Project report:

-Istiak Mohammad

Contents

Analysis…………………………………………………………………….2
Design…………………………………………………………………….…..14
Development and Testing
 Iteration 1…………………………………………………………….….25
 Iteration 2……………………………………………………………….32
 Iteration 3……………………………………………………………….40
 Iteration 4………………………………………………………………44
Evaluation…………………………………………………………….…54
Reference ……………………………………………………………….58

Analysis

The problem

Finding the right product quickly in a big store, comparing with other products and making the purchase /checkout easier by booking a product.

After working at a retail supermarket for part time , I found out that the supermarkets/large malls don't have any customer navigation map to find a product quickly except some display boards (e.g. Fruit, veg, meat etc.) Then when I looked at other supermarkets it became clear that almost all of them use only display boards to navigate their product aisles.

As you can see there is very limited information about products, which is not an issue for small stores and can be found in sufficient or little time. But the issue occurs when the store size is massive/superstore and customers are not familiar with the store..

If a computational approach is taken to this situation , a system could be created which helps the customer/user to be able to search for a product and navigate through the store to find the product easier and quicker. This would save a lot of time, especially because some customers have to verbally ask a store colleague and sometimes they don’t remember the right aisle. With this system it will all be automated and really easy to find a product in a superstore or a large mall.

Problem Decomposition

The main problem I am faced with is dealing with the navigation route which is found in most map systems. This includes the route finding path to decide which path to take in a limited time range.

This problem can be decomposed into smaller steps to help achieve what I am aiming for.

These steps include:

1. The user can properly move across the map freely (with no restrictions). This is
essentially designing the paths which the user will follow.
2. Implementing the new destination/feature one by one and developing it as I go. (e.g. doing the platforms, then the store and so on). This will follow an iterative approach.
3. Design and create maps. Receiving feedback from clients after every
map for feedback.
4. Designing and developing the remaining GUI (including main menu and search tab) and checking design with stakeholders.

Divide and Conquer

I can solve numerous problems through the divide and conquer computational approach. This can be done for each step described in the Problem Decomposition section. For instance, the second step can be divided by implementing the new elements one by one. So the size of the problem is decreased and can be tackled easily.

Abstraction

Abstraction can be used in many aspects of my game.
Technically,when designing the map of a store/mall , it is required in order to visualise how I will place the components. For example, I would draw a rectangle instead of an actual product section/ store to understand where it will be.

Additionally, the map and elements in my system will not be majorly sophisticated in design. They will still be distinguishable as to what they are, though they will be as simple as possible. This is to reduce storage requirements as well as for the simplicity of the game and easiness on the eye for the user. A good example would be a sketch map. (such as the Westfield shopping centre etc.)

Backtracking

Due to how I want to face my problem, It will force me to use previously developed
code to satisfy the new features. Most functions overlap and affect one another (as they will
mainly affect either the display depending on the scenario) therefore it means that
certain variables and procedures/routines will need to change or be updated.
In the map, there could be experiences of backtracking when a user realises a certain
route/path in the map will be less efficient than another, and then goes back to take the better
route.

Stakeholders

The main stakeholders for my system would be superstore/ large malls, specifically Asda superstore, Old Kent road / Sainsbury superstore, Whitechaple. I will speak to the store manager … of Asda superstore, Old Kent road , and target him as my stakeholder.

For my end user, I will have to choose those who don't shop in a supermarket/ large mall often as they will be less familiar with a store and would not know the exact position of a store/product.Having said that, they could be any age group who can read,write and able to use basic search tab.

Research

Westfield mall navigation screen

.

Its main aim is to search a stores and analyse the input from the database and display the desire stores

What would I include?

It has a simple GUI which doesn’t require any technical knowledge, and a user friendly interface.

What would I not include?

I did not like the fact that there is no navigation route feature which will show the path from initial position to target position.This will help users to find the shortest route to their store.

Santander cycle

It is a cycle hire scheme where they allow users to plan their journey via apps/docking stations. Users can choose where to start from / GPS location of the device and the end points then it shows the journey time by categorising “Easy- longest time to get end point ”, “Moderate- average time to get the end point”, “Fast- shortest time to get end point”. It also shows the speed for each category, for Easy-12 km/hr, Moderate-12 km/hr, Fast-12 km/hr. After selecting the mode it shows a route line from start to end and also shows each direction to get to the end point.

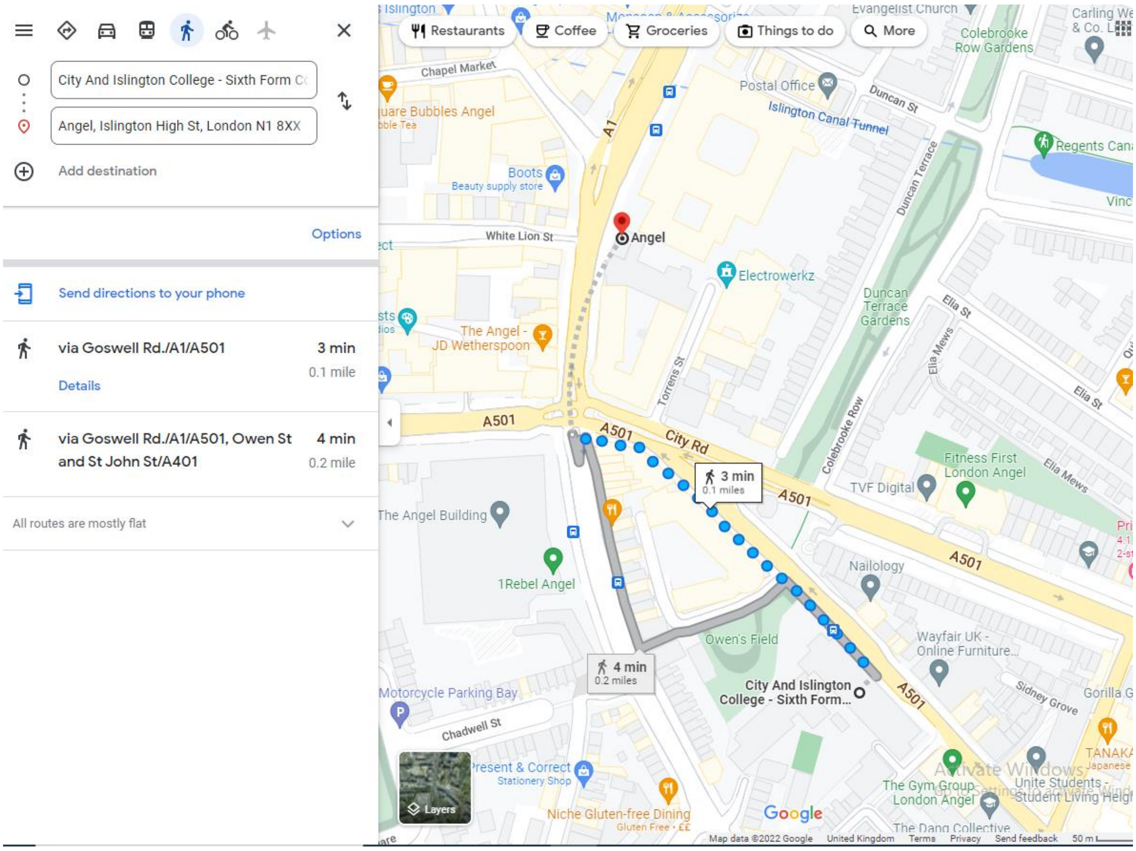
The map that they are showing is provided by Google Map API.

What would I include?
I will be adding the feature start and end points. The categorising of the mood(easy, moderate, quick) and time will be useful for my system as there will be different types of people with different circumstances(some want to find the store quicker).

What would I not include?
I will not add the GPS location feature it requires more data to store about every specific point. Because packet switching needs to take place or i might need to use the API form a provider like Google Map which would not cover the exam board requirement.

Google Map

Widely used map navigation system, has tons of features and algorithms to find the shortest or other routes. Shows real time traffic, dotted path, opposite route and many more.



City And Islington College - Sixth Form C

Angel, Islington High St, London N1 8XX

Add destination

Options

Send directions to your phone

via Goswell Rd./A1/A501 3 min 0.1 mile

Details

via Goswell Rd./A1/A501, Owen St and St John St/A401 4 min 0.2 mile

All routes are mostly flat

What would I include?

It has quite a lot of functions but I will only focus on the walking direction feature as there would not be a transport system in malls or big stores.

It also shows the distance from the starting point which will be useful for users to determine if they will take the route or not .

I think the “add another destination” feature will be useful to accommodate the next destination if the user wants to visit 2 places at once.

What would I not include?

I will not add the opposite path to the destination as it will require more calculations and confusions to the user.

I will not add the real time traffic

Limitations

There are few restrictions that I have to follow.

This cannot be used online as that would require using different languages (php, JavaScript etc.) to store data and packet switching which is not the skill/knowledge I have so connecting user with server is not possible.

This application cannot interact with multiple users at the same time. It's only limited to a single user at a time meaning more than one user cannot interact with my application at same time. However, as my application was created for single user use, it will not be an important feature that is not included.

GUI cannot be complex. My system could confuse users if the gui is too complex. So, I will design the user interface as simple as possible and not make the display overcrowded. I will test it with stakeholders during development of the application to ensure my UI is simple but does the job.

