Sam Milward

 N0690641

Week 10 programming project – sam milward

Contents

[Analysis 2](#_Toc468569554)

[Overview 3](#_Toc468569555)

[Design 4](#_Toc468569556)

[Functions of the game 11](#_Toc468569557)

[Testing 15](#_Toc468569558)

[Improvements 18](#_Toc468569559)

[Evaluation 20](#_Toc468569560)

# Analysis

For my end of term project, I have decided to take on the challenge of making a graphical game. Completing this challenge will show that I have great understanding of how to use the programming language Python, as well as being able to use an extension of Python called PyGame.

I focus my end user on a young generation who are being introduced to the world of computer gaming. Due to this focus I will be designing my project to be relatively simplistic so that it is easily playable by all ages.

I have found out that my focus group prefers more cartoon orientated characters, as this is what they have grown up with from influences such as kid’s TV shows and films. I have decided to use this to give the user a familiar background so that it is easy for them to adapt to the game. I am also going to use a range of bright colours so that the game is visually stimulating to the target audience.

The controls for this game will be made as easy as possible, I am expecting my game to only require the use of the arrow keys on the computers keyboard as I believe this will be the simplistic.

From this information, I have managed to conclude some key objectives for my project:

1. To produce a fully functional game that is simplistic enough for the younger generation.
2. For the game to be easily navigable so that non-computer-literate people can use it
3. The game to include objects for the player to avoid and other items that the player should aim to collect.
4. The game should be able to save important information such as the players name and score
5. The player name should be validated so only certain information can be input
6. The high scores should be displayed through the program

I aim to meet objective one with the use of a library called Pygame. This will allow me to create a simple game which will also allow for user input using the keyboard to control the game. My idea the second objective is to use another module called Tkinter. The functions within this module allow me to create a user interface using windows and widgets. This will give my project a GUI that can be navigated with the use of buttons. For the game to have objects to avoid and collect, I will first make the player and the object move. If the coordinates of the two objects match then the item will be collected, or the game will end. Players scores and names will be saved using a text file. At the end of the game I will save the score to a text file. This file can be read at the start of the game to obtain and display the high scores. I will use another function which will validate the players name that they input. I will use a function that compares the strings input using ASCII values, if characters input is any other value than letters, the game will not accept the string.

If I can manage to implement these objectives how I have just stated then my program will run exactly how I intend it to.

# Overview

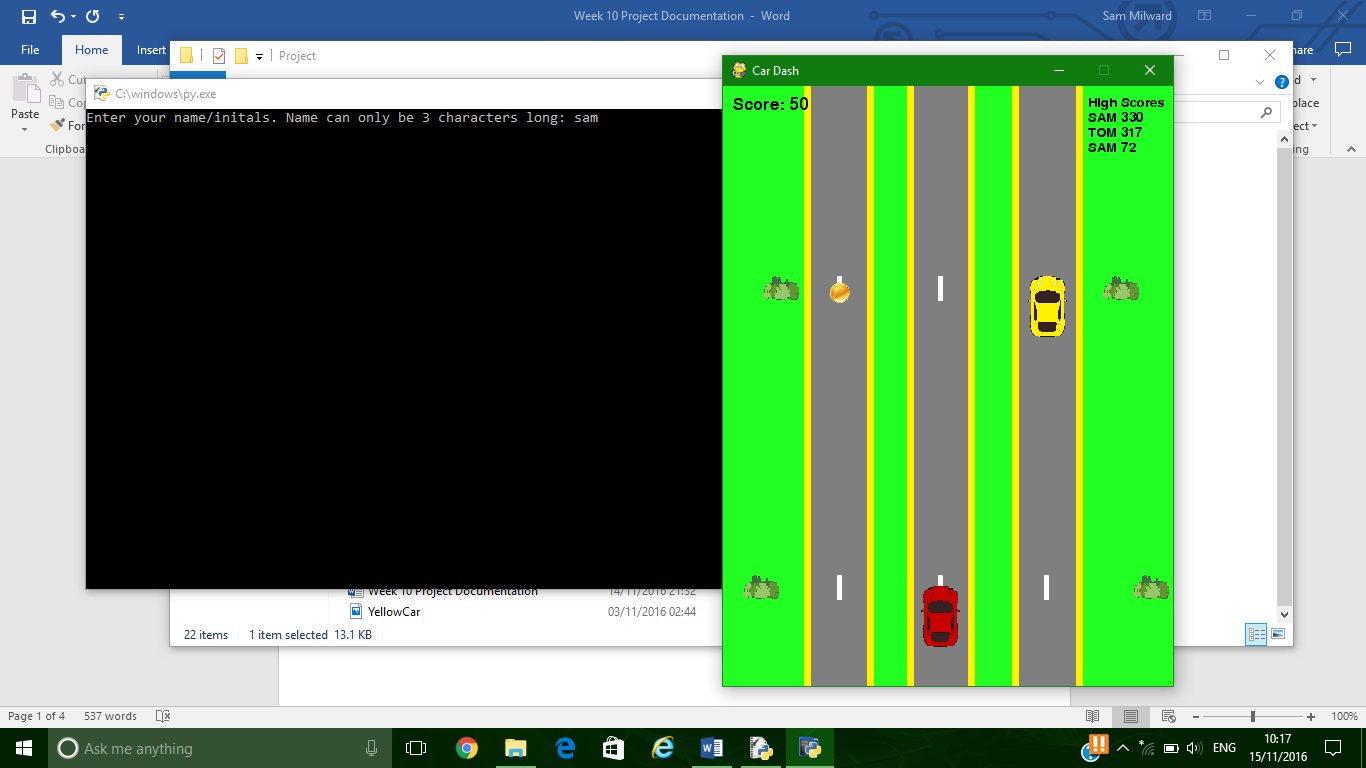
The theme for my game is an old arcade car game. This involves the players car [1] moving from road to road avoiding other traffic [2] to prevent a collision. This, resulting in the end of the game. Whilst playing the user also has the chance to collect any coins [3] that appear on the screen. This will result in an incrementation to their score (Displayed top left corner of the game window) [4]. The players receive one point for every second that they surive playing the game; aswell as receiving five points when they collect a coin.

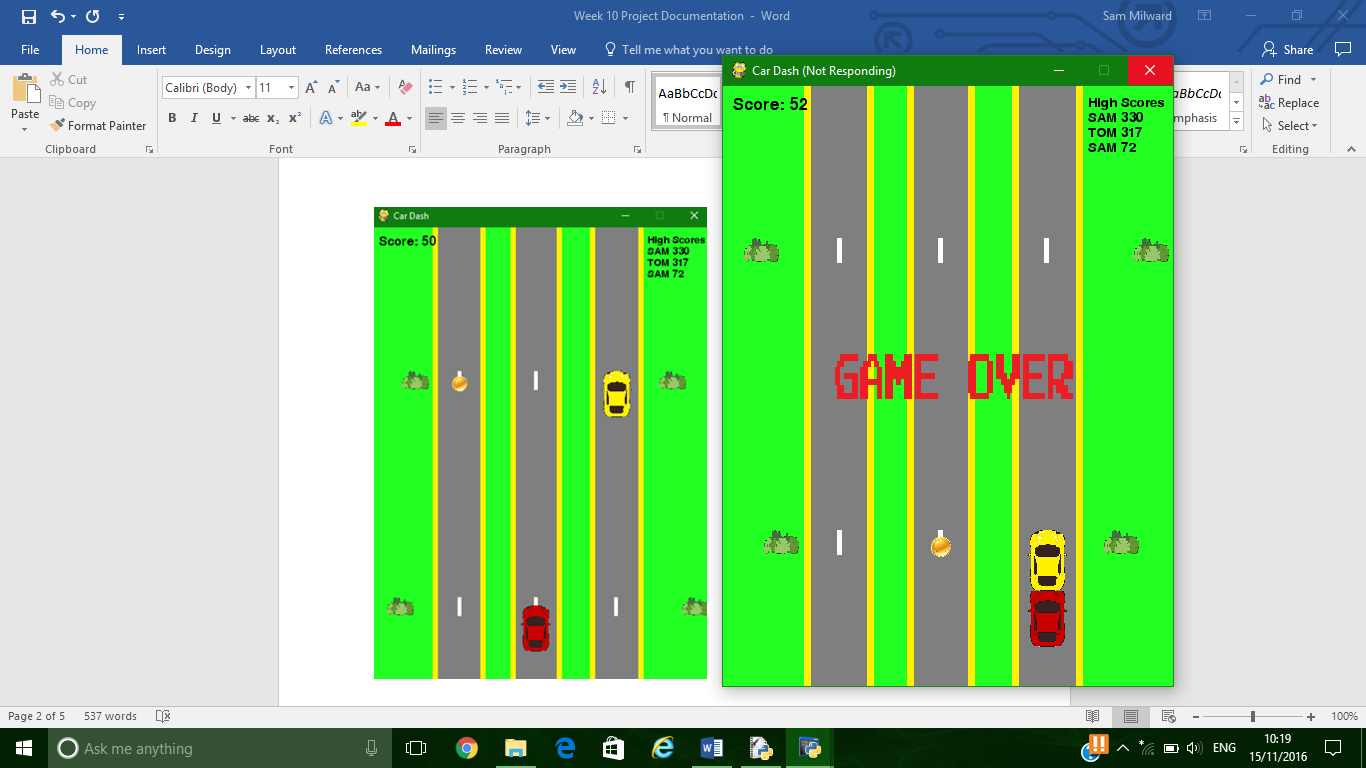
The player controls the vehical with the arrow keys on the keyboard. The play must use the up arrow key to move to the centre lane. From the centre lane the user can either use the left arrow to move into the left lane, or the right arrow key to go to the right lane. The car can only move one lane across at a time. Therefore to go from the left lane to the right lane, the player must first go through the middle lane.

The players score at the end of the game is taken alongisde with their name (entered at the start of the game) to be written to a text file.

The top three high scores are read from the text file and displayed in the top right hand side of the

When the player crashes into another car, the game ends and “GAME OVER” is displayed [6]

The following are screenshots taken from the game labelled with features mentioned.



