<Game  Record> appendGame | Retrieving List of Game Records from Games File | Number of Saved Game Records: **0** | Empty List of Game Records retrieved | An empty List of Game Records was retrieved | Pass | Null |  |
| 4.4 | appendFile<Game  Record> appendGame | Saving List of Game Records to Games File | Number of Game Records: 1 | List of Game Records saved to File | The list of Game Records was saved to the Games File | Pass | Null |  |
| 4.5 | appendFile<Game  Record> appendGame | Retrieving List of Game Records from Games File | Number of Saved Game Records: **10** | List of Game Records retrieved from File | The List of Game Records was retrieved from the Games File | Pass | Null |  |
| 4.6 | appendFile<Game  Record> appendGame | Saving List of Game Records to Games File | Number of Game Records: **11** | List of Game Records saved to File | The list of Game Records was saved to the Games File | Pass | Null |  |

**Game Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | buttonClicked (Button\_Custom button) | Updating the Players’ Score Labels | Player selected two consecutive buttons that are from the same Pair | Players’ score labels updated | The Players’ score labels were updated | Pass | Null |  |
| 2.1 | buttonClicked (Button\_Custom button) | Adding the Game Record to the Game Records File | No button Pairs left – Game Finished | Game Record added to existing Game Records saved in Games File | The Game Record was added to existing Game Records saved in Games File | Pass | Null |  |
| 3.1 | buttonClicked (Button\_Custom button) | Switching the Current Player | Current Player: Player 1 | Current Player switched to Player 2 | The Current Player was switched to Player 2 | Pass | Null |  |
| 3.2 | buttonClicked (Button\_Custom button) | Switching the Current Player | Current Player: Player 2 | Current Player switched to Player 1 | The Current Player was switched to Player 1 | Pass | Null |  |

**Game\_Settings\_Form Class**

No methods/functions called each other in the Game Settings Form. No Integration Tests were carried out.

**Register\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | btn\_Submit.Click | Checking if a TextField contains text or not | TextField text: “” | Output: **false** – No valid text entered | The Output: **false** | Pass | Null |  |
| 1.2 | btn\_Submit.Click | Checking if a TextField contains text or not | TextField text: “ ” | Output: **false** – No valid text entered | The Output: **false** | Pass | Null |  |
| 1.3 | btn\_Submit.Click | Checking if a TextField contains text or not | TextField text: “jonny” | Output: **true** – Valid text entered | The Output: **true** | Pass | Null |  |
| 2.1 | btn\_Submit.Click | Checking if the username is unique | ***Predecessor*:**  **Username textbox not empty**  Username: “**andreabriffa**” | Output: **false** – Username already exists | The output was **false** | Pass | Null |  |
| 2.2 | btn\_Submit.Click | Checking if the username is unique | ***Predecessor*:**  **Username textbox not empty**  Username: “**jonny**” | Output: **true** – Username does not exist | The output was **true** | Pass | Null |  |
| 3.1 | btn\_Submit.Click | Appending the User to the existing Users | ***Input*:**  **New User instance** | User appending to existing Users | The registered User was appended to the existing Users | Pass | Null |  |
| 4.1 | btn\_Submit.Click | Updating the current Users | Input:  Users saved in Users file | The current Users were replaced with the Users saved in the Users File | Current Users updated | Pass | Null |  |

**Statistics Form**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | populateUser  RanksDtGridView (DataGridView dtGridView) | Updating the Current Users | Users stored in Library: **1** | Current Users updated from the Users File | The current Users were updated | Pass | Null |  |
| 1.2 | populateUser  RanksDtGridView (DataGridView dtGridView) | Updating the Current Users | Users stored in Library: **5** | Current Users updated from the Users File | The current Users were updated | Pass | Null |  |
| 2,1 | populateUser  RanksDtGridView (DataGridView dtGridView) | Transitioning a numerical value inside a Label | Numerical Value: **0**  Label: l**bl\_Word\_Count** | Numerical Value inside text transitioned | The Numerical Value inside the Label was set directly to **0** | Pass | Null |  |
| 2.2 | populateUser  RanksDtGridView (DataGridView dtGridView) | Transitioning a numerical value inside a Label | Numerical Value: **0**  Label: l**bl\_Word\_Count** | Numerical Value inside text transitioned | The Numerical Value inside the Label was transitioned | Pass | Null |  |
| 2.3 | populateUser  RanksDtGridView (DataGridView dtGridView) | Transitioning a numerical value inside a Label | Numerical Value: **0**  Label: l**bl\_Word\_Count** | Numerical Value inside text transitioned | The Numerical Value inside the Label was transitioned | Pass | Null |  |

