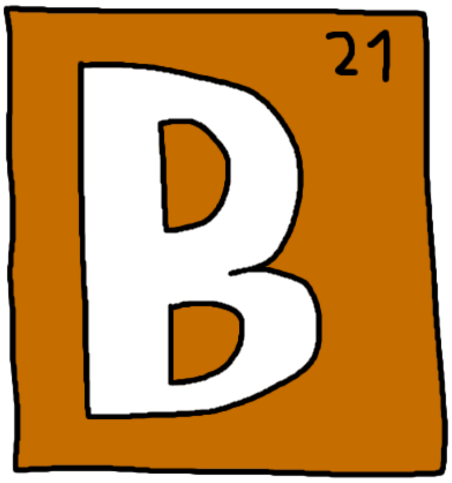
Software Design and Development
Boggle Game Project Report



List of Contents

Section A – Project Management Report

1. Problem Definition

The problem was to design a game for a software development assignment. This was to be done with the use of Visual Studio as an IDE (Integrated Development Environment) and the Visual Basic .NET Framework as the coding language. The game had to be either Boggle, Yahtzee, a Memory Puzzle, or other word-based game. The solution must also contain the following skills which we had learnt in class: calculations, case statements, random number generator, use of timer, simple graphic use, and word count.

2. Ideas and Target Audience

Target Audience

The target audience would primarily be towards students of all ages. It would also be ideal if the game could also be played by anyone regardless of their age. As such, creating a game that is easy to play combined with a simple User Interface would be key in targeting as large an audience as possible.

Ideas

A number of ideas for the application included:

- Various games to choose from
  - At first, 3 games were planned to be included. But as the scope of the project increased with planned difficulties for each game, only 2 games would be implemented instead.
- Different difficulties for each game
  - This would offer the user different ways to play the games as well as an added challenge if they wished.
- Saving of top scores
  - The best scores for each game and difficulty would be saved, and the user would be promoted to break their old high scores.
- A way to allow the user to customise their games
  - In order to add customizability, the user would be able to change different aspects of the game (within reason).
- Different UI themes
  - Another way to add customizability to the entire program. Different colour themes would be available to choose from, which would be complemented by the rest of the program.
- Audio that could be played in the background and as feedback to the user
  - This idea was added extremely lately into the development of the software (as can be seen on the Gantt Chart). After some thought, it was decided that having audio would help with making the software feel less “empty”

3. Screen Design Sketches

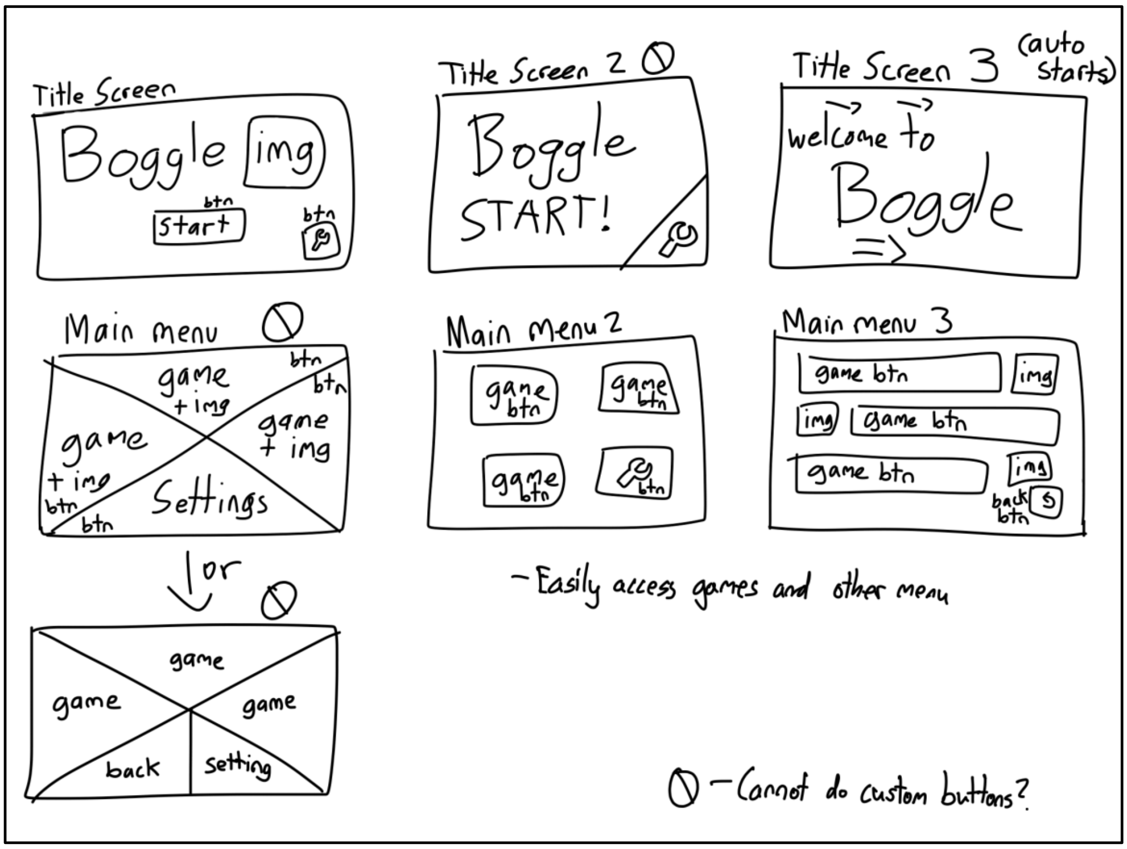
Personal Tips

Various points were kept in mind when designing the games and User Interface:

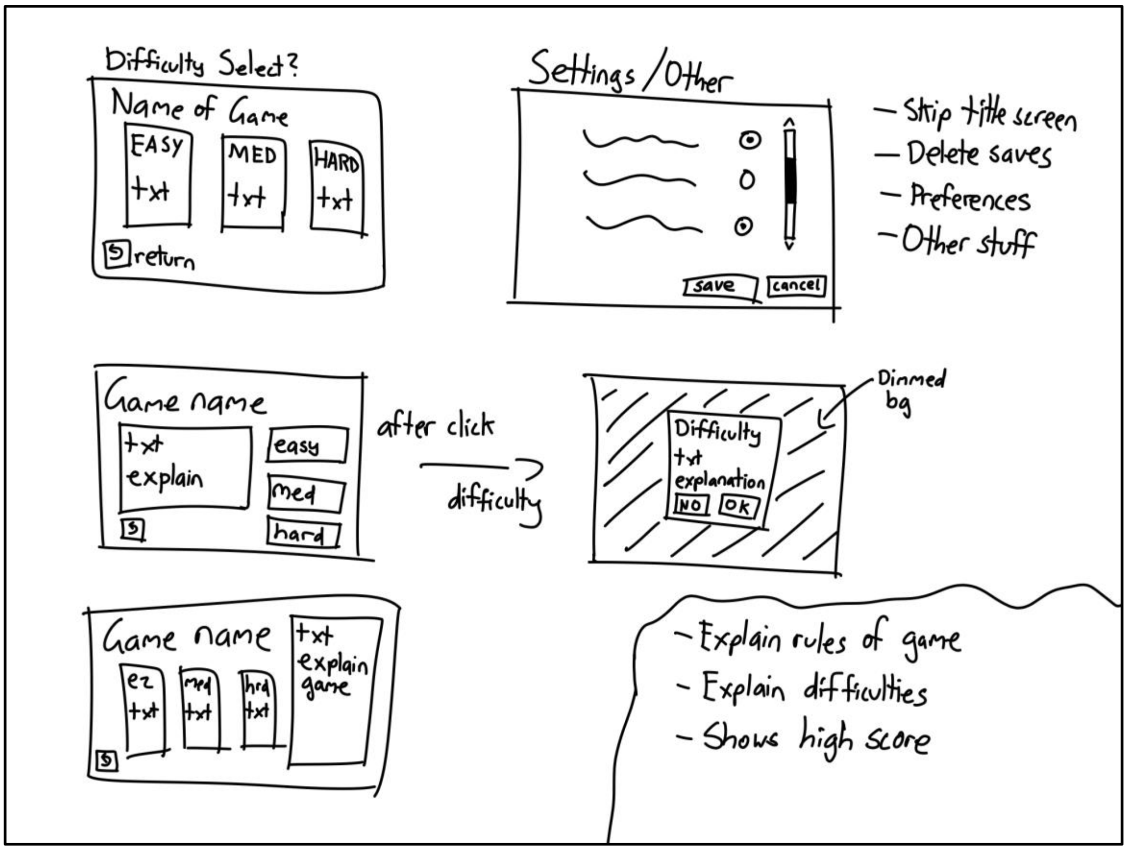
- Keep it within the scope of Windows Forms
- Keep things simple, concise, and clear
- Don't make people "re-learn things'
- Try not overload information
- Bigger things = more important
- Give feedback so people know something is happening
- Anticipate mistakes, and make ways to reverse it

Sketches of the Software

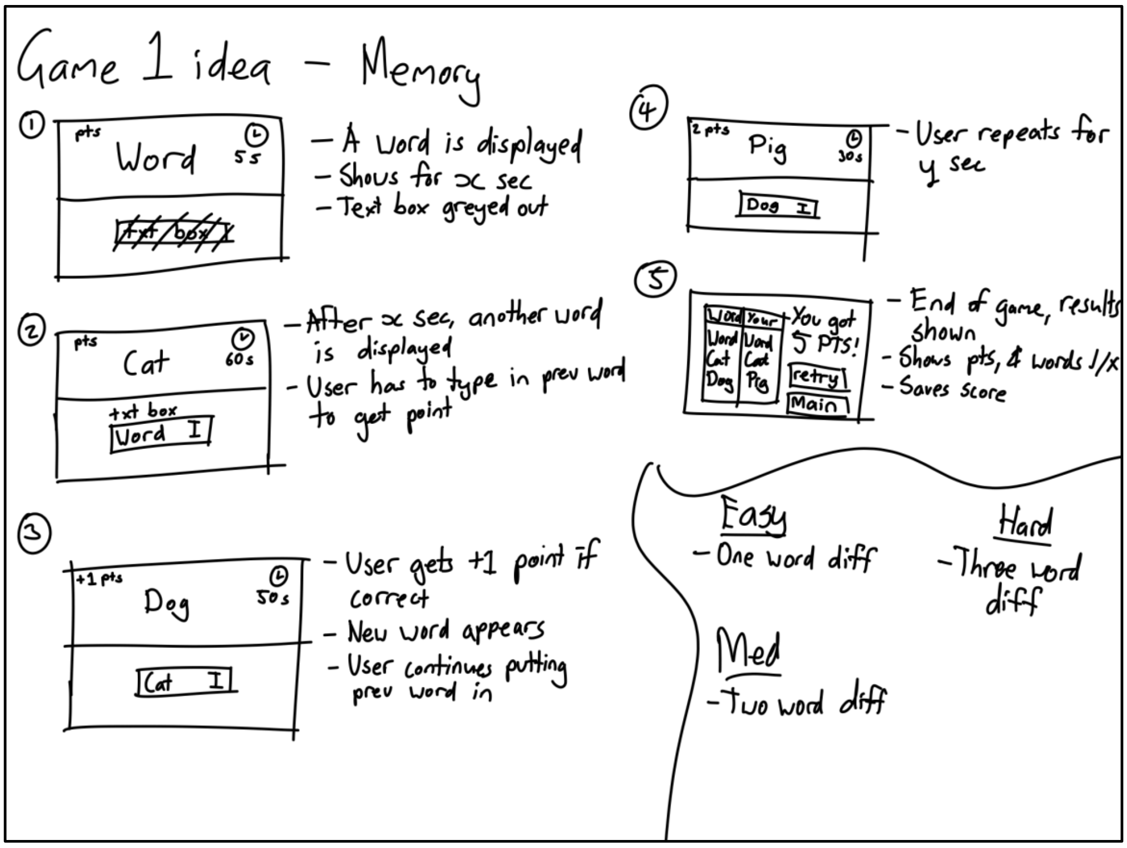
Various sketches had been made on how the UI would look and how each game would be played. Some of the UI sketches took inspiration from a number of popular video games including Super Smash Bros. and Persona 5, which helped with the layout of certain features. The sketches are shown below:



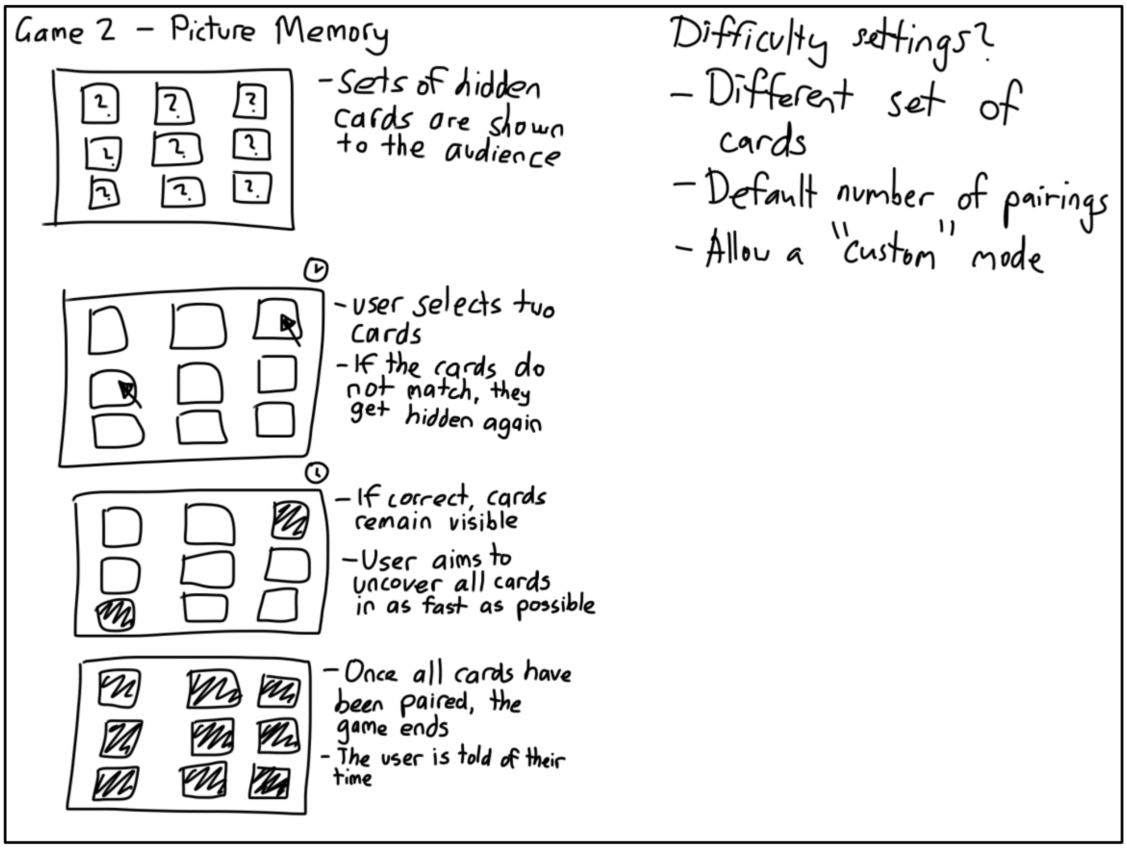
Title Screen and Main Menu



Difficulty Select and Settings



A memory game based on words

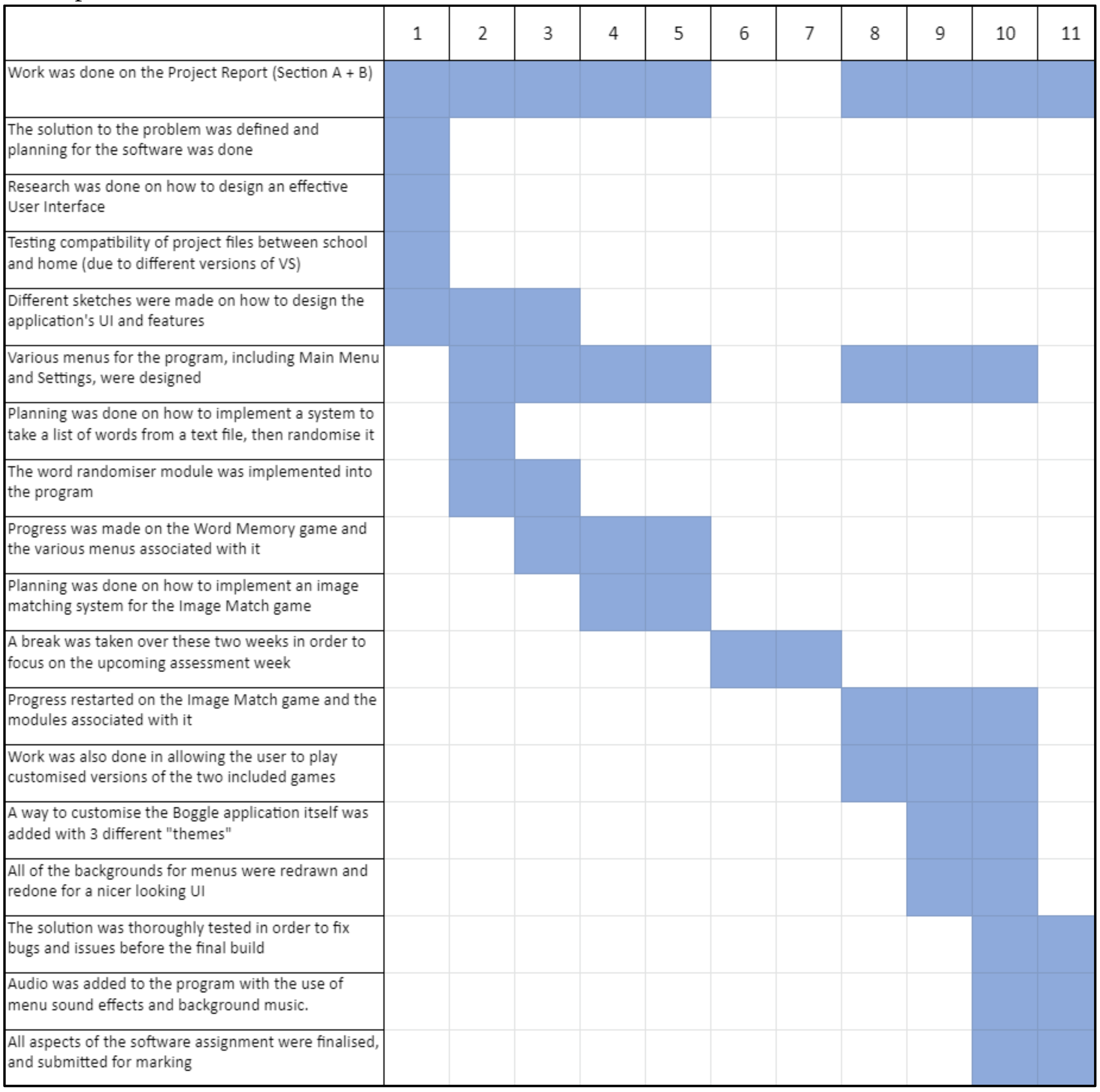

Game 2 – Picture Memory
-Sets of hidden cards are shown to the audience
-user selects two cards
-If the cards do not match, they get hidden again
-If correct, cards remain visible
-User aims to uncover all cards in as fast as possible
-Once all cards have been paired, the game ends
-The user is told of their time
Difficulty settings?
– Different set of cards
– Default number of pairings
– Allow a "custom" mode

A memory game based on images

Sketches were also made to help understand some of the code before they were used. Due to their resolution, it would be impractical to add them straight to this document. If you would like to view these images, please go to the \Documentation\Extra Images\ folder.

4. Gantt Chart

A screenshot of the Gantt Chart is included in this document. If you wish to view the full chart, please refer to the Gantt Chart document in \Documentation\.



	1	2	3	4	5	6	7	8	9	10	11
Work was done on the Project Report (Section A + B)	🟦	🟦	🟦	🟦	🟦			🟦	🟦	🟦	🟦
The solution to the problem was defined and planning for the software was done	🟦										
Research was done on how to design an effective User Interface	🟦										
Testing compatibility of project files between school and home (due to different versions of VS)	🟦										
Different sketches were made on how to design the application's UI and features	🟦	🟦	🟦								
Various menus for the program, including Main Menu and Settings, were designed		🟦	🟦	🟦	🟦			🟦	🟦	🟦	
Planning was done on how to implement a system to take a list of words from a text file, then randomise it		🟦									
The word randomiser module was implemented into the program		🟦	🟦								
Progress was made on the Word Memory game and the various menus associated with it			🟦	🟦	🟦						
Planning was done on how to implement an image matching system for the Image Match game				🟦	🟦						
A break was taken over these two weeks in order to focus on the upcoming assessment week						🟦	🟦				
Progress restarted on the Image Match game and the modules associated with it								🟦	🟦	🟦	
Work was also done in allowing the user to play customised versions of the two included games								🟦	🟦	🟦	
A way to customise the Boggle application itself was added with 3 different "themes"									🟦	🟦	
All of the backgrounds for menus were redrawn and redone for a nicer looking UI									🟦	🟦	
The solution was thoroughly tested in order to fix bugs and issues before the final build										🟦	🟦
Audio was added to the program with the use of menu sound effects and background music.										🟦	🟦
All aspects of the software assignment were finalised, and submitted for marking										🟦	🟦

Section B – Implementing the Solution

1. Software Approach

The software approach which has been chosen for the solution is the RAD Approach.