Requirements

Functionality

Requirement	Details	Justification
Display all map elements and once	When a user is in the system, the screen should display all the elements such as nearby shops including the user position.	User has to know where he/she is on the map and all the objects around him/her. Otherwise it would be useless to show user position without map or the objects without user position.
Display the search tab	This option has to be displayed for all the time, to show the user what they searched for and if	User must be able to edit, modify and cancel the product name or the shop he/she is looking

	they wish to change the destination they will be able to do that by this.	for. The system should allow the user to rewrite the data subject. So users can search and change their search object.
The images can be change by using button.	This system should allow user to change the floor.	User must be able to change the floor. The system should allow the user to move between floors.
Store the data	The system should store all the data for maps and the products including the pathfinder which calculates the path for point x to point y.	The system is aware of where the x and y is, then it will display the data that is stored for that specific product or destination.
Calculating the path	From user position to the destination, the system must find a path and display it in real time by depending on the mood selected.	User is aware which path to take and whether he/she wants to go there quickly or slowly.
Calculating the time	From user position to destination, the system will display a time to show how long the journey will take.	User is aware which route to take and how long it will take from his position to the destination.
User input	The system takes all the possible inputs by the user and detects when the user does input.	Without this key feature the system would not be able to work so it is necessary for my system.
Correct output	It must display the correct information after it recognised the input.	The system is worthless without outputting any instruction after detecting inputs.

Non-Functionality

Requirement	**Details**	**Justification**
Map layout	The map design should display in such a way where user disabilities would not affect like colour blind.	Most maps have a certain style colour scheme which helps users in the visual aspect of the system. Also, users could be in different difficulties so some colours would not be used like red and green.
Help screen	It shows how users can input and find the product or shop by using the search tab, following the path etc.	New users might not be familiar with the system so it is necessary to include this so they are informed to use the system.

Software

Windows/Mac/Linux Operating System	These are the only operating systems that are supported by python. Without these, the system could not be run.
Python Interpreter	As the coding will be written in python, an interpreter is required in order to perform the code

Hardware

Keyboard and mouse	These will be the main inputs for my system. These are essential as the system cannot be played without these
A computer with decent specs (minimum)	The system will require a decent amount of processing power (especially as it will need to do lots of calculations and processes), therefore, a decent computer will be needed in order to get the optimal experience from this system. I.e. no lagging or slowing down.

Success Criteria

Criteria	**How to collect evidence**
Simple main menu	Screenshot of the main menu to ensure that it opens as soon as the software opens.
Search tab and button	Screenshot of the search tab and button before and after the mouse click. Making sure it is at the right output for each path.
Displaying the map	Screenshot of the map at the beginning and middle of testing to ensure that the user is displayed correctly.
Maps and objects placed correctly	Screenshot of the screen to check the

	user and the objects/shops are in the correct position. This will ensure my map is set properly.
checking the object position by mouse click	Screenshot of the screen to check the user and the objects/shops are in the correct position. This will ensure my map is set properly.
Finding the position of objects	Screenshot of the coordinates to check the region clicked display accordingly. This will ensure my map is set properly.
The images can be change by using button.	Screenshot of the button to check the image change accordingly. This will ensure my map is set properly.
Users can view the starting and ending position	Screenshot of the screen to check the starting and ending position is displayed after mouse click. This will ensure my map is set properly.
Users can search for a type of product (shoes, clothes etc.)	Screenshot of the search tab after the search button is clicked to check the code properly.
User can choose which mood to use (easy, moderate, quick)	Screenshot of the software after the mouse clicked the search button. This is to make sure the mood feature works.
The map can display the destination	Screenshot of the map after the search button is clicked. This is to ensure the destination place is showing correctly on the map.
The map can navigate to the	Screenshot of the map after the mood

destination	is selected to check the algorithm/ path finder works the way it should.
The map shows the time of the journey	Screenshot of the map mood is selected. Each mood will show different time range (e.g., Easy mood will show the longest path and bigger time range while Quick will show the shortest and smallest time range)
Router button to find the path between two positions.	Screenshot of the router button after the implementation to check the algorithm/ path finder works the way it should.

Design

<u>General Breaking down into smaller problems.</u>

The map will be tackled by aiming to solve different smaller problems which will produce to form the full solution. The smaller problems are as follows:

-Displaying the map: This will be the first problems within the system to be tackle as it is a fundamental and required for the map. Without this feature the map would not be functional.

-Button to change floor: This is to change from one floor to another desired to the as desired position can be choose from.

-Displaying the selected position: This will be useful to see for user that what position is selected, so user can modify their selection.

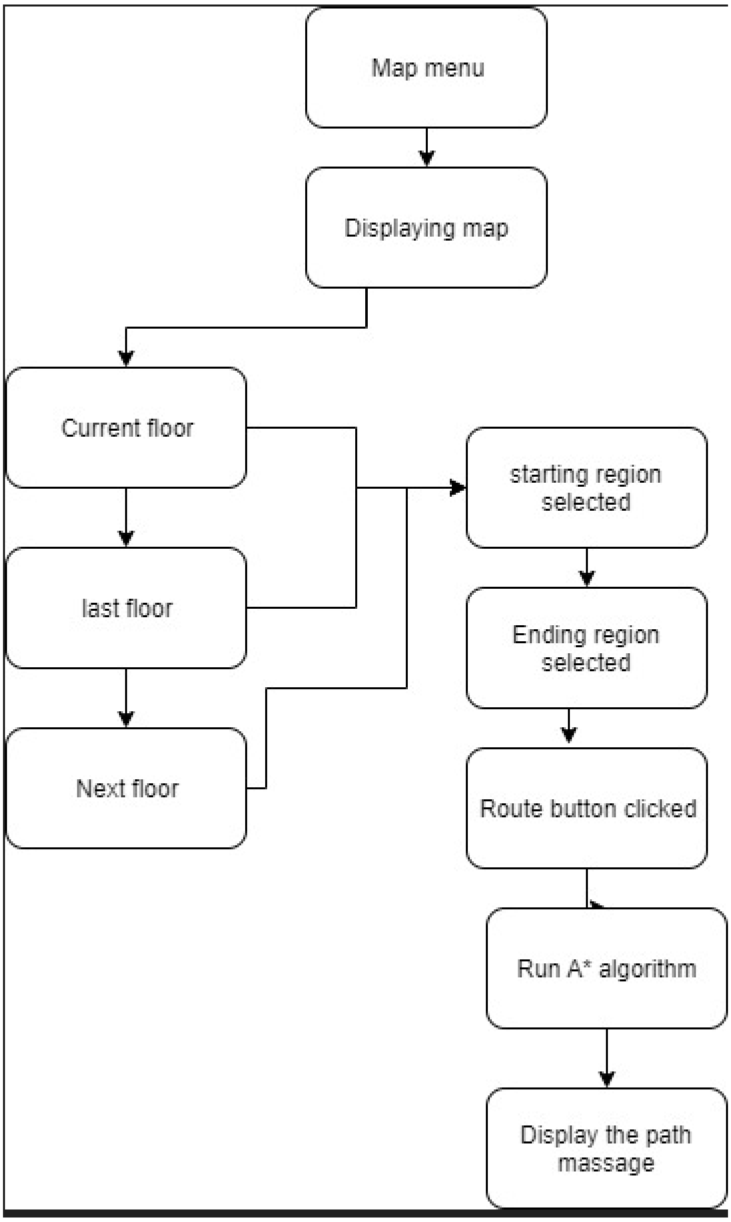
-GUI and algorithm linking: This is one of the crucial features. Without the algorithm the map would be only displayable without functionality. I will aim to develop this feature.

- Path massage output: This feature is necessary as the user would not be able to understand abstract graph. It will deliver readable massage for the path. I will add this feature as long as I have enough time.

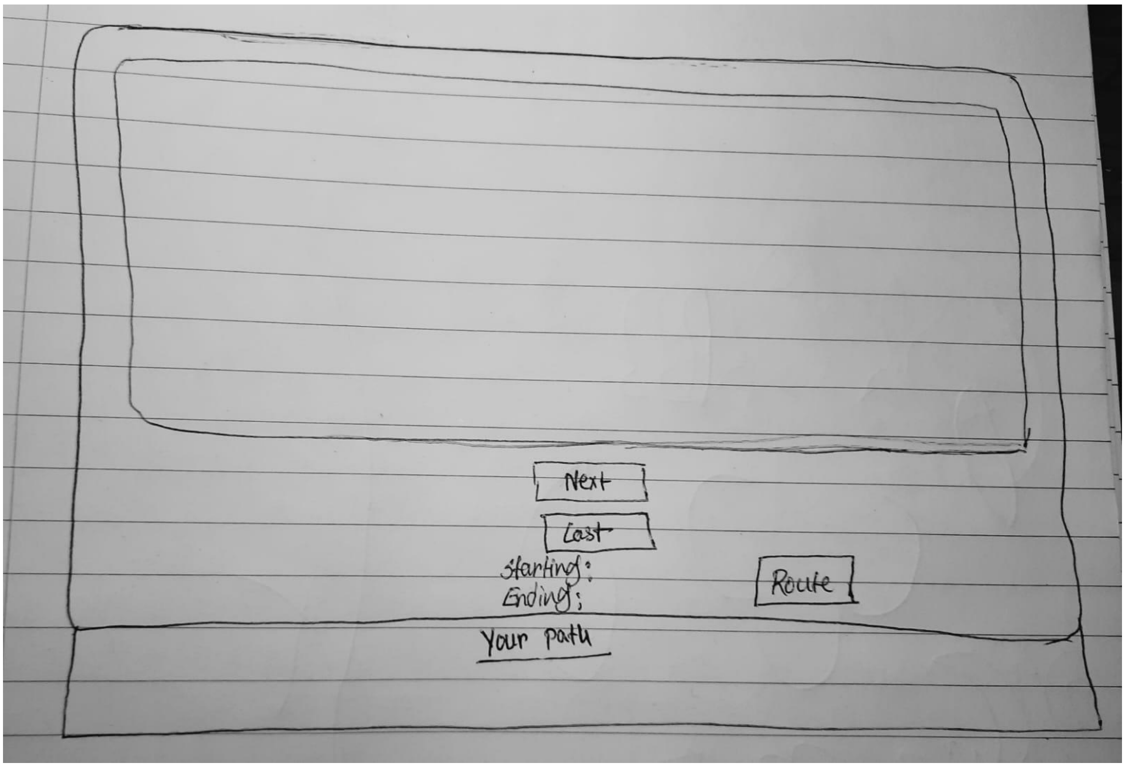
These are the main problems which will be tackled in order to form a full solution.

