CS ALL Projects – Project Two: Virtual Robot Bargain Hunt (VRBH)
Individual Project Portfolio (IPP)

Student Name: Asis Rai	**SID: 6528683**

PWP – the link to your up-to-date Project Work Portfolio (PWP) is https://raia10coventryacuk.wordpress.com/

Link to my GitHub is https://github.com/GroupD5Cov/D5VirtualRobotProject

Project Work Portfolio (PWP)

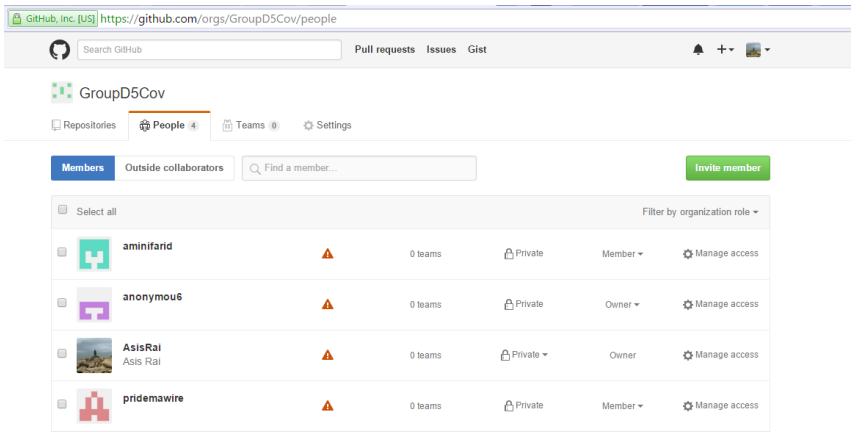
Week 1 - 18-22 January

This was the first week and I think the least productive week for all of us. Our project tutor introduced the project two to us: Virtual Robot Bargain Hunt (VRBH).

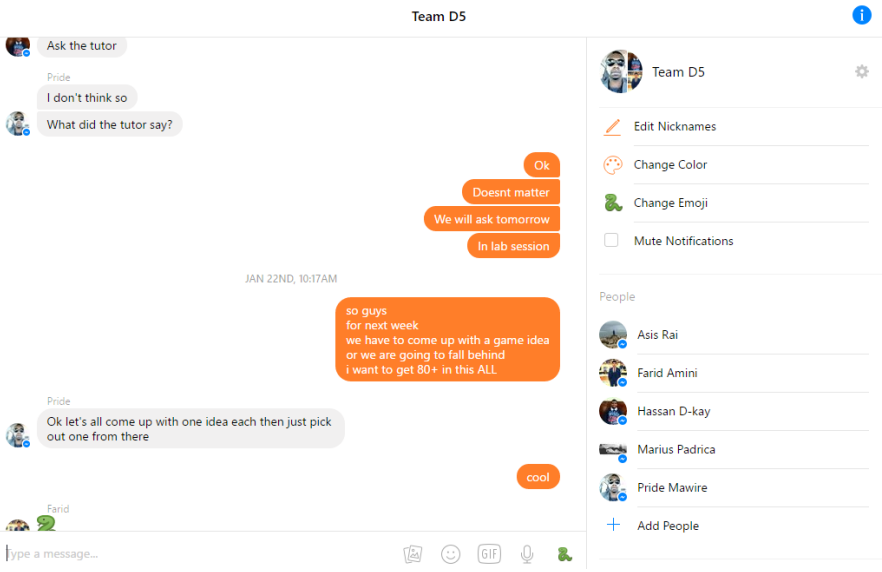
Group work requirements: (contributed to 75% of project marks)
Your team is required to develop a program with a graphical user interface which enables virtual robots to search for bargains. The basic elements of the program are required as follows: (contributed to 70% of project marks)

- To set up different locations with items, each of which should have a name, type (e.g., clothing, jewellery), price and quantity
- To enable a list of items to be specified by the user for a robot to search for bargains
- To enable a location specified by the user for a robot to start the searching for a list of requested items
- To display the bargain items (i.e. cheapest) requested by the user, along with their locations, on the screen
- To allow a robot to gather as many bargain items requested by the user as possible within the given time specified by the user
- To display a robot's movement through different locations when gathering the bargain items
- To allow a robot to sort the bargain items gathered in an ascending or descending order specified by the user according to certain criteria, such as name or price, determined by the user.
- To display the sorted items on the screen Individual work with extra features: (contributed to 25% of project marks) Each team member may produce extra features, in addition to the above required for the group work.

After we were told to go and meet our respective team members. I found out that I was allocated to Group D5 and my group members were – Pride Mawire, Hassan Hersi and Marius Padrica. After meeting with the group, we discussed for an hour of the lesson brainstorming ideas and sharing our experiences from our last project. We then decided to create a GitHub group so that we can all manage the program developed through GitHub and also work on the codes collaboratively.



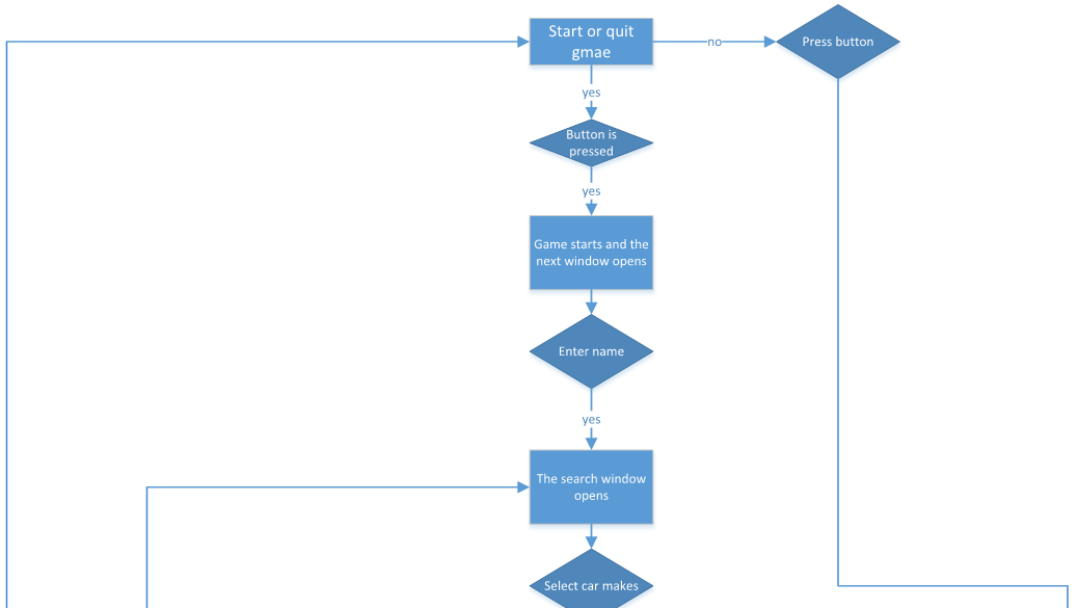
After everyone was connected in the GitHub Group we decided to connect on Facebook as well, so that we could all communicate efficiently.

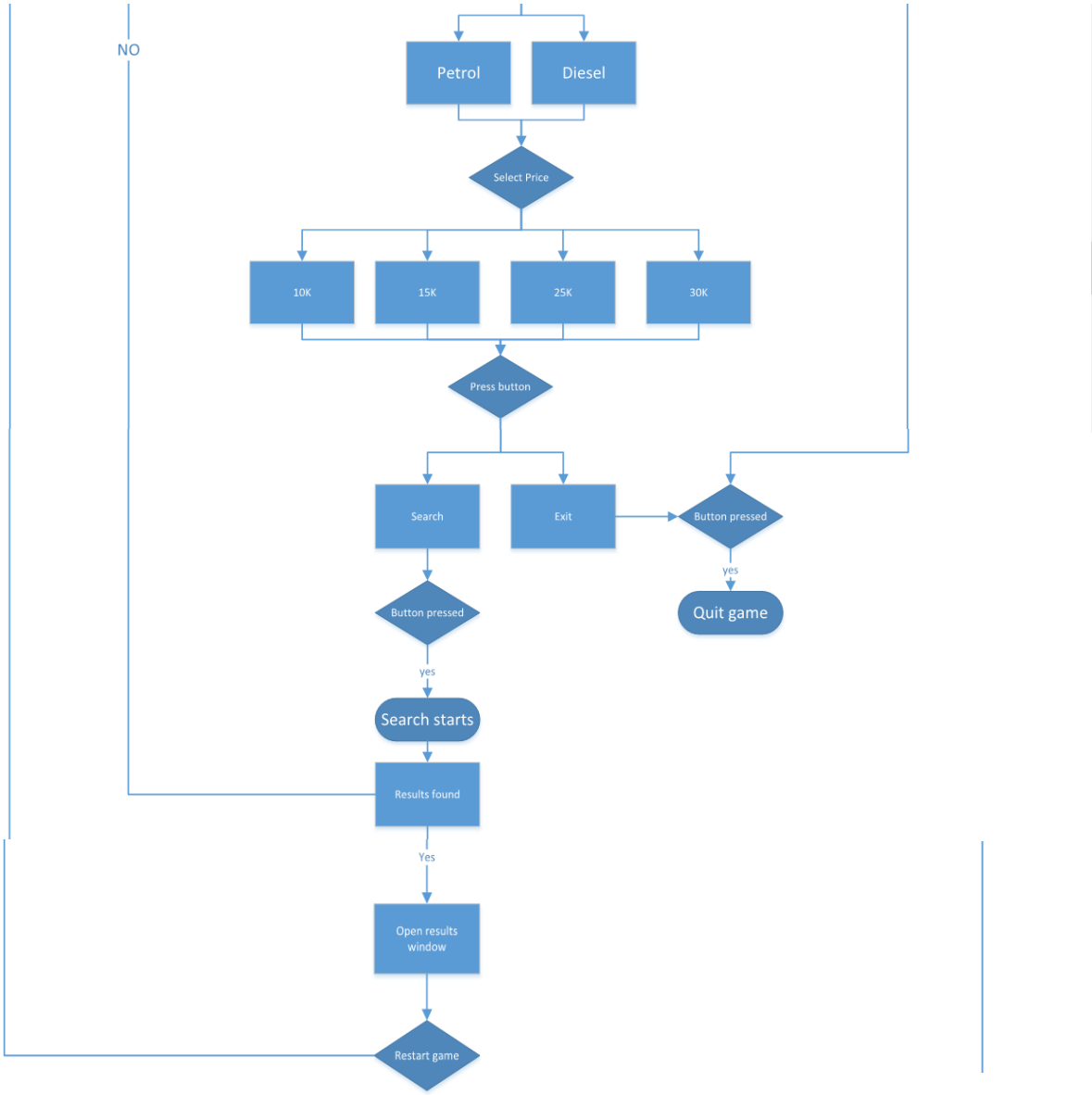


All of us were not sure what we were doing in the first week because this project was new to some group members and they stated that they needed time to understand the whole project tasks and requirements before they could contribute anything to the group and therefore we agreed that in week one all of us would learn the Python GUI (Graphical User Interface) basic features provided in the Tkinter package, so that we would come up with 4-5 ideas and then we could pick out the best possible one which contribute the highest marks. We decided to work on the programming tasks in the GUI tutorials individually however if anyone needed help then we would communicate on Facebook and help each other.