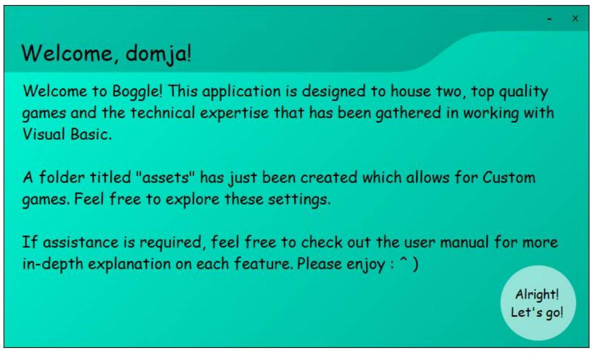
The RAD (Rapid Application Development) Approach is defined as being able to create a usable software version in the shortest time possible, with as little cost to the client. It has a lack of formal, distinct stages. And developers often reuse existing modules and routines to save time and money. Computer Aided Software Engineering (CASE) Tools are also utilised which assists the developers in producing the solution.

This approach was used for this solution as many of its criterias were closely aligned with the software assignment in mind. There were strict time constraints in order to submit our solution by a certain deadline, and there was not a budget allocated towards the solution. Formal stages were also not followed as different parts of development would have been occurring at the same time, as seen with the Gantt Chart. Issues were essentially being dealt with as they arose. Using the internet helped with different modules and resources that were adapted to work for the solution. CASE Tools such as Visual Studio were also heavily dependent upon which greatly assisted in terms of language syntax and organisation of code.

Since many of the requirements of the assignment were closely correlated with the RAD Approach, it was the reason this approach was used for the solution.

2. Solution to the Problem

The Menus

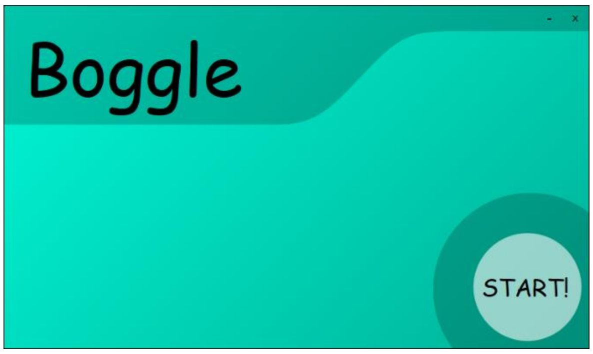


Welcome, domja!

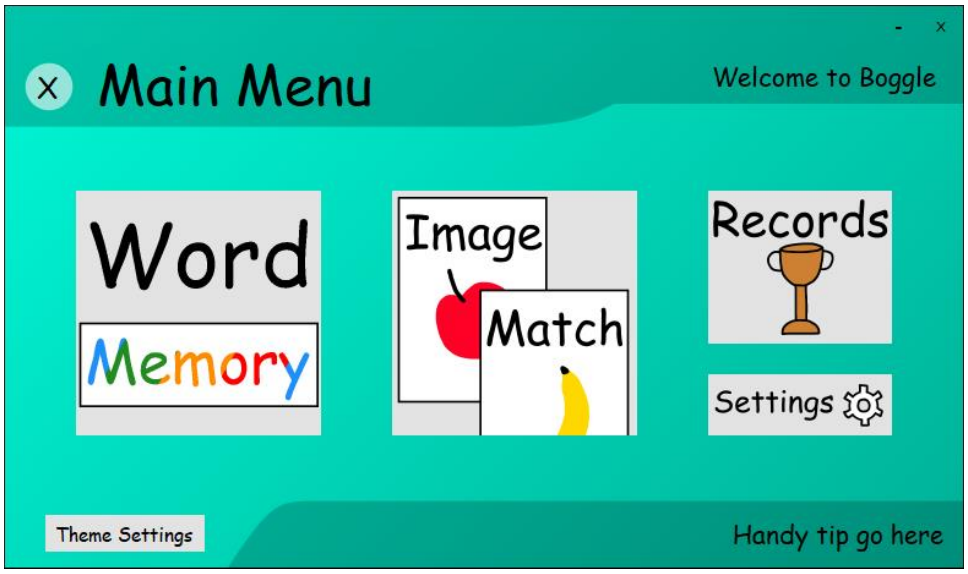
Welcome to Boggle! This application is designed to house two, top quality games and the technical expertise that has been gathered in working with Visual Basic.

A folder titled "assets" has just been created which allows for Custom games. Feel free to explore these settings.

If assistance is required, feel free to check out the user manual for more in-depth explanation on each feature. Please enjoy : ^)

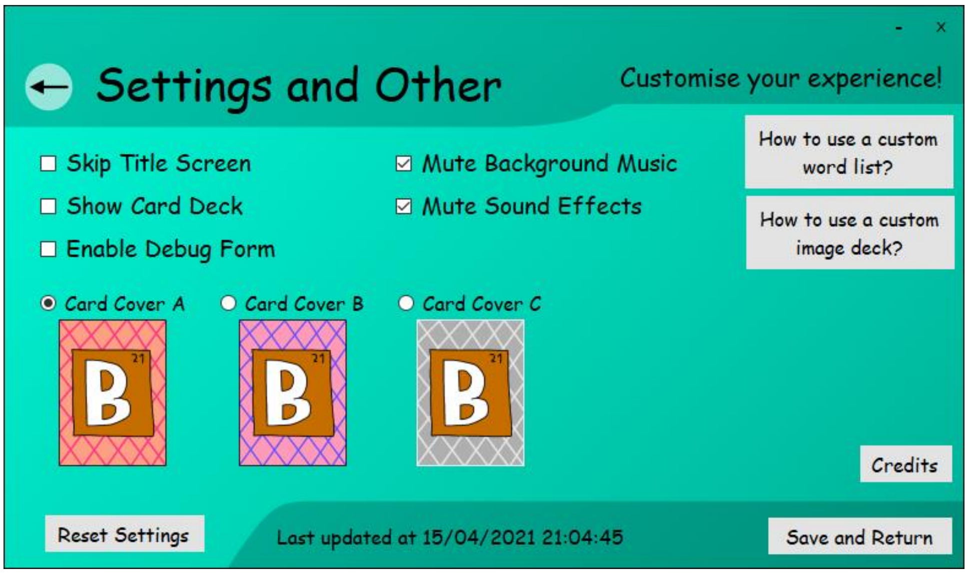


In order to present a compelling software that is used by everyone, the software's menus must be clear, concise, and easy to use. As such, when the game loads, a Welcome screen greets the users with some suggestions on how to use the software. This screen will only appear once, or after the settings are fully reset. Afterwards, they are sent to the Title Screen, and then the Main Menu.



On the Main Menu, all of the buttons have been specifically sorted by size in order of importance. Front and center are the two main attractions of the software, the Word Memory and Image Match game. This is to show the user that these games are the most important part of the software, which is also the case with this assignment. Followed by those games are Records (which displays the user's high scores) and then the Settings and Theme Settings.

Note that every essential menu contains a minimise and close button. These buttons are contained in a custom-made drag bar so that the user can also move the form anywhere. This drag bar exists on all forms and is indicated when the user hovers over it.



Settings and Other

Customise your experience!

- [ ] Skip Title Screen
- [ ] Show Card Deck
- [ ] Enable Debug Form
- [x] Mute Background Music
- [x] Mute Sound Effects

[✓ Card Cover A] Card Cover B Card Cover C

Last updated at 15/04/2021 21:04:45

How to use a custom word list?
How to use a custom image deck?
Credits
Reset Settings
Save and Return

The Settings menu has been sorted by grouping similar settings together. In the top left quadrant are the toggle and audio settings. Underneath them are the Card Colour settings. To the right are Custom Game settings and Credits. There is also an option to reset all settings if the user wishes to.

Both of the Custom game settings include instructions on what is required of a Custom game. Due to the language that is used with these instructions, less advanced users may find them a little confusing. However the software will still automatically create and open the required files for the user and ease of use.

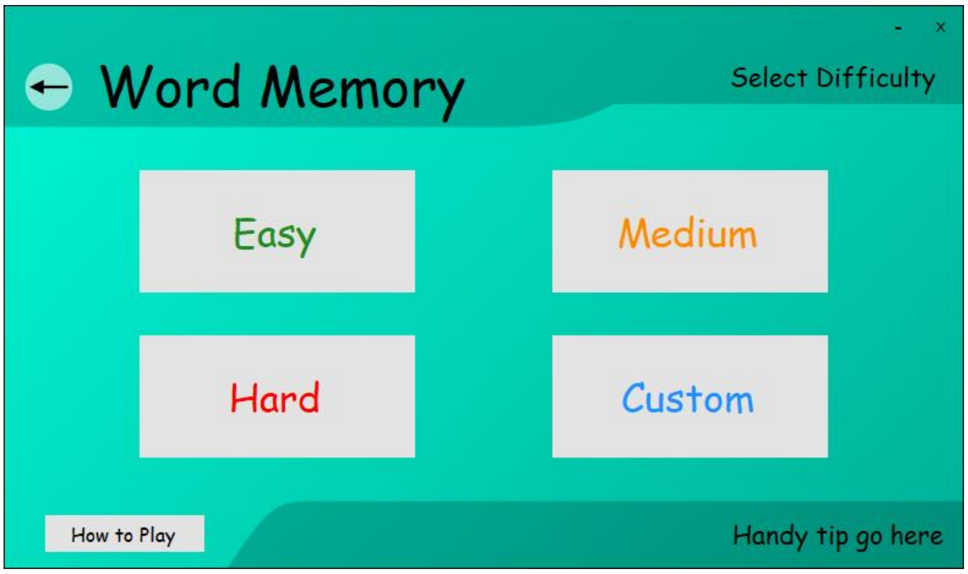
The Games

In order to solve the problem, a game must be developed that would tick off the criterias. Hence it was decided that having 2 games would be best suited for showing off the skills that have been gathered in working with Visual Basic. As such, the two games that have been chosen was:

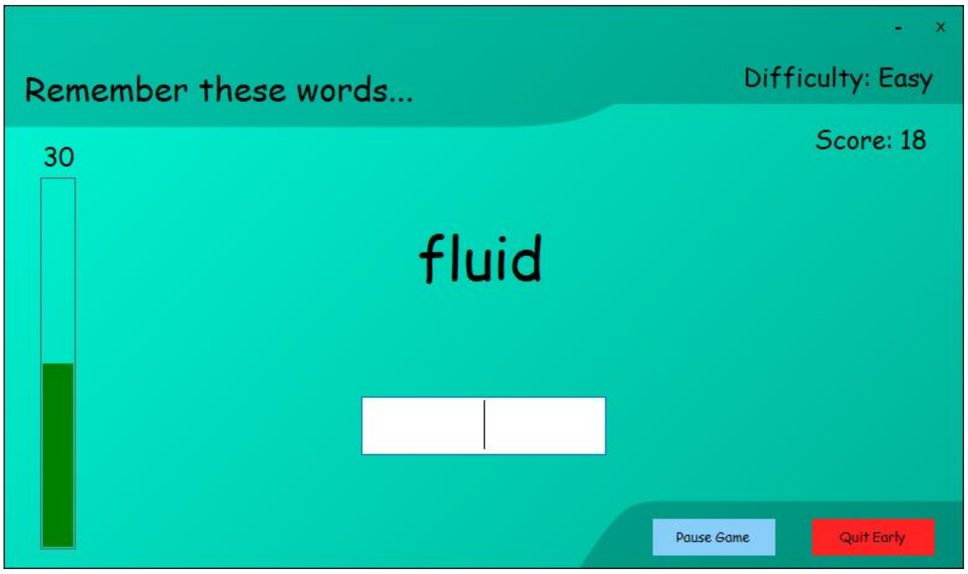
1. ***A custom word memory game***

Word Memory showcases the manipulation of controls, calculations, word logic, arrays/lists, and a randomiser/random number generator. The aim of the game is to test the user's memory and typing speed as they attempt to input the word which had previously appeared on screen while on a time limit. The words are chosen from a scrambled list of about 2000 common English words.

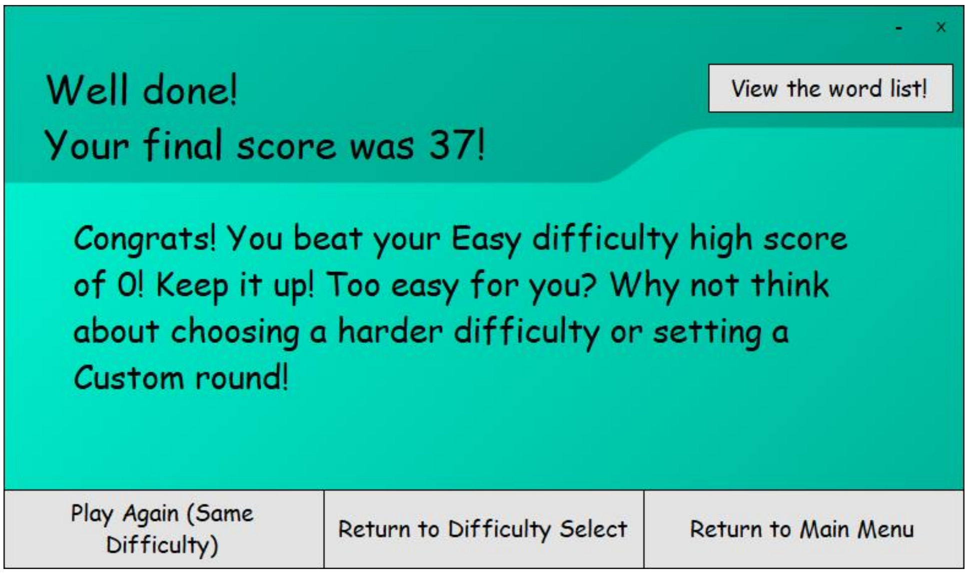
The Custom difficulty works a little differently. The word gap, time in between each word before input, and the length of the rest of the game is manually set by the user. The program also uses an externally saved list of words that is randomised instead. This list can be edited by the user. Custom games do not have their records saved, however, as modification of the word list and parameters could be used as an advantage.



The Difficulty Select allows for the user to very easily choose out of the four difficulties. Hovering over each button reveals their high score for that difficulty so that they can easily see their best score. And the ‘How to Play’ button reveals instructions on how to play Word Memory. The colours on each of the text on the buttons correspond to their difficulty, where each button is equally sized and spaced out as they are equally important.



In the game, the focus word is placed largely and centrally on the screen as it is the user’s main objective. There are various controls to pause or quit the game on the bottom right corner. The top right corner shows the current difficulty and score of the game. As the user is most likely focused on the central word, they would not want to be distracted by their score or other buttons during the game, hence those being more spread out. As the game is on a countdown, the progress bar on the left continues decreasing to show the user how much time they have left without distracting them from the game. It also doubles as a way to indicate to the user with the bar colour whether their previous word was correct (green) or incorrect (red).

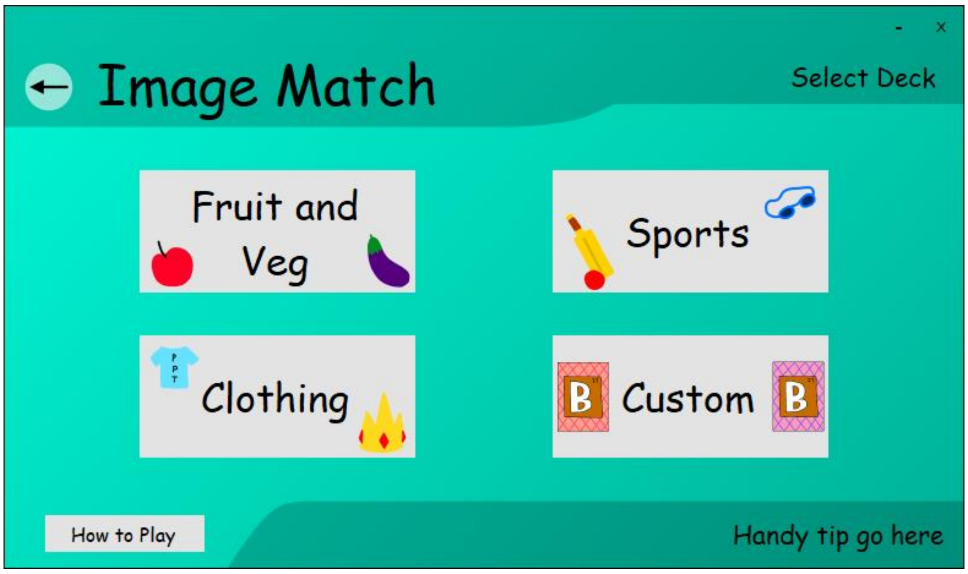

Well done!
Your final score was 37!

Congrats! You beat your Easy difficulty high score of 0! Keep it up! Too easy for you? Why not think about choosing a harder difficulty or setting a Custom round!

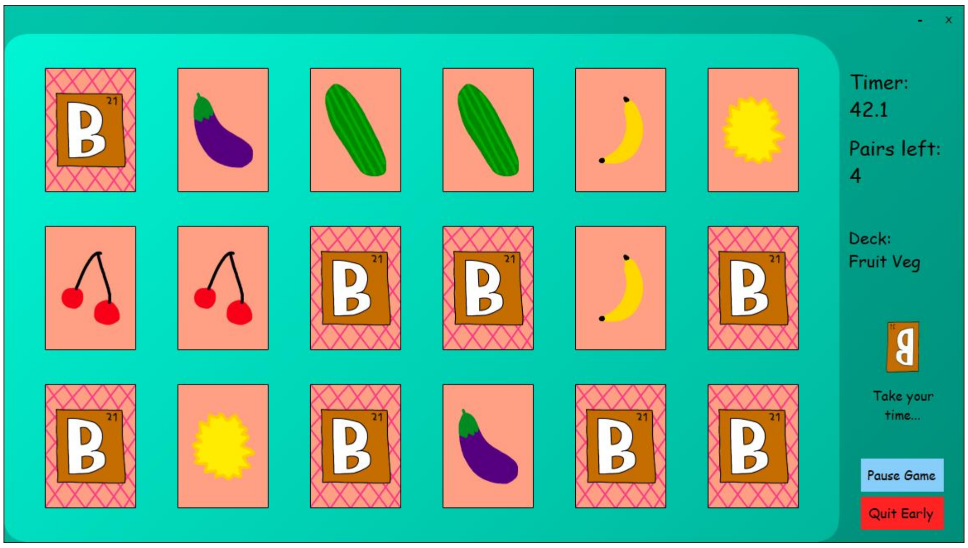
Once the game is over, the user is sent to the results screen where their final score is compared to their high score. The text aims to inspire the user in beating their score or to choose a different difficulty. The user also has the option to view the list of words they just used through the button on the top right corner. The three buttons on the bottom also makes its purpose clear to the user and gives them a chance to replay the game, or do something else.

2. A standard image match game

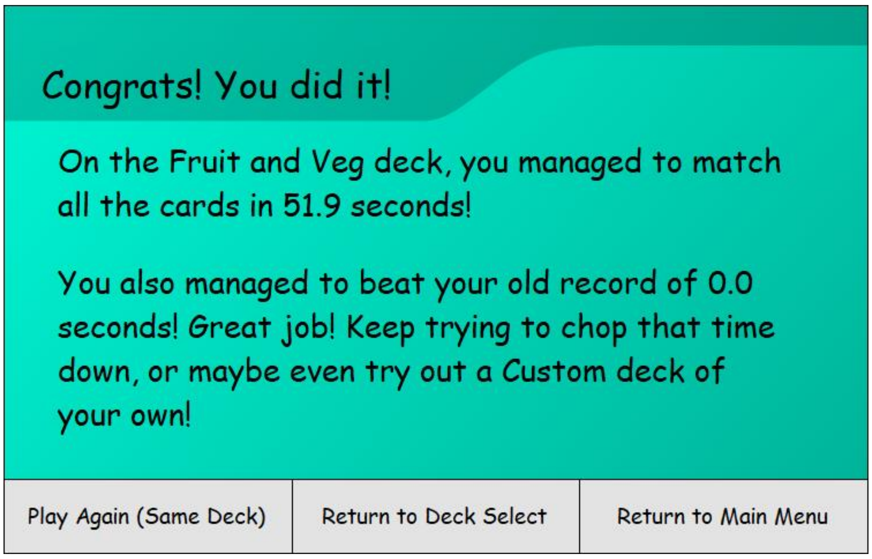
Image Match was designed to show off case statements, the wide use of graphics, timers, resources, and file linking. The aim of the game is to match all 18 cards with their pairs in the shortest time possible. All sets of cards have been compiled with the game's code for ease of access. For a Custom deck, the game will check an external folder for whether the required images exist or not. For similar reasons to Word Memory, Custom deck scores will not be saved.