**Modify\_Word\_Sets Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | btn\_Save\_  Language.Click | Saving the Current Languages | ***Predecessor*:**  **Valid Languages/ Word Sets entered inside DataGridView**  ***Input*:**  **2 Languages** | Languages present inside DataGridView saved to Languages File | The Languages entered inside the DataGridView were saved to the Languages File | Pass | Null |  |
| 1.2 | btn\_Save\_  Language.Click | Saving the Current Languages | ***Predecessor*:**  **Valid Languages/ Word Sets entered inside DataGridView. Row Deleted**  ***Input*:**  **3 Languages** | Languages present inside DataGridView saved to Languages File | The Languages entered inside the DataGridView were saved to the Languages File | Pass | Null |  |
| 2.1 | btn\_Delete\_Row.Click | Saving the Current Languages, following deletion of Row | ***Predecessor*:**  **Valid Languages/ Word Sets entered inside DataGridView. Row Deleted**  ***Input*:**  **2 Languages** | Languages present inside DataGridView saved to Languages File | The Languages entered inside the DataGridView were saved to the Languages File | Pass | Null |  |
| 2.2 | btn\_Delete\_Row.Click | Saving the Current Languages, following deletion of Row | ***Predecessor*:**  **Valid Languages/ Word Sets entered inside DataGridView**  ***Input*:**  **3 Languages** | Languages present inside DataGridView saved to Languages File | The Languages entered inside the DataGridView were saved to the Languages File | Pass | Null |  |
| 3.1 | Btn\_Delete.Click | Deleting a Language | ***Predecessor*:**  **More than 2 existing Languages**  ***Input*:**  **Current Number of Languages: 3** | Language Deleted | The Language was successfully Deleted | Pass | Null |  |
| 3.2 | Btn\_Delete.Click | Deleting a Language | ***Predecessor*:**  **More than 2 existing Languages**  ***Input*:**  **Current Number of Languages: 4** | Language Deleted | The Language was successfully Deleted | Pass | Null |  |
| 4.1 | Btn\_Restore.Click | Writing Backup Languages to Languages File | ***Input*:**  **List of Languages** | Languages File overwritten with Backup Languages | The Languages File was overridden with the Backup Languages | Pass | Null |  |
| 5.1 | Btn\_Restore\_Default  \_Languages.Click | Writing initial Languages to Languages File | ***Input*:**  **List of Languages** | Languages File overridden with initial Languages | The Languages File was overwritten with the initial Languages | Pass | Null |  |
| 6.1 | FormClosing | Saving the Languages when Form is closing | ***Predecessor*:**  **User verified to Save Changes made. Word Sets valid for saving.**  ***Input*:**  **List of Languages** | Current Languages within DataGridView saved to Languages File | The current languages present in the DataGridView were saved to the Languages File | Pass | Null |  |
| 7.1 | saveLanguages () | Saving the Current Languages | **Predecessor:**  **Languages present in DataGridView are valid**  ***Input*:**  **List of Languages present in DataGridView** | Current Languages present inside the Modify Word Set DataGridView were saved to Languages File | The current Languages present inside the DataGridView were saved to the Languages File | Pass | Null |  |
| 7.2 | saveLanguages () | Updating current list of Languages | **Predecessor:**  **Languages present in DataGridView are valid**  ***Input*:**  **Current List of Languages updated with Languages present in DataGridView** | Current Languages present inside the Modify Word Set DataGridView were set as the Current Languages | The current Languages present inside the DataGridView were set as the Current Languages | Pass | Null |  |
| 8.1 | DeleteLanguage () | **Removing selected Language from List of Saved Languages** | ***Predecessor*:**  **Language word sets are valid. More than 2 remaining Languages**  ***Input*:**  **List of remaining Languages** | Languages saved in File updated | The languages that were saved in the Languages File were updated | Pass | Null |  |