[4]

[5]

[2]

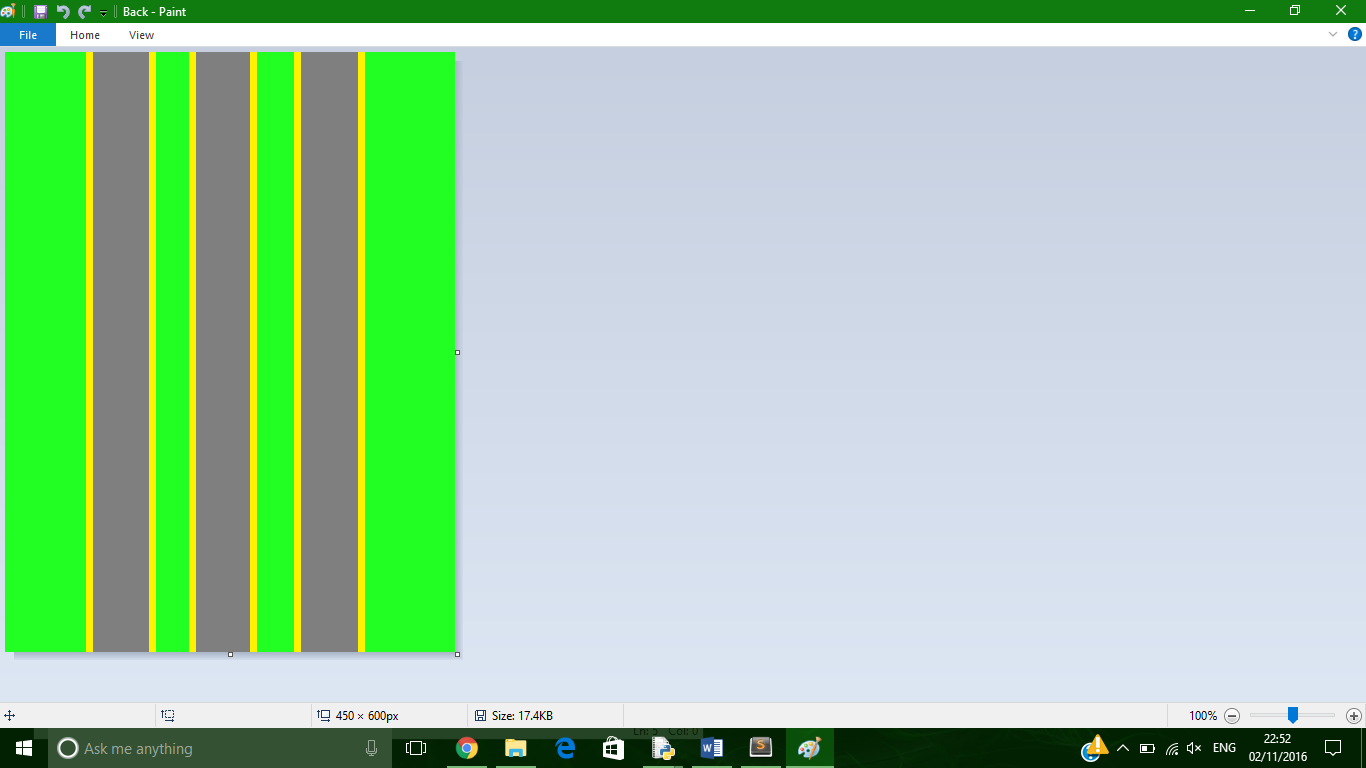
[3]

[6]

[1]

# Design

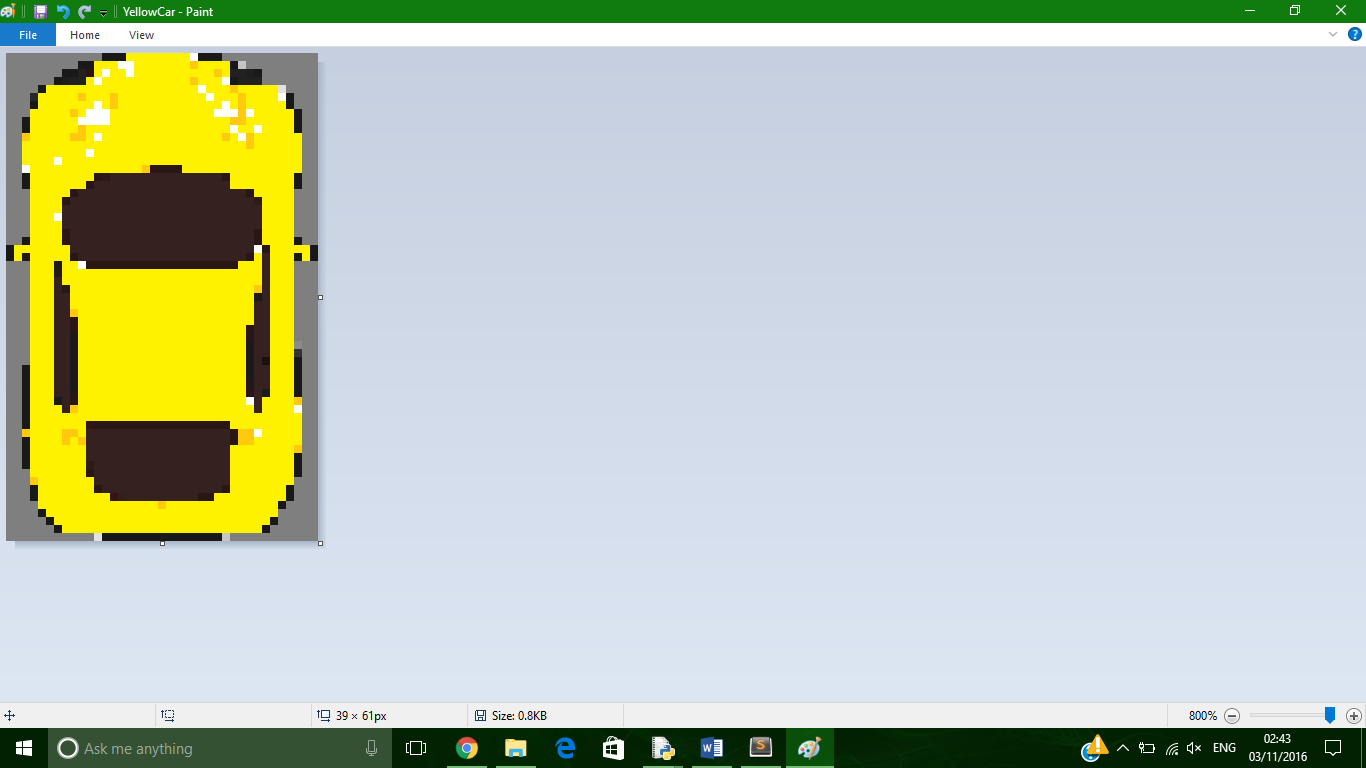
Game Design

I have decided to create the game background myself using Microsoft Paint. This is the first layer of the background that will be stationary throughout the game play. I will give the illusion of the player moving forward by making the other objects in the game move down the page (increasing their y axis value in pygame). The background is just the grass and the three roads that the player moves on.

For the player’s car, I used an image I found on the internet. This image is of a red cartoon car from a top view. I then imported this picture into Microsoft paint to resize, orientate and add details to the image. The link to the website containing the original image is below:

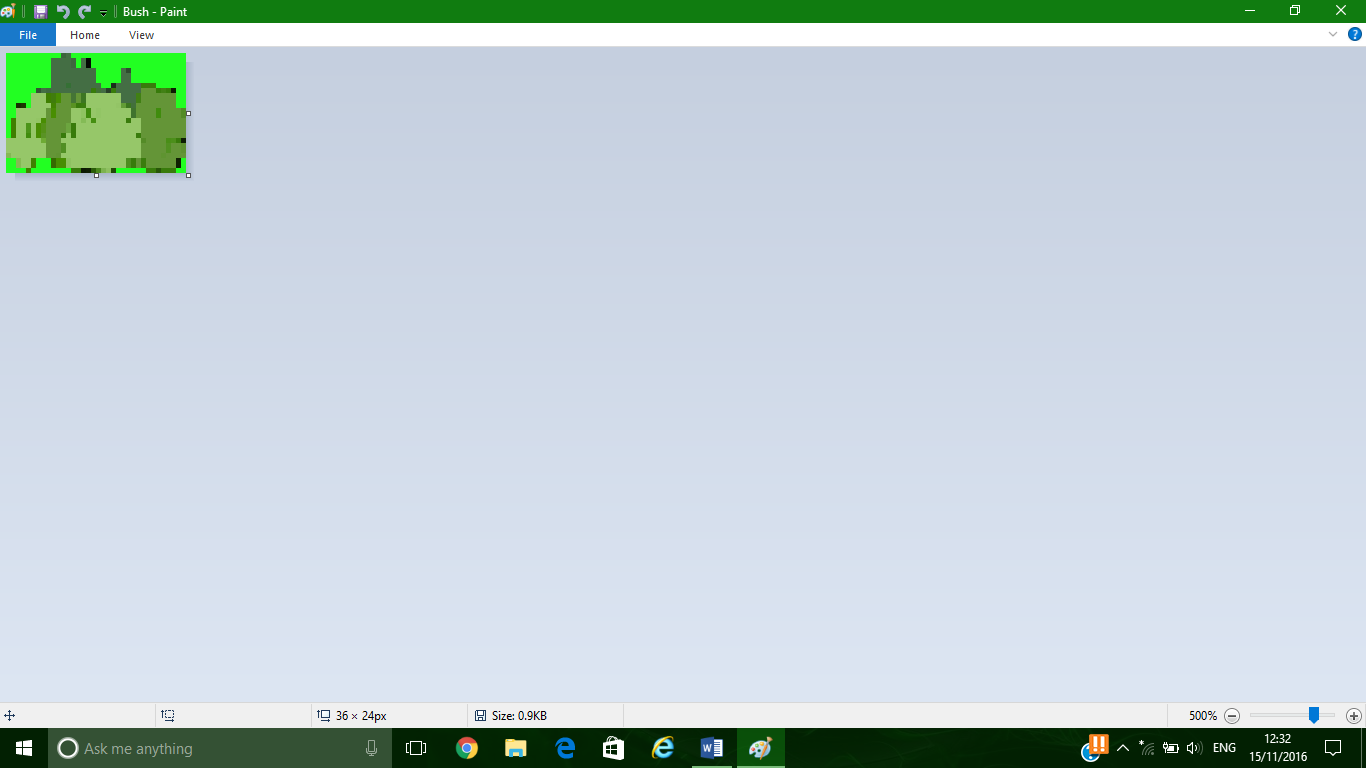
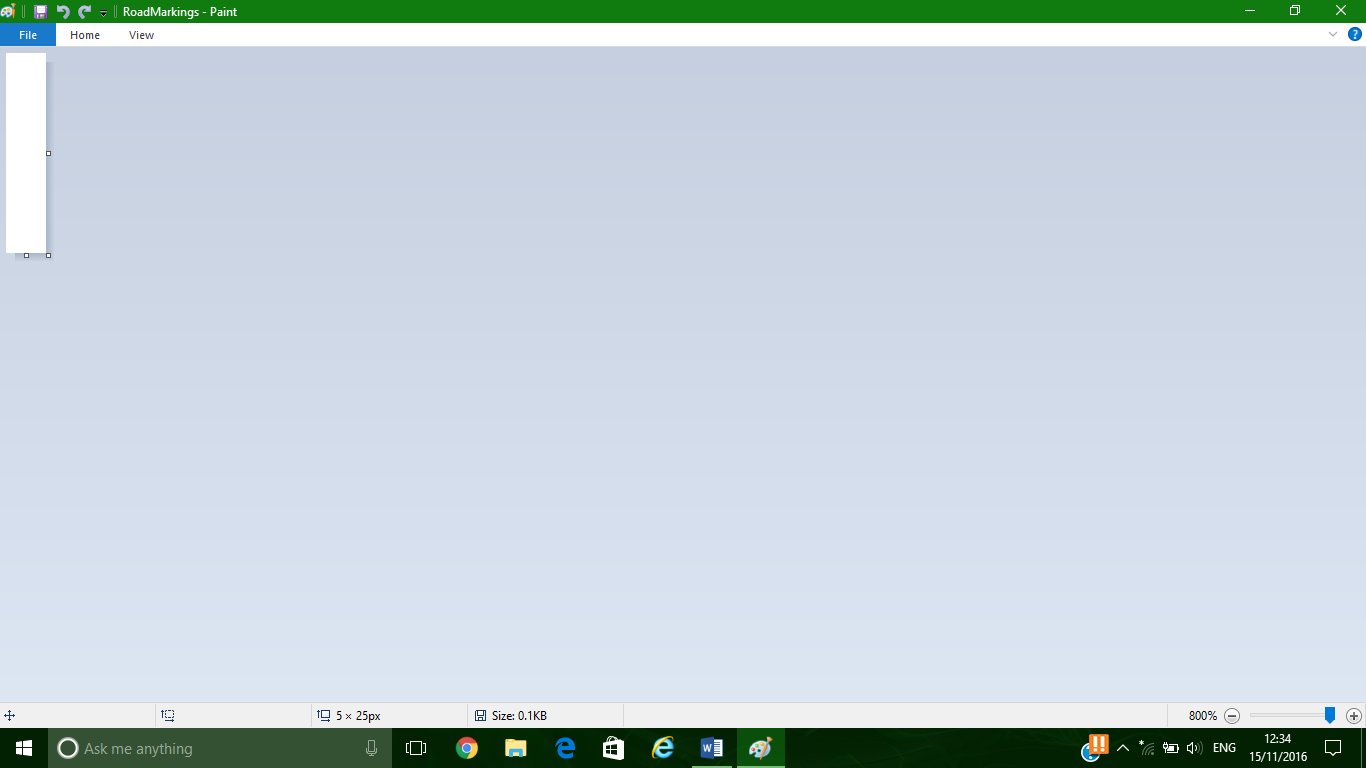
Red car reference - <http://www.clker.com/clipart-red-sports-car-top-view.html>

This is one of the cars that takes place in the game. I created this through manipulating the image of the red car that I downloaded previously. I changed the image size to 40 \* 60 pixels. I changed the colour of the image from red to yellow. I also rotated the image 90 counter clockwise so that the car has the appearance of driving along the road.

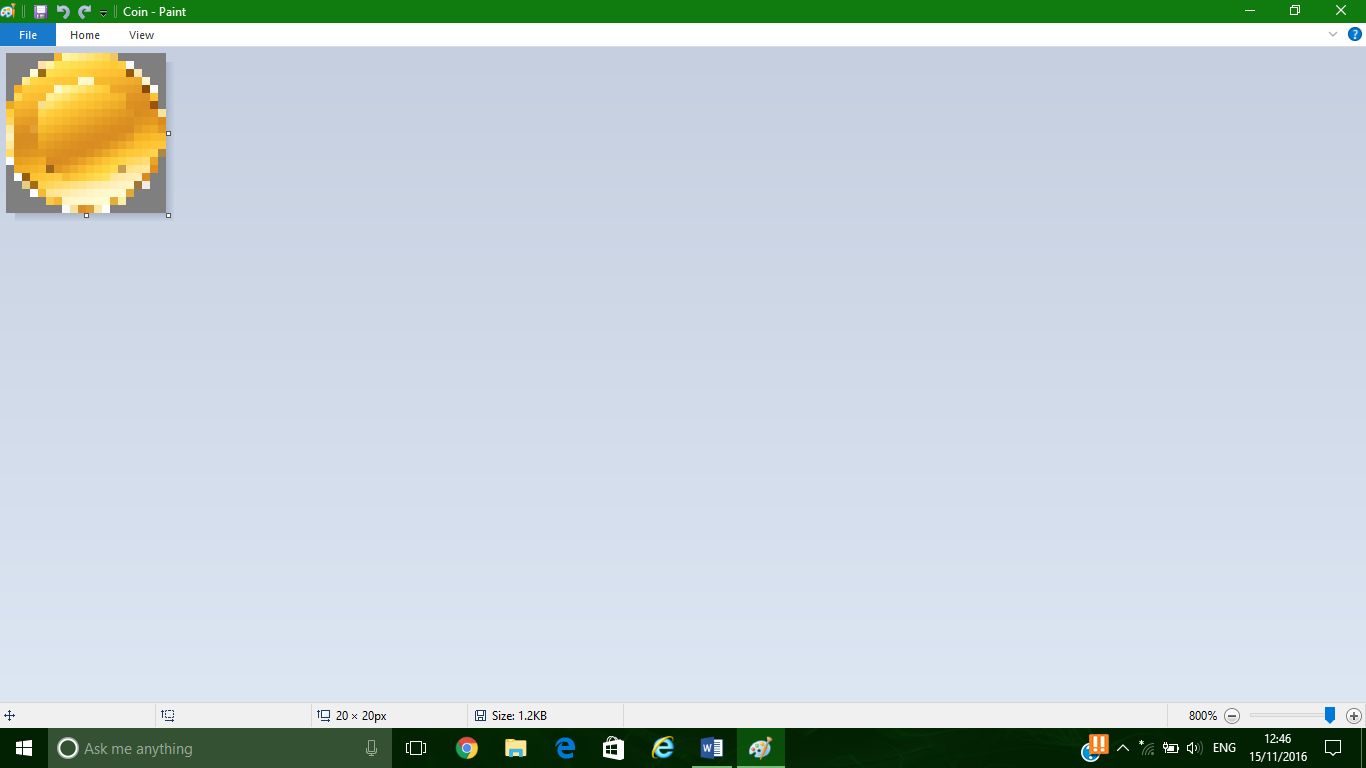


I used the original red car image to create all the vehicles within the game. I used Microsoft Paint to make all the edits to this image. The other cars just have their colours changed to either blue, green, red, or yellow.

For decoration to the game I added a view extra items that move throughout the game. I added an image of a bush that is placed at the side of the game on the grass to add more detail to the game. I also added six road markings for the three roads, these move down the road giving the illusion that the player is moving forwards.

Both of these were created through Paint as I needed the size of the images to be perfect in order to fit them in to the game seemlessly.

Throughout the game coins are spawned for the user to collect to boost their score. These coins were created by finding an image online and then rescaling it and changing the back ground of them.

I changed to colour of the coin’s background to the same colour as the road, therefore there will be no box surrounding the coin even though it is a square image.

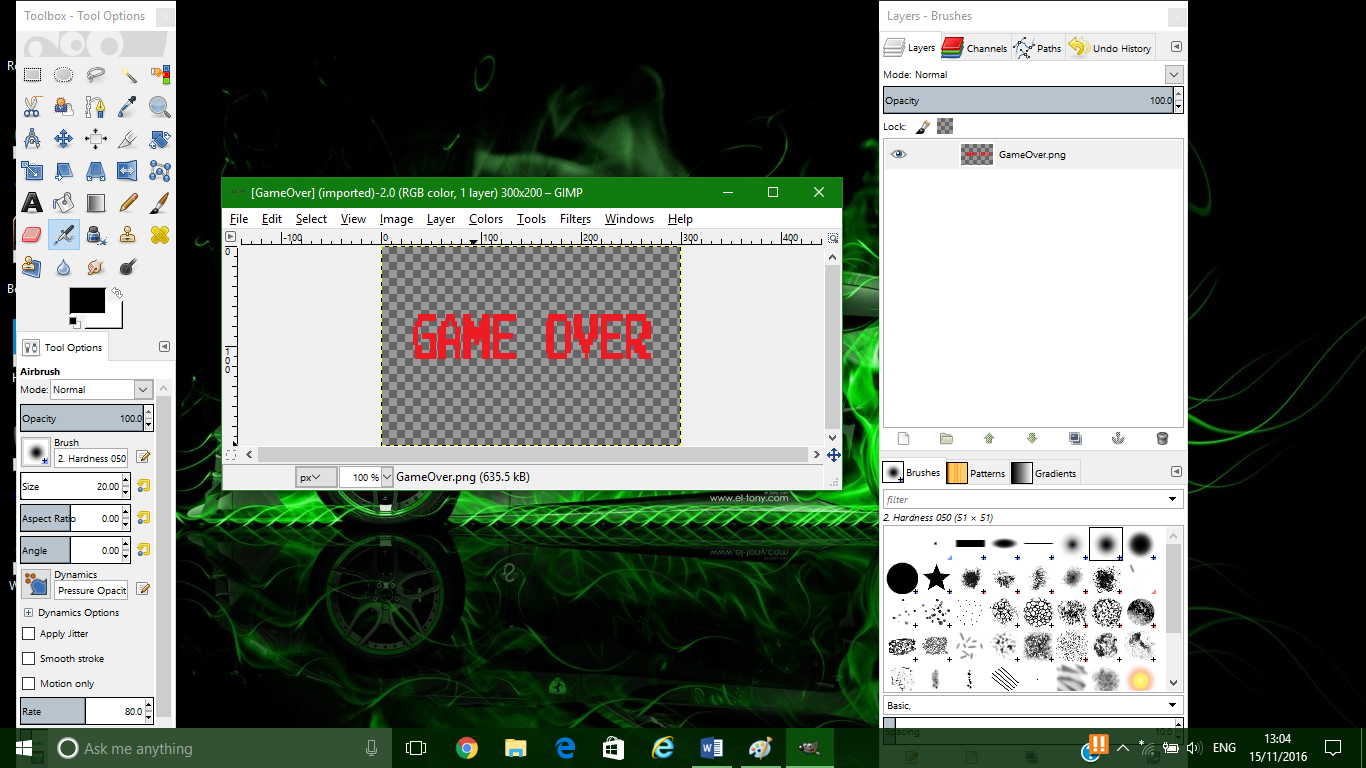
The original coin image is shown on this website below:

Coin reference - <http://www.keyword-suggestions.com/Z29sZCBjb2luIGljb24/>

Finally, the last piece of graphics to be displayed is the game over banner at the end of the game. When the player crashes into another car, the game ends, and the words “GAME OVER” appear on screen, before the menu appears again.

The original image of the game over is on this website: <http://game-over-dex.wikia.com/wiki/Pro_Wrestling_(NES)>.

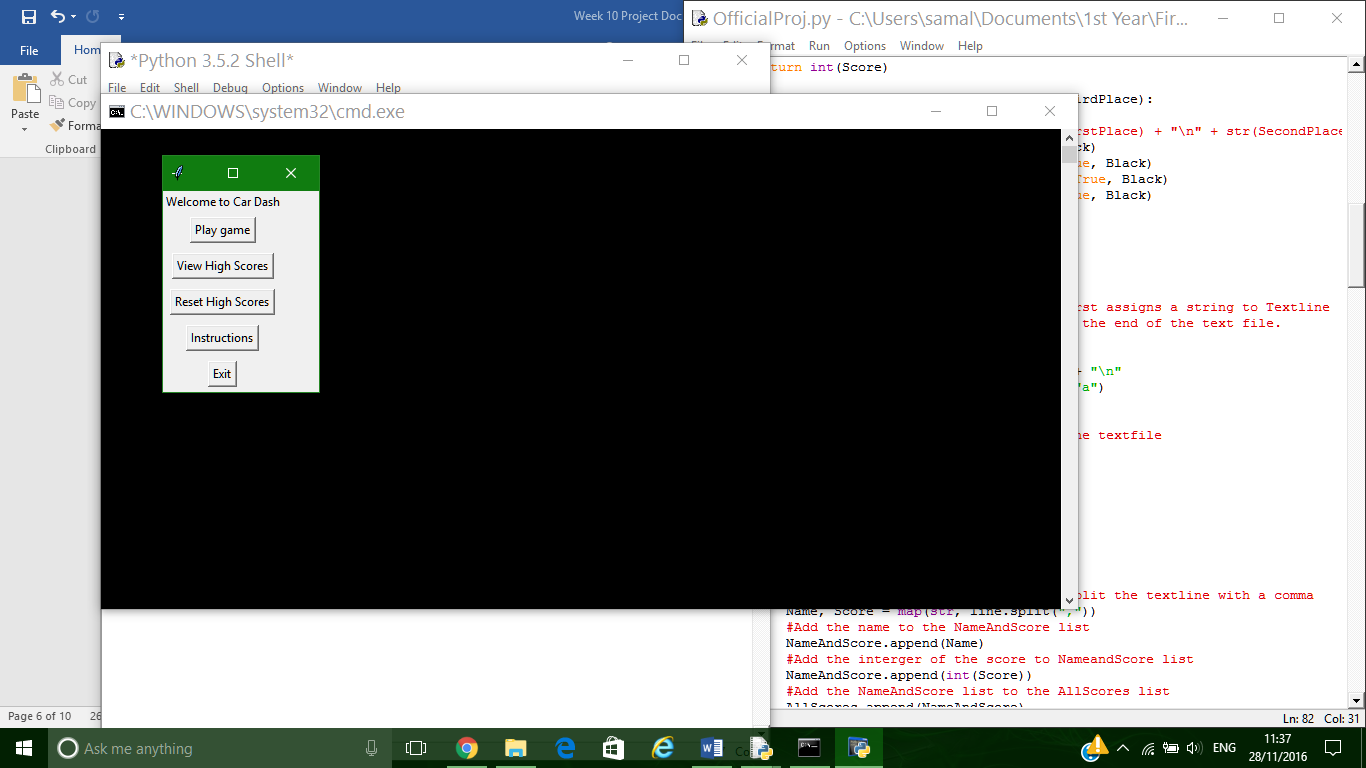
In order for this image to fit in to my game well I had to edit the size of it through paint. However this was not the only adjustment I had to make to the image. The original image has a black background. I downloaded an image editor software called GIMP to remove this background.



Now that I have removed the background of the image it means that it will appear on the gaming screen as the text only.

Menu Design

For my menu interface, I decided to use Tkinter. This is due to it being a simplistic way of generating a GUI that can easily be navigated by users who have little experience with computers. I decided to make my Start Up menu as simplistic as possible, therefore the menu consists only of buttons that navigate to different Tkinter files that lead to the execution of a function.

Here is a screenshot of the Start Up menu that appears when the program is loaded, and again once the game has ended.