The Deck Select for Image Match is identical to Word Memory in order to remain consistent with the "difficulty" select screens. However instead of difficulties, Image Match has 3 sets of decks, and the option for the Custom deck. Each deck contains 9 sets of cards which are duplicated while playing the game. This leads to a total of 18 cards that are used in the play area.



Due to more space needed, Image Match is contained in a separate form. All of the information and buttons have been put on the right side of the window so that they're all in one place. This is combined with a nice curve that is pleasant to the eyes and brings focus towards the play area. An animated gif plays above the button controls to indicate that the game has started. The timer is a standard label now as there is less importance in having the time run out. But it is recorded in milliseconds instead to give more precise results. The pause and quit buttons are located in a similar location to Word Memory with the exact same colours to keep it familiar. Whenever the user uncovers a pair, a positive sound effect is played to give audio feedback to the player.



Congrats! You did it!

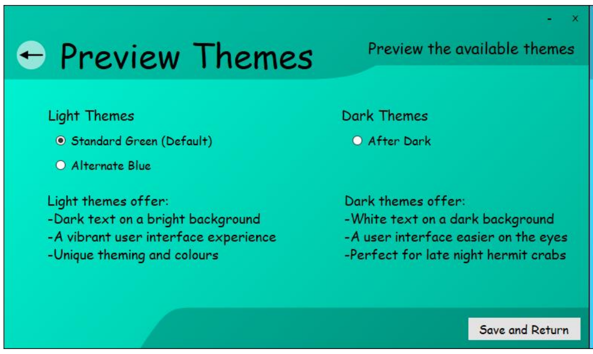
On the Fruit and Veg deck, you managed to match all the cards in 51.9 seconds!

You also managed to beat your old record of 0.0 seconds! Great job! Keep trying to chop that time down, or maybe even try out a Custom deck of your own!

Once all pairs have been found, the results screen is brought up. This is also identical to Word Memory's results screen, apart from the button in the top right corner. Similarly, the text aims to inspire the user to keep improving their time, or to try something else instead. The buttons along the bottom are still clearly labeled to avoid any confusion while remaining the same as Word Memory.

UI Themes
In order to appeal to a wider audience, 3 different colour themes were included in the software. It was decided that the different themes would be based on the colours Green, Blue, and Black.

Both the "Standard Green" and "Alternate Blue" themes come under the "Light" theme. Meaning that colours would be brighter, more vibrant, and eye catching. All of the text and buttons would follow the same black text on a bright background pattern to keep everything consistent.


Preview Themes
Preview the available themes

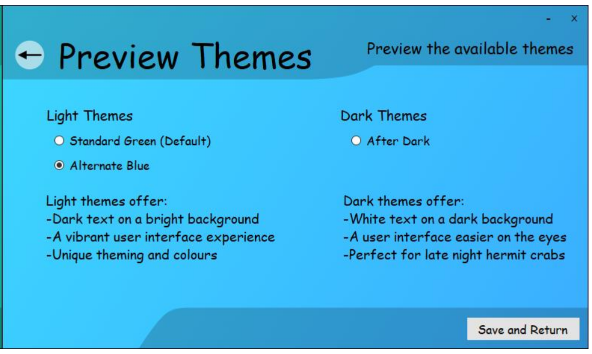
Light Themes
[✓ Standard Green (Default)]
Alternate Blue

Dark Themes
After Dark

Light themes offer:
-Dark text on a bright background
-A vibrant user interface experience
-Unique theming and colours

Dark themes offer:
-White text on a dark background
-A user interface easier on the eyes
-Perfect for late night hermit crabs

Save and Return


Preview Themes
Preview the available themes

Light Themes
Standard Green (Default)
[✓ Alternate Blue]

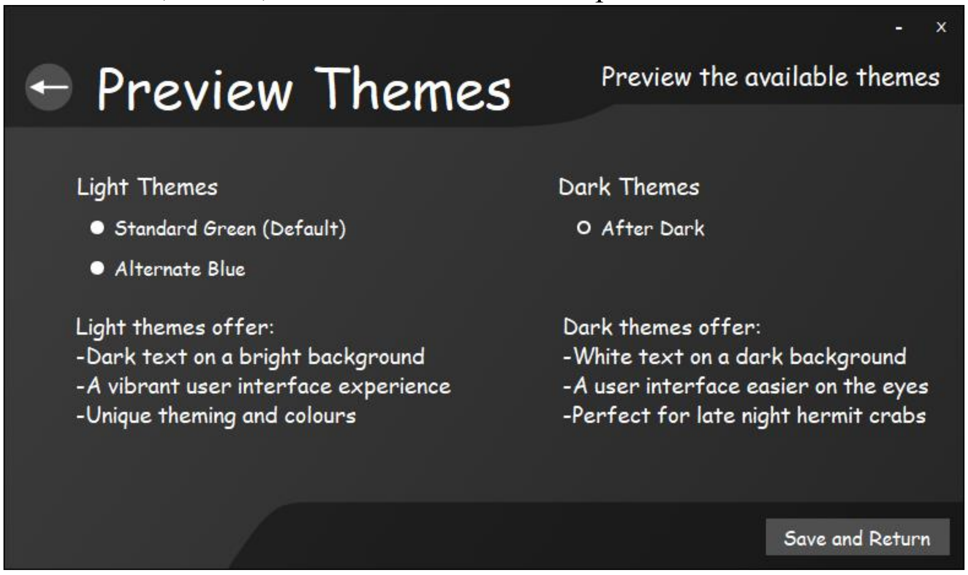
Dark Themes
After Dark

Light themes offer:
-Dark text on a bright background
-A vibrant user interface experience
-Unique theming and colours

Dark themes offer:
-White text on a dark background
-A user interface easier on the eyes
-Perfect for late night hermit crabs

Save and Return

However, the "After Dark" theme comes under the "Dark" theme. This is aimed at people who prefer a more monochrome look which isn't as bright on the eyes. Most of the colours have been reversed with light text on a dark background instead. Due to a complete change in colours, all of the text, menus, and elements had to be updated for this "Dark" theme.


Preview Themes
Preview the available themes

Light Themes
Standard Green (Default)
Alternate Blue

Dark Themes
After Dark

Light themes offer:
-Dark text on a bright background
-A vibrant user interface experience
-Unique theming and colours

Dark themes offer:
-White text on a dark background
-A user interface easier on the eyes
-Perfect for late night hermit crabs

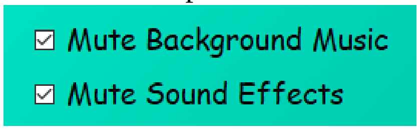
Save and Return

Debug Form

The Debug Form is an extra Form that can be enabled from the Settings menu. It allows the user to view the currently saved settings and some variables that are updated in real time. As this Form was mostly used to help troubleshoot issues during development, it does not receive the same styling as everything else. It is only meant to be a technical sneak peek into the parts of the program.

Audio

As seen in the Settings menu, there are two options for Audio.



☑ Mute Background Music

☑ Mute Sound Effects

In the software, background music is played and certain actions create sound effects. The background music is used to help set a relaxing mood. And the sound effects help give audio feedback to the user that something has happened. If wanted, the user can disable one or both settings independently.

While the use of audio was not required as a part of the assignment, it does help to make the software feel “less empty” and more relaxing.

As audio was taken from the internet, the proper attributes should be given. A list of the audio files that were used can be found in this document, and in the Credits section of the software.

Audio Attributes

"Bleeping Demo" Kevin MacLeod (incompetech.com)
Licensed under Creative Commons: By Attribution 4.0 License
http://creativecommons.org/licenses/by/4.0/

"UI Confirmation Alert, A4.wav" by InspectorJ (www.jshaw.co.uk) of Freesound.org

"Beep Boing.wav" by IndigoRay, https://freesound.org/s/339132/

"Positive Blip Effect" by Eponn, https://freesound.org/s/531512/

"UI_2-1 NTFO Triangle(Sytrus,arpegio,multiprocessing,rsmpl).wav" by newlocknew, https://freesound.org/s/515829/

"Cartoon UI Back/Cancel" by plasterbrain, https://freesound.org/s/423168/

User Guide

A user guide was also created as a part of the solution. As such, it will be included with the final submission. It can be found in \Documentation\.

Problems Encountered

- Limitations in Windows Forms prevented some ideas from the screen design sketches from being possible, including the ideas for diagonal menus. This was overcome by coming up with other ideas which would be possible to pull off within Windows Forms. Implementing different themes helped to make up for a lack of originality.

- Before starting to code, some tests had to be run to make sure progress could still be made at home (as seen in the Gantt Chart). This was due to different versions of Visual Studio which could have meant project files were not compatible. However, it was found that older project files would work fine on newer versions of Visual Studio. Therefore, the project file was just created at school, then brought home whenever work was planned to be done.

- For menus, it was preferred that there was a method that would allow them to be opened and closed without the need for opening a form every single time. This was solved by some research and the discovery of UserControls. These allowed for entire menus to be organised into its own control which could then be shown, hidden, or manipulated just like forms. (For detailed code, please go to “The Menus and Themes” on Page 28)

- Early on in development, it was decided that having a borderless form would allow for more control over the program. But a form without a border would also prevent the form from being dragged across the screen. With some more research, some code was found that would allow for the form to be moved when a certain area was clicked. This combined with some extra buttons to minimise and close the program was able to solve this problem. (For detailed code, please go to “Dragging Windows” on Page 33)

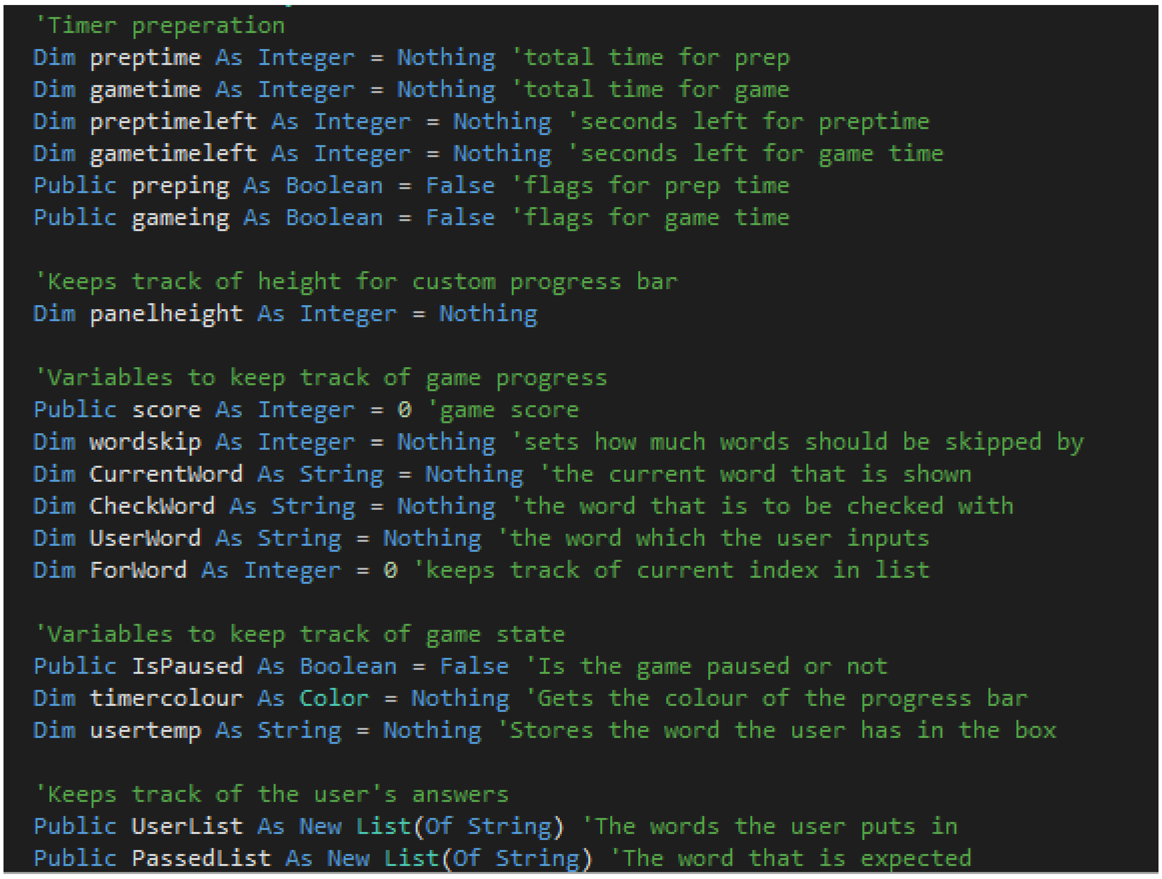
- As shown in Word Memory, a vertical progress bar was used to indicate time remaining. However, vertical progress bars do not normally exist within the .NET Framework. Instead, they were created by overlaying two panels on top of each other, then coding some maths based on the time remaining and height of the bar to make it seem like it is an actual progress bar. (For the maths behind it, please go to “Word Memory” on Page 18)

- During early builds of Image Match, it was discovered that trying to replay the game would result in the deck not appearing at all. Upon further investigating, it was found the code that handled the images was not being called when the game was reset. Therefore, all the code that handled the setup of the game was just combined into a single module in order to simplify the process.

3. Intrinsic Documentation

Word Memory

Word Memory was completely written from scratch for the purpose of showing off the manipulation of controls, calculations, word logic, arrays/lists, and a randomiser/random number generator.



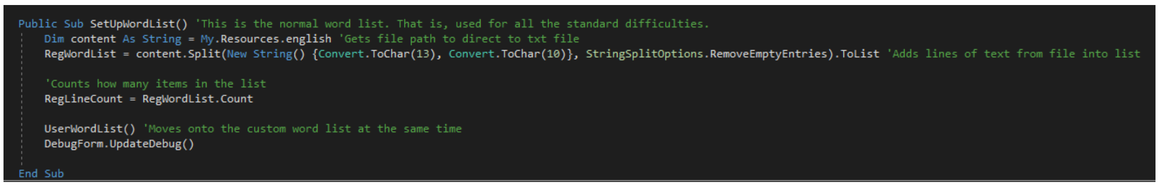
```
'Timer preperation
Dim preptime As Integer = Nothing 'total time for prep
Dim gametime As Integer = Nothing 'total time for game
Dim preptimeleft As Integer = Nothing 'seconds left for preptime
Dim gametimeleft As Integer = Nothing 'seconds left for game time
Public preping As Boolean = False 'flags for prep time
Public gameing As Boolean = False 'flags for game time

'Keeps track of height for custom progress bar
Dim panelheight As Integer = Nothing

'Variables to keep track of game progress
Public score As Integer = 0 'game score
Dim wordskip As Integer = Nothing 'sets how much words should be skipped by
Dim CurrentWord As String = Nothing 'the current word that is shown
Dim CheckWord As String = Nothing 'the word that is to be checked with
Dim UserWord As String = Nothing 'the word which the user inputs
Dim ForWord As Integer = 0 'keeps track of current index in list

'Variables to keep track of game state
Public IsPaused As Boolean = False 'Is the game paused or not
Dim timercolour As Color = Nothing 'Gets the colour of the progress bar
Dim usertemp As String = Nothing 'Stores the word the user has in the box

'Keeps track of the user's answers
Public UserList As New List(Of String) 'The words the user puts in
Public PassedList As New List(Of String) 'The word that is expected
```

These are the list of variables used for the Word Memory game. As seen, they all have comments with a quick explanation of what they are used for. In particular:

- ‘preping’ and ‘gameing’ indicates what state the game is currently in. This is especially useful for when the game needs to be paused so that the correct text can be replaced once the game starts again.
    - When ‘preping’ is true, it means the game is still in a phase where the user does not have input of the text box.
    - When ‘gameing’ is true it means the game has fully begun and the user can begin answering.

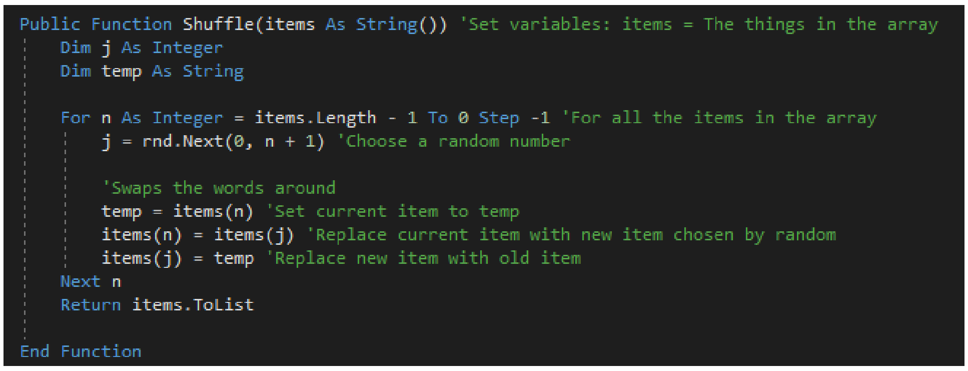
- ‘ForWord’ keeps track of the current word in the index. After every word, is it stepped by 1 so that the next word in the randomised list can be shown.

- ‘UserList’ and ‘PassedList’ are both lists used to keep track of the words that have already passed and the corresponding answer that the user entered. This is essential for the Results screen.



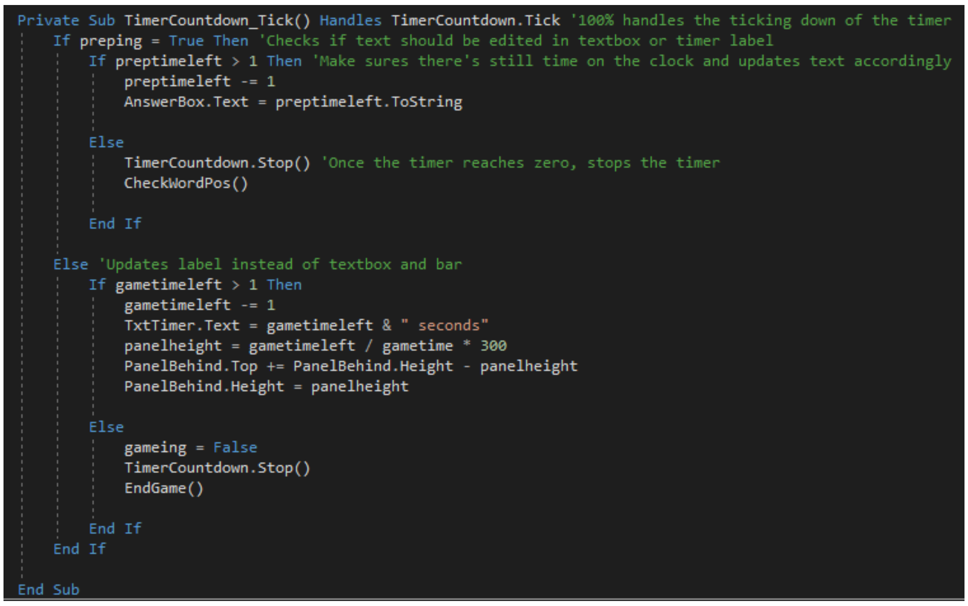
```
Public Sub SetUpWordList() 'This is the normal word list. That is, used for all the standard difficulties.
    Dim content As String = My.Resources.english 'Gets file path to direct to txt file
    RegWordList = content.Split(New String() {Convert.ToChar(13), Convert.ToChar(10)}, StringSplitOptions.RemoveEmptyEntries).ToList 'Adds lines of text from file into list

    'Counts how many items in the list
    RegLineCount = RegWordList.Count

    UserWordList() 'Moves onto the custom word list at the same time
    DebugForm.UpdateDebug()

End Sub
```

When the program first loads, the subroutine ‘SetUpWordList()’ is called from the ‘Randomise’ module. This gets a text file from ‘My.Resources.english’ and puts all the words into a list called ‘RegWordList’. This will always be used for subsequent plays of the game.