**User\_Account\_Form Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | User\_Account\_Form (User user) | Transitioning numerical Value inside Label | Numerical Value: **0** | Label’s text set directly to 0 | The Label’s text was set directly to 0 | Pass | Null |  |
| 1.2 | User\_Account\_Form (User user) | Transitioning numerical Value inside Label | Numerical Value: 1 | Numerical Value transitioned inside Label | The Numerical value transitioned inside the Label | Pass | Null |  |
| 1.3 | User\_Account\_Form (User user) | Transitioning numerical Value inside Label | Numerical Value: **22** | Numerical Value transitioned inside Label | The Numerical value transitioned inside the Label | Pass | Null |  |
| 2.1 | User\_Account\_Form (User user) | Populating the User’s Game Details | Input:  DataGridView: **dtGridViewGame**  User: **user** | User’s Game Details displayed inside the DataGridView | The User’s Game Details were displayed inside the DataGridView | Pass | Null |  |
| 3.1 | User\_Account\_Form (User user) | Calculating the User’s average Game time | User: **user** | User’s average time was calculated | The User’s average time was calculated | Pass | Null |  |
| 4.1 | User\_Account\_Form (User user) | Calculating the Words covered by the User | User: **user**  Label: **lbl\_Words\_Covered** | The total percentage of Words covered by the User was calculated | The total percentage of Words covered by the User was calculated | Pass | Null |  |

**Log\_In Class**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test**  **ID** | **Method/**  **Function** | **Test Variables** | **Environmental Conditions** | **Expected Outcome** | **Actual Outcome** | **Pass/**  **Fail** | **Comments** | **Snippets** |
| 1.1 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > **FormClosing ()** | Cancelling the Game Setup | Player confirmed to Cancel Game Setup | Game Setup Cancelled | The Game Setup was cancelled | Pass | Null |  |
| 2.1 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > Check\_Box\_Show  \_Password  .**CheckedChanged** | Obfuscating/Showing Clearly the text inside the Password Textbox | Checkbox State: **Checked** | Text inside TextBox shown clearly | The text inside the textbox was shown clearly | Pass | Null |  |
| 2.2 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > Check\_Box\_Show  \_Password  .**CheckedChanged** | Obfuscating/Showing Clearly the text inside the Password Textbox | Checkbox State: **Unchecked** | Text inside Checkbox Obfuscated | The text inside the textbox was Obfuscated | Pass | Null |  |
| 3.1 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > **Btn\_LogIn.Click** | Appending User to Playing Users’ List | Input:  User instance | User added to Playing Users’ List | The User was added to the Playing Users’ List | Pass | Null |  |
| 4.1 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > **Btn\_Play\_As\_Guest.Click** | Appending Guest to Playing Guests’ List | Input:  Guest instance | Guest added to Playing Guests’ List | The Guest was added to the Playing Guests’ List | Pass | Null |  |
| 5.1 | Log\_In (string playerNo, addUser appendUser, addGuest appendGuest, breakLogin breakGame) > **FormClosing ()** | Cancelling the Game Setup | Player confirmed to Cancel Game Setup | Game Setup Cancelled | The Game Setup was cancelled | Pass | Null |  |
| 6.1 | Log\_In () > Check\_Box\_Show  \_Password  .**CheckedChanged** | Obfuscating/Showing Clearly the text inside the Password Textbox | Checkbox State: **Checked** | Text inside TextBox shown clearly | The text inside the textbox was shown clearly | Pass | Null |  |
| 6.2 | Log\_In () > Check\_Box\_Show  \_Password  .**CheckedChanged** | Obfuscating/Showing Clearly the text inside the Password Textbox | Checkbox State: **Unchecked** | Text inside Checkbox Obfuscated | The text inside the textbox was Obfuscated | Pass | Null |  |

## **Test Summary**

Before testing began, the most possible information about the use and scope of the system was acquired from the document provided.

The testing was carried out with all scenarios in mind when testing every function/test case. This will expose the task to all possible scenarios. Test cases were also given inputs and scenarios where they were most likely to fail. Each test was carried out using a Test Table.

When a failure occurred, it was documented, along with the changes made.

Overall, the testing led to implementing measures and fixes to the revealed system’s vulnerabilities. This does not mean that the system is defect-free.

# **Conclusion**

Through Testing, the system’s faults were revealed and fixed, then tested again to verify that the fault was fixed.

The system is documented effectively to aid the readability and maintenance of the system.

A high-level description of the system was developed in the class diagram, accompanied by a detailed description.

The Scope of the Game is to make the players think and remember, instead of just guessing and hoping they would match the button pairs.