1

2

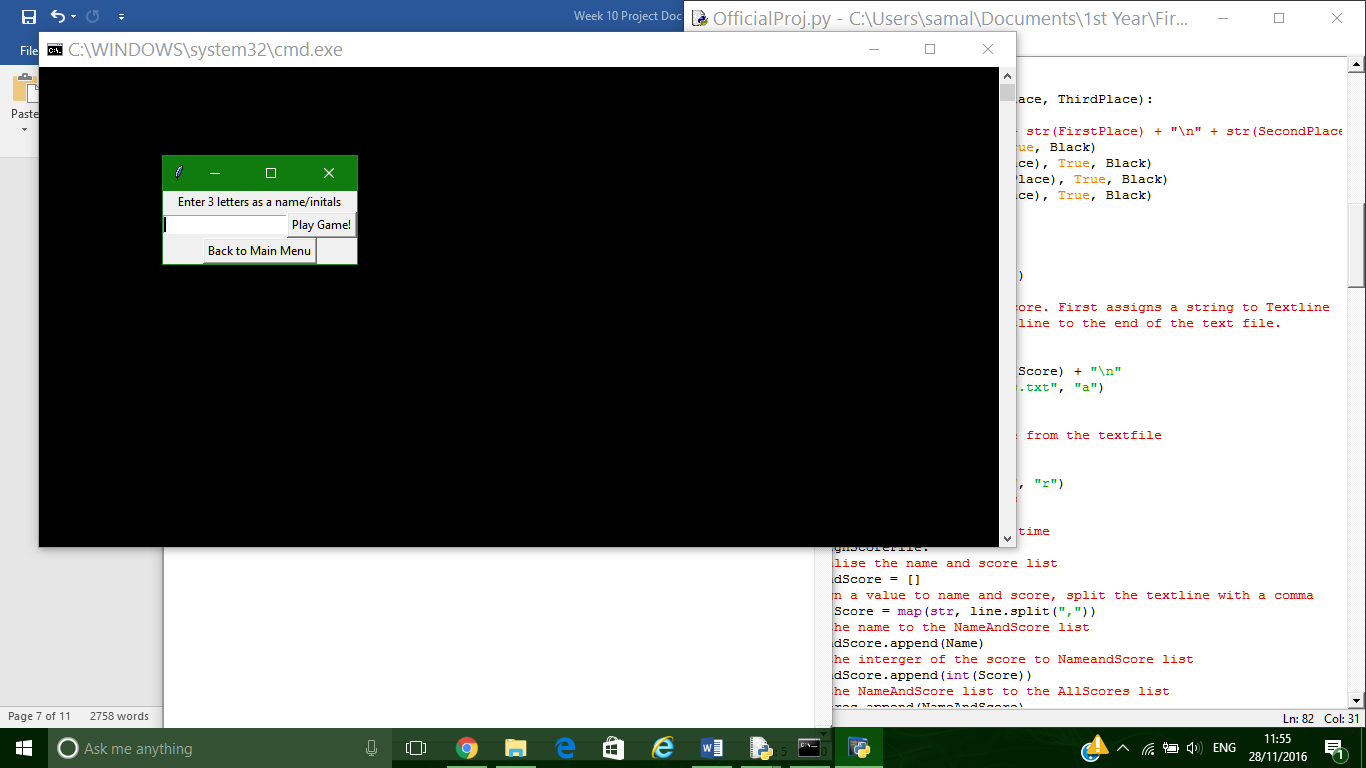
3

4

5

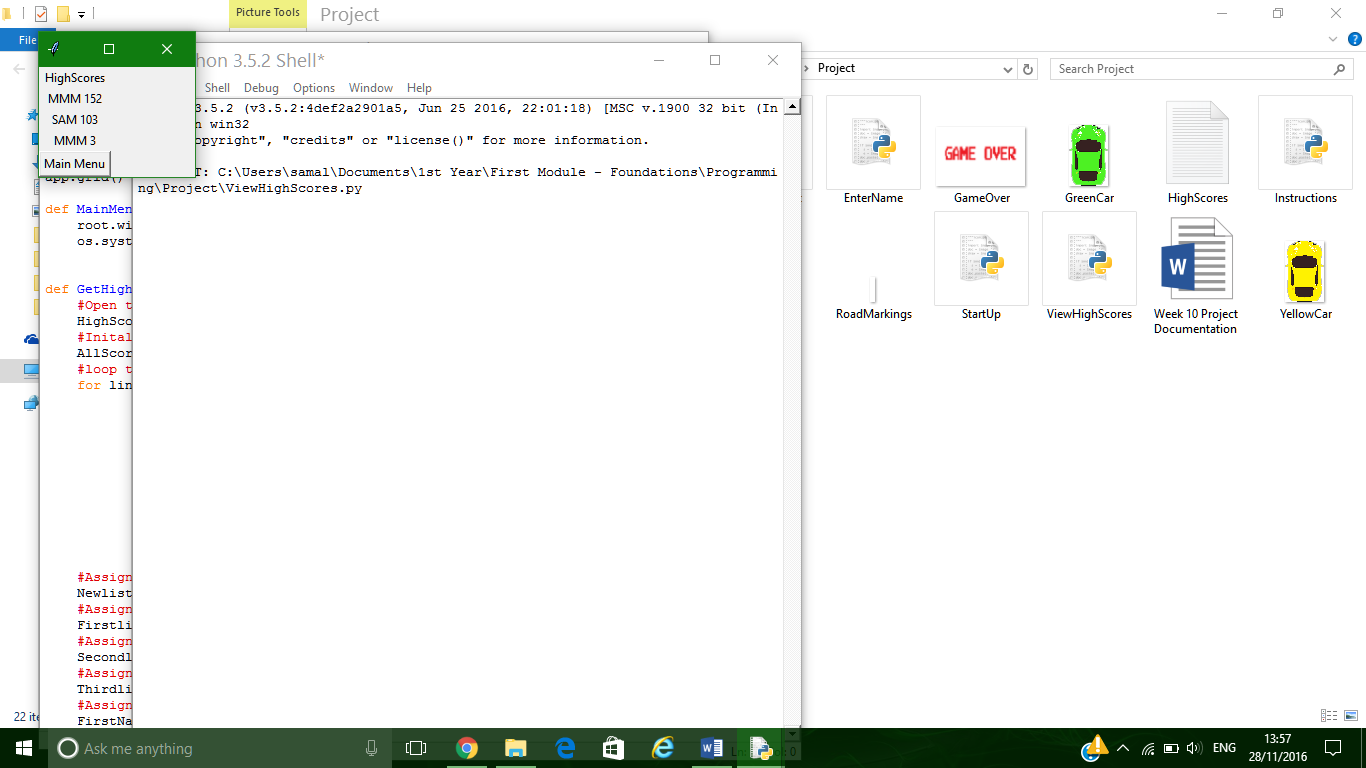
From this menu, the user has 5 options. They can either start to play ‘Car Dash’, View the high scores of the game, reset the high scores of the game, view the instructions, or exit the program completely.

When the user clicks on Play Game [1], the current window is shut down and a new file, “EnterName.py” is loaded. This new program that is loaded gives the user the ability to enter 3 characters that they want to represent their name if they make it on to the high scores board.



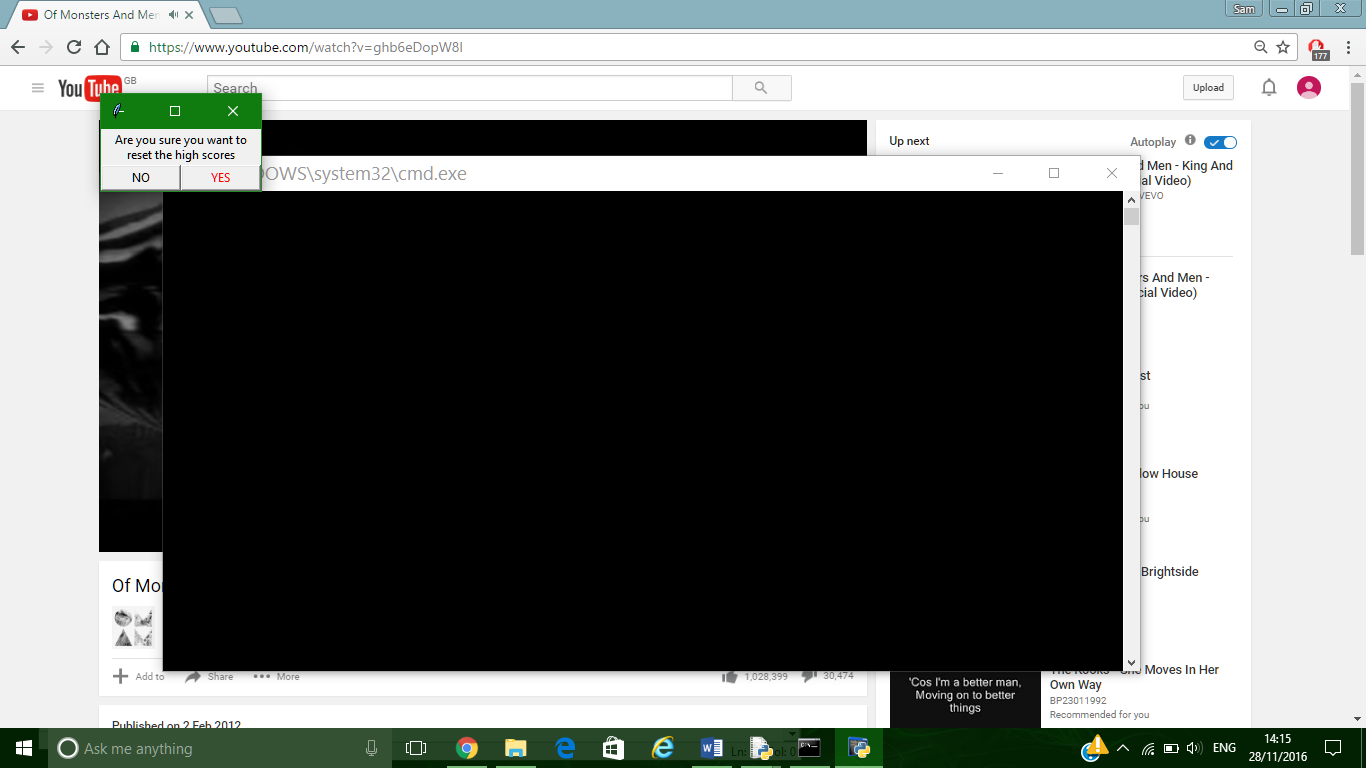
This window has a text box entry that allows input from the user. The game will only start when the player enters a 3-character long string. If the length of the string entered is not 3 then the text above the text box stating ‘Enter 3 letters as a name/initials’ will turn red attracting the users eye to the text. If the user changes their mind they can go back to the main menu with the ‘Back to Main menu’ button.

The view high scores [2] button takes the user to another Tkinter file. This page displays the top three high scores from the game, just like the game does.



This form uses the same function GetHighScores() as the game program. Therefore, only the top 3 high scores are displayed from the text file.

Reset high scores [3] takes the user to a confirmation page to make sure that they want to reset the high score.



If the Yes button is clicked, an algorithm re writes the file:

**def ResetHigh():**

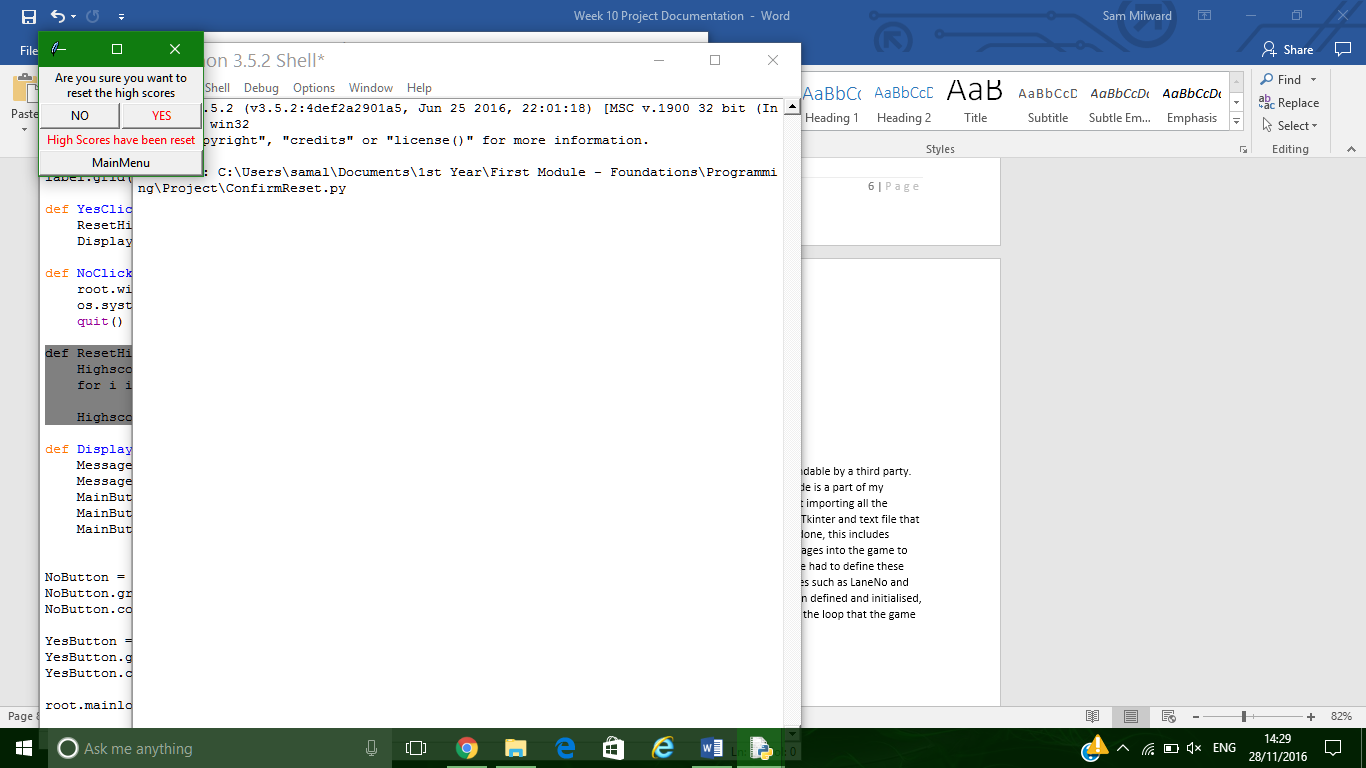
**Highscores = open("Highscores.txt", "w")**