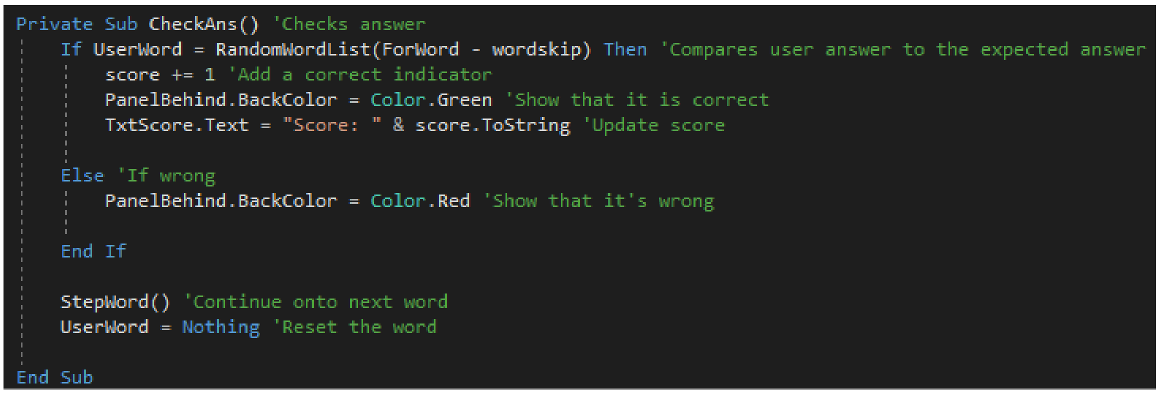
```
Public Function Shuffle(items As String()) 'Set variables: items = The things in the array
    Dim j As Integer
    Dim temp As String

    For n As Integer = items.Length - 1 To 0 Step -1 'For all the items in the array
        j = rnd.Next(0, n + 1) 'Choose a random number

        'Swaps the words around
        temp = items(n) 'Set current item to temp
        items(n) = items(j) 'Replace current item with new item chosen by random
        items(j) = temp 'Replace new item with old item
    Next n
    Return items.ToList

End Function
```

When the user begins playing the game, ‘RegWordList’ is input into a function ‘Shuffle’. The function randomises the word list so that every game will be different from one another. By using a random number generator, it randomly chooses a number which will become the index for the list. Then the item at the last index is swapped with the item at the random index. This is repeated up the list until every word is thoroughly randomised. The output is then returned to a different list, ‘RandomWordList’.



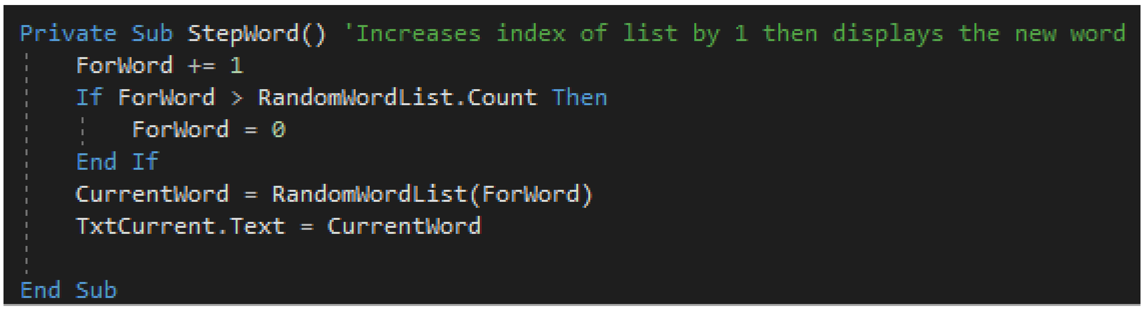
```
Private Sub TimerCountdown_Tick() Handles TimerCountdown.Tick '100% handles the ticking down of the timer
    If preping = True Then 'Checks if text should be edited in textbox or timer label
        If preptimeleft > 1 Then 'Make sures there's still time on the clock and updates text accordingly
            preptimeleft -= 1
            AnswerBox.Text = preptimeleft.ToString

        Else
            TimerCountdown.Stop() 'Once the timer reaches zero, stops the timer
            CheckWordPos()

        End If

    Else 'Updates label instead of textbox and bar
        If gametimeleft > 1 Then
            gametimeleft -= 1
            TxtTimer.Text = gametimeleft & " seconds"
            panelheight = gametimeleft / gametime * 300
            PanelBehind.Top += PanelBehind.Height - panelheight
            PanelBehind.Height = panelheight

        Else
            gameing = False
            TimerCountdown.Stop()
            EndGame()

        End If
    End If

End Sub
```

In order to keep track of the game, a timer is used which activates code every second. Depending on the status of ‘preping’, the code will either edit text from the text box, or update the progress bar. Once the game has properly begun, the code will perform some maths based on the total amount of time, the time remaining, and height of the progress bar panel to adjust its height and position accordingly. This works as a workaround to the absence of a vertical vertical bar, as mentioned earlier.



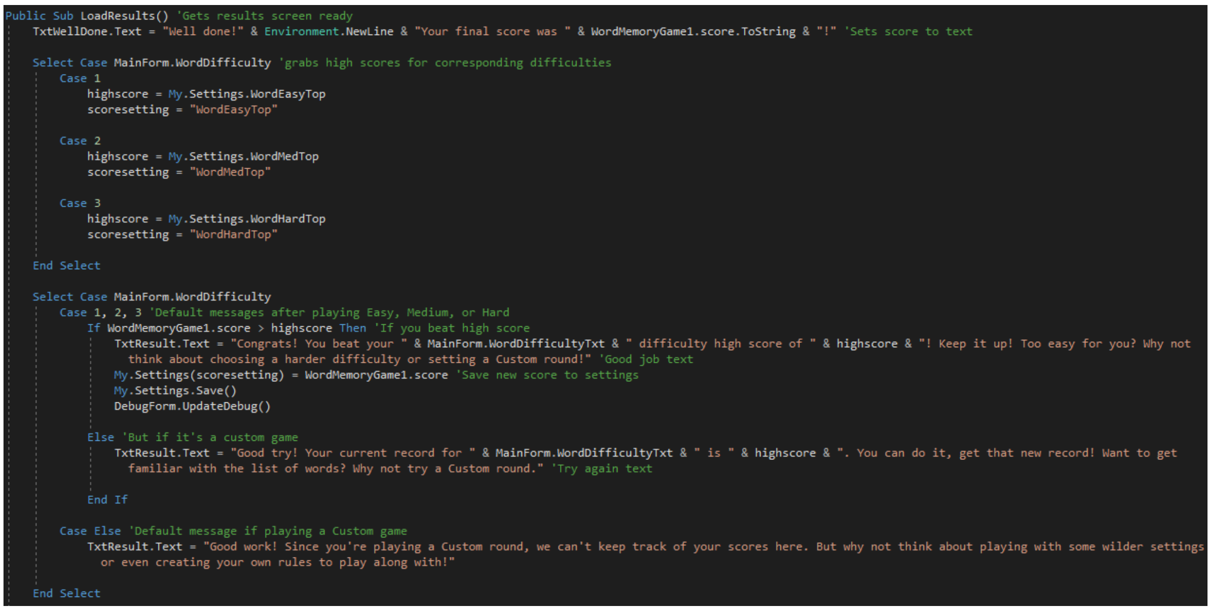
```
Private Sub CheckAns() 'Checks answer
    If UserWord = RandomWordList(ForWord - wordskip) Then 'Compares user answer to the expected answer
        score += 1 'Add a correct indicator
        PanelBehind.BackColor = Color.Green 'Show that it is correct
        TxtScore.Text = "Score: " & score.ToString 'Update score

    Else 'If wrong
        PanelBehind.BackColor = Color.Red 'Show that it's wrong

    End If

    StepWord() 'Continue onto next word
    UserWord = Nothing 'Reset the word

End Sub
```

Once the user has entered a word, it is to be checked with the word in the randomised list. As there is a gap between the currently shown word and the expected answer, ‘wordskip’ adjusts this depending on the difficulty. After the inputted word is manipulated to be all lowercase, it is compared with the expected word. If it is correct, +1 is added to the score and the progress bar turns green. If the word is wrong, then the progress bar turns red and leaves the score alone.



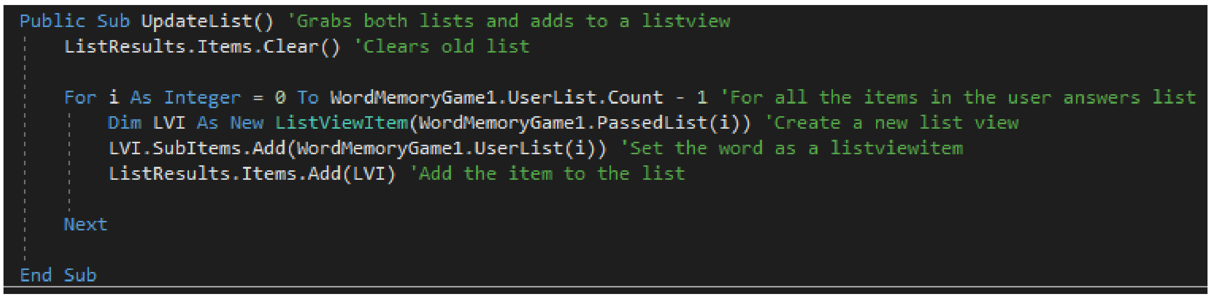
```
Private Sub StepWord() 'Increases index of list by 1 then displays the new word
    ForWord += 1
    If ForWord > RandomWordList.Count Then
        ForWord = 0
    End If
    CurrentWord = RandomWordList(ForWord)
    TxtCurrent.Text = CurrentWord

End Sub
```

After a word has been submitted, the next word has to be displayed. The subroutine ‘StepWord’ adds 1 to ‘ForWord’ which keeps track of the current index of the word list. This word is then displayed to the user. If the user happens to reach the end of the word list (which may happen with a custom game), it is told to repeat to the front of the list.



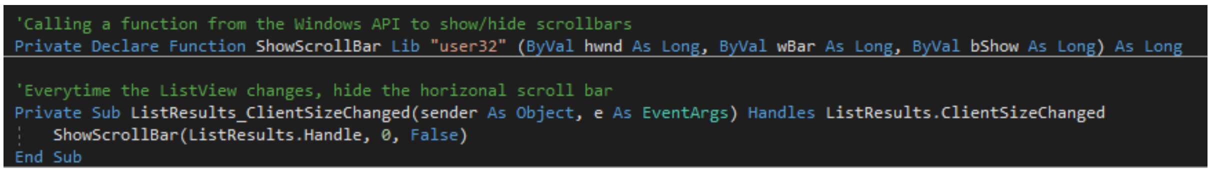
```
Public Sub LoadResults() 'Gets results screen ready
    TxtWellDone.Text = "Well done!" & Environment.NewLine & "Your final score was " & WordMemoryGame1.score.ToString & "!" 'Sets score to text

    Select Case MainForm.WordDifficulty 'grabs high scores for corresponding difficulties
        Case 1
            highscore = My.Settings.WordEasyTop
            scoresetting = "WordEasyTop"

        Case 2
            highscore = My.Settings.WordMedTop
            scoresetting = "WordMedTop"

        Case 3
            highscore = My.Settings.WordHardTop
            scoresetting = "WordHardTop"

    End Select

    Select Case MainForm.WordDifficulty
        Case 1, 2, 3 'Default messages after playing Easy, Medium, or Hard
            If WordMemoryGame1.score > highscore Then 'If you beat high score
                TxtResult.Text = "Congrats! You beat your " & MainForm.WordDifficultyTxt & " difficulty high score of " & highscore & "! Keep it up! Too easy for you? Why not
                  think about choosing a harder difficulty or setting a Custom round!" 'Good job text
                My.Settings(scoresetting) = WordMemoryGame1.score 'Save new score to settings
                My.Settings.Save()
                DebugForm.UpdateDebug()

            Else 'But if it's a custom game
                TxtResult.Text = "Good try! Your current record for " & MainForm.WordDifficultyTxt & " is " & highscore & ". You can do it, get that new record! Want to get
                  familiar with the list of words? Why not try a Custom round." 'Try again text

            End If

        Case Else 'Default message if playing a Custom game
            TxtResult.Text = "Good work! Since you're playing a Custom round, we can't keep track of your scores here. But why not think about playing with some wilder settings
              or even creating your own rules to play along with!"

    End Select
```

Once time runs out, the user is sent to the results screen. The game checks whether the user is playing a Custom game or not. And then it checks whether the user has beaten their old high score or not. The corresponding text is shown to the user and the high score in settings is updated if needed.



```
Public Sub UpdateList() 'Grabs both lists and adds to a listview
    ListResults.Items.Clear() 'Clears old list

    For i As Integer = 0 To WordMemoryGame1.UserList.Count - 1 'For all the items in the user answers list
        Dim LVI As New ListViewItem(WordMemoryGame1.PassedList(i)) 'Create a new list view
        LVI.SubItems.Add(WordMemoryGame1.UserList(i)) 'Set the word as a listviewitem
        ListResults.Items.Add(LVI) 'Add the item to the list

    Next

End Sub
```

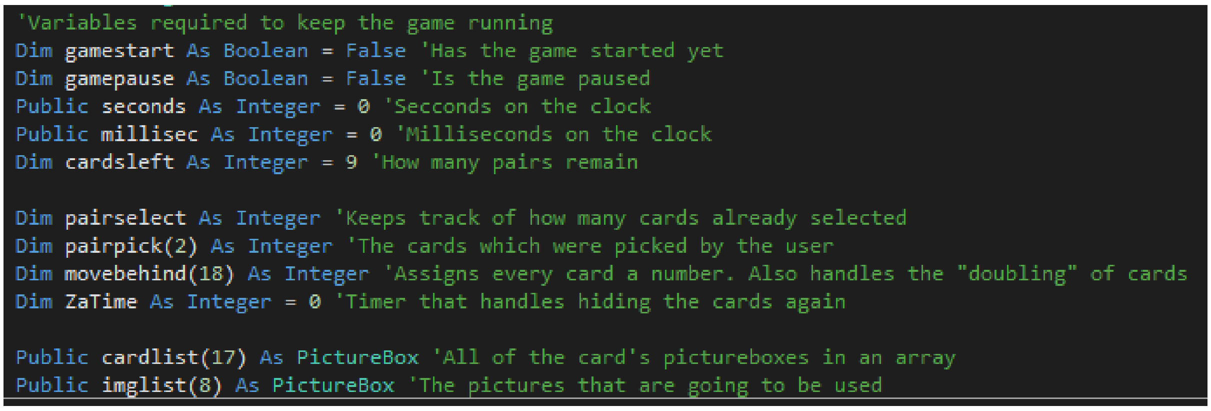
In the results screen, the user is able to see the list of words they got right or wrong. The list of words they entered and the words that were expected are entered into a listview by using ‘For… Next’ for every item in the list.



```
'Calling a function from the Windows API to show/hide scrollbars
Private Declare Function ShowScrollBar Lib "user32" (ByVal hwnd As Long, ByVal wBar As Long, ByVal bShow As Long) As Long

'Everytime the ListView changes, hide the horizonal scroll bar
Private Sub ListResults_ClientSizeChanged(sender As Object, e As EventArgs) Handles ListResults.ClientSizeChanged
    ShowScrollBar(ListResults.Handle, 0, False)
End Sub
```

If there are enough words in the results list, a scrollbar is shown which allows the user to scroll through the list. But due to how scrollbars are added to listviews, the Private Declare ‘ShowScrollBar’ uses the Windows32 API (Application Programming Interface) to prevent the horizontal scrollbar from showing.

Image Match
The original code for Image Match was taken from GodeGuru and adapted to work for this software. In particular, the game will automatically start instead of requiring user input, and will display a results screen once all pairs have been found. Image Match was made to highlight the use of case statements, various graphics, timers, resources, and file linking.


```
'Variables required to keep the game running
Dim gamestart As Boolean = False 'Has the game started yet
Dim gamepause As Boolean = False 'Is the game paused
Public seconds As Integer = 0 'Secconds on the clock
Public millisec As Integer = 0 'Milliseconds on the clock
Dim cardsleft As Integer = 9 'How many pairs remain

Dim pairselect As Integer 'Keeps track of how many cards already selected
Dim pairpick(2) As Integer 'The cards which were picked by the user
Dim movebehind(18) As Integer 'Assigns every card a number. Also handles the "doubling" of cards
Dim ZaTime As Integer = 0 'Timer that handles hiding the cards again

Public cardlist(17) As PictureBox 'All of the card's pictureboxes in an array
Public imglist(8) As PictureBox 'The pictures that are going to be used
```
Here is a list of variables that are used for Image Match. Some of them are further explained:

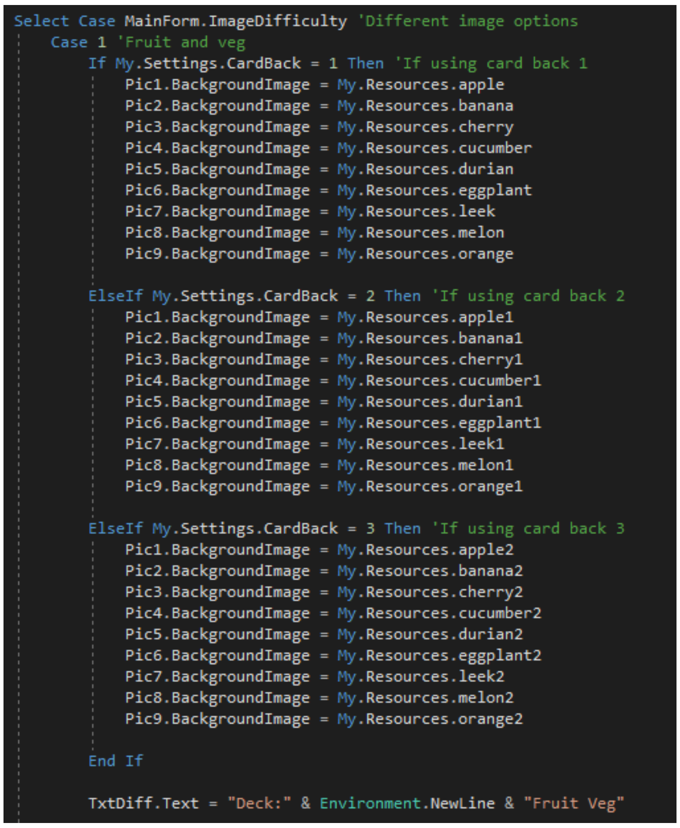
- ‘ZaTime’ is an integer which keeps track of how much time has passed since the user has flipped their second card. With the use of a timer that increments the integer, it will hide the cards after a few seconds.

- ‘pairpick’ is an array which holds information about the two cards the user has picked. By passing the array through a subroutine, we are able to check whether two cards are the same or not.

- ‘movebehind’ assigns every card an integer which helps duplicate the images for each pair. It is used in randomising the deck and checking of the chosen cards.

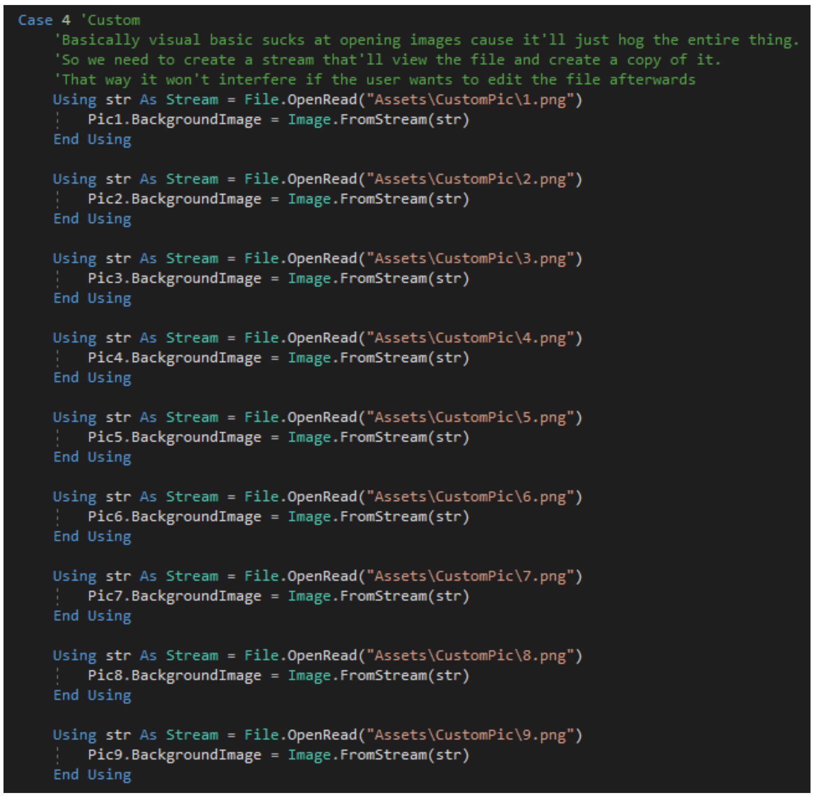
- ‘imglist’ is a set of picture boxes which contains the images that will be used for the game. When the game first loads, it sets the required images to these picture boxes. Once the randomised deck has been decided, these picture boxes are used to determine what picture should appear when the card is flipped. These picture boxes can either be shown or hidden with the use of a toggle in the Settings menu.