Week 2 (25-29 January)

In week 2 we all met up on Monday 25th and decided to hear out the ideas that we had each came up with. All our ideas were good but it was all vague and not very clear to all of us so we decided to find the similarities in our ideas and pick the best similarity and then come up with a flowchart which will give us a guide line to work to create the prototype graphical interface design for the VRBH program. After we wrote down all of our idea into a paper and picked the best one, we made a flowchart to understand as a group of how the program might work but we decided the flowchart is only to be used as a guideline/prototype and it was not the final idea or flowchart. After we created the flowchart, we as a group sort-of understood what we had to prepare for the coming weeks.





Later in the week when we all met again, we decided to create some design to be implemented into the prototype graphical user interface design. It was decided as a group that Farid would create the design on Photoshop and I would also be involved in the process, I will solely work on the code that would make it work on Python because we thought out of everyone I was the most capable, Pride and Marius will continue work to create the storyboard.

After Farid and I finished creating the design, Pride and Marius worked on the storyboard with all of us helping out explaining and discussing the processes in the storyboard.

Storyboard

1. This will be the start page as you click on the game. When this window opens you will be able to start the game or exit the game if you wish to. The window will also display the name of the game.

2. The second window will display a bar that will ask the user to enter their name. The users name will be saved to show the results for the user when they carry out the search on the next window.

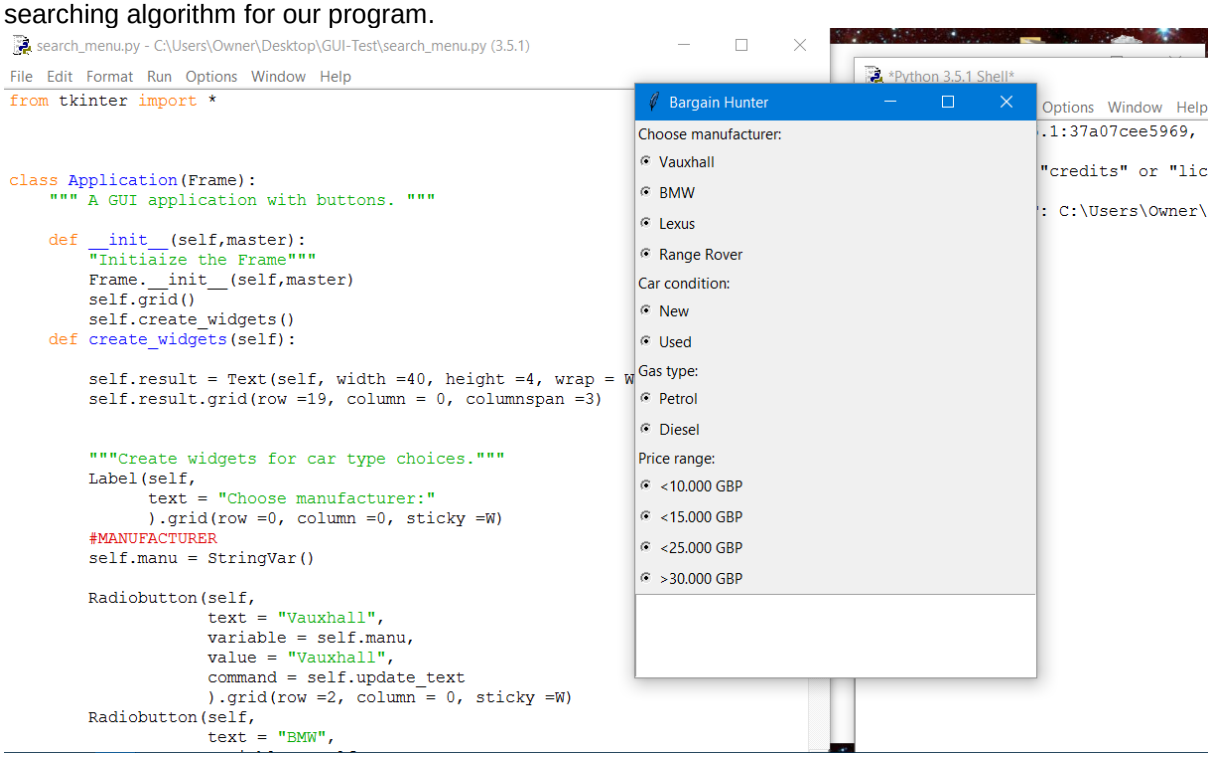
3. This window will display the search engine on the left and an image of the world map on the right. This window will be used to carry out the search. The user will be able to search for new or used cars. The user can search for four makes of cars: BMW, Vauxhall, Range Rover and Lexus. Also different fuel types: diesel and petrol. And the price range. This window will also contain two buttons which are to search or exit.

4. The last window will display an image of the world map which will show the robot moving around on the world image to find the location of the item and display the results.

After this I started working on the code where all the images will be implanted into the GUI and in the right places and this would be our prototype graphical user interface design for the VRBH program. We decided to call our VRBH program 'Bargain Inspector'. In this week we also started to prepare our IPP which will be used to asses us after week 8.

Week 3 (1-5 February)

Week 3 I started to work on the Start menu of our program and I planned to finish the start menu by the end of the week so that we could start working on the GUIs associated tasks to get better understand of Python. We as a group started researching and understanding the basics of searching algorithms so that we can start adding searching algorithms to our programs and decide which will be the most appropriate and relevant searching algorithm for our program.



Later in the week we started to document the program flow in UML activity diagrams which is needed for the IPP file later in the weeks. We also started to prepare for our report of our research and conclusion on searching algorithms. We decided to meet up again later in the week or next week as a team to discuss the situation of the team and to address who is most up-to-date with the GUIs tasks and whoever is the most up-to-date can start working on the searching algorithms and the others can catch-up whilst doing other activities such as updating flowchart, UML activity diagrams and making more pictures in Photoshop for the program etc.

Week 4 (8-12 February)

This week we as a group carried on doing GUIs associated tasks and this week was not very productive as most of the group members were missing because they were ill. Therefore, whoever came in for the project session started learning basic sorting algorithms and furthermore carried out a research on a sorting algorithm of our choice. We decided that this week, we were all going to be focused in researching sorting algorithms and learning about it as much as we can so that we understand how to implement them in our program and the missing group members can catch-up when they come in again while we start to implement the sorting algorithms on the code as we would have gone through the algorithms already.

Later in the week we hoped that we would have every single group member in the session, however they had not recovered in the week and therefore again we could decide which sorting algorithm we wanted to implement in the code and therefore we started doing our report of our research on sorting algorithm which would be included in the IPP file.

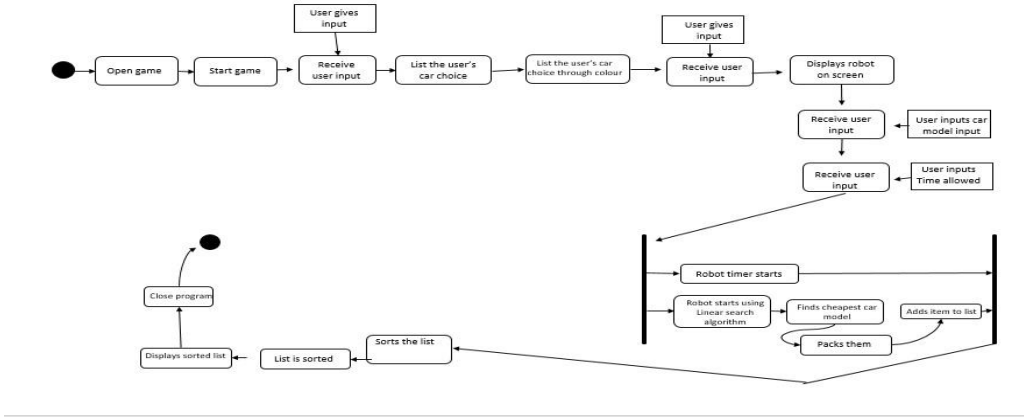
Weeks 5 (15-19 February)

This week luckily the whole group was in the session and we managed to tell the missing group members about what they missed in the last session. We all came together and I showed what I had done so far. They were happy with what I had done so far which was create the start screen with radio buttons, however we were falling behind and we decided to tweak our idea so that It would be faster and easy for us to finish our program. This meant flowcharts and storyboard had to be changed as well. We decided to change the idea because some group members were becoming unreliable and if we had stayed with our current idea that meant we each had to be assigned with a specific task for the program to work such as creating a database, searching algorithms, sorting algorithms and we decided to tweak our idea where instead of a database we would use arrays in the program which will make it easier and faster for us to program.

Because I had done a start-up screen before for our initial idea, I was tasked to create a new start-up screen for the program using pygame. Our game idea of choosing car models changed to choosing colours where each car models would be assigned to a specific colour and that colour would be assigned to arrays and then with the help of random numbers the robot would give the car models each random colour (car model) price, name. Through this sorting and searching algorithms would be connected. However in-order for the program to do this the user has to input values into the right from the start screen and then user has to input 4 values, the colour they want(colour assigned to a car model, choose where they want to search from(the map stay the same from previous storyboard and will be displayed on the screen and the user choose when they want to start from using co-ordinates), how many models of the chosen car models the user hope to find and how long the user will allow the robot to search for the values the user just inputted. This meant that we needed a new flow-chart but because this week we had to start our UML diagrams as well and therefore we decided to update our new idea into the UML diagram. I then started to work on the UML diagram and after I will also start working on the start screen and a new storyboard.