**for i in range (1, 4):**

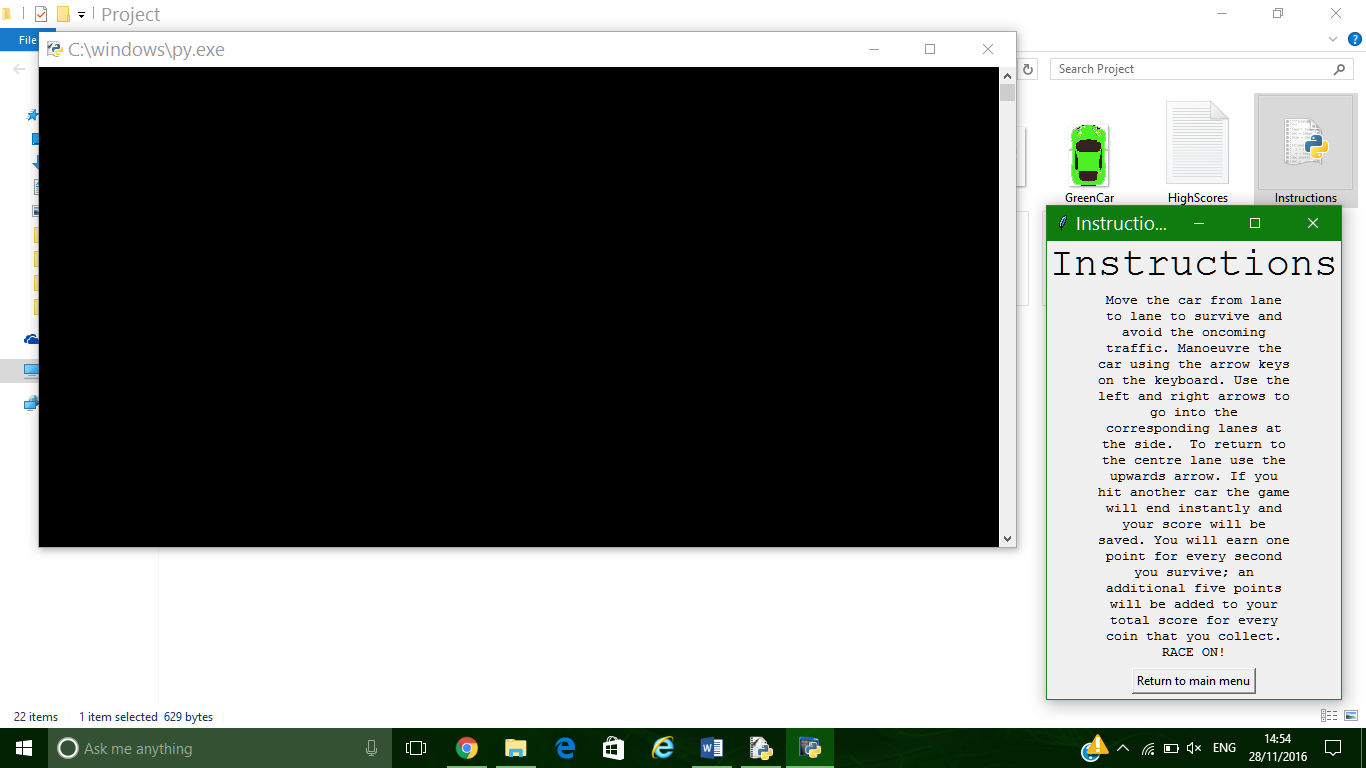
**Highscores.write("N/A,0 \n")**

**Highscores.close()**

Along side this the form also changes to the following to show that the file has been reset. From here the user can then go back to the main menu.



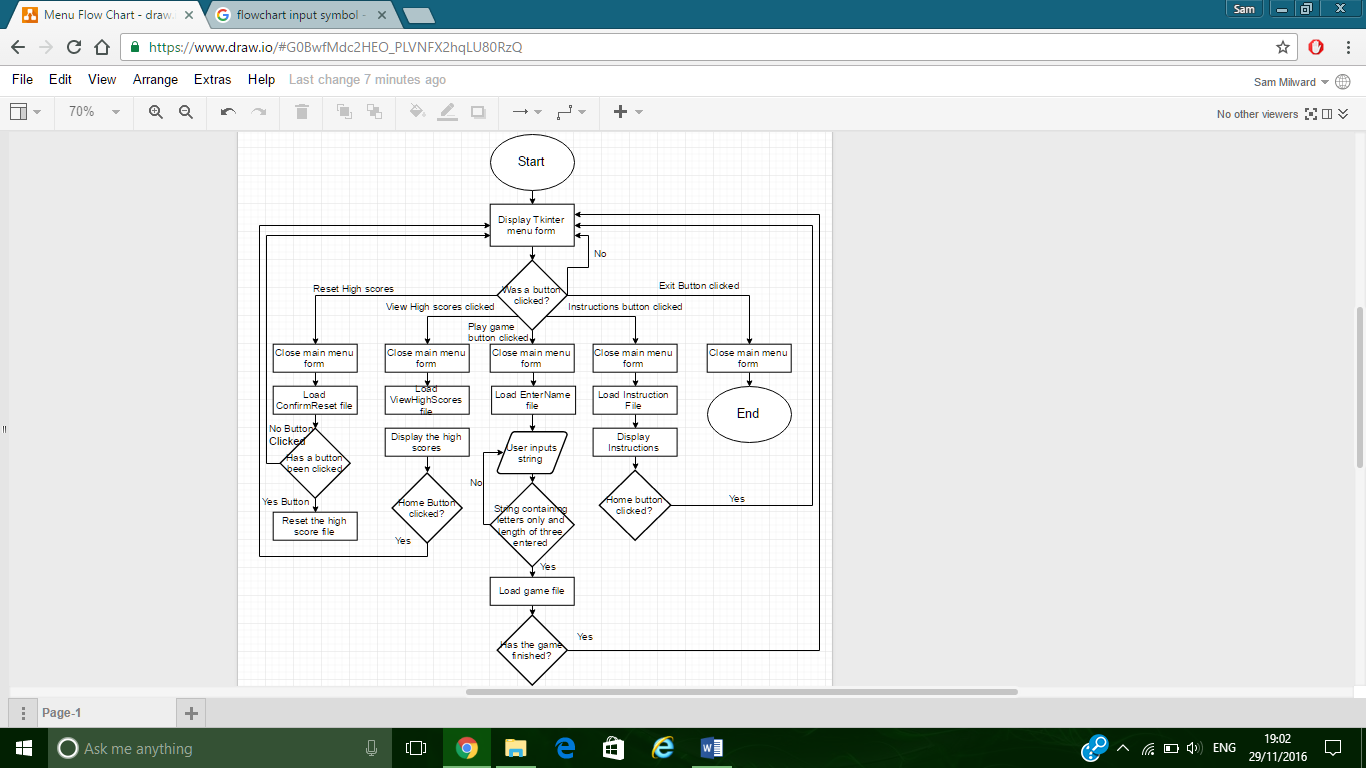
Instructions [4] take the user to a page that explains the rules and controls of the game. The instructions are read from a textfile.

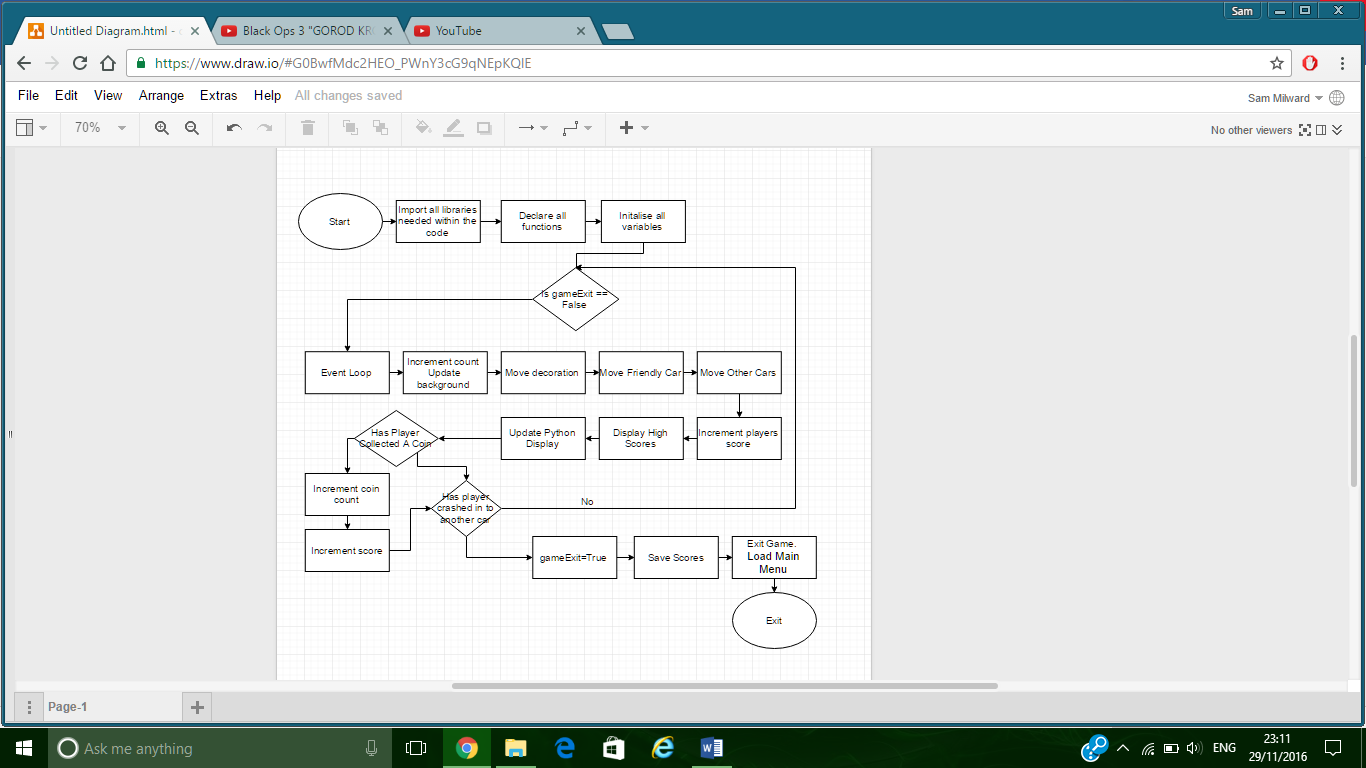


I formatted the text so that it looked more user friendly. This included a bigger title and larger text as the user will be reading a larger amount of text.