- ‘cardlist’ is also a set of picture boxes which are used by the user. They assist in hiding or showing the cards when they are selected. By using the indexes of the array, it helps corresponds to the integers in ‘movebehind’



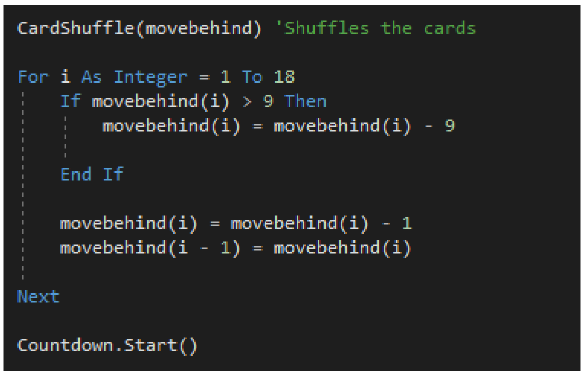
```
Select Case MainForm.ImageDifficulty 'Different image options
    Case 1 'Fruit and veg
        If My.Settings.CardBack = 1 Then 'If using card back 1
            Pic1.BackgroundImage = My.Resources.apple
            Pic2.BackgroundImage = My.Resources.banana
            Pic3.BackgroundImage = My.Resources.cherry
            Pic4.BackgroundImage = My.Resources.cucumber
            Pic5.BackgroundImage = My.Resources.durian
            Pic6.BackgroundImage = My.Resources.eggplant
            Pic7.BackgroundImage = My.Resources.leek
            Pic8.BackgroundImage = My.Resources.melon
            Pic9.BackgroundImage = My.Resources.orange

        ElseIf My.Settings.CardBack = 2 Then 'If using card back 2
            Pic1.BackgroundImage = My.Resources.apple1
            Pic2.BackgroundImage = My.Resources.banana1
            Pic3.BackgroundImage = My.Resources.cherry1
            Pic4.BackgroundImage = My.Resources.cucumber1
            Pic5.BackgroundImage = My.Resources.durian1
            Pic6.BackgroundImage = My.Resources.eggplant1
            Pic7.BackgroundImage = My.Resources.leek1
            Pic8.BackgroundImage = My.Resources.melon1
            Pic9.BackgroundImage = My.Resources.orange1

        ElseIf My.Settings.CardBack = 3 Then 'If using card back 3
            Pic1.BackgroundImage = My.Resources.apple2
            Pic2.BackgroundImage = My.Resources.banana2
            Pic3.BackgroundImage = My.Resources.cherry2
            Pic4.BackgroundImage = My.Resources.cucumber2
            Pic5.BackgroundImage = My.Resources.durian2
            Pic6.BackgroundImage = My.Resources.eggplant2
            Pic7.BackgroundImage = My.Resources.leek2
            Pic8.BackgroundImage = My.Resources.melon2
            Pic9.BackgroundImage = My.Resources.orange2

        End If

        TxtDiff.Text = "Deck:" & Environment.NewLine & "Fruit Veg"
```

When the game first loads, a case statement is used to determine which deck should be used. This is based on the option that the user chooses in Deck Select and the colour they have chosen to use from Settings. Every single card and colour combination exists in ‘My.Resources’ for ease of access.



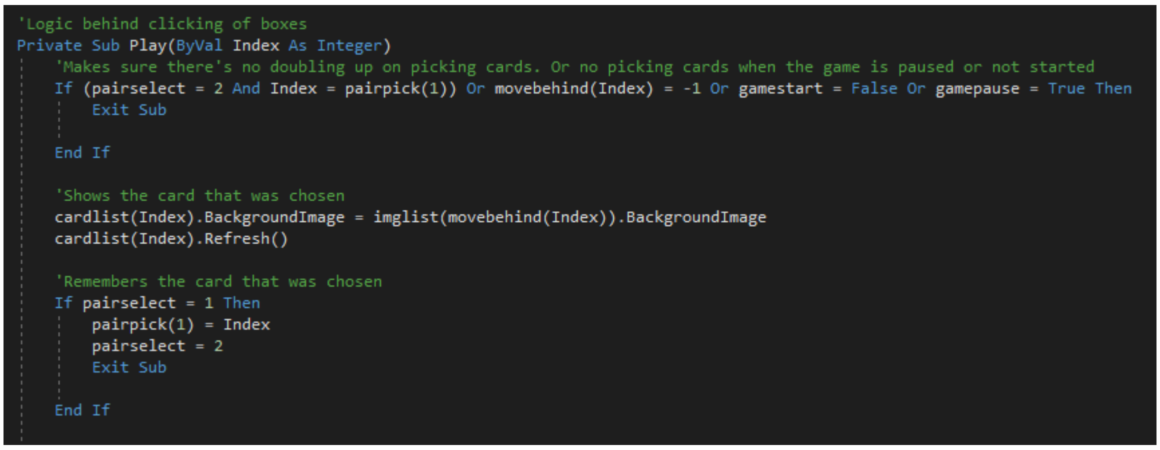
```
Case 4 'Custom
    'Basically visual basic sucks at opening images cause it'll just hog the entire thing.
    'So we need to create a stream that'll view the file and create a copy of it.
    'That way it won't interfere if the user wants to edit the file afterwards
    Using str As Stream = File.OpenRead("Assets\CustomPic\1.png")
        Pic1.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\2.png")
        Pic2.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\3.png")
        Pic3.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\4.png")
        Pic4.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\5.png")
        Pic5.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\6.png")
        Pic6.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\7.png")
        Pic7.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\8.png")
        Pic8.BackgroundImage = Image.FromStream(str)
    End Using

    Using str As Stream = File.OpenRead("Assets\CustomPic\9.png")
        Pic9.BackgroundImage = Image.FromStream(str)
    End Using
```

For a Custom deck, the game will check the ‘Assets\CustomPic’ folder for images titled “1.png” up to “9.png”. Due to how Visual Basic opens files, a Stream is required to insert the images. The Stream copies the original image from the file location and inserts it into the corresponding picture box. This way, the user is able to change the custom deck after a game without having to close the program.



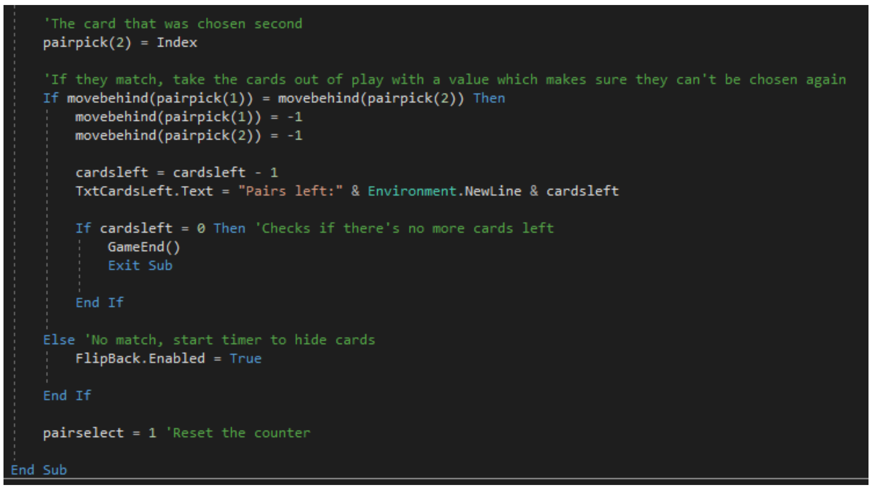
```
CardShuffle(movebehind) 'Shuffles the cards

For i As Integer = 1 To 18
    If movebehind(i) > 9 Then
        movebehind(i) = movebehind(i) - 9

    End If

    movebehind(i) = movebehind(i) - 1
    movebehind(i - 1) = movebehind(i)

Next

Countdown.Start()
```

Once all of the arrays have been filled and populated, ‘movebehind’ is shuffled using a similar randomiser to Word Memory. The difference is that it accepts integers instead of strings. The shuffled list is then set up so that the numbers from 1 to 9 have a corresponding duplicate in the array. This is done by taking all numbers above 9, and subtracting 9 from it. This is essential in being able to pair the cards up. The game then begins with a 5 second countdown before the user can select a card.



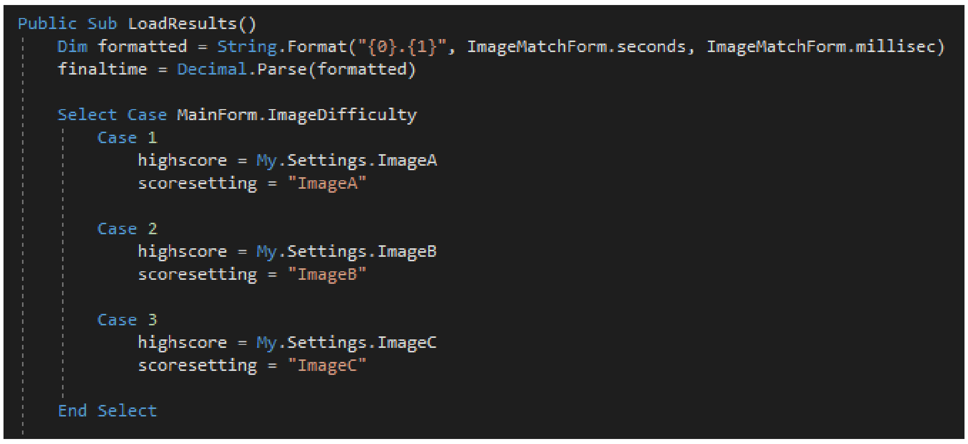
```
'Logic behind clicking of boxes
Private Sub Play(ByVal Index As Integer)
    'Makes sure there's no doubling up on picking cards. Or no picking cards when the game is paused or not started
    If (pairselect = 2 And Index = pairpick(1)) Or movebehind(Index) = -1 Or gamestart = False Or gamepause = True Then
        Exit Sub

    End If

    'Shows the card that was chosen
    cardlist(Index).BackgroundImage = imglist(movebehind(Index)).BackgroundImage
    cardlist(Index).Refresh()

    'Remembers the card that was chosen
    If pairselect = 1 Then
        pairpick(1) = Index
        pairselect = 2
        Exit Sub

    End If
```

When the user selects a card, the subroutine ‘Play’ checks that the card has not already been chosen, or that the game is paused. If the card has already been chosen or the game is currently paused, it will not execute the rest of the code by exiting the subroutine. If allowed, the selected card is then revealed to the player by using the value from ‘movebehind’, and the first index of ‘pairpick’ is filled out with the card’s information. ‘pairselect’ is then increased to 2 so that the game knows that the next card will be the second card.



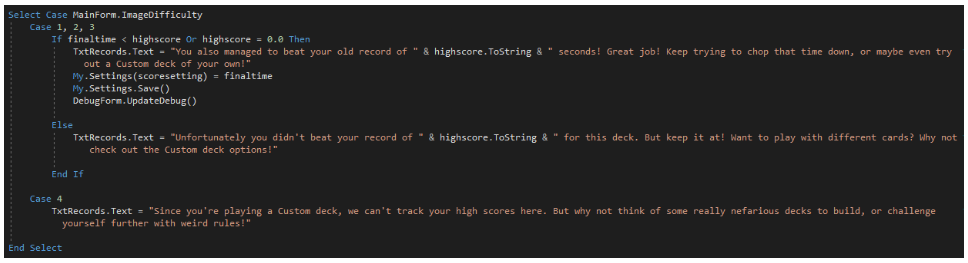
```
        'The card that was chosen second
        pairpick(2) = Index

        'If they match, take the cards out of play with a value which makes sure they can't be chosen again
        If movebehind(pairpick(1)) = movebehind(pairpick(2)) Then
            movebehind(pairpick(1)) = -1
            movebehind(pairpick(2)) = -1

            cardsleft = cardsleft - 1
            TxtCardsLeft.Text = "Pairs left:" & Environment.NewLine & cardsleft

            If cardsleft = 0 Then 'Checks if there's no more cards left
                GameEnd()
                Exit Sub

            End If

        Else 'No match, start timer to hide cards
            FlipBack.Enabled = True

        End If

        pairselect = 1 'Reset the counter

    End Sub
```

Once the second card has been chosen, the two cards are compared. If they have the same 'movebehind' integer, they get moved out of play and the pair remaining counter gets updated. At this point, the game checks if there are no more pairs left, in which case it will end the game. But if the two cards do not match, a second timer is activated which will hide the mis-matched cards after a few seconds.



```
Public Sub LoadResults()
    Dim formatted = String.Format("{0}.{1}", ImageMatchForm.seconds, ImageMatchForm.millisec)
    finaltime = Decimal.Parse(formatted)

    Select Case MainForm.ImageDifficulty
        Case 1
            highscore = My.Settings.ImageA
            scoresetting = "ImageA"

        Case 2
            highscore = My.Settings.ImageB
            scoresetting = "ImageB"

        Case 3
            highscore = My.Settings.ImageC
            scoresetting = "ImageC"

    End Select
```

When the user finds all pairs of cards, the results screen will display with the total time the user took. By formatting the seconds and the millisecond variables into a decimal number, it can be stored in My.Settings if it is a high score.

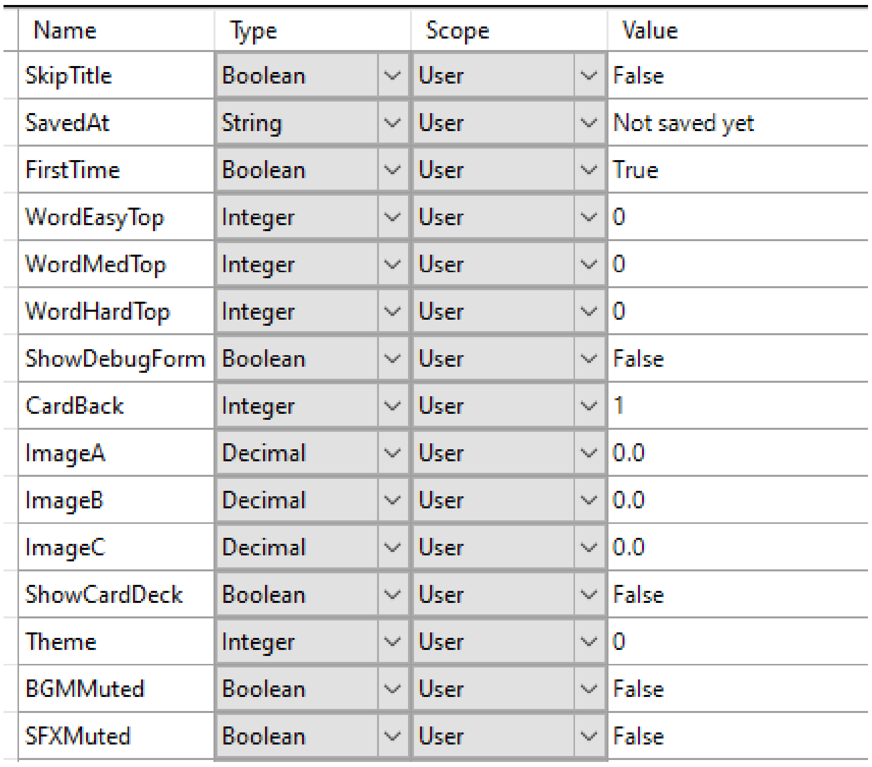


```
Select Case MainForm.ImageDifficulty
    Case 1, 2, 3
        If finaltime < highscore Or highscore = 0.0 Then
            TxtRecords.Text = "You also managed to beat your old record of " & highscore.ToString & " seconds! Great job! Keep trying to chop that time down, or maybe even try
              out a Custom deck of your own!"
            My.Settings(scoresetting) = finaltime
            My.Settings.Save()
            DebugForm.UpdateDebug()

        Else
            TxtRecords.Text = "Unfortunately you didn't beat your record of " & highscore.ToString & " for this deck. But keep it at! Want to play with different cards? Why not
               check out the Custom deck options!"

        End If

    Case 4
        TxtRecords.Text = "Since you're playing a Custom deck, we can't track your high scores here. But why not think of some really nefarious decks to build, or challenge
          yourself further with weird rules!"

End Select
```

Another case statement in the results check whether a Custom deck was played or not, and then uses the user's time to compare to their best time. The corresponding text is shown to the user and if the user achieved a new high score, the time is saved to My.Settings.

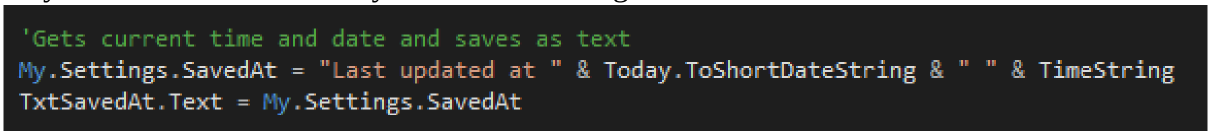
The Settings

In order to save all high scores and settings, Visual Basic's Settings manager is used. This allows for the settings to be manipulated in code by the use of 'My.Settings'. Essentially, these are variables which are saved after the software is closed.



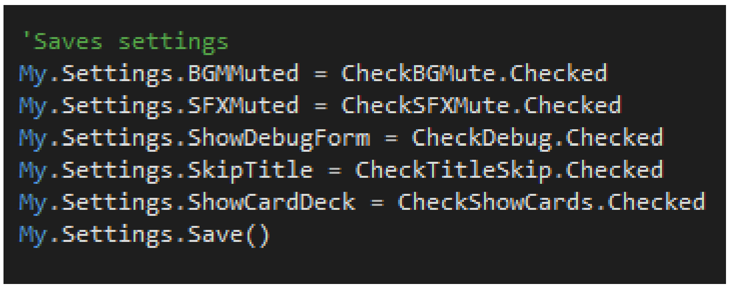
Name	Type	Scope	Value
SkipTitle	Boolean	User	False
SavedAt	String	User	Not saved yet
FirstTime	Boolean	User	True
WordEasyTop	Integer	User	0
WordMedTop	Integer	User	0
WordHardTop	Integer	User	0
ShowDebugForm	Boolean	User	False
CardBack	Integer	User	1
ImageA	Decimal	User	0.0
ImageB	Decimal	User	0.0
ImageC	Decimal	User	0.0
ShowCardDeck	Boolean	User	False
Theme	Integer	User	0
BGMMuted	Boolean	User	False
SFXMuted	Boolean	User	False

Some of these settings are Boolean as they are only affected by a CheckBox with a value of True or False. Others are Integers/Decimal as they store the user's best scores, which are numerical. 'SavedAt' is a string as it retrieves the time and date from the user's PC so that they can see the last time they saved their settings.



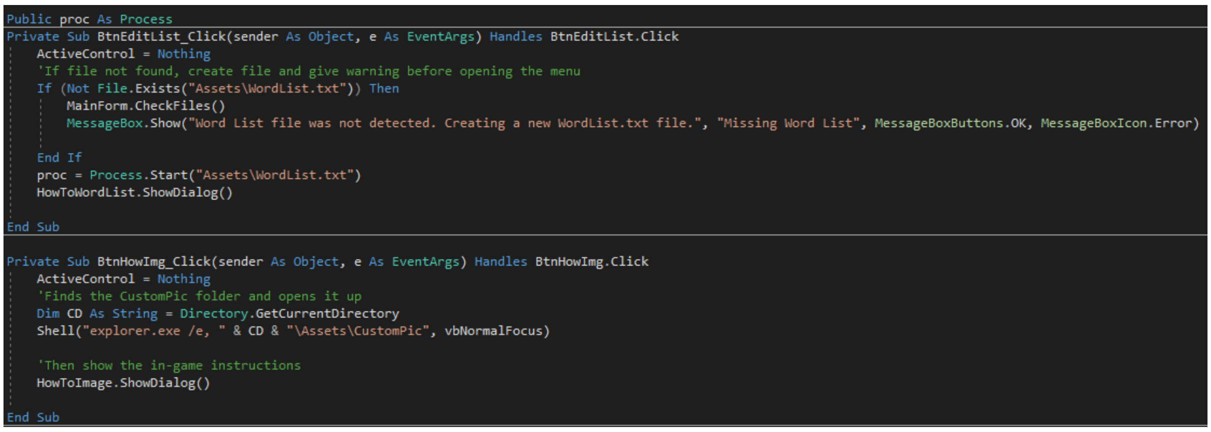
```
'Gets current time and date and saves as text
My.Settings.SavedAt = "Last updated at " & Today.ToShortDateString & " " & TimeString
TxtSavedAt.Text = My.Settings.SavedAt
```

'FirstTime' is the only setting which can not be manipulated by the user unless they reset all settings. It's default value is 'True' and signifies whether it's the user's first time opening the software or not. If it is, the Welcome screen is shown to them. If it is not the user's first time, the setting would be 'False' and the Welcome screen will not be shown to the user.