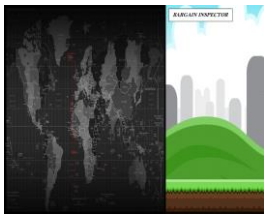
Week 6 (22-26 February)

This week we were supposed to start testing however our code was not done and therefore we did not do testing this week. By this week I finished the new UML diagram and the new storyboard and started to work on the new start-up screen using pygame.

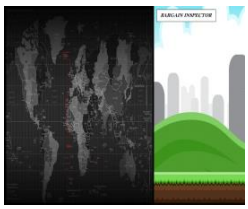


New Storyboard

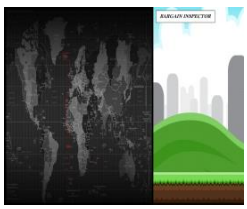
1. First screen would stay the same as before. This will be the first screen when the program starts and two buttons would be added to it, 'Start' button and 'Quit' button. Start button will start the game and take the user to the next screen and Quit game button will close the code and exit the program. Furthermore, instructions will be added to the screen for the user considering one of the principle from 106CR, Designing for usability.



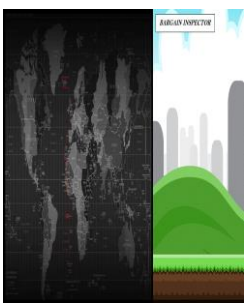
2. This will be the second screen the user will see after the 'Start' button is pressed. On the right hand side of the screen the user will see a text box and a question. The question will say 'Enter colour Red for BMW, colour Blue for Vauxhall, colour Green for Land Rover and colour pink for Lexus. The user will enter the input the colour type in the text box. If the user has entered anything other than the colours allowed, then they will see error message giving the user instruction again to what the user can input and the user may try again.



3. This will be the third screen the user will see and only allowed if the user has inputted the correct colour type from previous screen. In this screen the user will see be able to see the map clearly on the screen and in the right hand side of the screen the user will be able to input the co-ordinates to which side of the map the user wants the robot to search from for the car models. There will be a clear instruction in the right hand side of the screen telling the user what to do in this screen. The screen will move on to the next screen after the user has inputted the co-ordinates in this screen.



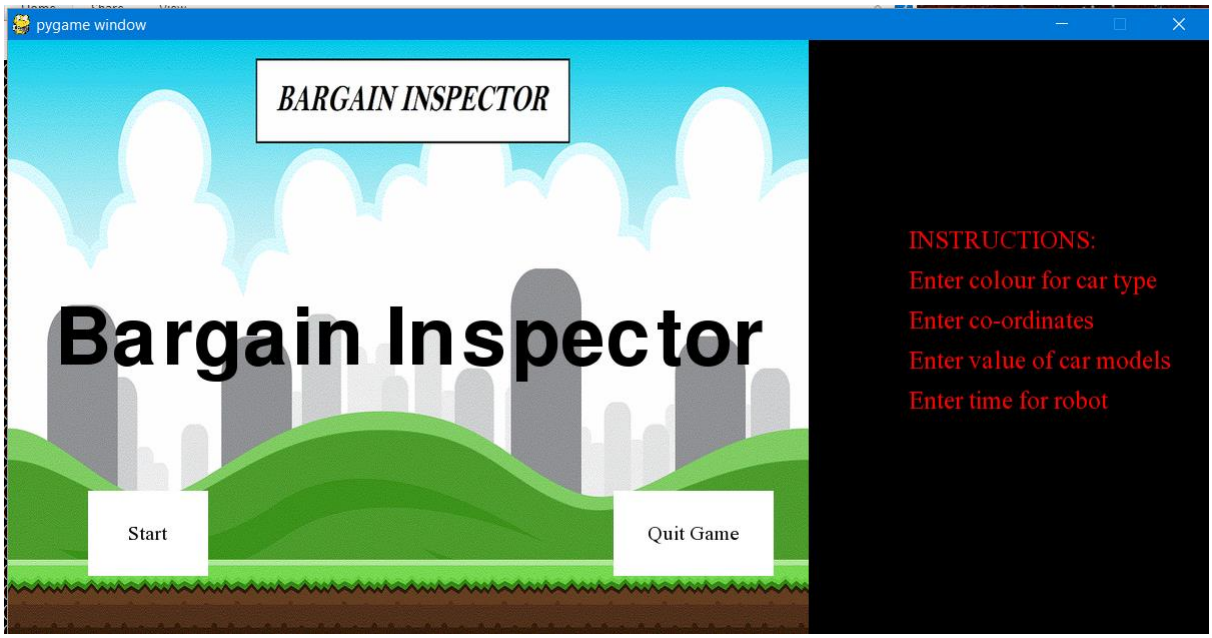
4. This will be the forth screen and the user will only be able to move to this screen after the user has inputted the co-ordinates in the previous screen, instructing the robot where to start searching from. This screen will follow the same procedure as the last one, however in this screen the user will be asked to input how many car models the user wants the user to search for and the user has to input the number of car models accordingly. E.g. 1



5. This will be the fifth screen and in this screen the user will be able to input the number of minutes(time) the user would like the robot to search for the car models. After the user has inputting the input the time, all the input has been recorded by the robot and now the robot will start searching for the car models by using the values provided by the user and display the robot's movement in left side of the map. After it has finished searching, the robot will have displayed the results found in the right side of the map with the results and the time it took for the robot to find the results, the results will be displayed in ascending order.

Week 7 (29 February – 4 March)

This week I was very busy with reiving for my advantage module CISCO which has a lot of content and I was going to be busy for the next coming week as well because that is when I have my CISCO exam. I finished making the user-screen for our program using pygame last weekend and I gave the finished code to the group to work and finish off the program.

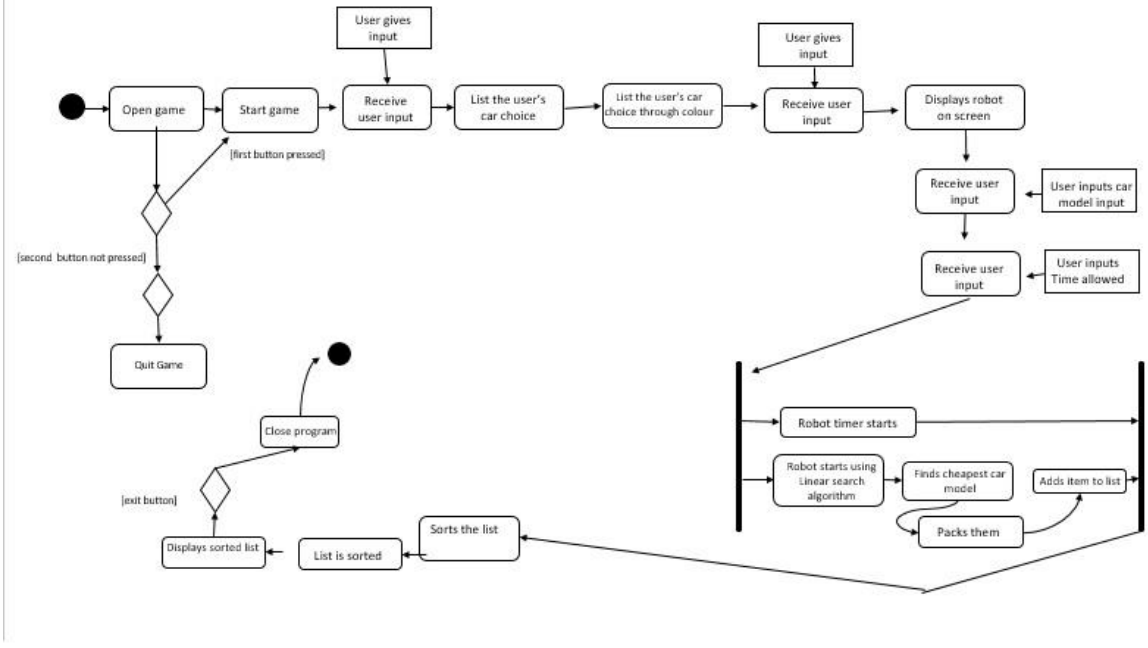


In this week We also had to for the presentation for next week which would require us to do a 10-minute group presentation and discuss with the class:

- What you have achieved as a group
- What problems you have encountered and how you resolved them
- What you have learnt from working as a group (in terms of skills and knowledge)

Week 8 (7 – 11 March)

This week we delivered our presentation to the class and the tutors, however we did not have a demo to show to the class because we still had not finished the code fully however was ¾ done and just needed a little bit of time. We managed to explain to the class and the tutors what our program is does and the outcome with the help of updated UML activity diagrams and storyboard.



Later in the week we finished the program and was ready for testing. I tested the program and made improvements to the program individually as well as with my group.

Test NO.	Description	Expected Result	Actual Result	feedback	Improvements
1	Start-up Screen	The code is run, and then code should bring a GUI with start-screen and instructions displayed for the user. Buttons to start and quit the game should also appear.	When the code was run the code was compiled and returned with a 'USABLE' GUI with the instructions to guide the user, also the buttons appeared and can be used by following the instructions.	N/A	N/A
2	Play button	Clicking the button should take it to the main screen where now the user will follow the on-screen texts to interact with the game	Button pressed and taken to the main screen and able to enter texts into the texts box	Texts font and colour seems hard to see as it is too small, need to increase the font size or change the font style and font colour.	Texts font changed from aerial to times and size changed from 10-14 and now able to see the text displayed on the screen clearly.
3	User input for car model with colours, where red = BMW, Green = Vauxhall, Blue = Landover and Pink = Lexus	Textbox accepts all four inputs and upper case or lowercase will input does not matter. If Inputs other than allowed entered should generate type error.	Textbox accepted all four inputs and upper case or lowercase will input did not matter. BMW inputted and type error came with a message error.	Change the error message to something appropriate for the first-time users.	Error changed to please enter the colour type matching car model type.
4	User input for co-ordinates	User input the co-ordinates of map to search from. The map displayed on the screen and looking that user can enter co-ordinates to search from a specific country.	Entered 'E' for East and started from east side of the map.	N/A	N/A

5	User input for how many	Users should be able to search for	Users should be able to search for	Feedback received were vague and not	Feedback received were vague and not
	number of car models to find	items for more than 14 car models.	items for more than 14 car models.	enough time to do improvements.	enough time to do improvements.
6	User input for how long the user should wait for the search	No more than 60 minutes should be accepted as it is too large for user and for the code.	180 minutes were accepted as well as the short minutes such as a minute.	Reduce the time allowed to 60 minutes max.	Program made to accept no more than 40 minutes as a group we thought it was more than enough time.
7	Robot's movement visibility	Robot's movement must be displayed on the screen.	Robot's movement were clearly displayed on the screen with white smoke left by the robot behind it.	N/A	N/A

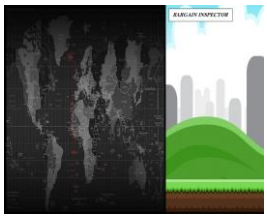
After testing everything was done and then we started working on our IPP.