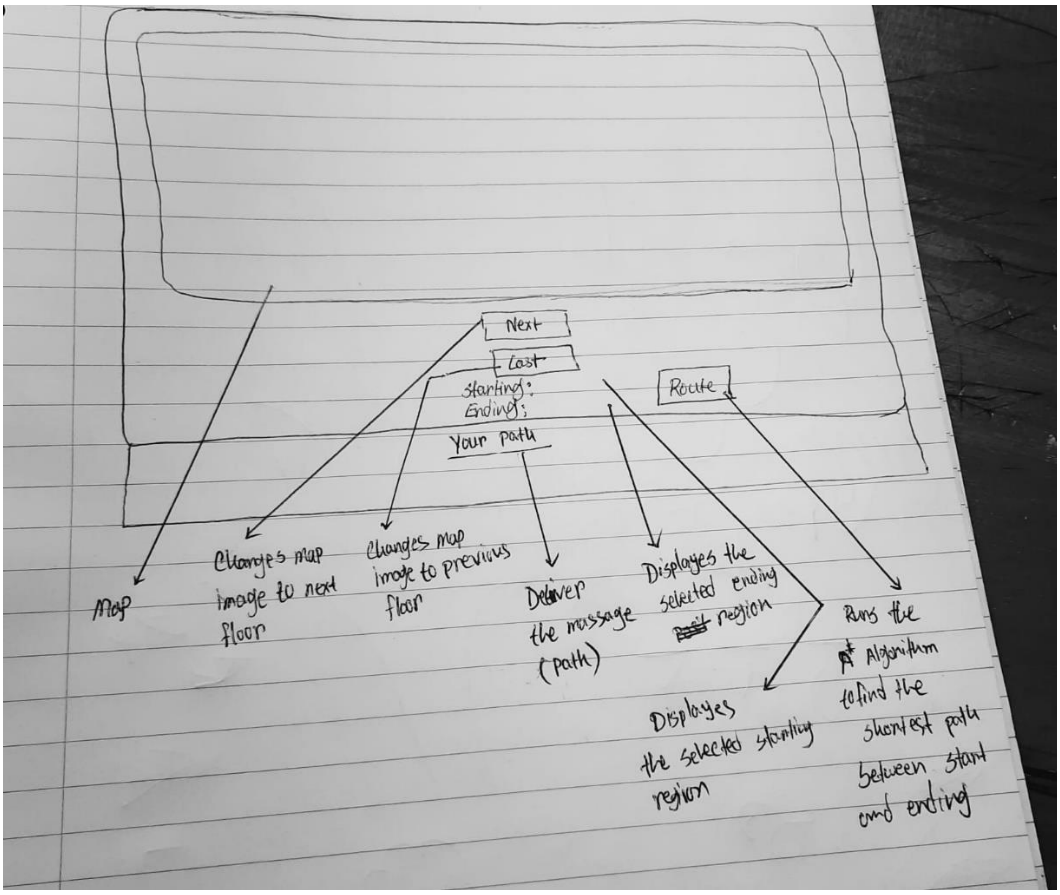
Initial decomposition

Here is the flow diagram of the problem breaking down into smaller sub problem.



Map menu



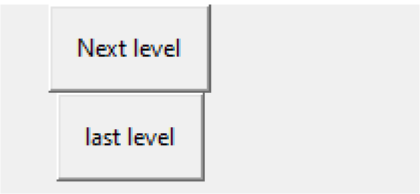

Next
Last
Starting:
Ending;
Your path
Route
Map
Changes map image to next floor
Changes map image to previous floor
Deliver the massage (path)
Displayes the selected ending region
Displayes the selected starting region
Runs the A* Algorithm to find the shortest path between start and ending

Map menu

This is the map menu which will be displayed when user starts the program. The design is simple and consist of all the features at one window. It shows user the map image and has button to change between floors. User can pick the desire starting and ending position by mouse click. As soon as any starting or ending position is selected it will display on the screen. Once the starting and ending position is selected, the "Route" button will run the a* algorithm to find the shortest path of the route. The path will display on "Your path" section.

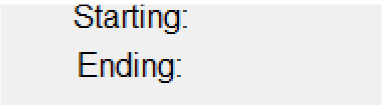
Justification

I made the map size large so user can easily view the details of the map, which allows user to have an easier experience with UI. There are no other modes for this map which is essential to because the map has less data to handle and output quick result, considering majority of the user will be using this in busy time in mall or those who are unfamiliar with the area. More details in GUI could confuse users leaving a bad experience in the map.



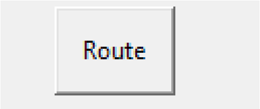
Changing floor

Because of easy view and detail viewing experience I create two buttons. Initially the picture it displays when the program is open is consistent. The "Next" button displays the next floor picture, and the "last" button displays the previous floor picture when it pressed. Therefore, the user can always change their desired floor as they want. Without this the user would not be able to change the floor and the program would not be functional.



Starting and ending position

The text box displays the position name when they are selected. When the first mouse click will occur inside the picture that will be selected as starting position. The second mouse click event will be selected as ending position and both of the position name will be displayed. Without this the user would not be able to input data and the program would not be functional.

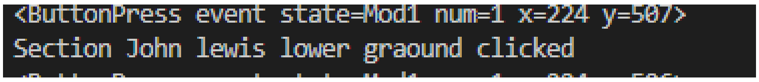


Route button

This takes the starting and ending value and pass it to the algorithm to find the shortest part between position. Without this the user would not be able to find the path and the program would not be functional.

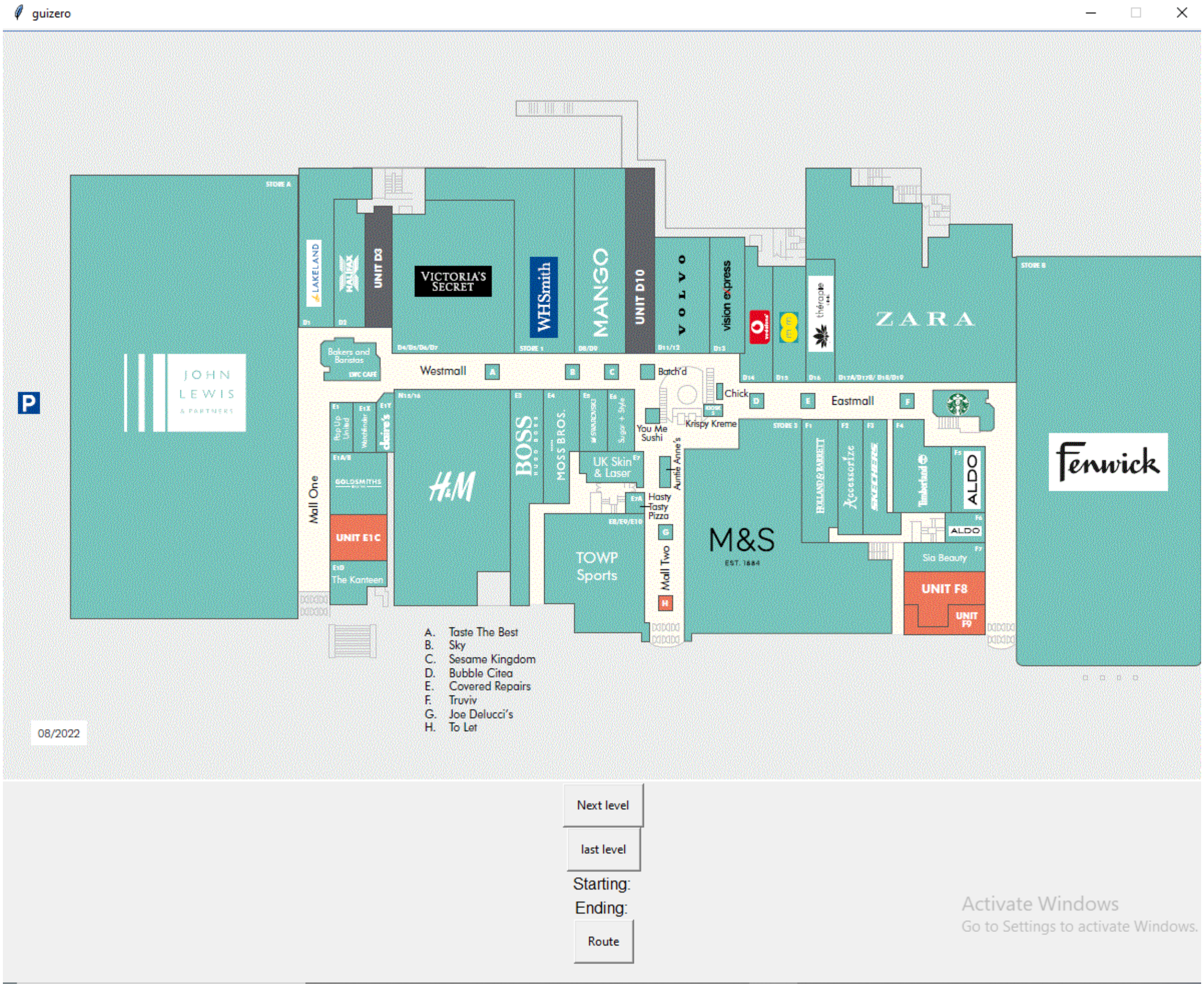
Your path

This will be a simple where it will display the route massage for user. The massage will consist of string only, so using array will be more effective as array is quickest compared to list and tuple and homogeneous data type.



```
<ButtonPress event state=Mod1 num=1 x=224 y=507>
Section John lewis lower graound clicked
```

Here is the GUI appear after implementation.