Exit[5] ends the program.

The following flow chart shows all possible functions that can be executed from the main menu. This includes all the different forms being loaded and the main game to start to run. All of the separate forms can link back to the main menu, so that the user can navigate through the system easily.



The next flow chart describes the steps of the main game. There is a main game loop that calls functions until the player crashes into another car. After the game ends the start menu is loaded again so that the user can navigate through the other functions within the program.

no

yes

# Functions of the game

I have formatted my code in a way so that it is easily readable and understandable by a third party. At the top of the page I use a commented-out section to explain that this code is a part of my programming project. The first part of code that is executed is the statement importing all the modules that my program uses. The players name is then received from the Tkinter and text file that was called before the game is loaded. After this the setup of the PyGame is done, this includes initialising PyGame, setting the dimensions of the window, and importing images into the game to be used. After this I defined all my functions and procedures in groups. I have had to define these functions before initialising all my variables for the program as some variables such as LaneNo and CoinNo I retrieve their initial value from a function. Once everything has been defined and initialised, the main loop that the game runs starts to be execute. The following code is the loop that the game runs.

**while gameExit == False:**

**#Lets user quit the game using the red cross in the top corner**

**for event in pygame.event.get():**

**if event.type == pygame.QUIT:**

**pygame.quit()**

**quit()**

**#If a key is pressed down, left key move a lane to the left, right move a lane to the right**

**if event.type == pygame.KEYDOWN:**

**if PlayersCarLane == 198.5:**

**if event.key == pygame.K\_LEFT:**

**PlayersCarLane = 97.5**

**elif event.key == pygame.K\_RIGHT:**

**PlayersCarLane = 305.5**

**elif PlayersCarLane == 97.5:**

**if event.key == pygame.K\_UP:**

**PlayersCarLane = 198.5**

**elif PlayersCarLane == 305.5:**

**if event.key == pygame.K\_UP:**

**PlayersCarLane = 198.5**

**#Count is calculated by the result of the function SpeedIncrement**

**count = SpeedIncrement(count)**

**#DecorationStartPoint is the y axis value of the decorations(Roadmarkings and the bushs)**

**#When the Y value gets so high it is set to 0 to be increased again**

**DecorationStartPoint = DecorationStartPoint + (1\*(count))**

**if DecorationStartPoint > 590:**

**DecorationStartPoint = 0**

**#This is the algorithm to increment OtherMove. This variable is the y axis**

**#For the other cars and some decoration. When it reaches 590 function LaneNo**

**#and WhichCar are called**

**OtherMove = OtherMove + (1\*(count))**

**if OtherMove > 590:**

**OtherMove =0**

**LaneNo, CoinLane = RandomLane()**

**AlreadyCollected = False**

**Car = WhichCar()**

**#Build and show the background to cover the previous frame**

**GameScreen.blit(BackGround, (0,0))**

**#Call DecorationMove with the parameter DecorationStartPoint**

**DecorationMove(DecorationStartPoint)**

**#Calls the FriendlyCar Function with parameters PlayersCarLane and PlayersVerticalVal**

**FriendlyCar(PlayersCarLane,PlayersVerticalVal)**

**#Assigns value to OtherCarLane from the return of OtherCars function**

**OtherCarLane = OtherCars(OtherMove, LaneNo, Car)**

**#Assigns a value to the varable PlayerScore whilst also running the function**

**#score with FirstTime as a parameter**

**PlayerScore = Score(FirstTime, CoinCollection(PlayersCarLane, OtherMove, OtherCarLane), AmountofCoins)**

**SpawnCoin(OtherMove, CoinLane, Coin, AlreadyCollected)**

**DisplayHighScore(FirstPlace, SecondPlace, ThirdPlace)**

**#Updates the display**

**pygame.display.update()**

**#Call function to see if a coin has been collected**

**if CoinCollection(PlayersCarLane, OtherMove, CoinLane) == True:**

**AlreadyCollected = True**

**AmountofCoins = AmountofCoins + 1**

**Score(FirstTime, CoinCollection(PlayersCarLane, OtherMove, CoinLane), AmountofCoins)**

**#If TestCrash returns True then stop the loop**

**if TestCrash(PlayersCarLane, OtherMove, OtherCarLane) == True:**

**SaveScore(PlayersName, PlayerScore)**

**GameScreen.blit(GameOver, (80,200))**

**pygame.display.update()**

**time.sleep(2)**

**gameExit = True**

**break**

**#Calls the function SaveScore to store PlayersName and PlayerScore to a textfile**

**pygame.quit()**

**os.system('StartUp.py')**

This loop is repeated for the whole duration of the game. The event loop (at the start of the game loop) allows the player to move from lane to lane whenever they wish using the arrow keys on their keyboard. Count is then incremented by the SpeedIncrement() function. Count is used to speed up the movement of the objects moving downwards in the game. DecorationStartPoint is incremented every time the loop is executed by adding the calculation (1\*count). The DecorationStartPoint is used as the y coordinate for the decoration images. When the value of this reaches 590 (Very close to the bottom of the game window) then the value is set to 0 so that the image is displayed back at the top of the window. OtherMove has the same algorithm as DecorationStartPoint, however when this value reaches 590, it also is reset to 0, but additionally LaneNo, CoinNo, and Car are all assigned a value from the functions RandomLane() and WhichCar(). AlreadyCollected is set to False so that the player can collect another coin again when it appears at the top of the page.

GameScreen.blit(Background, (0,0)) puts the background on top of the window so that the old images of the decoration and cars are covered. DecorationMove(DecorationStartPoint) is then called, this function then places the images of all the decoration on to the background with the new y coordinate which was passed from the main game loop. FriendlyCar() is then passed with the parameters PlayersLane and PlayersVerticalVal. This function places the players car image on top of the back ground with the coordinates passed from the main program.

OtherCarLane is then assigned the return value from the function OtherCars(). This function deciphers the other cars X value, and the places the image on top of the background. It passed the value of the OtherCars lane back to the main program for crash test comparison.

PlayersScore is then given the return value of the returned variable of Score(). This function calculates the players score by adding 1 point for every second the player survives the game, plus 5 points for every coin collected. SpawnCoin is the next function to be called, this function places the image of the coin on the screen with the coordinated passed through the main loop.

DisplayHighScores() then uses the three parameters it is passed with in order to display the top three high scores on top of the background.

The screen is then updated so the user can see all of the changes that have happened to the objects on the game screen. Once the screen has been updated then the tests are then implemented. The first check is if the player has collected a coin. This is done by sending the coordinates of the player and the coordinates of the coin to the function CoinCollection(). If the function returns true as the coordinates are the same, then the AmountofCoins variable is incremented, and the score function is then ran to calculate the new score.

The second test run is to test whether the player has collided with another car. This is done by comparing the players coordinates with the other cars. If the coordinates match the return value of the function turns true. When this condition is true then the players score is saved, the game over text is displayed, and then the main game loop is broken. The code on the outside of the loop closes the py game window, and then loads the start menu Tkinter file.

The players score is saved with this algorithm(see below). The HighScoresFile is opened in append mode so that the new players score can be added on the end to the file without replacing any other scores. The algorithm appends a string variable, TextLine, to the end of the file. TextLine is created by concatonating the players name, a comma, and then the players score. The newline string is added to the end so that the next time a score is saved it will be on the next line within the file. A comma is placed between the name and score so that when retrieving the information, a split function can be used to retrieve the two values of the variables.

**def SaveScore(Name, Score):**

**TextLine = str(Name) + "," + str(Score) + "\n"**

**HighScoresFile = open("HighScores.txt", "a")**

**HighScoresFile.write(TextLine)**

**HighScoresFile.close()**

The algorithm that retrieves the three highest scores is displayed below. This algorithm is called at the start of the game so that the first second and third scores can be displayed in the top right hand corner of the game. It is also called within a tkinter file accessed from the start up menu called ‘View high scores’ This displays a graphical text box of the highest scores.

**def GetHighScores():**

**HighScoreFile = open("HighScores.txt", "r")**

**AllScores = []**

**for line in HighScoreFile**

**NameAndScore = []**

**Name, Score = map(str, line.split(","))**

**NameAndScore.append(Name)**

**NameAndScore.append(int(Score))**

**AllScores.append(NameAndScore)**

**Newlist = sorted(AllScores, key=lambda NameAndScore: NameAndScore[1], reverse=True)**

**Firstlist = Newlist[0]**

**Secondlist = Newlist[1]**

**Thirdlist = Newlist[2]**

**FirstName, FirstScore, SecondName, SecondScore, ThirdName, ThirdScore = Firstlist[0], Firstlist[1], Secondlist[0], Secondlist[1], Thirdlist[0], Thirdlist[1]**

**First = FirstName + " " + str(FirstScore)**

**Second = SecondName + " " + str(SecondScore)**

**Third = ThirdName + " " + str(ThirdScore)**

**return First, Second, Third**

**HighScoresFile.close()**

This algorithm firstly opens the textfile in read mode, so that it can access all of the data that has been written within. The program then loops the following for every line within the textfile. Name is given the value of the string before the comma (split) and Score is given the string value of the text after the comma (the score). The name variable is then appended to the list NameAndScore, then the interger of score is added to the same list. The list NameAndScore is then added to another list, AllScores[]. Once the loop has been complete and the program has executed the previous code for all lines within the textfile, the program then uses a statement to sort the list AllScores. This code sorts the list based on the Second Value within the NameAndScore list (which is the players score). The list is sorted based on scores going from highest to lowest. Because of this, therefore the first three values of the AllScores lst will contain the top three high scores. Therefor I assign the values of FirstList, SecondList, ThirdList to the first second and third values of the AllScores list. Furthermore the names and scores of all three new variables are then broken down further. A new set of three strings are then created by the concatonation of the players name and score with a space inbetween, these variables (First,Second,Third) are then returned from the function.