122COM

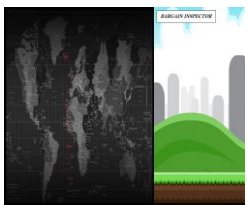
A graphical user interface design prototype (i.e. a storyboards with screen outputs) for the VRBH program with your contribution to the GUI design as an individual clearly identified on the prototype.

Design of the graphical user interface design prototype - a storyboards with screen outputs

- First screen would stay the same as before. This will be the first screen when the program starts and two buttons would be added to it, 'Start' button and 'Quit' button. Start button will start the game and take the user to the next screen and Quit game button will close the code and exit the program. Furthermore, instructions will be added to the screen for the user considering one of the principle from 106CR, Designing for usability.



- This will be the second screen the user will see after the 'Start' button is pressed. On the right hand side of the screen the user will see a text box and a question. The question will say 'Enter colour Red for BMW, colour Blue for Vauxhall, colour Green for Land Rover and colour pink for Lexus. The user will enter the input the colour type in the text box. If the user has entered anything other than the colours allowed, then they will see error message giving the user instruction again to what the user can input and the user may try again.



- This will be the third screen the user will see and only allowed if the user has inputted the correct colour type from previous screen. In this screen the user will see be able to see the map clearly on the screen and in the right hand side of the screen the user will be able to input the co-ordinates to which side of the map the user wants the robot to search from for the car models. There will be a clear instruction in the right hand side of the screen telling the user what to do in this screen. The screen will move on to the next screen after the user has inputted the co-ordinates in this screen.

- This will be the forth screen and the user will only be able to move to this screen after the user has inputted the co-ordinates in the previous screen, instructing the robot where to start searching from. This screen will follow the same procedure as the last one, however in this screen the user will be asked to input how many car models the user wants the user to search for and the user has to input the number of car models accordingly. E.g. 1

- This will be the fifth screen and in this screen the user will be able to input the number of minutes(time) the user would like the robot to search for the car models. After the user has inputting the input the time, all the input has been recorded by the robot and now the robot will start searching for the car models by using the values provided by the user and display the robot's movement in left side of the map. After it has finished searching, the robot will have displayed the results found in the right side of the map with the results and the time it took for the robot to find the results, the results will be displayed in ascending order.

A report of your research on search algorithms supported with in-text citations/references. (A maximum of 500 words)

To start off with the research on search algorithms I used the help of 'Python Basics Lab 6' in Computer Science Module. The lab sheet consisted of two search algorithms which are Linear Search and Binary search.

Linear Search is used to find an item in a list which do not have to be in order. [1] This form of search algorithm is one of the most basic search algorithm. When Linear Search is used the program will start at the beginning of the list and then the program will continue searching the list until the search has reached the end of the list of the searched item is found. Where this may be the most basic search algorithm and easy to implement however it could also be very time consuming because if the searched items are not in the list then the search start from beginning and end at the end of the list regardless to generate a result, therefore Linear search algorithm is basic but time consuming.

Binary Search consists of two ways where a programmer can implement Binary search. They are Depth-First search and Breadth-First search. These are algorithms to find connecting nodes in a graph.

Depth-First Search algorithm works by starting at the root also called the starting node, and then following of the branches connected to the root, keep following the branch till you find the desired node or reached the last node of the selected branch. If the desired node is not found, then return to the root and go to the other branch and repeat the last process. [2]

Breadth-First search works by starting at the root node (starting node) and then moving to the second level node and scan from the furthest left node of the second level to the farthest right node of the second level, if the desired node is found then stop, otherwise move on to the third level node and repeat the process.

Researching these algorithms were fascinating and I wanted to extend my knowledge further by researching more search algorithms. While searching on the internet I found out there are many other searching algorithms but the most relevant to the program I was creating was Naïve string search.

[3] Naïve string search can be interpreted graphically sliding a pattern between two strings. The naïve string search simply tests all the possible pattern between texts/string. This will be particularly useful if my program were to search for a match between a given text/string and a text/string in the program. It is a simple and efficient search algorithm.

For our program we used decided to implement Binary search as we thought it would work well with the new tweaked idea of ours and if we managed to do it we thought would help perhaps to achieve a higher mark and good understanding within us.

[1] University, C., 2016. *Python Basics Lab 6.* [Online]
Available at: https://cumoodle.coventry.ac.uk/pluginfile.php/663997/mod_resource/content/2/Python%20Basics%20Lab%206%20-%20Search%20Algorithms%20v2.pdf
[2] Cprogamming.com, 2016. *Cprogramming.com.* [Online]
Available at: http://www.cprogramming.com/discussionarticles/sorting_and_searching.html
[Accessed 09 March 2016].
[3] http://www.personal.kent.edu/, 2016. *http://www.personal.kent.edu/.* [Online]
Available at: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/naiveStringMatch.htm
[Accessed 07 March 2016].

```
def binarySearch(self, alist, item):
    """This will search a list for the search item and returns all infomation about that item"""
    first = 0
    last = len(alist)-1
    found = False

    while first<=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint][0] == item:
            return(alist[midpoint])
        else:
            if item < alist[midpoint][0]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found
```

A report of your research on sorting algorithms supported with in-text citations/references. (A maximum of 500 words)

A sorting algorithm is an algorithm that places the elements of a list in a certain order. There are numerous sorting algorithms out there and I have found few of the predominate ones in practice in my research. [4]

Bubble sort sorting algorithm is one of the most basic and simplest to understand. Bubble sort works by bubbling up smallest or largest elements, depending on the sorting order, then it will move to the other second largest element and bubble the elements, it will keep doing this until it reaches the end of the list, then the processes is repeated again until it has sorted out all the elements. Bubble sort is one of the inefficient sorting algorithms and is good if the programmer is working with a small list however it is not recommended for use with big lists as it will consume too much time bubbling elements in the entire list and waste time and productivity.

Insertion sort is another sorting algorithm that I found. I found out that insertion sort is inefficient in large lists like bubble, however easy to implement and more efficient that bubble sort sorting algorithm. Insertion sort takes elements from an unsorted list and inserts them in a new sorted list which is empty in the beginning. Insertion sort is more efficient than bubble sort because the unsorted and sorted lists stay the same and therefore programmers can use the same list to link and represent the sorted and unsorted sections.

Selection sort sorting algorithm is similar to bubble sort however differs by only making one exchange for each pass through the list. Selection sort looks for the largest value as it passes the value and after completing the pass it places the largest value in the correct location where it is placed as sorted. Just like bubble sort, it keeps repeating the process of finding the first largest element and placing it in its proper location and moving into the second largest element and placing the element into its proper place. For this process to continue requires n-1 for passes and sort n items.

Merge sort is more efficient sorting algorithm that subdivides the list into two sub-lists, sorts them and them backup into their respective lists. Merge sort uses a recursive call, when it is made it will divide each lists into sub lists and they will eventually become individual elements and constructed into final sorted lists.

Other sorting algorithms I researched were Heap sort, Quick sort and Shell sort. Heap sort is known for its insertion and merge sorting. Only a constant number of array value are stored outside the input array. This sorting algorithm is not appropriate for the program I am creating. [5]

[6] Quick sort algorithm is similar to Merge sort algorithm, both of them use a recursive call, executed by using divide and conquer strategy, two pointers are used at the end of each element list and then both are compared

[4] University, C., 2016. *Python Basics Lab 7.* [Online]
Available at: https://cumoodle.coventry.ac.uk/pluginfile.php/748776/mod_resource/content/6/Python%20Basics%20Lab%207%20-%20Sorting%20Algorithms.pdf
[Accessed 06 March 2016].

[5] http://staff.ustc.edu.cn/, 2016. *CHAPTER 7: HEAPSORT.* [Online]
Available at: http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap07.htm
[Accessed 06 mar 2016].

[6] RSI, 2016 . *The Quick Sort.* [Online]
Available at: https://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuickSort.html
[Accessed 08 March 2016].

with each other, then they are finally compared with each other and picked out the best depending on the sorting order.

Shell sort algorithm is an exchange sort algorithm that works by comparing values at intervals and reducing the size of the interval after each pass through the list. Shell sort is similar to Bubble sort, however faster because the values can move out of place to be put into closer to their destination depending on the sort criteria.

As a group we decided to use bubble sort and quick sort because we knew the lists generated by arrays will be small and these algorithms will be efficient in sorting the lists.

```
def bubbleSort(self,colourL):
    """ This will sort the list in order of price generated by random, cheapest first"""
    for passnum in range(len(colourL)-1,0,-1):
        for i in range(passnum):
            if colourL[i][2]>colourL[i+1][2]:
                temp = colourL[i]
                colourL[i] = colourL[i+1]
                colourL[i+1] = temp
```

```
def quick_sort(self,items):
    """ This will sort a list in order searched by binary search"""
    if len(items) > 1:
        pivot_index = len(items) // 2
        smaller_items = []
        larger_items = []

        for i, val in enumerate(items):
            if i != pivot_index:
                if val < items[pivot_index]:
                    smaller_items.append(val)
                else:
                    larger_items.append(val)
```

Program implementation – GitHub commented code including algorithms (e.g. searching and sorting algorithms) and external libraries/APIs with screen shots of key functionality testing