Object detection

To be able to select an object in on the map I have to use coordinates to measure the region and compare it with the defined data. Because the map program is developing in python so I can use the top-left corner and bottom-right corner to select a rectangular region. The coordinates are stored in a tuple which then compares with the current selected region to see the area selected is matched or not.

Justification

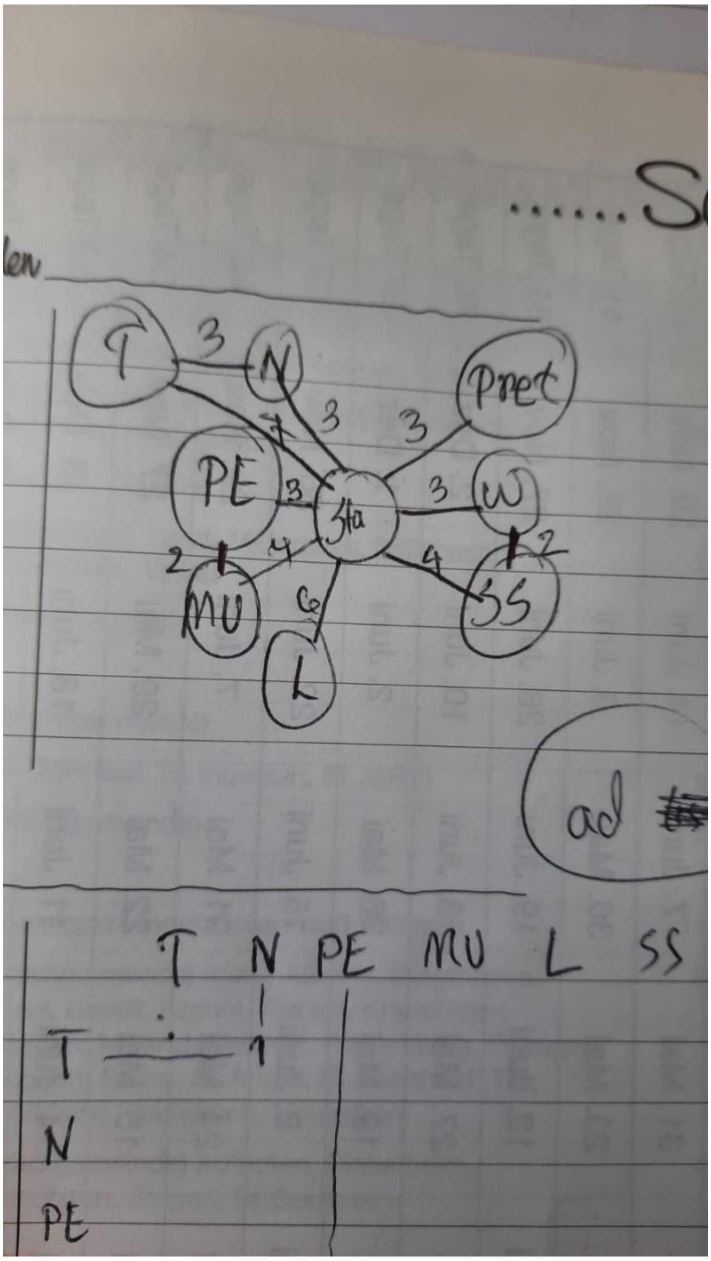
I choose tuple to store object data because it can contain more than single datatype. Also, tuples are faster than list to access and retrieve data.

Graph

The entire map is developed through the use of array, tuple, list and graph. This would be the most efficient way to represent the map as the graph is an abstract data type used to represent a collection of nodes and the edges between them. It helps simplify complex problem by providing a clear interface for manipulating data. It is important because this will help me organise modify the data in a clear and consistent way.

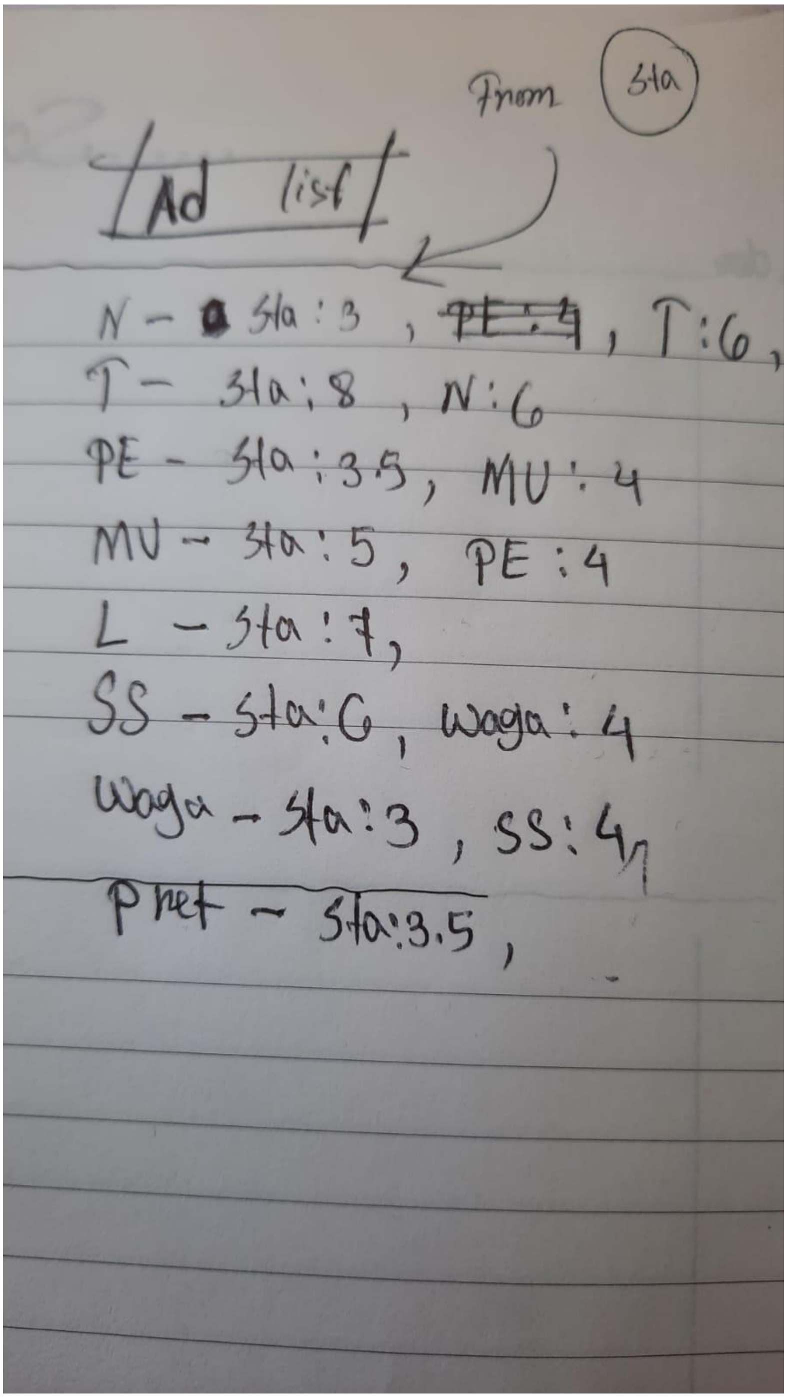
For implementation I will be using adjacency list as it would be more space efficient than adjacency matrix. I will be implemented as list of dictionaries, with the key in each dictionary being the node and the value, the edge weight.

The graph I started with is this:



	T	N	PE	MU	L	SS
T		1				
N						
PE						

For upper floor I draw a graph by using a ruler. To build an adjacency list I collect all the weight of each for node. After collecting weight, my list appears like this:



From Sta

Ad list

N - Sta : 3 , ~~PE : 4~~ , T : 6 ,
T - Sta : 8 , N : 6
PE - Sta : 3.5 , MU : 4
MU - Sta : 5 , PE : 4
L - Sta : 7 ,
SS - sta : 6 , Waga : 4
Waga - Sta : 3 , SS : 4,
Pret - Sta : 3.5 ,

The datatype I will be using to implement the map will be dictionary. Which is memory efficient. It shows which object relates to other. The value on the right is the cost of the node for which I measure by ruler.