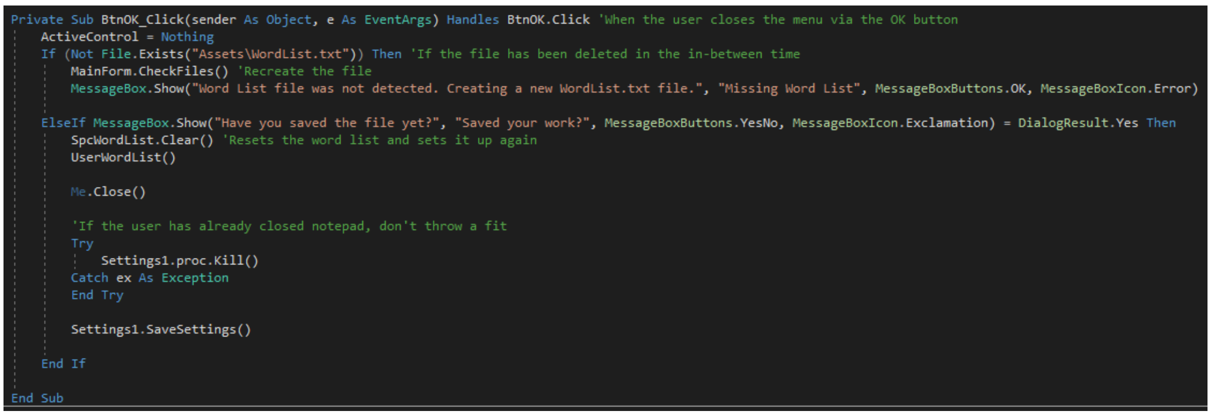
```
'Saves settings
My.Settings.BGMMuted = CheckBGMute.Checked
My.Settings.SFXMuted = CheckSFXMute.Checked
My.Settings.ShowDebugForm = CheckDebug.Checked
My.Settings.SkipTitle = CheckTitleSkip.Checked
My.Settings.ShowCardDeck = CheckShowCards.Checked
My.Settings.Save()
```

When the user decides to save their settings and return to the Main Menu, the states of the CheckBoxes are used to determine what should be updated in My.Settings. If required, this will also automatically adjust the DebugForm or adjust the background music.



```
Public proc As Process
Private Sub BtnEditList_Click(sender As Object, e As EventArgs) Handles BtnEditList.Click
    ActiveControl = Nothing
    'If file not found, create file and give warning before opening the menu
    If (Not File.Exists("Assets\WordList.txt")) Then
        MainForm.CheckFiles()
        MessageBox.Show("Word List file was not detected. Creating a new WordList.txt file.", "Missing Word List", MessageBoxButtons.OK, MessageBoxIcon.Error)

    End If
    proc = Process.Start("Assets\WordList.txt")
    HowToWordList.ShowDialog()

End Sub

Private Sub BtnHowImg_Click(sender As Object, e As EventArgs) Handles BtnHowImg.Click
    ActiveControl = Nothing
    'Finds the CustomPic folder and opens it up
    Dim CD As String = Directory.GetCurrentDirectory
    Shell("explorer.exe /e, " & CD & "\Assets\CustomPic", vbNormalFocus)

    'Then show the in-game instructions
    HowToImage.ShowDialog()

End Sub
```

When the user wishes to update the word list, they must go through the settings in order to allow the word list to be updated. For convenience, the word list is automatically opened for them through their default text editor. This is easily done by starting a new process (opening an application) for the desired file.



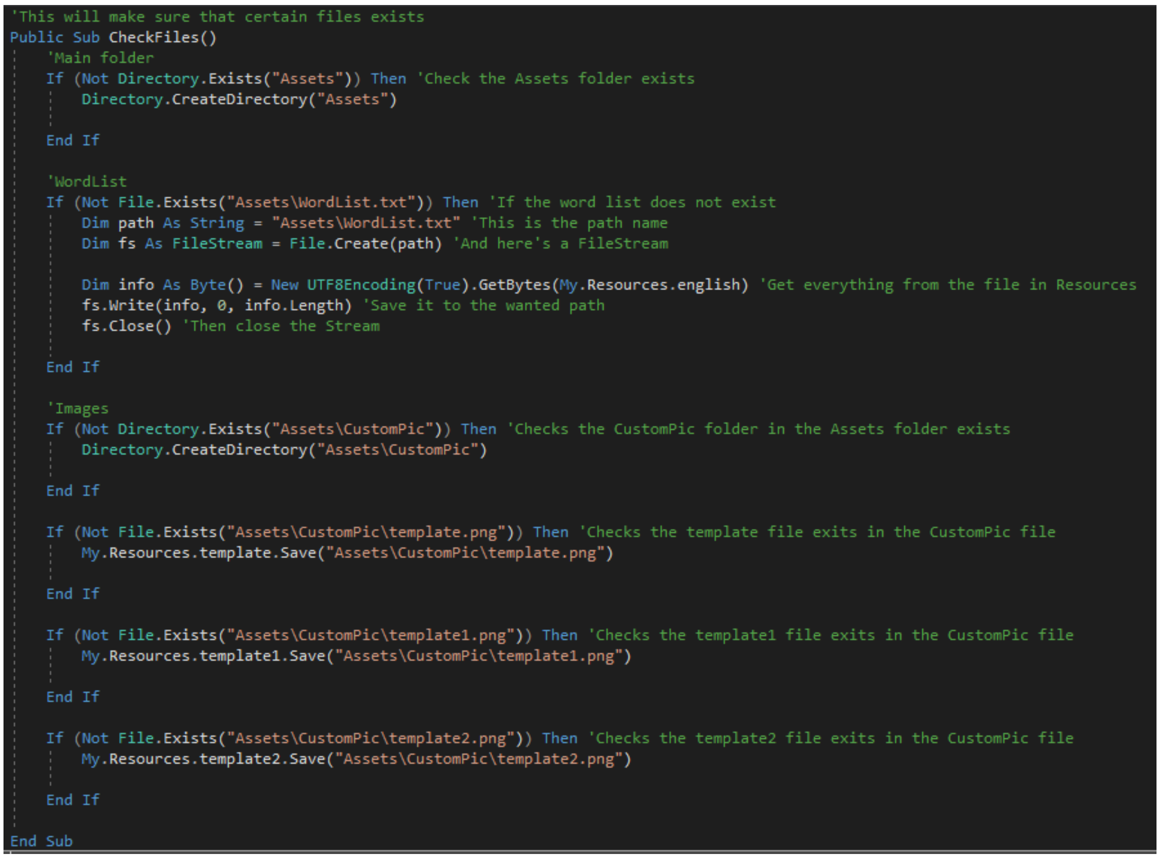
```
Private Sub BtnOK_Click(sender As Object, e As EventArgs) Handles BtnOK.Click 'When the user closes the menu via the OK button
    ActiveControl = Nothing
    If (Not File.Exists("Assets\WordList.txt")) Then 'If the file has been deleted in the in-between time
        MainForm.CheckFiles() 'Recreate the file
        MessageBox.Show("Word List file was not detected. Creating a new WordList.txt file.", "Missing Word List", MessageBoxButtons.OK, MessageBoxIcon.Error)

    ElseIf MessageBox.Show("Have you saved the file yet?", "Saved your work?", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation) = DialogResult.Yes Then
        SpcWordList.Clear() 'Resets the word list and sets it up again
        UserWordList()

        Me.Close()

        'If the user has already closed notepad, don't throw a fit
        Try
            Settings1.proc.Kill()
        Catch ex As Exception
        End Try

        Settings1.SaveSettings()

    End If

End Sub
```

Once the user has finished editing the text file, an attempt is made to close the process once the user presses OK. If the text editor has already been closed, an error would occur. This will not cause the program to crash, but it will show an error message to the user which is inconvenient and very annoying. Therefore, a “Try” and “Catch” is required in order to prevent the error from appearing to the user. These tell the program that the error has been handled and it doesn’t need to notify the user.

My Resources

As said in Image Match, Resources are used to contain all of the images used for the cards. However, it is also used to contain all the images used in the UI, the predetermined word list for Word Match, and all of the audio files. In total, over 150 files exist in Resources. The reason everything is saved in Resources is because when the program is compiled, Resources are also compiled with the code. This makes it very easy to reference files that will be used in the program with the use of ‘My.Resources’, and it doesn’t require the user to download any additional files. This is also helpful in making sure none of the images or audio are tampered with.

However for Custom games to be able to run, the user must be able to interact with the game’s files in some way. Therefore, everytime the game loads, it runs a check for specific files and folders.



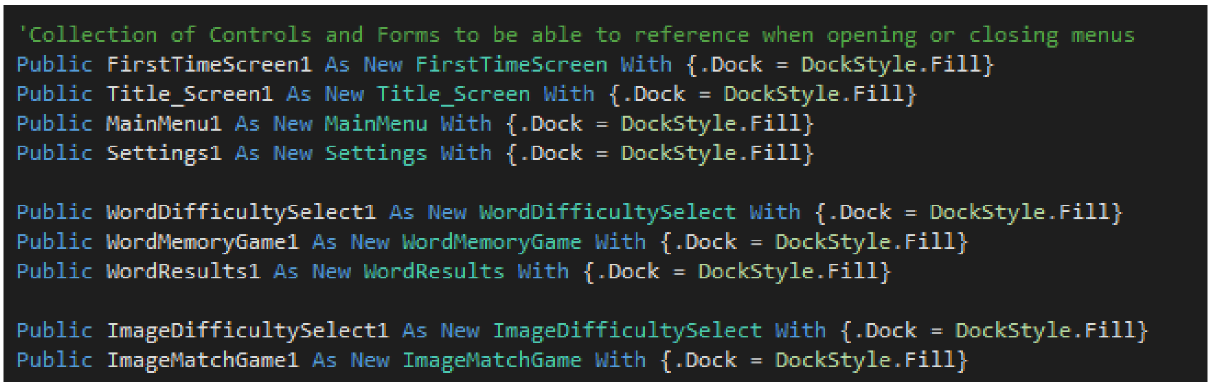
```
'This will make sure that certain files exists
Public Sub CheckFiles()
    'Main folder
    If (Not Directory.Exists("Assets")) Then 'Check the Assets folder exists
        Directory.CreateDirectory("Assets")

    End If

    'WordList
    If (Not File.Exists("Assets\WordList.txt")) Then 'If the word list does not exist
        Dim path As String = "Assets\WordList.txt" 'This is the path name
        Dim fs As FileStream = File.Create(path) 'And here's a FileStream

        Dim info As Byte() = New UTF8Encoding(True).GetBytes(My.Resources.english) 'Get everything from the file in Resources
        fs.Write(info, 0, info.Length) 'Save it to the wanted path
        fs.Close() 'Then close the Stream

    End If

    'Images
    If (Not Directory.Exists("Assets\CustomPic")) Then 'Checks the CustomPic folder in the Assets folder exists
        Directory.CreateDirectory("Assets\CustomPic")

    End If

    If (Not File.Exists("Assets\CustomPic\template.png")) Then 'Checks the template file exits in the CustomPic file
        My.Resources.template.Save("Assets\CustomPic\template.png")

    End If

    If (Not File.Exists("Assets\CustomPic\template1.png")) Then 'Checks the template1 file exits in the CustomPic file
        My.Resources.template1.Save("Assets\CustomPic\template1.png")

    End If

    If (Not File.Exists("Assets\CustomPic\template2.png")) Then 'Checks the template2 file exits in the CustomPic file
        My.Resources.template2.Save("Assets\CustomPic\template2.png")

    End If

End Sub
```

With the use of If statements, ‘Directory.Exists’, and ‘File.Exists’, we are able to check whether specific files and folders exist, and copy them from Resources or create folders if required. This way, the user has access to the word list for Custom Word Memory, and the templates for the images that were used in Image Match.

Maths is also vital when the program looks for the custom images. For every required file, the program will either calculate 1 * 1 if the file exists, or 1 * 0 if a file does not exist. This way, if there is at least 1 file missing, the answer will always return a 0, and the program will not begin Custom Image Match.

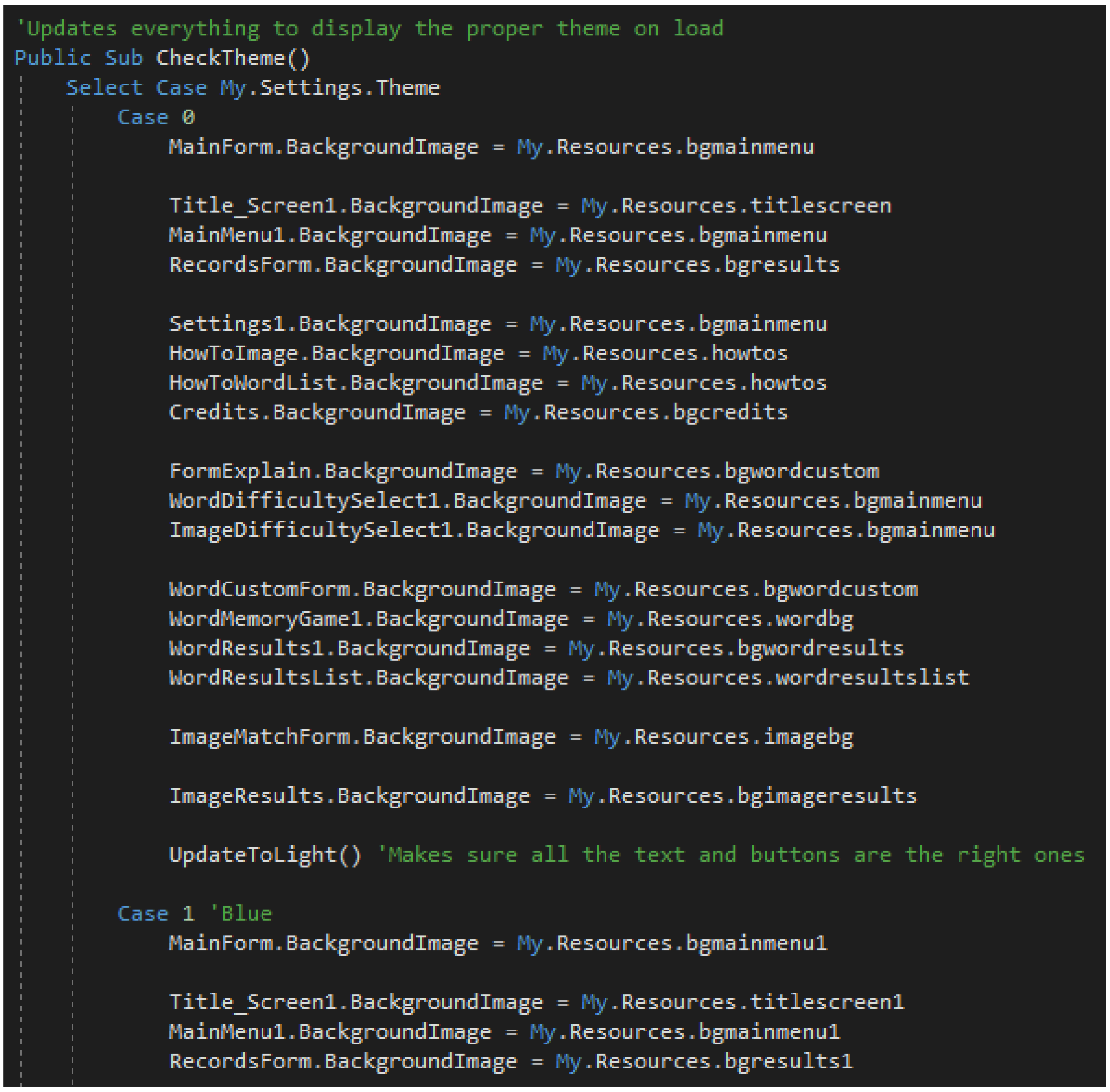
The Menus and Themes

When the program first loads, all of the UserControls have to be referenced so that they can be used. Hence a collection of every UserControl can be found at the top of the 'UserControls' module.



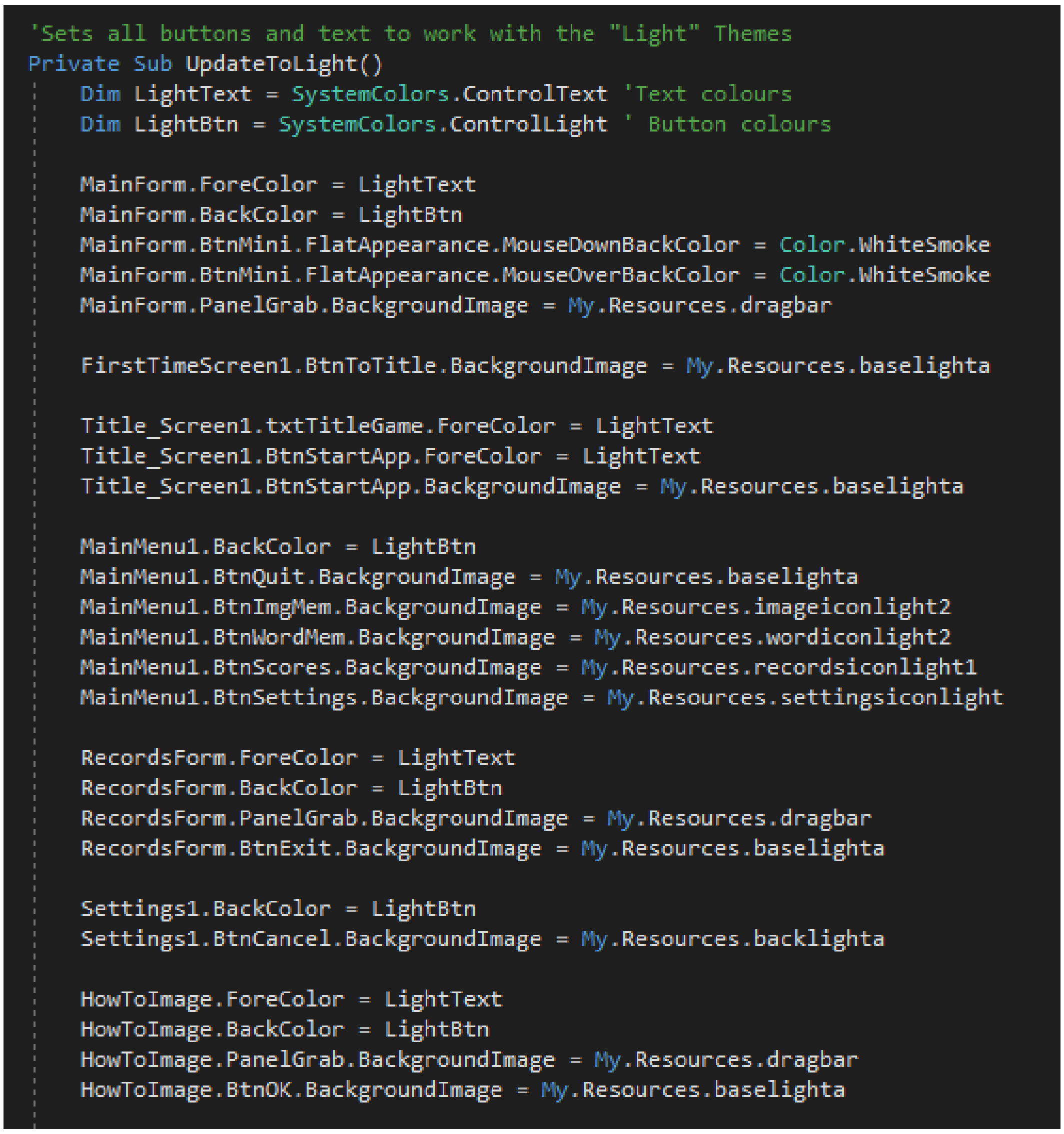
```
'Collection of Controls and Forms to be able to reference when opening or closing menus
Public FirstTimeScreen1 As New FirstTimeScreen With {.Dock = DockStyle.Fill}
Public Title_Screen1 As New Title_Screen With {.Dock = DockStyle.Fill}
Public MainMenu1 As New MainMenu With {.Dock = DockStyle.Fill}
Public Settings1 As New Settings With {.Dock = DockStyle.Fill}

Public WordDifficultySelect1 As New WordDifficultySelect With {.Dock = DockStyle.Fill}
Public WordMemoryGame1 As New WordMemoryGame With {.Dock = DockStyle.Fill}
Public WordResults1 As New WordResults With {.Dock = DockStyle.Fill}

Public ImageDifficultySelect1 As New ImageDifficultySelect With {.Dock = DockStyle.Fill}
Public ImageMatchGame1 As New ImageMatchGame With {.Dock = DockStyle.Fill}
```

All of them have 'With {.Dock = DockStyle.Fill}' so that they will automatically take up the entire space of the MainForm.