AsisRai Added files via upload — 99f3e6d an hour ago

1 contributor

516 lines (445 sloc) | 20.2 KB — Raw | Blame | History

```
#importing in functions that we need for the program to work
import pygame as pg
import random
from textbox import TextBox

pg.init()

#setting up colour RGB values
white = (255,255,255)
red = (255,0,0)
blue = (0,0,255)
green = (0,255,0)
black = (0,0,0)
pink = (255,20,147)
bright_red = (200,0,0)
bright_green = (0,200,0)

bg = pg.image.load('startscreen.png')

clock = pg.time.Clock()

font = pg.font.Font(None, 25)
frame_count = 0
frame_rate = 60
start_time = 180

menuDisplay = pg.display.set_mode((1200,600))
gameDisplay = pg.display.set_mode((1200, 600))
display_width = 800
```

```
display_width = 800
display_height = 600

gameExit = False

KEY_REPEAT_SETTING = (200,70)#textbox to appear in the same position after action

"""setting instruction colour and font"""
def instruction(i,Space,List):
    intrs = List
    font = pg.font.SysFont("Times", 25)
    message = intrs[i]
    rend = font.render(message, True, pg.Color("red"))
    return (rend, rend.get_rect(topleft=(900,35+Space)))

"""setting font and colour of the name displayed at the start"""
def text_objects(text, font):
    textSurface = font.render(text, True, black)
    return textSurface, textSurface.get_rect()

"""quit game button function"""
def quit_game():
    pg.quit()
    quit()
"""start game button function"""
def start_game():
    app = MainProgram()
    app.main_loop()


#Right hand side title screen colour
def game_intro():
    x=0
    y=0
    i = 0
```

```
    intro = True
    gameDisplay.fill(black)
    while intro:
        for event in pg.event.get():
            if event.type == pg.QUIT: #quit function added
                pg.quit()
                quit()

#Now creating the start-up screen

        gameDisplay.blit(bg,(x,y)) #displaying in starmenu bg being pygame, x and y being the position
        largeText = pg.font.Font('freesansbold.ttf',80)#font and font size
        TextSurf, TextRect = text_objects("Bargain Inspector", largeText)#Program name
        TextRect.center = ((display_width/2),(display_height/2)) #text allignment
        gameDisplay.blit(TextSurf, TextRect)

        button("Start",80,450,120,85,white,bright_green,start_game) #Button which starts the program, position,size, colour and linked to t
        button("Quit Game",605,450,160,85,white,bright_red,quit_game) # Button which closes the program, position,size, colour and linked t

        intrs = ["INSTRUCTIONS:","Enter colour for car type", "Enter co-ordinates", "Enter value of car models", "Enter time for robot",]#I
        space = int(150) #position of the instruction on the screen

        while i != 5: #5 total strings
            prompt = instruction(i,space,intrs)#i=number of instructions, space = position, intrs = intructions
            gameDisplay.blit(*prompt)
            space = space + 40 #how close to each other the instructions
            pg.display.update() #for clock speed functuon
            i = i+1


        pg.display.update()
#limit the clock speed to 15 FPS (to prevent overflow)
        clock.tick(15)
```

```

#buttons function defined, event driven action (for the Start game and Quit button)
def button(msg,x,y,w,h,ic,ac,action=None):
    mouse = pg.mouse.get_pos()
    click = pg.mouse.get_pressed()
    print(click)
    if x+w > mouse[0] > x and y+h > mouse[1] > y: #when mouse button clicked outcome (1,0,0)
        pg.draw.rect(gameDisplay, ac,(x,y,w,h))
        if click[0] == 1 and action != None: #if mouse position (0,0,0 = no action, otherwise event driven action)
            action()
    else:
        pg.draw.rect(gameDisplay, ic,(x,y,w,h))
    smallText = pg.font.SysFont("Times",20)
    textSurf, textRect = text_objects(msg, smallText)
    textRect.center = ( (x+(w/2)), (y+(h/2)) )
    gameDisplay.blit(textSurf, textRect)

#Initialising the main program with class attribute
class MainProgram(object):
    def __init__(self):
        """The initialisation function of key components of the main program"""
        pg.init()
        pg.display.set_caption("D5's Bargain Inspector")
        bg = pg.image.load('mainScreen.png')
        gameDisplay.blit(bg,(0,0))
        self.red = []
        self.blue = []
        self.green = []
        self.pink = []
        self.colourA =[]
        self.colour = ""
        self.num_items = 0
        self.finishedList =[]
```

```
        self.time = 0
        self.frame_count = 0
        self.frame_rate = 60
        self.start_time = 180
        self.screen = menuDisplay
        self.clock = pg.time.Clock()
        self.robot_loc = []
        self.fps = 60.0
        self.done = False
        self.input = TextBox((900,200,200,40),command=self.get_input,          #setting the size and position of the text box
                             clear_on_enter=True,inactive_on_enter=False)
        self.user_input = ""
        self.color = white
        self.prompt = self.make_prompt('Enter Red:BMW, Blue:Vauxhall, Green:Land Rover, Pink:Lexus')
        pg.key.set_repeat(*KEY_REPEAT_SETTING) #textbox to appear in the same position after action

    def make_prompt(self,Message):
        """ Function to create the labels, called everytime a new input is entered """
        pg.draw.rect(menuDisplay , white,(820,165,400,30)) #1 is left right position, 2 is up down, 3 is width, 4 is height
        font = pg.font.SysFont("Times", 14)
        message = Message
        rend = font.render(message, True, pg.Color("black"))
        return (rend, rend.get_rect(topleft=(820,165)))#position of the text in the screen

    def event_loop(self):
        """ A continuous FOR loop which allows an exit for our main program"""
        for event in pg.event.get():
            if event.type == pg.QUIT:
                self.done = True
            self.input.get_event(event)

    def random_types(self):
        """Randomly generates colours into the screen and randomly gives them a price and a name eg. red-bmw,price """
        names = ["BMW", "Vauxhall", "Land Rover", "Lexus"]
```

```
        names = ["BMW", "Vauxhall", "Land Rover", "Lexus"]
        for i in range(50):
            item = random.randint(1,3)
            radx = random.randint(0,790)
            rady = random.randint(0,590)
            radp = random.randint(1,20)
            radnum = random.randint(1,4) - 1
            radn = names[radnum]
            coords = [radx,rady,radp,radn]

            if item == 1:
                pg.draw.rect(menuDisplay , red,(radx,rady,10,10))
                self.red.append(coords)
            elif item == 2:
                pg.draw.rect(menuDisplay , blue,(radx,rady,10,10))
                self.blue.append(coords)
            elif item == 3:
                pg.draw.rect(menuDisplay, green,(radx,rady,10,10))
                self.green.append(coords)
            elif item == 4:
                pg.draw.rect(menuDisplay, pink,(radx,rady,10,10))
                self.pink.append(coords)
            i = i +1

    def get_input(self,id,input):
        """ allows the user to search for cars by enterin a specific colour """
        try:
            input = input.lower()
            self.user_input = input
            self.colour = input
            if self.user_input == "red" or self.user_input == "blue" or self.user_input == "green" or self.user_input == "pink":
                self.prompt = self.make_prompt('Where do you want to start : e.g. NW')
                self.input = TextBox((900,200,200,40),command=self.robot_start, # textbox position
                             clear_on_enter=True,inactive_on_enter=False)
```

```
            if input == "red":
                for coord in self.red:
                    x = coord[0]
                    y = coord[1]
                    pg.draw.rect(menuDisplay, red,(x,y,15,15))
                    self.colourA = self.red

            elif input == "blue":
                for coord in self.blue:
                    x = coord[0]
                    y = coord[1]
                    pg.draw.rect(menuDisplay, blue,(x,y,15,15))
                    self.colourA = self.blue

            elif input == "green":
                for coord in self.green:
                    x = coord[0]
                    y = coord[1]
                    pg.draw.rect(menuDisplay, green,(x,y,15,15))
                    self.colourA = self.green

            elif input == "pink":
                for coord in self.pink:
                    x = coord[0]
                    y = coord[1]
                    pg.draw.rect(menuDisplay, pink,(x,y,15,15))
                    self.colourA = self.pink

            else:
                self.prompt = self.make_prompt('Please enter the colour type given')
            self.screen = menuDisplay

        except ValueError:
```

```
            print("ERROR")

    def robot_start(self,id,input):
        """ Allows the user to choose the starting position of the robot"""
        input = input.upper()
        self.robot_loc = input
        if input == "N":
            pg.draw.rect(menuDisplay, red,(400,0,20,30))
            self.robot_loc = [400,0]
        elif input == "E":
            pg.draw.rect(menuDisplay, blue,(750,300,20,30))
            self.robot_loc = [750,300]
        elif input == "S":
            pg.draw.rect(menuDisplay, pink,(400,550,20,30))
            self.robot_loc = [400,550]
        elif input == "W":
            pg.draw.rect(menuDisplay, green,(10,300,20,30))
            self.robot_loc = [10,300]
        elif input == "NW":
            pg.draw.rect(menuDisplay, bright_green,(10,10,20,30))
            self.robot_loc = [10,10]
        elif input == "NE":
            pg.draw.rect(menuDisplay, bright_red,(750,10,20,30))
            self.robot_loc = [750,10]
        elif input == "SW":
            pg.draw.rect(menuDisplay, red,(10,550,20,30))
            self.robot_loc = [10,550]
        elif input == "SE":
            pg.draw.rect(menuDisplay, pink,(750,550,20,30))
            self.robot_loc = [750,550]
        else:
            self.prompt = self.make_prompt('Please enter a valid co-cordinate for the robot to search')
        if input == "N" or input == "E" or input == "S" or input == "W" or input == "NW" or input == "NE" or input == "SW" or input == "SE"
            self.prompt = self.make_prompt('Please enter the number of car types you will like to find?')
```

```
            self.input = TextBox((900,200,200,40),command=self.number_of_items, #textbox position
                        clear_on_enter=True,inactive_on_enter=False)

    def number_of_items(self,id,input):
        """ This will allow the user to enter the number of chosen car models they want to find"""

        if input.isdigit() and (int(input) <= len(self.colourA)):
            self.num_items = int(input)
            self.prompt = self.make_prompt('Enter the minutes you want the robot to search for?')
            self.input = TextBox((900,200,200,40),command=self.input_time, #textbox pisition
                            clear_on_enter=True,inactive_on_enter=False)

        else:
           self.prompt = self.make_prompt('Please enter how many chosen car models to find?')

    def input_time(self,id,input):
        """ Allows the user to enter the time for the robot to search for car types"""

        if input.isdigit() and int(input) <= 15:
            self.time = input
            self.start_time = int(self.time) * 60

        else:
            self.prompt = self.make_prompt('Please enter a valid time, e.g 1 for 1 minute')

    def collide(self,c1, p1, p2, p3,xORy):
        """ Tests to see if the next pixals are not white"""
        locations = [p1,p2,p3]
        self.Collide = False
        i = 0
        if xORy == "X":
            while i != 3:
                colour = menuDisplay.get_at((c1,locations[i]))  # gets the colour of the pixal at the coordinates

                if (c1 >= self.nextX and c1 <= (self.nextX + 15)) and (p1 >= self.nextY and p1 <= (self.nextY + 15)):
                    i=i+1
                    continue
                elif (colour[0] != 255 or colour[1] != 255 or colour[2] != 255):
                    self.Collide = True
                    break
                else:
                    i=i+1
                    continue
        elif xORy == "Y":
            while i != 3:
                colour = menuDisplay.get_at((locations[i],c1))
                if (c1 >= self.nextY and c1 <= (self.nextY + 15)) and (p1 >= self.nextX and p1 <= (self.nextX + 1)):
                    i=i+1
                    continue
                elif (colour[0] != 255 or colour[1] != 255 or colour[2] != 255):
                    self.Collide = True
                    break
                else:
                    i=i+1
                    continue

    def bubbleSort(self,colourL):
        """ Used to sort the list in order of price, cheapest first"""
        for passnum in range(len(colourL)-1,0,-1):
            for i in range(passnum):
                if colourL[i][2]>colourL[i+1][2]:
                    temp = colourL[i]
                    colourL[i] = colourL[i+1]
                    colourL[i+1] = temp

    def binarySearch(self, alist, item):
        """Used to search a list for the search item and returns all infomation about that item"""
```

```
        first = 0
        last = len(alist)-1
        found = False

        while first<=last and not found:
            midpoint = (first + last)//2
            if alist[midpoint][0] == item:
                return(alist[midpoint])
            else:
                if item < alist[midpoint][0]:
                    last = midpoint-1
                else:
                    first = midpoint+1

        return found

    def quick_sort(self,items):
        """ Used to sort a list in order by x coords for binary search"""
        if len(items) > 1:
            pivot_index = len(items) // 2
            smaller_items = []
            larger_items = []

            for i, val in enumerate(items):
                if i != pivot_index:
                    if val < items[pivot_index]:
                        smaller_items.append(val)
                    else:
                        larger_items.append(val)


            self.quick_sort(smaller_items)
            self.quick_sort(larger_items)
            items[:] = smaller_items + [items[pivot_index]] + larger_items
```

```
    def robot_move(self):
        """Makes the robot move visually and makes a countdown timer that countdowns from the users input"""
        i = 0
        if self.colour == "red":
            self.bubbleSort(self.red)
            locations = self.red
        elif self.colour == "blue":
            self.bubbleSort(self.blue)
            locations = self.blue
        elif self.colour == "green":
            self.bubbleSort(self.green)
            locations = self.green
        elif self.colour == "pink":
            self.bubbleSort(self.pink)
            locations = self.pink
#pg.draw.rect(menuDisplay, white,(self.robot_loc[0],self.robot_loc[1],20,30))
        print(locations)
        while i != self.num_items :   #Makes the robot move visually
            self.event_loop()
            nextX = locations[i][0]
            nextY = locations[i][1]

            if self.robot_loc[0] == nextX and self.robot_loc[1] == nextY:
                pg.draw.rect(menuDisplay, black,(nextX,nextY ,15,15))
                self.finishedList.append(locations[i][0])
                i = i + 1

            elif self.robot_loc[0] < nextX:
                pg.draw.rect(menuDisplay, white,(self.robot_loc[0],self.robot_loc[1],20,30))
                self.robot_loc[0] = self.robot_loc[0] + 1
                pg.draw.rect(menuDisplay, pink,(self.robot_loc[0],self.robot_loc[1],20,30))
                self.input.draw(self.screen)
```