Abstraction

This allows me to work with a simpler and more understandable way of the system, without having to worry about the complexity of the map. It is important for this program because it allows me to manage complexity and build modular and reusable program. By breaking a system down into smaller, simpler abstractions, I can focus on solving one problem at a time, and then combine the solutions into a larger program. This makes it easier to design, develop, and maintain program like this.

A* algorithm

To find the shortest path between 2 position this algorithm is essential. It is also efficient than other path finding algorithms. It uses graph data structure to find the shortest path. It based on a heuristic search approach, which means it uses an estimate of the distance between a current node and the goal node to make decisions about which path to take next.

The cost of each node is calculated as the sum of its actual cost (g(n)), which is the cost to reach the node from the start node, and its heuristic cost (h(n)), which is the estimated cost to reach the goal node from the current node. The total cost of a node is f(n) = g(n) + h(n).

By selecting nodes with the lowest total cost (f(n)), the algorithm process nodes that are closer to the goal node and takes the most efficient path to reach it. The algorithm terminates when the goal node is found.

I will be using the algorithm provided by raspberry pi foundation. Because writing similar algorithm on my own would-be time consuming and the algorithm is already pre-tested. So, it would be effective approach for me. Here are the steps I will be using:

- Define the start node and the target node.
- Create 2 list. The unvisited list contains nodes that are being considered for the path, and visited list contains nodes that have already been explored.
- Add the start node to the unvisited list.
- While the unvisited list is not empty:
  Select the node with the lowest total cost (f(n)) from the open set.

If the selected node is the goal node, return the path. c. Otherwise, remove the node from the unvisited list and add it to the visited list. Generate the adjacent nodes of the selected node.
For each adjacent nodes: If the it is already in the visited list, ignore it. If it is not in the unvisited list, add it and calculate its cost (f(n)). If the it is already in the unvisited list, update its cost if a lower cost path has been found.

Key Variables

Name	Data Type	How it is used and justification
section clicked	String	
current_image	integer	
coordinates x1, y1, x2, y2	integer	
coordinates [4]	string	
event.x , event.y	integer	
selecting_start	boolean	
start_text.value , end_text.value	string	
current_image	integer	
image_list	list	

Test Plan

Description	Test Data (if applicable)	Expected Results	Test pass?
Check the map menu displaying		The program should start with the map displayed	
Testing of all the buttons	Clicking with left mouse	The screen would show as the interface when a specific button is pressed	
Displaying the map		The screen would display the map	
Maps and objects placed correctly		All the object displays in the screen can be identifiable.	
checking the object position by mouse click	coordinates x1, y1, x2, y2, coordinates [4]	The display will show the correct position when clicked.	
Finding the position of objects	event.x , event.y	The display will show the x, y coordinates of the position clicked.	

The images can be change by using button.		The display will show the next or previous image according to the button pressed.	
The map can display the destination		The screen will show the ending position name.	
The map can navigate to the destination from starting position		The display will show the shortest path between starting and ending position.	

Development and Testing

ITERATION 1

In this iteration, I will focus on developing the following aspects of my success criteria:

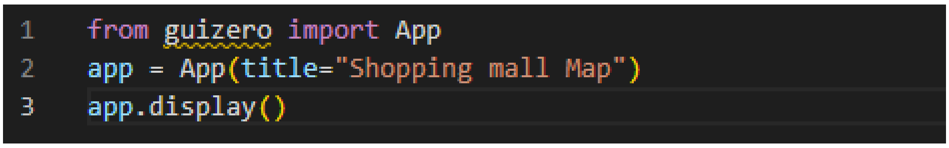
-Display the main window

-Display the map

-The user can change the floor of the map

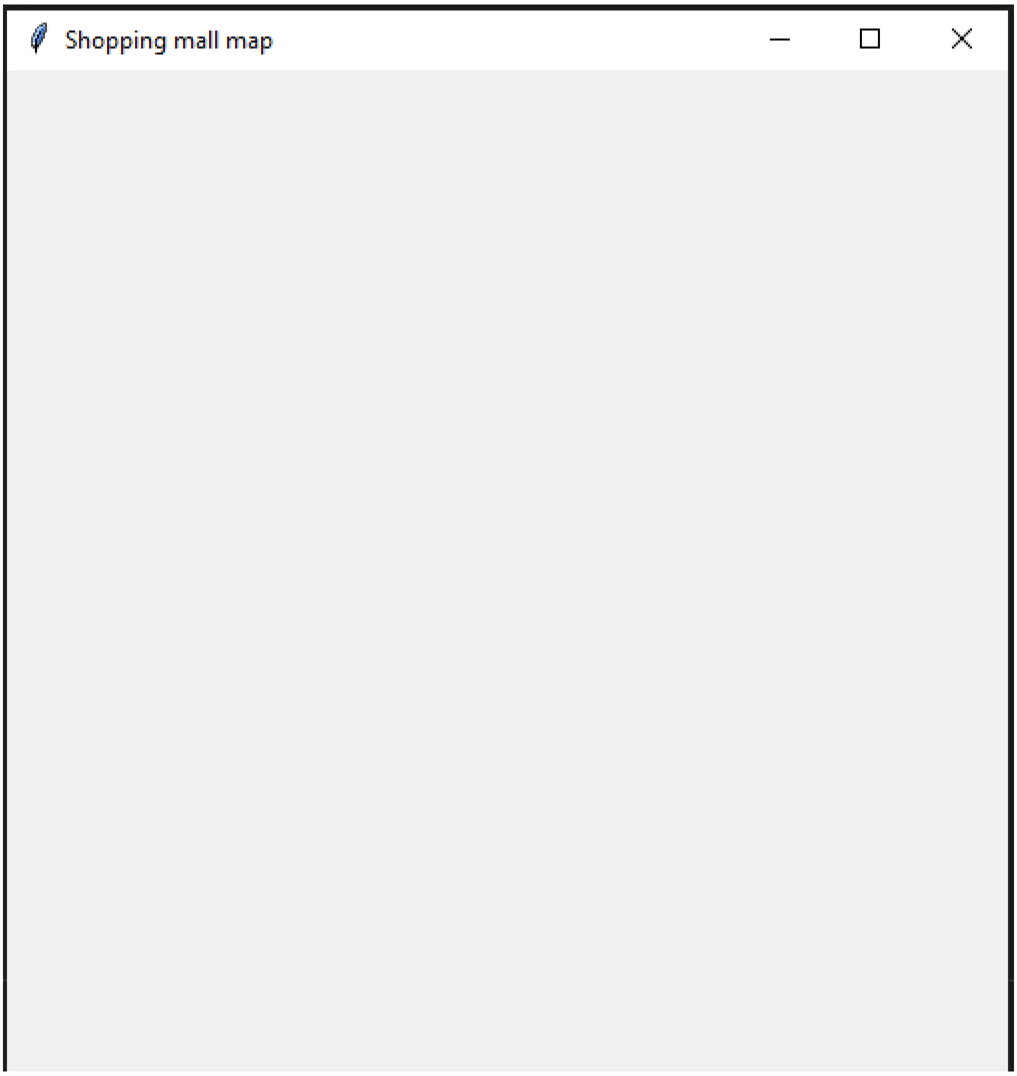
To do this iteration, I will initialise and create a guizero window. Then the button to change the floor will be created.

Firstly, I need to set up the window. Here is the code I have started with:



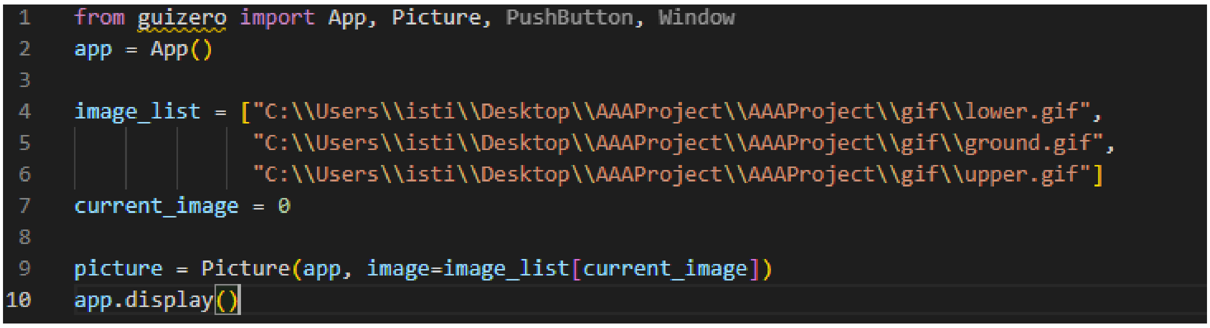
```
from guizero import App
app = App(title="Shopping mall Map")
app.display()
```

The code is simple. It initialises the App module and setting the caption as “Shopping mall map” which is the name of the program will be.
This is what is displayed when program is run:



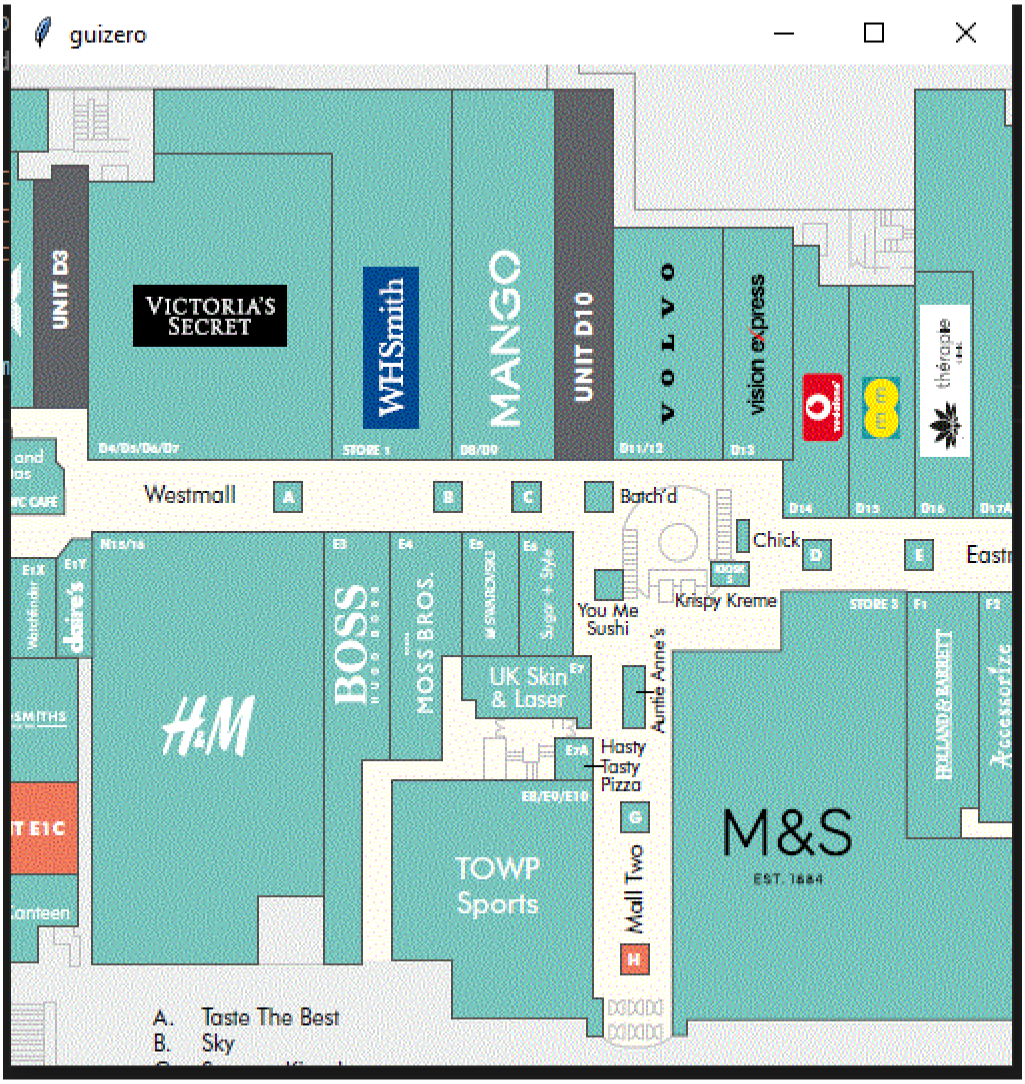
It only displays the window. Next, I will be adding the main background as pictures of the map in the window and the program is successfully closed when I pressed “X” in the corner.

Starting with the background, I have selected 3 gif images of the map. It takes the gif image and displays it onto the screen. I also use Picture module from guizero to display my image. Here is my code:

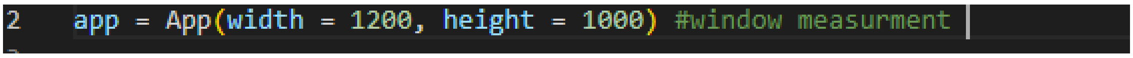


```python
from guizero import App, Picture, PushButton, Window
app = App()

image_list = ["C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\lower.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\ground.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\upper.gif"]
current_image = 0

picture = Picture(app, image=image_list[current_image])
app.display()
```

The gif images stored in a list called image_list. I have initialised the path of the images. The picture module takes the image and display on the window. The map always starts with the first image(lower.gif) so current_image is set to 0. Here is what is displayed:

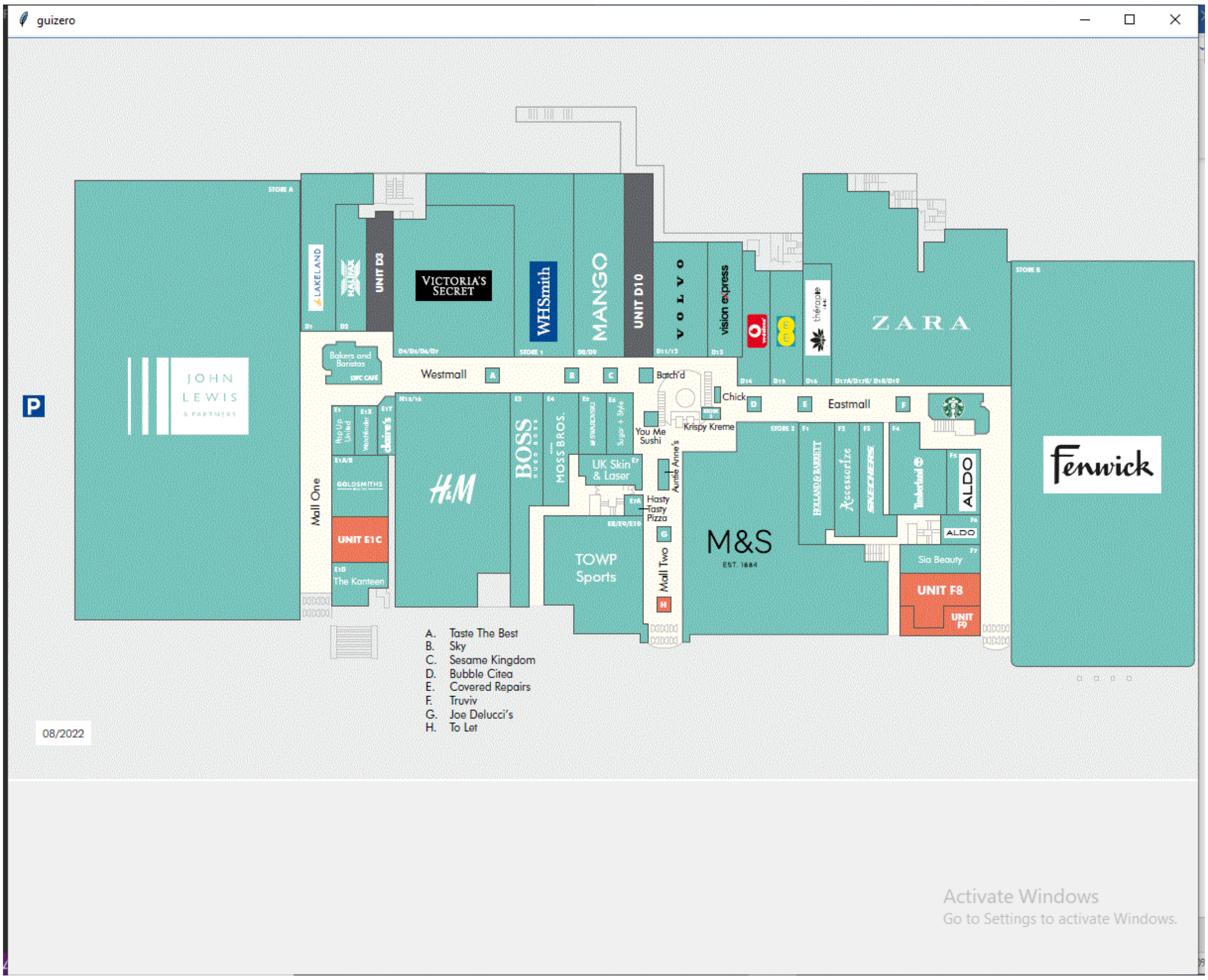


As I did not initialise any measurement of the window it only displayed a part of the full image. So, I add this code for the height and width to the App module to initialise the window measurement. Here is my code:



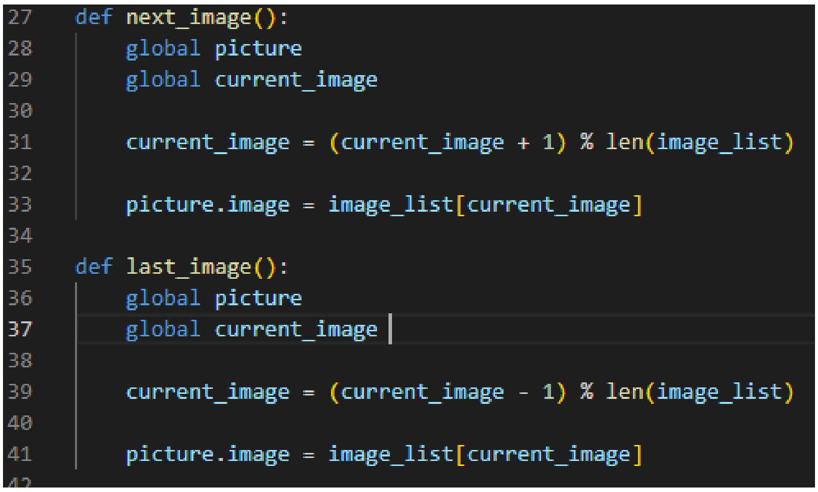
```
app = App(width = 1200, height = 1000) #window measurment
```

Now, it displays the full image as I wanted.