# Testing

Introduction

To test my projects quality I will perform several tests on the code I have designed. I will perform a series of black box tests and compare the expected output compared to the actual output.

Testing the Main Game code

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test | Expected Result | Actual Result | Actual result matches expected? Yes/No |
| 1 | Loads and displays the top three high scores on the game window. | High scores are found and then displayed in the top right hand side of the games window. | High scores are shown in the top right hand corner of the screen for the player to see. | Yes |
| 2 | Pressing the left arrow key when player in centre lane. | For the player’s car to move to the left lane. | The car moved to the left lane | Yes |
| 3 | Pressing the right arrow key when player in centre lane. | For the player’s car to move to the right lane. | The car moved to the right lane | Yes |
| 4 | Pressing the up-arrow key when player in centre lane. | For the player’s car to remain in the centre lane. | The car stayed in the same lane | Yes |
| 5 | Pressing the left arrow key when player in left lane. | For the player to remain in the left-hand lane | The car stayed in the same lane | Yes |
| 6 | Pressing the right arrow key when player in left lane. | For the player to remain in the left-hand lane. | The car stayed in the same lane | Yes |
| 7 | Pressing the up-arrow key when player in left lane. | For the player to move to the central lane. | The car moved to the middle lane | Yes |
| 8 | Pressing the left arrow key when player in right left. | For the player to remain in the right-hand lane | The car stayed in the same lane | Yes |
| 9 | Pressing the right arrow key when player in right lane. | For the player to remain in the right-hand lane | The car stayed in the same lane | Yes |
| 10 | Pressing the up-arrow key when player in right lane. | For the player to move to the central lane. | The car moved to the middle lane. | Yes |
| 11 | Score incrementation over time. | Score to increment by a point for every coin collected. | The score is incremented and continuously displayed in the top left corner. | Yes |
| 12 | Collecting a coin | The coin to disappear and the players score to be incremented by 5 points. | The player’s car collected the coin, the image of the coin was removed and the players score was increased by 5 points. | Yes |
| 13 | Crashing in to another car. | The game loop to exit, movement of images to stop, and the game over to appear. Shortly afterwards the start-up menu should be loaded. | The game stops, movement of everything on the game window stops, “game over” is displayed and the start menu appears. | Yes |
| 14 | Players score saved at the end of the game | The players score to be appended to the end of the “HighScores” text file in ‘PlayersName,Score’ format. | The players name and score is saved in the correct text file in the appropriate format. |  |

Testing on the Tkinter menu forms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test | Expected Result | Actual Result | Actual result matches expected? Yes/No |
| StartUp.py file. (Main Menu) | | | | |
| 15 | From the main menu clicking play game button. | The new form for Entering the players name will appear, the start-up menu will disappear. | The form was loaded and the start-up menu closed. | Yes |
| 16 | From the main menu clicking view high scores. | The high scores window will appear. It will have loaded the top three scores within the text file. Start-up menu will have closed. | The window did appear, whilst closing the start-up menu. The top three high scores from the text file were displayed in the correct order. | Yes |
| 17 | Clicking on the reset high scores button from the main menu. | Takes the user to another window “Confirm Reset HighScores”, main menu is closed. | Menu was closed and the new window appeared. | Yes |
| 18 | Instruction button selected from the main menu. | New instructions window is loaded and main menu is hidden. | Instruction window is displayed and the main menu is closed. | Yes |
| 19 | Exit button selected | The program is ended and all menus are hidden. | The program is killed | Yes |
| EnterName.py | | | | |
| 20 | Enter a 3-letter long string and click “Play Game”. | The name to be written to a text file, the game to start, and the game to be passed the players name. | Name is written to a text file and is then read from the text file in the game code. The game is also started. | Yes |
| 21 | Enter a string with length not equal to three and click “Play Game”. | The game not to be run and the text on the form to turn red to highlight the users error. | The game was not run and the text was highlighted red. | Yes |
| 22 | Enter a string containing numbers and click “Play Game”. | The game not to be run and the text on the form to turn red to highlight the users error. | Name is written to a text file and is then read from the text file in the game code. The game is also started. | No |
| 23 | Enter a string containing symbols and click “Play Game”. | The game not to be run and the text on the form to turn red to highlight the users error. | Name is written to a text file and is then read from the text file in the game code. The game is also started. | No |
| 24 | Clicking the main menu button. | To form to close and the Start-up menu to load up. | The Start-up menu is loaded up and the current form is closed. | Yes |
| ConfirmReset.py | | | | |
| 25 | No button clicked | Form to close and main menu to load. | Form closes and menu is loaded. | Yes |
| 26 | Yes button clicked | High scores text file is reset. New label and button Is displayed on the form. | High scores are reset and the new button and label are displayed at the bottom of the page. | Yes |
| 27 | Main Menu button clicked | Form to close and main menu to load. | Form closes and menu is loaded. | Yes |

Summary of testing

Most of my program ran successfully as planned. However, there were a few errors that I would like to solve within my code. The errors are test numbers: s22, and 23.

The first error, tests 22 and 23 are from poor validation within the EnterName Tkinter File. The code successfully manages to perform a length validation check; however, it does not provide a character validation method that prevents users from inputting digits or symbols in to the text box.

Whilst testing the program, I found out another error that could not be detected using the black box method. The problem is lack of efficiency within my code when I store and read the high scores of the game. All the players scores are saved within this text file, but only ever the top three high scores are displayed throughout the game. This therefore effectively means that the program loops through the text file comparing lots of scores, when only the top three scores need to be saved within the file.

In the next section of this document I will be creating solutions to fix the problems that were identified through my testing.

# Improvements

Within this section I am displaying evidence of the improvement of my code after the errors that were identified through my testing were solved.

The first error I will be addressing is the validation error created on the EnterName.py form. The error here is that the code allowed for a user to input both digits and symbols for the players name. This error is shown in tests 22 and 23.

This is the current algorithm within the program:

**def PlayClick():**

**PlayersName = InputBox.get()**

**if len(PlayersName) == 3:**

**WriteName(PlayersName.upper())**

**root.withdraw()**

**os.system('OfficialProj.py')**

**quit()**

**else:**

**#Run error window explaining 3 letter required length**

**Title = Label(app, text = "Enter 3 letters as a name/initals", fg = 'red')**

**Title.grid(row = 0, columnspan = 2)**

The only validation this function provides is whether the string entered is equal to three or not. If not, then red text appears to highlight the users error.

To solve this error, I will use ASCII values. I will create a loop that will cycle through the characters that are entered to check if they are within the given ASCII values of upper and lower case letters. From an ASCII table that I found online, the decimal values of 65-90 are the value for all upper-case letters, and the values 97-122 represent lower case letters. Therefore, if all the characters within the string are between these ranges of numbers, only letters can be inputted.

With this information, I found from my research I have managed to produce the following algorithm:

**def PlayClick():**

**PlayersName = str(InputBox.get())**

**if len(PlayersName) == 3:**

**for Char in PlayersName:**

**if (ord(Char)>= 65 and ord(Char) <= 90) or (ord(Char) >=97 and ord(Char) <=122):**

**Valid = True**

**else:**

**Valid = False**

**break**

**if Valid == True:**

**WriteName(PlayersName.upper())**

**root.withdraw()**

**os.system('OfficialProj.py')**

**quit()**

**else:**

**#Run error window explaining 3 letter required length**

**Title = Label(app, text = "Enter 3 letters as a name/initals", fg = 'red')**

**Title.grid(row = 0, columnspan = 2)**

**else:**

**#Run error window explaining 3 letter required length**

**Title = Label(app, text = "Enter 3 letters as a name/initals", fg = 'red')**

**Title.grid(row = 0, columnspan = 2)**

This algorithm only allows a string of 3 characters’ length, all containing letters to be entered. Therefore, I have removed the original error from the program.

The last problem that I found with my code was the efficiency of storing the players score within the text file. I realised that I only needed to save the top three highest scores, not every player. This will effectively speed up the search process for the high scores.

My idea to solve this problem is to retrieve the highest three scores, using the same method I use within the function GetHighScores(). Once I have got these scores. I will pass them to a different function called SortHighScores(). Within this function I will open up the text file in Write mode. This will over write any information within the file. Then I will write the top three high scores passed from the other function. This will result in storing only three scores in the text file.

The following for GetHighScores() is the extension I have made for this adjustment in the program. It creates three new strings that are sent as parameters to the new function SortHighScores. These three string variables are written to the text file.

**def GetHighScores():**

**#Creates variables for new file. Uses , for split and \n for new line.**

**sortFirst = FirstName + "," + str(FirstScore) +"\n"**

**sortSecond = SecondName + "," + str(SecondScore)+"\n"**

**sortThird = ThirdName + "," + str(ThirdScore)+"\n"**

**SortHighScores(sortFirst, sortSecond, sortThird)**

**return First, Second, Third**

**def SortHighScores(First, Second, Third):**

**#open file in write mode to remove any information currently held in the file.**

**HighScoreFile = open("HighScores.txt", "w")**

**#Writes all three high scores in the text file**

**HighScoreFile.write(First)**

**HighScoreFile.write(Second)**

**HighScoreFile.write(Third)**

**#Closes file**

**HighScoreFile.close()**

SortHighScores() simply opens the text file in write mode to overwrite any information within the text file. Then the top three high scores are written to the file. This means only three scores are saved in the file at one time. This function is called from the GetHighScores() function.

# Evaluation

Overall I believe that my project has been successful. I have successfully met all the aims of my program that I stated within the analysis chapter of this documentation. This includes creating a fully functional game, which can easily be navigated through. The game allows the player (The Red Car) to manoeuvre around other cars and collect coins, this meets the specification of objective three. The game also uses text files to write and read high scores to/form. This information is then displayed in the top hand corner of the game its self and can be accessed through the main menu.