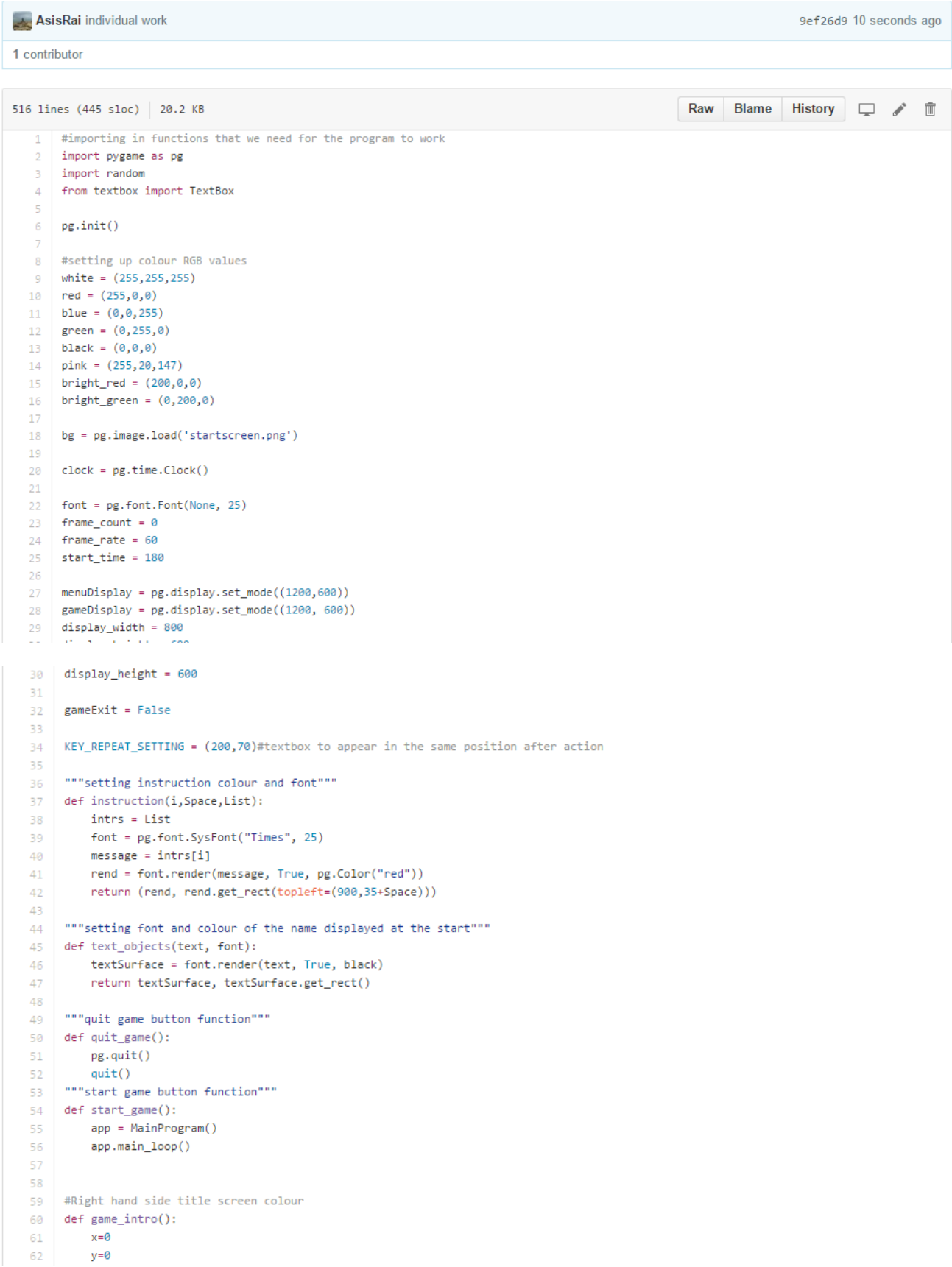
```
397
398             elif self.robot_loc[1] < nextY:
399                 pg.draw.rect(menuDisplay,white,(self.robot_loc[0],self.robot_loc[1],20,30))
400                 self.robot_loc[1] = self.robot_loc[1] + 1
401                 pg.draw.rect(menuDisplay, pink,(self.robot_loc[0],self.robot_loc[1],20,30))
402                 self.input.draw(self.screen)
403
404             elif self.robot_loc[0] > nextX:
405                 pg.draw.rect(menuDisplay, white,(self.robot_loc[0],self.robot_loc[1],20,30))
406                 self.robot_loc[0] = self.robot_loc[0] - 1
407                 pg.draw.rect(menuDisplay, pink,(self.robot_loc[0],self.robot_loc[1],20,30))
408                 self.input.draw(self.screen)
409
410             elif self.robot_loc[1] > nextY:
411                 pg.draw.rect(menuDisplay, white,(self.robot_loc[0],self.robot_loc[1],20,30))
412                 self.robot_loc[1] = self.robot_loc[1] - 1
413                 pg.draw.rect(menuDisplay, pink,(self.robot_loc[0],self.robot_loc[1],20,30))
414                 self.input.draw(self.screen)
415
416             self.event_loop()
417             # Starts the timer countdown
418             pg.draw.rect(menuDisplay, green,(810,540,400, 60))
419             total_seconds = self.frame_count // self.frame_rate
420             total_seconds = self.start_time - (self.frame_count // self.frame_rate)
421             if total_seconds < 0:
422                 total_seconds = 0
423             minutes = total_seconds // 60
424             seconds = total_seconds % 60
425             output_string = "Time left: {0:02}:{1:02}".format(minutes, seconds)
426             text = font.render(output_string, True, black)
427             menuDisplay.blit(text, [810, 540])
428             if output_string == "Time left: 00:00":
429                 self.done = True
```

```
430              self.frame_count += 1
431              clock.tick(frame_rate)
432              pg.display.flip()
433
434              self.input.draw(self.screen)
435              self.screen.blit(*self.prompt)
436
437              pg.display.update()
438
439         if self.colour == "red":
440             self.quick_sort(self.red)
441         elif self.colour == "blue":
442             self.quick_sort(self.blue)
443         elif self.colour == "green":
444             self.quick_sort(self.green)
445         elif self.colour == "pink":
446             self.quick_sort(self.pink)
447
448         self.clock.tick(self.fps)
449         if self.time != 0:
450             self.done = True
451
452     def output_lists(self,i,Space):
453         """Displays the list of cheapest items picked up"""
454         if self.colour == "red":
455             output = self.binarySearch(self.red, self.finishedList[i])
456         elif self.colour == "blue":
457             output = self.binarySearch(self.blue, self.finishedList[i])
```

```
        elif self.colour == "green":
            output = self.binarySearch(self.green, self.finishedList[i])
        elif self.colour == "pink":
            output = self.binarySearch(self.pink, self.finishedList[i])

        font = pg.font.SysFont("Times", 20)
        message = str(output[3]) + " | " + str(output[2])
        rend = font.render(message, True, pg.Color("black"))
        return (rend, rend.get_rect(topleft=(820,35+Space)))

    def main_loop(self):
        """ Makes the program loops and call certain function only if an event has been met"""
        i = 0



        """adds sound to the code"""
        pg.mixer.music.load('programsound.wav')
        pg.mixer.music.play(-1)

        space = 0
        self.random_types()
        while not self.done:
            self.event_loop()
            self.input.update()
            self.input.draw(self.screen)
            self.screen.blit(*self.prompt)

            pg.display.update()
            self.clock.tick(self.fps)
            if self.time != 0:
                self.done = True
        self.done = False
```

```
            if self.time != 0:
                self.robot_move()
                pg.draw.rect(menuDisplay , green,(810,0,450,540))
                pg.display.update()
                while i != self.num_items:
                    self.prompt = self.output_lists(i,space)
                    self.screen.blit(*self.prompt)
                    space = space + 20
                    pg.display.update()
                    i = i+1
                    self.done = False
                    #self.main_program()
                while not self.done:
                    self.event_loop()

#Sets up the start-up screen
menuDisplay.fill(white)
pg.draw.rect(menuDisplay , black,(800,0,10,600))
#Calls mainprogram function to start the game
game_intro()


pg.display.update()
pg.quit()
quit()
```