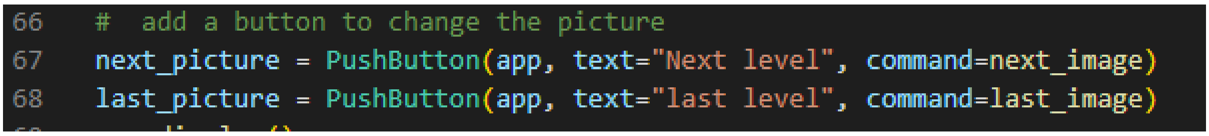
Now the simple part finished, I will add button which will change the floor image.

To do this I write 2 function each initialise the global variables picture and current_image.The function next_image() the current_image incremented by 1 and in last_image() it decremented by 1. Here is my code:



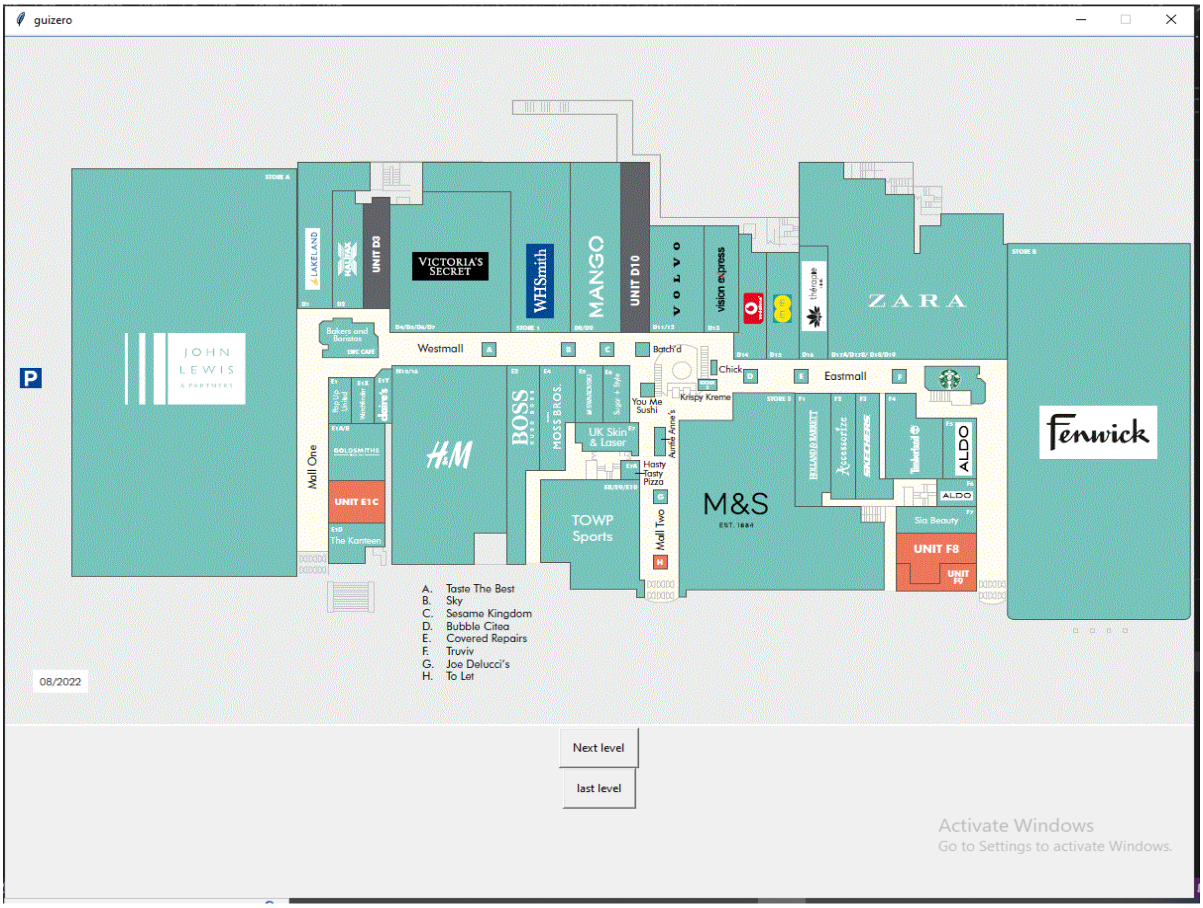
```python
def next_image():
    global picture
    global current_image

    current_image = (current_image + 1) % len(image_list)

    picture.image = image_list[current_image]

def last_image():
    global picture
    global current_image

    current_image = (current_image - 1) % len(image_list)

    picture.image = image_list[current_image]
```

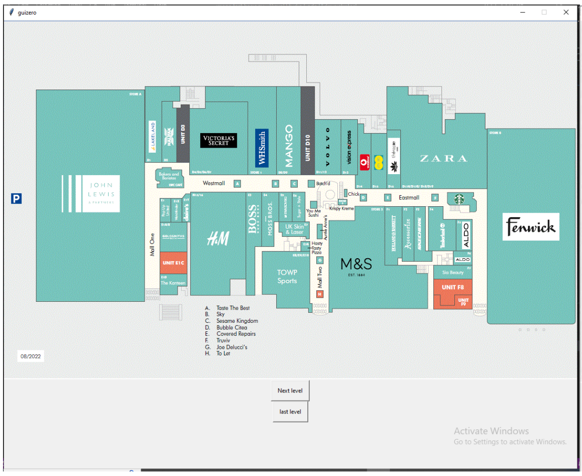
To create buttons, I use pushbutton and initialise the functions to the command.

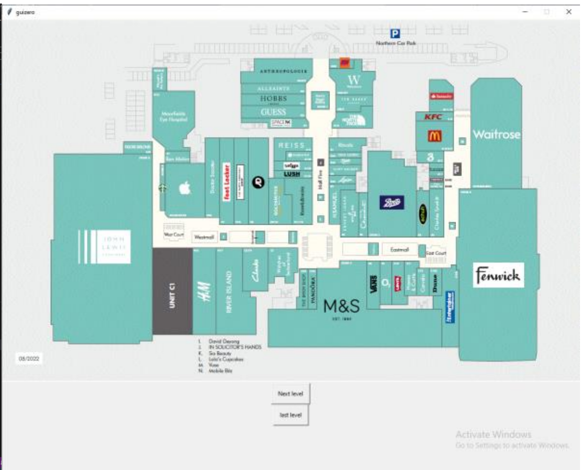


```python
#  add a button to change the picture
next_picture = PushButton(app, text="Next level", command=next_image)
last_picture = PushButton(app, text="last level", command=last_image)
```

Here is the window with fully functional button with image.