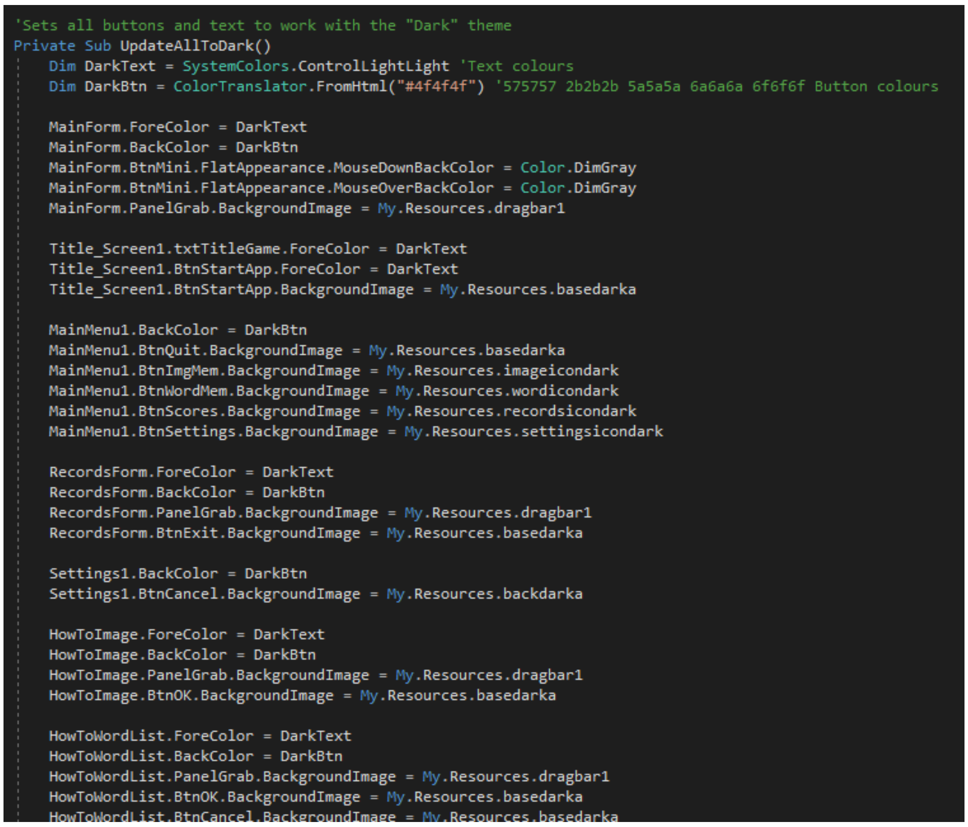
```
'Updates everything to display the proper theme on load
Public Sub CheckTheme()
    Select Case My.Settings.Theme
        Case 0
            MainForm.BackgroundImage = My.Resources.bgmainmenu

            Title_Screen1.BackgroundImage = My.Resources.titlescreen
            MainMenu1.BackgroundImage = My.Resources.bgmainmenu
            RecordsForm.BackgroundImage = My.Resources.bgresults

            Settings1.BackgroundImage = My.Resources.bgmainmenu
            HowToImage.BackgroundImage = My.Resources.howtos
            HowToWordList.BackgroundImage = My.Resources.howtos
            Credits.BackgroundImage = My.Resources.bgcredits

            FormExplain.BackgroundImage = My.Resources.bgwordcustom
            WordDifficultySelect1.BackgroundImage = My.Resources.bgmainmenu
            ImageDifficultySelect1.BackgroundImage = My.Resources.bgmainmenu

            WordCustomForm.BackgroundImage = My.Resources.bgwordcustom
            WordMemoryGame1.BackgroundImage = My.Resources.wordbg
            WordResults1.BackgroundImage = My.Resources.bgwordresults
            WordResultsList.BackgroundImage = My.Resources.wordresultslist

            ImageMatchForm.BackgroundImage = My.Resources.imagebg

            ImageResults.BackgroundImage = My.Resources.bgimageresults

            UpdateToLight() 'Makes sure all the text and buttons are the right ones

        Case 1 'Blue
            MainForm.BackgroundImage = My.Resources.bgmainmenu1

            Title_Screen1.BackgroundImage = My.Resources.titlescreen1
            MainMenu1.BackgroundImage = My.Resources.bgmainmenu1
            RecordsForm.BackgroundImage = My.Resources.bgresults1
```

When the MainForm begins to Load, a subroutine ‘CheckTheme’ uses a Case statement to get the current theme from Settings. 0 = Green, 1 = Blue, and 2 = Black. This will make sure all of the backgrounds for every menu and form will be set correctly. All of the background images have been pre-identified where nothing at the end of the file name means it’s for the Green theme, a ‘1’ at the end of the file name means it’s for the Blue theme, and a ‘2’ means it’s for the Black theme.



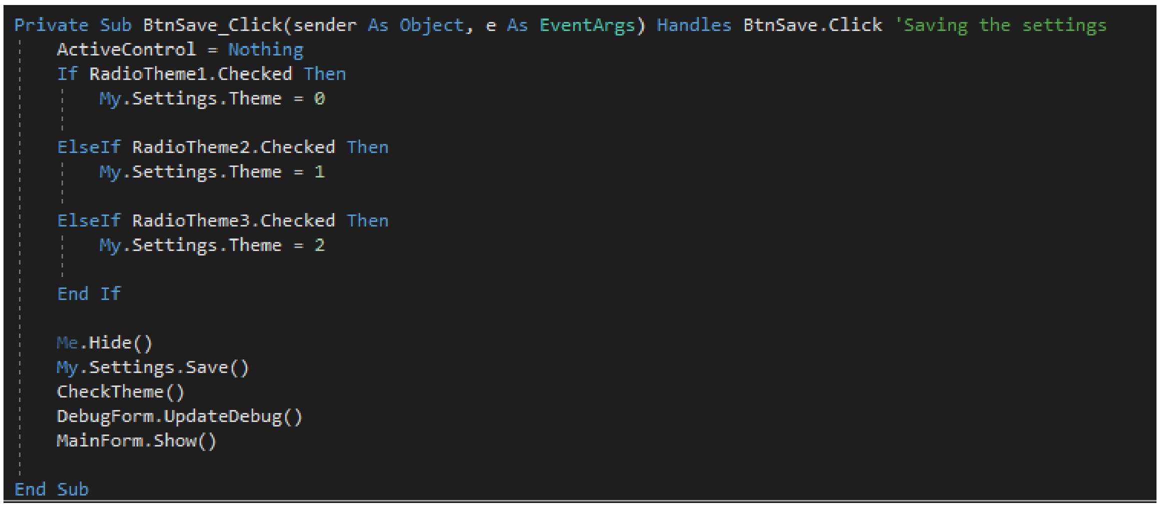
```
'Sets all buttons and text to work with the "Light" Themes
Private Sub UpdateToLight()
    Dim LightText = SystemColors.ControlText 'Text colours
    Dim LightBtn = SystemColors.ControlLight ' Button colours

    MainForm.ForeColor = LightText
    MainForm.BackColor = LightBtn
    MainForm.BtnMini.FlatAppearance.MouseDownBackColor = Color.WhiteSmoke
    MainForm.BtnMini.FlatAppearance.MouseOverBackColor = Color.WhiteSmoke
    MainForm.PanelGrab.BackgroundImage = My.Resources.dragbar

    FirstTimeScreen1.BtnToTitle.BackgroundImage = My.Resources.baselighta

    Title_Screen1.txtTitleGame.ForeColor = LightText
    Title_Screen1.BtnStartApp.ForeColor = LightText
    Title_Screen1.BtnStartApp.BackgroundImage = My.Resources.baselighta

    MainMenu1.BackColor = LightBtn
    MainMenu1.BtnQuit.BackgroundImage = My.Resources.baselighta
    MainMenu1.BtnImgMem.BackgroundImage = My.Resources.imageiconlight2
    MainMenu1.BtnWordMem.BackgroundImage = My.Resources.wordiconlight2
    MainMenu1.BtnScores.BackgroundImage = My.Resources.recordsiconlight1
    MainMenu1.BtnSettings.BackgroundImage = My.Resources.settingsiconlight

    RecordsForm.ForeColor = LightText
    RecordsForm.BackColor = LightBtn
    RecordsForm.PanelGrab.BackgroundImage = My.Resources.dragbar
    RecordsForm.BtnExit.BackgroundImage = My.Resources.baselighta

    Settings1.BackColor = LightBtn
    Settings1.BtnCancel.BackgroundImage = My.Resources.backlighta

    HowToImage.ForeColor = LightText
    HowToImage.BackColor = LightBtn
    HowToImage.PanelGrab.BackgroundImage = My.Resources.dragbar
    HowToImage.BtnOK.BackgroundImage = My.Resources.baselighta
```

After setting the backgrounds, all of the text, buttons, and other colours also have to match the theme. If using the Green or Blue theme, the subroutine ‘UpdateToLight’ is called. This will update every text to use ‘SystemColors.ControlText’ (basically black) and every button to ‘SystemColors.ControlLight’ (basically grey). Back and cancel buttons are also updated for it’s light variant. Other changes based on the light theme are also made here.



```
'Sets all buttons and text to work with the "Dark" theme
Private Sub UpdateAllToDark()
    Dim DarkText = SystemColors.ControlLightLight 'Text colours
    Dim DarkBtn = ColorTranslator.FromHtml("#4f4f4f") '575757 2b2b2b 5a5a5a 6a6a6a 6f6f6f Button colours

    MainForm.ForeColor = DarkText
    MainForm.BackColor = DarkBtn
    MainForm.BtnMini.FlatAppearance.MouseDownBackColor = Color.DimGray
    MainForm.BtnMini.FlatAppearance.MouseOverBackColor = Color.DimGray
    MainForm.PanelGrab.BackgroundImage = My.Resources.dragbar1

    Title_Screen1.txtTitleGame.ForeColor = DarkText
    Title_Screen1.BtnStartApp.ForeColor = DarkText
    Title_Screen1.BtnStartApp.BackgroundImage = My.Resources.basedarka

    MainMenu1.BackColor = DarkBtn
    MainMenu1.BtnQuit.BackgroundImage = My.Resources.basedarka
    MainMenu1.BtnImgMem.BackgroundImage = My.Resources.imageicondark
    MainMenu1.BtnWordMem.BackgroundImage = My.Resources.wordicondark
    MainMenu1.BtnScores.BackgroundImage = My.Resources.recordsicondark
    MainMenu1.BtnSettings.BackgroundImage = My.Resources.settingsicondark

    RecordsForm.ForeColor = DarkText
    RecordsForm.BackColor = DarkBtn
    RecordsForm.PanelGrab.BackgroundImage = My.Resources.dragbar1
    RecordsForm.BtnExit.BackgroundImage = My.Resources.basedarka

    Settings1.BackColor = DarkBtn
    Settings1.BtnCancel.BackgroundImage = My.Resources.backdarka

    HowToImage.ForeColor = DarkText
    HowToImage.BackColor = DarkBtn
    HowToImage.PanelGrab.BackgroundImage = My.Resources.dragbar1
    HowToImage.BtnOK.BackgroundImage = My.Resources.basedarka

    HowToWordList.ForeColor = DarkText
    HowToWordList.BackColor = DarkBtn
    HowToWordList.PanelGrab.BackgroundImage = My.Resources.dragbar1
    HowToWordList.BtnOK.BackgroundImage = My.Resources.basedarka
    HowToWordList.BtnCancel.BackgroundImage = My.Resources.basedarka
```

But if the Dark theme is used, ‘UpdateToDark’ is called. All of the text is updated to ‘SystemColors.ControlLightLight’ (basically white) and the buttons are changed to the hex code “#4f4f4f” (basically grey). Various buttons and other changes are made as required. Once the Menus and the Themes have loaded, the program continues running other code (Please go to “When the Main Form loads” on Page 34 for the rest of the startup sequence).

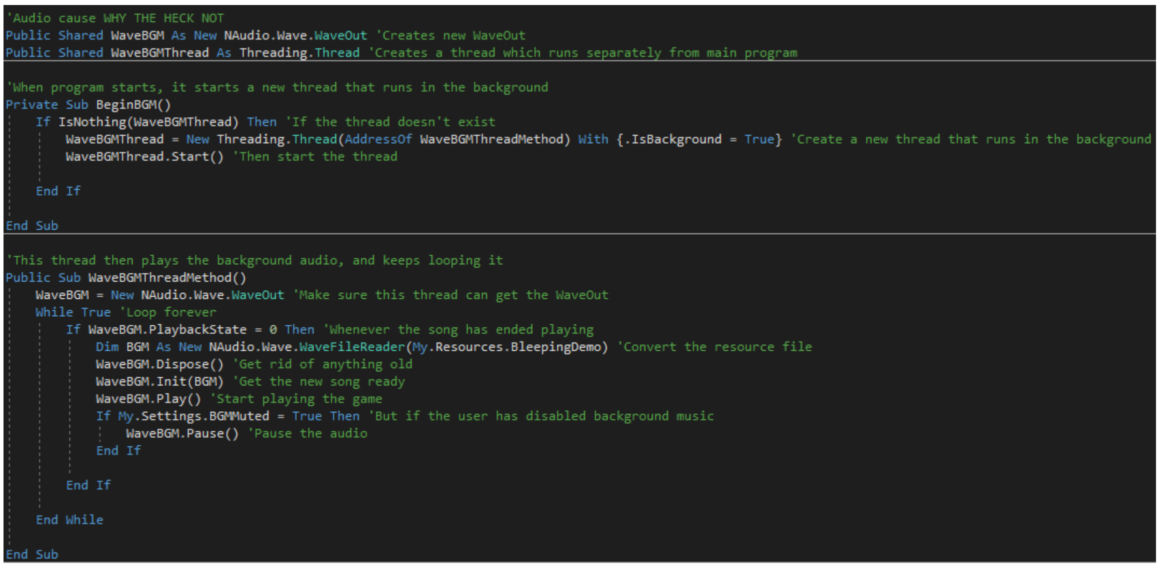


```
Private Sub BtnSave_Click(sender As Object, e As EventArgs) Handles BtnSave.Click 'Saving the settings
    ActiveControl = Nothing
    If RadioTheme1.Checked Then
        My.Settings.Theme = 0

    ElseIf RadioTheme2.Checked Then
        My.Settings.Theme = 1

    ElseIf RadioTheme3.Checked Then
        My.Settings.Theme = 2

    End If

    Me.Hide()
    My.Settings.Save()
    CheckTheme()
    DebugForm.UpdateDebug()
    MainForm.Show()

End Sub
```

The user is able to change the theme via the ‘Theme Settings’ button on the Main Menu. Once they’re happy with their selection and return to the Main Menu, the program saves their selection to ‘My.Settings.Theme’. Then the ‘CheckTheme’ subroutine is called which updates the UI. Finally, the Main Form is shown again with the corresponding theme.

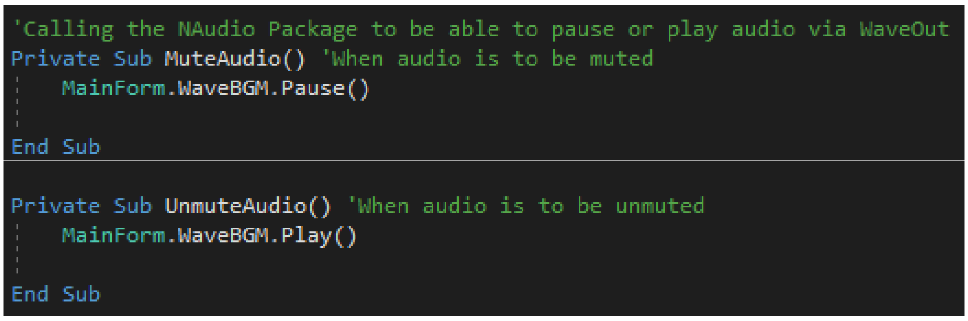
Audio Implementation

In the program, there is background audio and menu sound effects. This section is the one place where an external package has been used in order to handle the background audio. In particular, it is the NAudio package. NAudio uses the Windows MultiMedia API (Mmeapi) in order to control audio that is played through it.



```vb
'Audio cause WHY THE HECK NOT
Public Shared WaveBGM As New NAudio.Wave.WaveOut 'Creates new WaveOut
Public Shared WaveBGMThread As Threading.Thread 'Creates a thread which runs separately from main program

'When program starts, it starts a new thread that runs in the background
Private Sub BeginBGM()
    If IsNothing(WaveBGMThread) Then 'If the thread doesn't exist
        WaveBGMThread = New Threading.Thread(AddressOf WaveBGMThreadMethod) With {.IsBackground = True} 'Create a new thread that runs in the background
        WaveBGMThread.Start() 'Then start the thread

    End If

End Sub

'This thread then plays the background audio, and keeps looping it
Public Sub WaveBGMThreadMethod()
    WaveBGM = New NAudio.Wave.WaveOut 'Make sure this thread can get the WaveOut
    While True 'Loop forever
        If WaveBGM.PlaybackState = 0 Then 'Whenever the song has ended playing
            Dim BGM As New NAudio.Wave.WaveFileReader(My.Resources.BleepingDemo) 'Convert the resource file
            WaveBGM.Dispose() 'Get rid of anything old
            WaveBGM.Init(BGM) 'Get the new song ready
            WaveBGM.Play() 'Start playing the game
            If My.Settings.BGMMuted = True Then 'But if the user has disabled background music
                WaveBGM.Pause() 'Pause the audio
            End If

        End If

    End While

End Sub
```

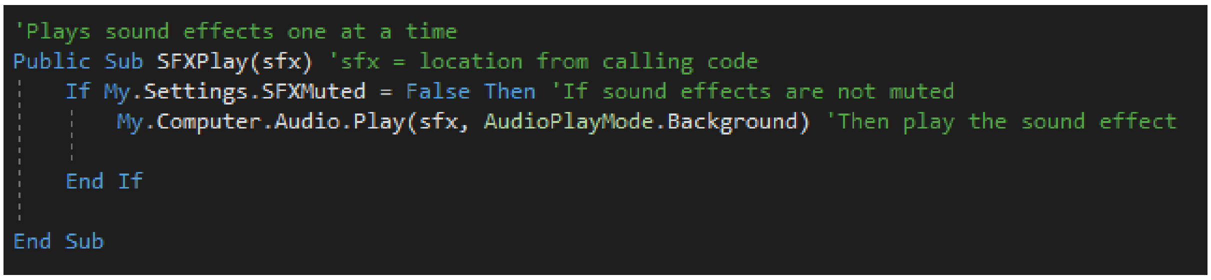
When the program first loads (after setting up themes), a new Thread, 'WaveBGMThreadMethod', is created. This Thread allows the program to execute code in the background while not pausing code that is running anywhere else. When started, the Thread gets the audio file from 'My.Resources', converts it into an applicable file format, gets the Audio Output ready to play the file, and then begins playing the file. If the user has disabled Background Music from the Settings, the song is instantly set to pause before any audio can be heard. This Thread also makes sure that when the audio file has finished playing, everything is reset so that it will loop indefinitely.



```vb
'Calling the NAudio Package to be able to pause or play audio via WaveOut
Private Sub MuteAudio() 'When audio is to be muted
    MainForm.WaveBGM.Pause()

End Sub

Private Sub UnmuteAudio() 'When audio is to be unmuted
    MainForm.WaveBGM.Play()

End Sub
```

When the user wants to mute or unmute the background music through settings, a command is sent to the Audio Output to pause (mute) or play (unmute) the song. The reason the volume is not changed instead is because it will affect the output of the application's entire volume instead of just the audio playing through NAudio, which causes issues with the sound effects (which are implemented differently).

In the same settings menu, the user can disable or enable the sound effects separately. These are played by the standard 'My.Computer.Audio.Play' instead. The reason the background music was not played this way is because 'My.Computer.Audio.Play' can only handle one audio file at a time. Therefore, two methods of playing audio was required.

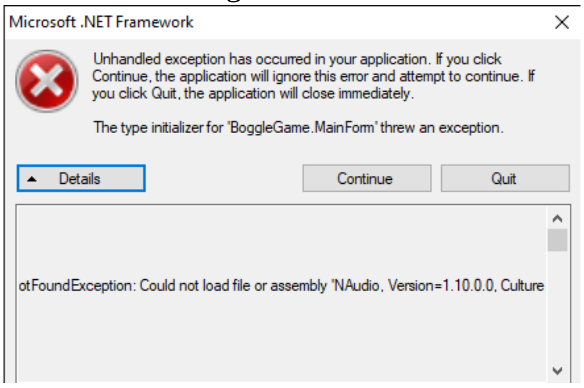


```
'Plays sound effects one at a time
Public Sub SFXPlay(sfx) 'sfx = location from calling code
    If My.Settings.SFXMuted = False Then 'If sound effects are not muted
        My.Computer.Audio.Play(sfx, AudioPlayMode.Background) 'Then play the sound effect

    End If

End Sub
```

If a sound effect is to be played, the calling code passes the location of the sound file (generally from My.Resources) into ‘SFXPlay’. If the user has not muted sound effects, then the sound clip will be played. ‘AudioPlayMode.Background’ determines that the program should not wait for the audio file to finish playing before continuing to execute code.

Remarks about NAudio

Since the Background Music uses an external reference that is not a part of the .NET Framework, it requires some extra files. When the application gets compiled, an external Dynamic-link library (DLL, allows for shared resources across applications) is created alongside the final executable. That DLL is required to be in the same folder as the executable in order for it to run. Even though it was not required for the application to be compiled, it would still have been nice to get a standalone version of the executable.