One additional feature implementation on GitHub (individual work)



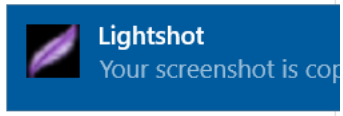
AsisRai individual work — 9ef26d9 10 seconds ago

1 contributor

516 lines (445 sloc) | 20.2 KB — Raw | Blame | History

```
#importing in functions that we need for the program to work
import pygame as pg
import random
from textbox import TextBox

pg.init()

#setting up colour RGB values
white = (255,255,255)
red = (255,0,0)
blue = (0,0,255)
green = (0,255,0)
black = (0,0,0)
pink = (255,20,147)
bright_red = (200,0,0)
bright_green = (0,200,0)

bg = pg.image.load('startscreen.png')

clock = pg.time.Clock()

font = pg.font.Font(None, 25)
frame_count = 0
frame_rate = 60
start_time = 180

menuDisplay = pg.display.set_mode((1200,600))
gameDisplay = pg.display.set_mode((1200, 600))
display_width = 800
display_height = 600

gameExit = False

KEY_REPEAT_SETTING = (200,70)#textbox to appear in the same position after action

"""setting instruction colour and font"""
def instruction(i,Space,List):
    intrs = List
    font = pg.font.SysFont("Times", 25)
    message = intrs[i]
    rend = font.render(message, True, pg.Color("red"))
    return (rend, rend.get_rect(topleft=(900,35+Space)))

"""setting font and colour of the name displayed at the start"""
def text_objects(text, font):
    textSurface = font.render(text, True, black)
    return textSurface, textSurface.get_rect()

"""quit game button function"""
def quit_game():
    pg.quit()
    quit()
"""start game button function"""
def start_game():
    app = MainProgram()
    app.main_loop()


#Right hand side title screen colour
def game_intro():
    x=0
    y=0
```

```
    i = 0
    intro = True
    gameDisplay.fill(black)
    while intro:
        for event in pg.event.get():
            if event.type == pg.QUIT: #quit function added
                pg.quit()
                quit()

#Now creating the start-up screen

        gameDisplay.blit(bg,(x,y)) #displaying in starmenu bg being pygame, x and y being the position
        largeText = pg.font.Font('freesansbold.ttf',80)#font and font size
        TextSurf, TextRect = text_objects("Bargain Inspector", largeText)#Program name
        TextRect.center = ((display_width/2),(display_height/2)) #text allignment
        gameDisplay.blit(TextSurf, TextRect)

        button("Start",80,450,120,85,white,bright_green,start_game) #Button which starts the program, position,size, colour and linked to t
        button("Quit Game",605,450,160,85,white,bright_red,quit_game) # Button which closes the program, position,size, colour and linked t

        intrs = ["INSTRUCTIONS:","Enter colour for car type", "Enter co-ordinates", "Enter value of car models", "Enter time for robot",]#I
        space = int(150) #position of the instruction on the screen

        while i != 5: #5 total strings
            prompt = instruction(i,space,intrs)#i=number of instructions, space = position, intrs = intructions
            gameDisplay.blit(*prompt)
            space = space + 40 #how close to each other the instructions
            pg.display.update() #for clock speed functuon
            i = i+1


        pg.display.update()
#limit the clock speed to 15 FPS (to prevent overflow)
        clock.tick(15)
```




```

#buttons function defined, event driven action (for the Start game and Quit button)
def button(msg,x,y,w,h,ic,ac,action=None):
    mouse = pg.mouse.get_pos()
    click = pg.mouse.get_pressed()
    print(click)
    if x+w > mouse[0] > x and y+h > mouse[1] > y: #when mouse button clicked outcome (1,0,0)
        pg.draw.rect(gameDisplay, ac,(x,y,w,h))
        if click[0] == 1 and action != None: #if mouse position (0,0,0 = no action, otherwise event driven action)
            action()
    else:
        pg.draw.rect(gameDisplay, ic,(x,y,w,h))
    smallText = pg.font.SysFont("Times",20)
    textSurf, textRect = text_objects(msg, smallText)
    textRect.center = ( (x+(w/2)), (y+(h/2)) )
    gameDisplay.blit(textSurf, textRect)

```

PWP – the link to your up-to-date Project Work Portfolio (PWP).

https://raia10coventryacuk.wordpress.com/

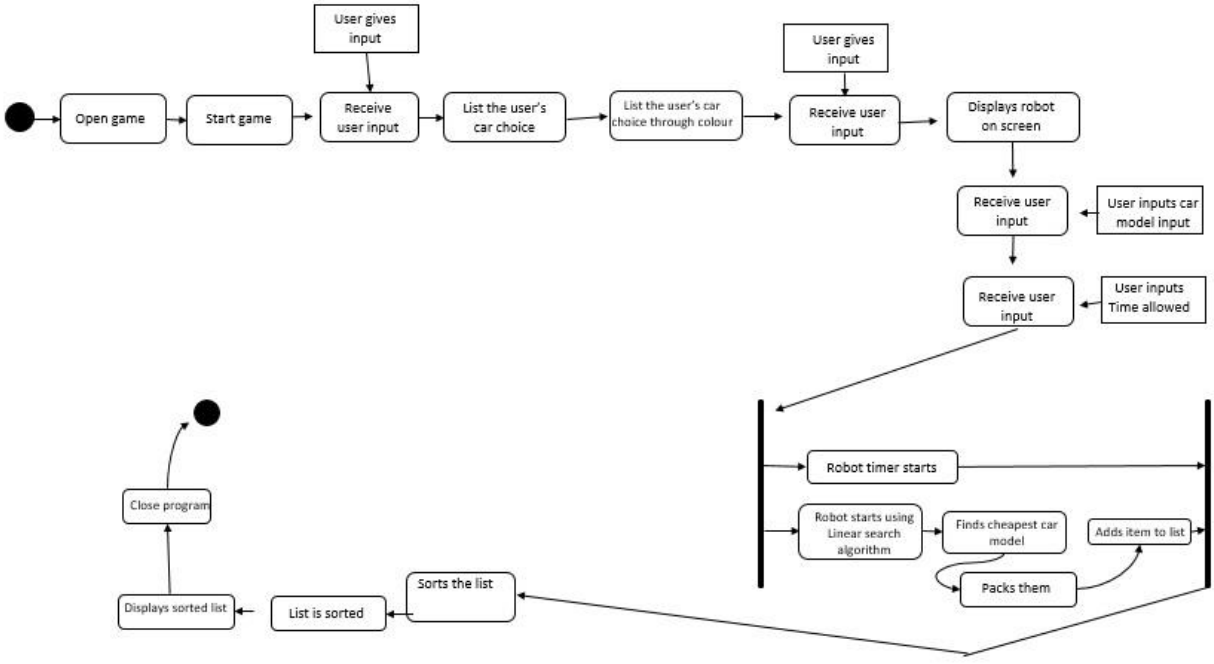
Program testing and results

Test NO.	Description	Expected Result	Actual Result	feedback	Improvements
1	Start-up Screen	The code is run, and then code should bring a GUI with start-screen and instructions displayed for the user. Buttons to start and quit the game should also appear.	When the code was run the code was compiled and returned with a 'USABLE' GUI with the instructions to guide the user, also the buttons appeared and can be used by following the instructions.	N/A	N/A
2	Play button	Clicking the button should take it to the main screen where now the user will follow the on-screen texts to interact with the game	Button pressed and taken to the main screen and able to enter texts into the texts box	Texts font and colour seems hard to see as it is too small, need to increase the font size or change the font style and font colour.	Texts font changed from aerial to times and size changed from 10-14 and now able to see the text displayed on the screen clearly.
3	User input for car model with colours, where red = BMW, Green = Vauxhall, Blue = Landover and Pink = Lexus	Textbox accepts all four inputs and upper case or lowercase will input does not matter. If Inputs other than allowed entered should generate type error.	Textbox accepted all four inputs and upper case or lowercase will input did not matter. BMW inputted and type error came with a message error.	Change the error message to something appropriate for the first-time users.	Error changed to please enter the colour type matching car model type.
4	User input for co-ordinates	User input the co-ordinates of map to search from. The map displayed on the screen and looking that user can enter co-ordinates to search from a specific country.	Entered 'E' for East and started from east side of the map.	N/A	N/A

5	User input for how many number of car models to find	Users should be able to search for items for more than 14 car models.	Users should be able to search for items for more than 14 car models.	Feedback received were vague and not enough time to do improvements.	Feedback received were vague and not enough time to do improvements.
6	User input for how long the user should wait for the search	No more than 60 minutes should be accepted as it is too large for user and for the code.	180 minutes were accepted as well as the short minutes such as a minute.	Reduce the time allowed to 60 minutes max.	Program made to accept no more than 40 minutes as a group we thought it was more than enough time.
7	Robot's movement visibility	Robot's movement must be displayed on the screen.	Robot's movement were clearly displayed on the screen with white smoke left by the robot behind it.	N/A	N/A

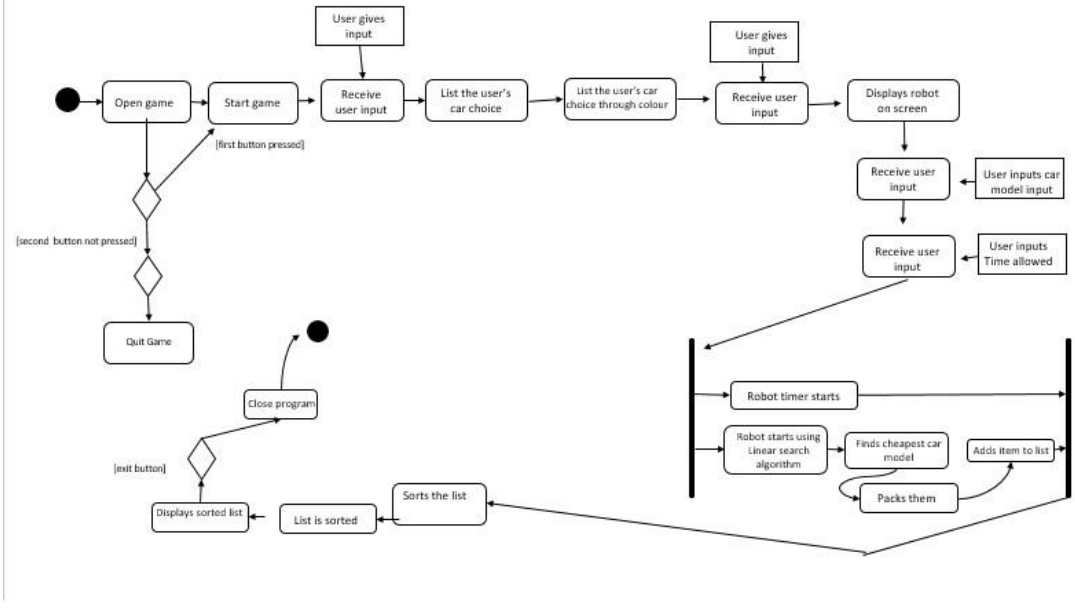
104KM

UML activity diagrams for the program flow with your contribution to the design as an individual clearly identified on the diagrams



I did the Start-up screen, added sound to the code, sorting algorithms and helped with the search algorithm as well.

Updated UML activity diagrams for the program flow with your contribution to the design as an individual clearly identified on the diagrams



I did the Start-up screen, added sound to the code, sorting algorithms and helped with the search algorithm as well.

106CR

A report of what design concepts and principles that have been considered and applied to the prototype design (a maximum of 500 words)

I have learned the design concepts and principles from 106CR and I have successfully implemented in the VRBH program. I found these design concepts and principles very important creating the program as I found out that without the right design concepts and principles, the program could turn out to be very unorthodox for the end user using the program and then this could prove to be very problematic.

From 106CR I have learned 12 important design concepts and principles by Beynon, Turner & Turner in 'Designing Interactive Systems, which I think is ideal to take into consideration when making a program for the end user because these design concepts and principles provide important guideline for good human interaction system design.

One of the principle I considered and implemented was visibility. This principle is concerned with how the data is displayed on the screen to the end user. This principle explains that the best way to data is in graphs or diagram and should be accessible to everyone. I have used this principle by making the text appear in small square and cube boxes which is coloured and should make the user feel very familiar with the program and should make the user feel ready and easy to interpret manner.

Second principle I thought must be considered and implemented was consistency. This principle made sure I kept things the same throughout such as using the same font, same text boxes, same shape buttons into same alignment. This will give the end users indications to which one is button and which one is a text box in the program and helps them to anticipate the next screen or next action.

Another principle I implemented was affordance, this meant that I had to design things to make it easy for the end user to know what each things displayed on the screen was for, for this when creating the storyboard and on the real program, I thought about using radio buttons to make the user to select only one option and using text boxes which made user know that they had to type values or texts into the boxes perhaps by using their past experiences.

Finally, another principle I learned from 106CR was navigation. Using this principle meant that I had to provide support for the end user to move around the system and indicate where else the end user can go to the system, for this I created a navigation in each screen such as in screen 1 there will be buttons with texts inside them to indicate what those buttons do when pressed and give indication to the end user to go further then system by clicking the 'Start' button or exit the program by clicking 'Exit' button. Similarly in screen 3, the end user is able to select only the options that are available and I made sure the option were fit to the end user and what the end user wanted by using other principles such as familiarity, in screen 3 the end user selects the values that the end user want to proceeded with, searched and sorted, to proceed with all of this the user has to proceed by clicking the 'Proceed' button which is colour coded with colour 'Green' which is a universal colour used for 'Go' or 'Safe'.

These were only few of many principles I learned in 106CR and implemented into the program, however they are all equally nevertheless. I believe these design concepts and principles have ensured that I have been guided me in the right steps and has helped me to evaluate and critique prototype design ideas which has made the program 'USABLE'.

Program usability testing – a report (a maximum of two pages) on the evaluation of the program using one of the usability methods (such as heuristic analysis, usability tests and cognitive walkthroughs)

I chose to do testing because a program or a programmer can only get better by testing and practicing. Testing can reveal many things that a programmer may have missed in the code or errors that are hard to capture such as logic errors and the chance of reducing errors can only be possible by doing testing. Testing cab be done to reveal more things other than a program functionality, such as to find out if the users can complete the program which will help reveal the difficulty of the program and therefore can be helpful to programmer to dictate whether the code should be more or less difficult.

I have chosen to do usability testing on participants using the program we created as a group. Usability testing reveals the problems real users experience with your product or web site under the actual conditions of use. We ask participants to carry out realistic tasks and note where they experience problems. [7] We think we have completed all the testing necessary on the code and implemented all the functionality. The test is being done because this test will allow me to get accurate data from real life scenarios by simply observing how a user or a group of users will use our program, how difficult they will find it and if the code generates any errors even though it has been tested beforehand.

During the usability tests I things I observed were effectiveness, efficiency, satisfaction of the user and the learnability because I believe that these are key factors when designing a program usability.
I planned to carry out my usability testing was by having people carry out tasks on the program which were considered important by me and then I observed them and saw how they did it. I told them to thinking out loud during the so that If they encounter an error then I would fix it on the spot and tell them to re-do the test again to see if the problem has been fixed.

I chose to pick 5 people at the ECG building, Coventry University to do the test. I laid out some important tasks in they can perform on the code and watched them do it. I had to make sure I was not making them test to link, load or code testing, I could not ask friends/family, online surveys or focus groups to do the test. I was hopeful by the end of the code I would know the effectiveness, efficiency, satisfaction and learnability of the program.

First task I laid out for my participants was that I loaded up the start-up screen and told them to start the main interface, which was to start the game by clicking the start button.
Result – This went great as everyone was able to start the program without me telling them what button to press or where to press which told me that my start-up screen was effective, efficient and needed no help which meant that no improvements were needed on the start-up main screen.

Second task I laid out was to follow the on-screen instructions and make the robot to find the cheapest car model of their choice from BMW, Vauxhall, Land Rover or Lexus.

Result – This also went great for 2 participants; however, the rest took more time than the others. The participants that finished the quickest were able to follow the on-screen instructions to find the cheapest car model of their choice which suggested to me that they have some computer knowledge and have a very good understanding of how the instructions were set. They pointed out that one of the things they found amazing was how they were able to see visually where in the map, which continent it was searching for and the county within the continent which helped me understand that they were able to give me a really good feedback and not a vague one. However, for the rest 3 participants it did not go well even though they were able to the main task that I laid out. I noticed it was because some of them were really slow at inputting the values needed and some took their time to read out the instruction fully and then read it again and again. This told me that they were not expert users and had a very little knowledge with computers. They told me that instructions were little bit hard to understand and that's why they kept reading it repeatedly. They also mistook the minutes allowed for hours. This told me that I had to change or word the instructions in a way that perhaps all users can understand and therefore I changed the instructions with giving examples next to it and told the 3 users to do the test again. In the second time I noticed that it took them a minute less to complete the task and told them to describe their experience. They told me that it was really good visually, however could be more fun if it had some kind of music in the background. I added a music to the background and told them to do the test again and this time they finished even early because they had done the test 3 times which relates to one of the principle 'familiarity' and they said the music fit the background really well and made it more interesting. With

[7] Userfocus.com, 2016. *Userfocus.com.* [Online] Available at: Userfocus.com [Accessed 06 march 2016].

the feedback received and changes I have made I can conclude that the program is flow of the program and the program itself is effective, efficient, satisfaction and learnability.

The final task I set them was that after the result is displayed, exit the program.

This went amazingly well for all 5 participants and relates to the metal mode as they were able to find the X button on the right side window which was colour coded to red and they would have been familiar with it.

Overall, I believe the program we created was a success and was able to meet the requirements of visual design and interface design. The program has proved to be very effective as all the participants were able to actually do specific tasks that I set with efficiency. I took their feedback of making the program more fun by adding a music to the background and added the music in the spot and the second prototype proved to be provide satisfaction. I took their advice of learnability, making the instructions clearer and changed it in the spot and second prototype provided to be more successful that the first one as the second program was changed with more clear instructions and tested again, with these changes I can conclude that I can use the second porotype as our final program as it meets all the requirements which are Effectiveness, Efficiency, Satisfaction and Learnability.

Bibliography

Cprogamming.com, 2016. *Cprogramming.com.* [Online]
Available at: http://www.cprogramming.com/discussionarticles/sorting_and_searching.html
[Accessed 09 March 2016].

http://staff.ustc.edu.cn/, 2016. *CHAPTER 7: HEAPSORT.* [Online]
Available at: http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap07.htm
[Accessed 06 mar 2016].

http://www.personal.kent.edu/, 2016. *http://www.personal.kent.edu/.* [Online]
Available at:
http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/naiveStringMatch.htm
[Accessed 07 March 2016].

RSI, 2016 . *The Quick Sort.* [Online]
Available at:
https://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuickSort.html
[Accessed 08 March 2016].

University, C., 2016. *Python Basics Lab 6.* [Online]
Available at:
https://cumoodle.coventry.ac.uk/pluginfile.php/663997/mod_resource/content/2/Python%20Basics%20Lab%206%20-%20Search%20Algorithms%20v2.pdf

University, C., 2016. *Python Basics Lab 7.* [Online]
Available at:
https://cumoodle.coventry.ac.uk/pluginfile.php/748776/mod_resource/content/6/Python%20Basics%20Lab%207%20-%20Sorting%20Algorithms.pdf
[Accessed 06 March 2016].

Userfocus.com, 2016. *Userfocus.com.* [Online]
Available at: Userfocus.com
[Accessed 06 march 2016].