Microsoft .NET Framework

Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

The type initializer for 'BoggleGame.MainForm' threw an exception.

```
otFoundException: Could not load file or assembly 'NAudio, Version=1.10.0.0, Culture
```

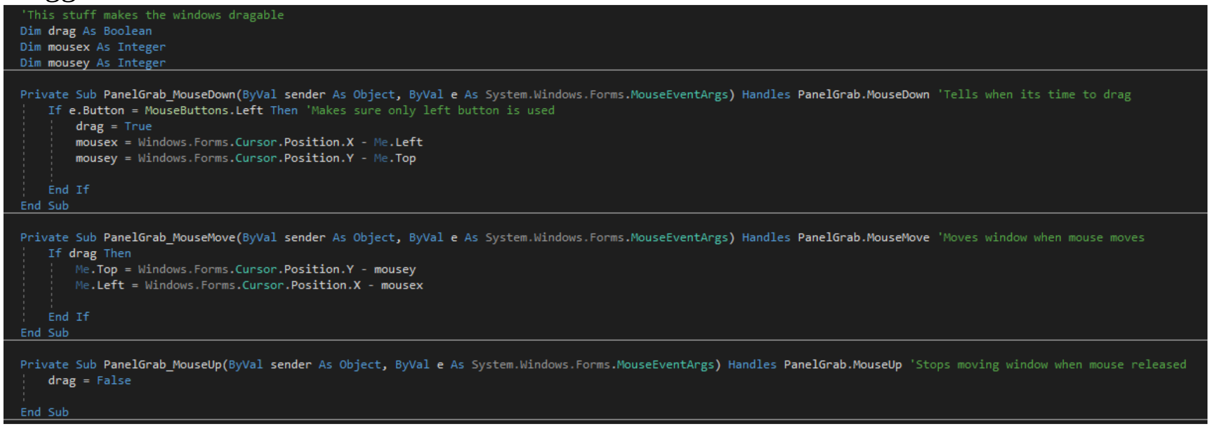
This led to the discovery of ILMerge, which could help merge the executable file and the DLL file. Thanks to this, we are able to have a standalone executable with all of its external references included.

While the original executable and DLL will still exist in the solution’s project files, a merged version of the executable will also exist in a separate folder. Both executables should be the exact same version with no differences apart from the DLL.

Miscellaneous Modules

Dragging Windows

In the software, most of the Forms use the 'None' Borderstyle to achieve a cleaner look. However, this also disables the default window border which allows the window to be dragged across the screen.

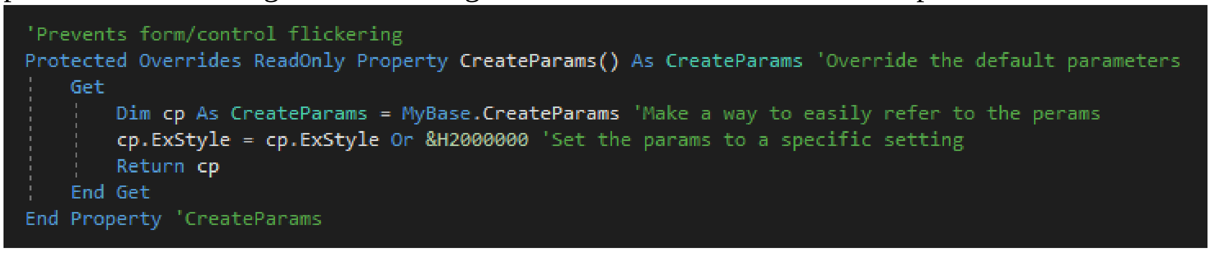


```
'This stuff makes the windows dragable
Dim drag As Boolean
Dim mousex As Integer
Dim mousey As Integer

Private Sub PanelGrab_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles PanelGrab.MouseDown 'Tells when its time to drag
    If e.Button = MouseButtons.Left Then 'Makes sure only left button is used
        drag = True
        mousex = Windows.Forms.Cursor.Position.X - Me.Left
        mousey = Windows.Forms.Cursor.Position.Y - Me.Top

    End If
End Sub

Private Sub PanelGrab_MouseMove(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles PanelGrab.MouseMove 'Moves window when mouse moves
    If drag Then
        Me.Top = Windows.Forms.Cursor.Position.Y - mousey
        Me.Left = Windows.Forms.Cursor.Position.X - mousex

    End If
End Sub

Private Sub PanelGrab_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles PanelGrab.MouseUp 'Stops moving window when mouse released
    drag = False

End Sub
```

To allow the Form to be dragged without the normal Windows border, some extra code was required which would achieve the same purpose. On the top of every Form is a Panel control which changes colour when the user hovers over it. When the left mouse button is clicked on it, it makes the 'drag' variable True. When 'drag' is true, and the mouse is moved, it will make the Form move along with it. But when the mouse button is released, 'drag' becomes False and so the Form can no longer be moved.

Flickering Windows Forms

Due to issues with Windows Forms, trying to update anything on the Form will result in flickering. While this isn't a major issue, it is a slight annoyance and could deter users from using the software due to this issue. However, with some extra lines of code, we are able to prevent the flickering from occurring whenever the form or control is updated.



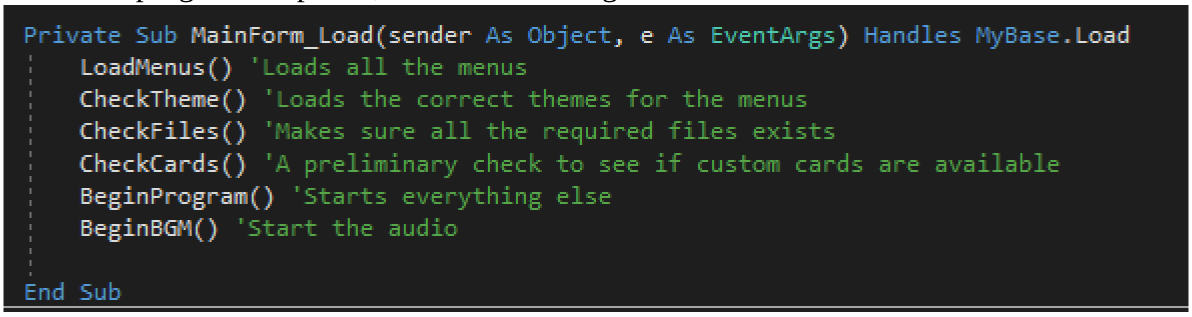
```
'Prevents form/control flickering
Protected Overrides ReadOnly Property CreateParams() As CreateParams 'Override the default parameters
    Get
        Dim cp As CreateParams = MyBase.CreateParams 'Make a way to easily refer to the perams
        cp.ExStyle = cp.ExStyle Or &H2000000 'Set the params to a specific setting
        Return cp
    End Get
End Property 'CreateParams
```

These lines of code override the default parameters which are used when the Form is created. By doing so, it prevents the Form from flickering when anything updates.

However, a side effect is that when the Form is minimised, everything becomes invisible once it returns. This was fixed by setting the Form's TransparencyKey into a colour that is not normally used.

When the Main Form loads

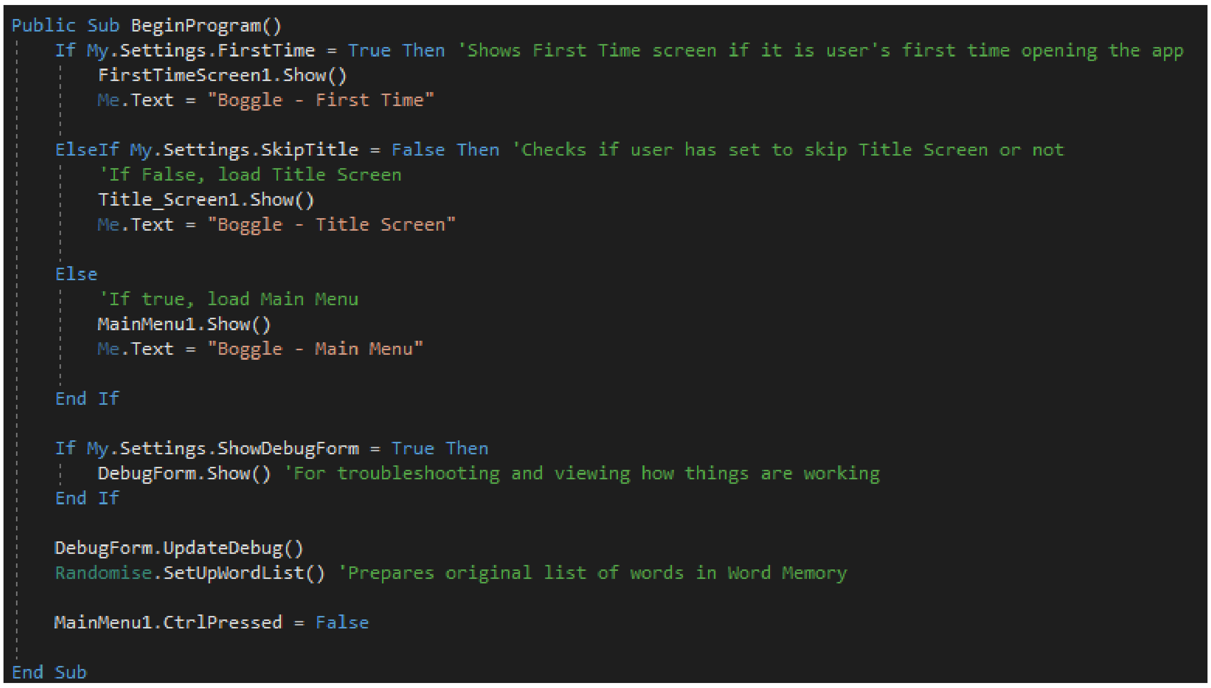
When the program is opened, a number of things are loaded one after another.



```
Private Sub MainForm_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    LoadMenus() 'Loads all the menus
    CheckTheme() 'Loads the correct themes for the menus
    CheckFiles() 'Makes sure all the required files exists
    CheckCards() 'A preliminary check to see if custom cards are available
    BeginProgram() 'Starts everything else
    BeginBGM() 'Start the audio

End Sub
```

First, all of the menus and themes are loaded in from the ‘UserControls’ module (see “The Menus and Themes” on Page 18). Then it’ll check for external files and folders. If these files are not found, it will create a copy of them from My.Resources (also see “My Resources” on Page 27).

Once all those have been done, the program will check for any menus that it has to display. This depends on whether it’s the user’s first time opening the program, or they want to skip the Title Screen. Similarly, it’ll check for the ‘Show Debug Form’ setting on whether it should display that form or not. Then it will set up the word list from the ‘Randomise’ module.

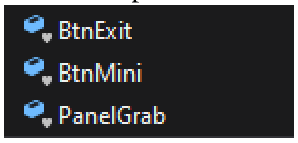


```
Public Sub BeginProgram()
    If My.Settings.FirstTime = True Then 'Shows First Time screen if it is user's first time opening the app
        FirstTimeScreen1.Show()
        Me.Text = "Boggle - First Time"

    ElseIf My.Settings.SkipTitle = False Then 'Checks if user has set to skip Title Screen or not
        'If False, load Title Screen
        Title_Screen1.Show()
        Me.Text = "Boggle - Title Screen"

    Else
        'If true, load Main Menu
        MainMenu1.Show()
        Me.Text = "Boggle - Main Menu"

    End If

    If My.Settings.ShowDebugForm = True Then
        DebugForm.Show() 'For troubleshooting and viewing how things are working
    End If

    DebugForm.UpdateDebug()
    Randomise.SetUpWordList() 'Prepares original list of words in Word Memory

    MainMenu1.CtrlPressed = False

End Sub
```

At last, it will begin the audio (also see “Audio Implementation” on Page 31) to complete the load sequence. These sequences of events happen every time the program is opened, including if the Main Menu’s restart shortcut is used.

Various Controls
A large number of controls were used in the making of this application. In order to keep track of the types of controls, most of their names have been simplified with an easy to read format.

For Example:

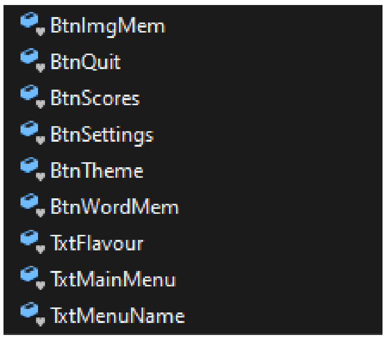


BtnExit
BtnMini
PanelGrab

These are the controls on the MainForm. As this form contains all the other menus, it does not have many controls. It can be seen that the first half of a name denotes the type of control it is. 'Btn' = Button. 'Panel' = Panel. The second half of the name denotes its purpose. 'BtnExit' is a Button made to Exit the application. BtnMini is a Button to Minimise the application. 'PanelGrab' is a Panel used to Grab the window to drag around.

A list of controls for essential menus are below, with explanations for a number of unique controls.

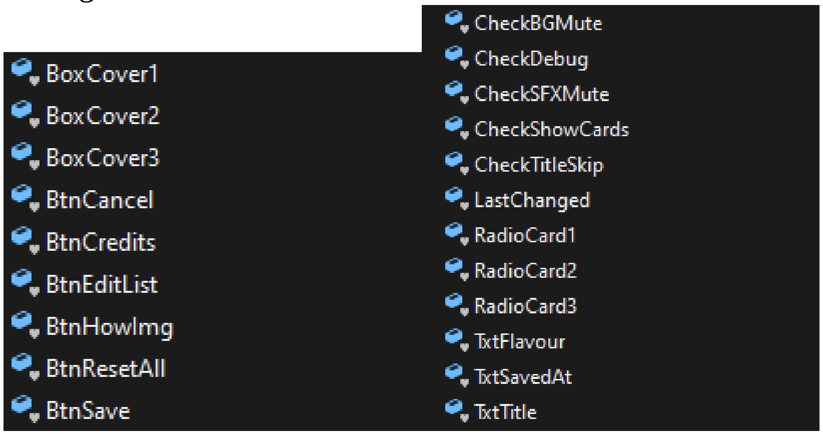
MainMenu:



BtnImgMem
BtnQuit
BtnScores
BtnSettings
BtnTheme
BtnWordMem
TxtFlavour
TxtMainMenu
TxtMenuName

Txt = Label (As Labels are often filled with Text)
TxtFlavour = Text which talks a bit about the menu (Flavour Text)

Settings:



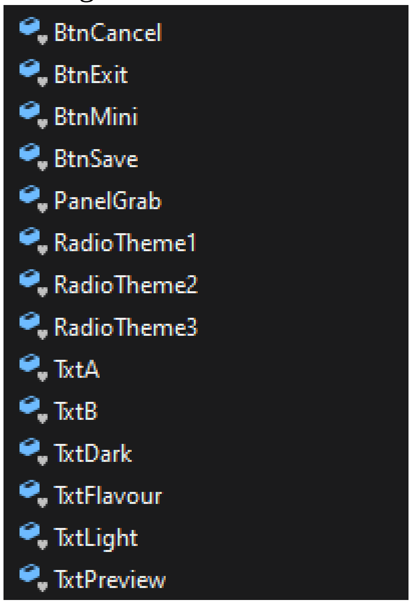
BoxCover1
BoxCover2
BoxCover3
BtnCancel
BtnCredits
BtnEditList
BtnHowImg
BtnResetAll
BtnSave

CheckBGMute
CheckDebug
CheckSFXMute
CheckShowCards
CheckTitleSkip
LastChanged
RadioCard1
RadioCard2
RadioCard3
TxtFlavour
TxtSavedAt
TxtTitle

Box = PictureBox (denotation only for Settings)
Check = CheckBox
Radio = RadioCard
BtnEditList = Button explaining the custom word list.

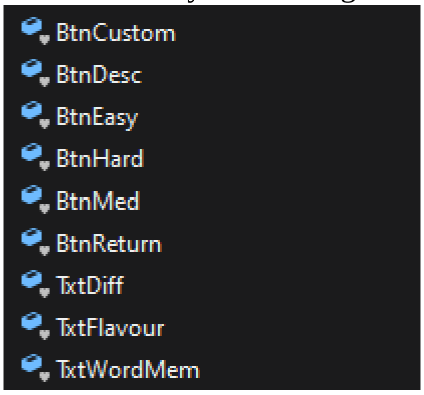
BtnHowImg = Button explaining custom images.

SettingsTheme:


BtnCancel
BtnExit
BtnMini
BtnSave
PanelGrab
RadioTheme1
RadioTheme2
RadioTheme3
TxtA
TxtB
TxtDark
TxtFlavour
TxtLight
TxtPreview

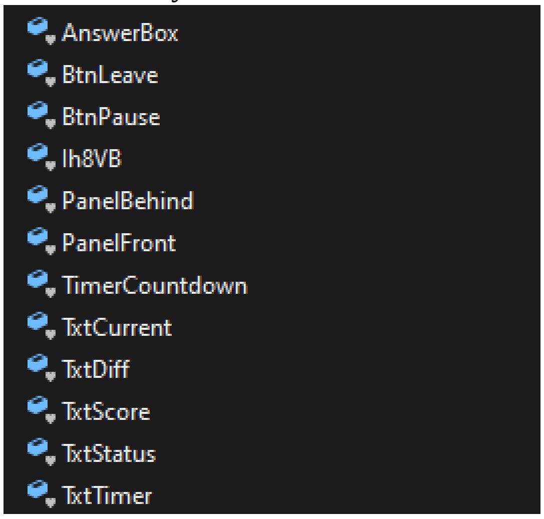
BtnExit, BtnMini and PanelGrab exist here as SettingsTheme is a separate form, thus needing its own panel to allow moving around.
TxtA and TxtB are the Labels denoting 'Light Theme' and 'Dark Theme'.
TxtDark and TxtLight explain the differences between Dark and Light themes.

WordDifficultySelect/ImageDifficultySelect:


BtnCustom
BtnDesc
BtnEasy
BtnHard
BtnMed
BtnReturn
TxtDiff
TxtFlavour
TxtWordMem

Diff = Difficulty
BtnDesc means Button Description, where Description = How to Play.
WordDifficultySelect and ImageDifficultySelect have the exact same controls, except for TxtWordMem. It is TxtImgMatch for ImageDifficultySelect instead.

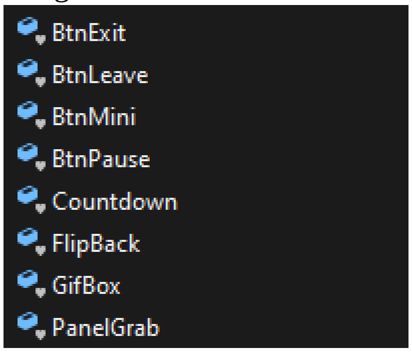
WordMemoryGame:



AnswerBox
BtnLeave
BtnPause
Ih8VB
PanelBehind
PanelFront
TimerCountdown
TxtCurrent
TxtDiff
TxtScore
TxtStatus
TxtTimer

AnswerBox = A textbox where the user can input text.
Ih8VB = A timer used to focus control onto the AnswerBox. Named as a developer joke due to frustrations with Visual Basic. It spells "I hate Visual Basic".
PanelBehind and PanelFront = Panels used for the vertical progressbar. PanelFront is a border around the bar. PanelBehind is the actual coloured bar that is manipulated.

ImageMatchForm:

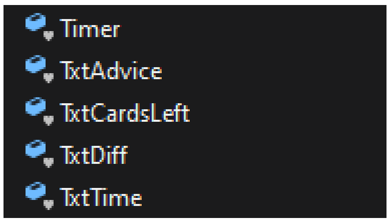


BtnExit
BtnLeave
BtnMini
BtnPause
Countdown
FlipBack
GifBox
PanelGrab

Countdown is a timer which counts down the time until the game properly starts.
FlipBack is another timer which hides the uncovered cards after a certain time.
GifBox is a PictureBox which contains a gif (animated image) inside.
BtnLeave exits the game while BtnExit exits the entire application.

PB1 to PB18
PictureBoxes which are the cards the user clicks on.

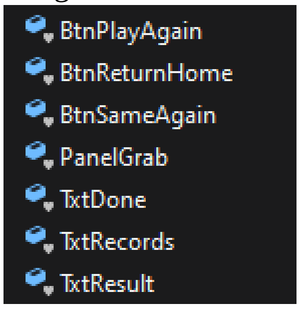
PBack, Pic1 to Pic9
PictureBoxes which contain the images which will be used in the game.



Timer
TxtAdvice
TxtCardsLeft
TxtDiff
TxtTime

Timer is a timer which counts how many seconds (and milliseconds) have passed.
TxtAdvice is the text under the GifBox.

ImageResults/WordResults:



BtnPlayAgain
BtnReturnHome
BtnSameAgain
PanelGrab
TxtDone
TxtRecords
TxtResult

BtnPlayAgain is if the user wants to return to the difficulty select screen.
BtnSameAgain is if the user wants to replay the game with the exact same settings.

The only difference between ImageResults and WordResults is that:

- WordResults has an extra Label, ‘TxtWellDone’ which is a different Label to congratulate the user.
- WordResults has an extra Button, ‘BtnResults’ to bring up the word list.
- WordResults does not have ‘PanelGrab’ as ImageResults is its own form.