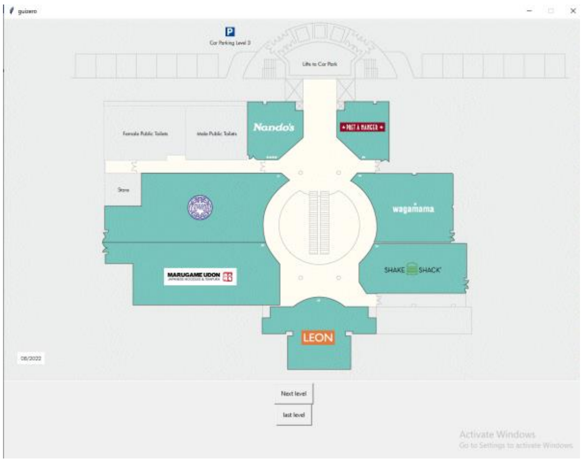


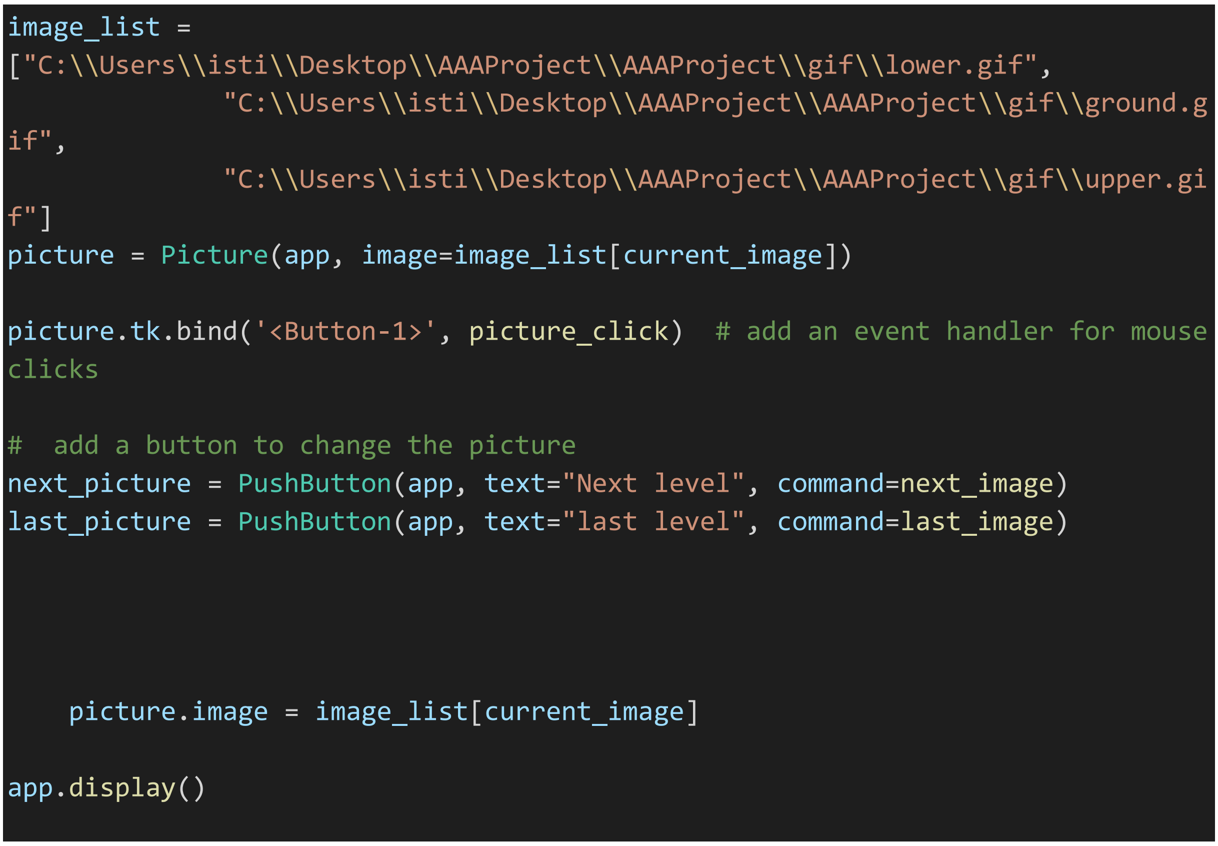
Figure 1First floor

Figure 2Ground

Figure 3Ground floor

Code in this stage

```
from guizero import App, Picture, PushButton, Window, Text
def next_image():
    global picture
    global current_image

    current_image = (current_image + 1) % len(image_list)

    picture.image = image_list[current_image]

def last_image():
    global picture
    global current_image

    current_image = (current_image - 1) % len(image_list)
app = App(width = 1200, height = 1000)
```



```
image_list =
["C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\lower.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\ground.g
if",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\upper.gi
f"]
picture = Picture(app, image=image_list[current_image])

picture.tk.bind('<Button-1>', picture_click)  # add an event handler for mouse
clicks

#  add a button to change the picture
next_picture = PushButton(app, text="Next level", command=next_image)
last_picture = PushButton(app, text="last level", command=last_image)




    picture.image = image_list[current_image]

app.display()
```

WHAT HAS BEEN DONE

I have start with the display of map while also implemented the initial button to change floor.

Main testing

Test No	Purpose	Data	Expected Results	Achieved?
1	See if the window displays with the image provided	app.display()	As the program opens it should display the image	YES -program displays the image provided.
2	See if the image of the program changes	Move floor/image using next_level or last_level button-	The screen should display the image moving accordingly	YES - current_image increases with next_level

		current_image value	to the button pressed	- current_image decreases with last_level

SUCCESS CRITERIA MET

-Displays the images

-The images can be change by using button.

- Maps and objects placed correctly

SUMMARY OF SYSTEM

So far, the system is only capable of opening a window where a picture of the map appears and is capable of moving previous and next by the button.

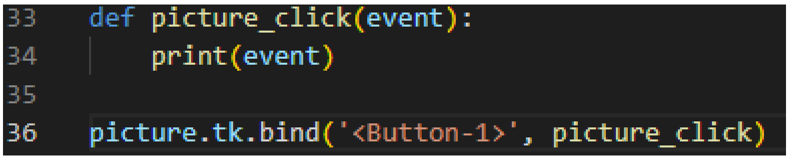
ITERATION 2

In this iteration, I will focus on developing the following aspects of my success criteria:

-Generate object coordinates.

-Detecting the object position by mouse click.

To do this iteration, I will initialise and use a python builds in event handler. Then the button to change the floor will be created. This handle the event when a mouse is clicked. Here is the code I start with:



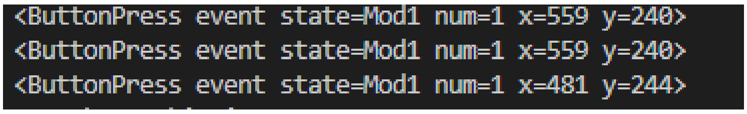
```python
def picture_click(event):
    print(event)

picture.tk.bind('<Button-1>', picture_click)
```

This generate coordinate when the image is clicked by left mouse button. The function picture_click takes the parameter “event” and print its out.

```python
picture.tk.bind('<Button-1>', picture_click)  # add an event handler for mouse clicks
```

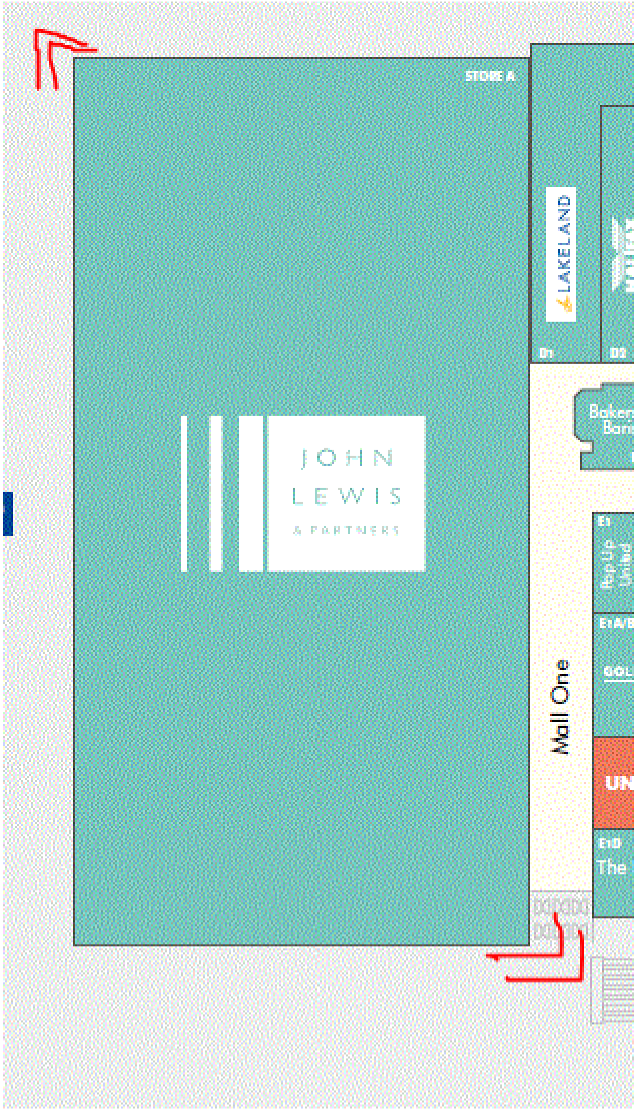
The variable picture uses tkinter library to define a area. The .tk.bind method is used to bind the (<Button-1>) (left mouse click) event to picture_click function. so that when the left mouse button is clicked inside the picture area, the picture_click function will be called with an event object containing information about the click event.

When the picture_click function is called, it simply prints the event object to the terminal, which will show coordinate about the mouse click. Here is the output of the code:

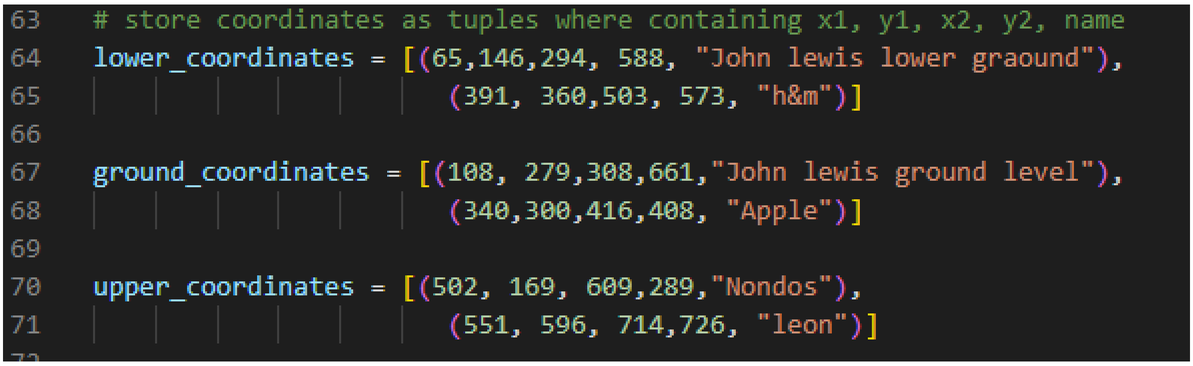


```
<ButtonPress event state=Mod1 num=1 x=559 y=240>
<ButtonPress event state=Mod1 num=1 x=559 y=240>
<ButtonPress event state=Mod1 num=1 x=481 y=244>
```

Now, I can generate coordinate.so I will generate few test coordinates of each floor and naming the object. To store each objects coordinate I will use a tuple as it can hold different datatype. where each tuple represents a rectangular region defined by four integer values representing the (x1, y1, x2, y2) coordinates of the top-left and bottom-right corners of the rectangle. The fifth value in each tuple will represent the name which will be a string. For each floor I will use a list to store multiple tuples.



To generate coordinates, I clicked the top-left and bottom-right corners of the region of an object, in this case “john lewis” that gives me coordinates of x=65 y=146 for top-left and x=294 y=588 for bottom-right corners.
Here is the code I wrote:

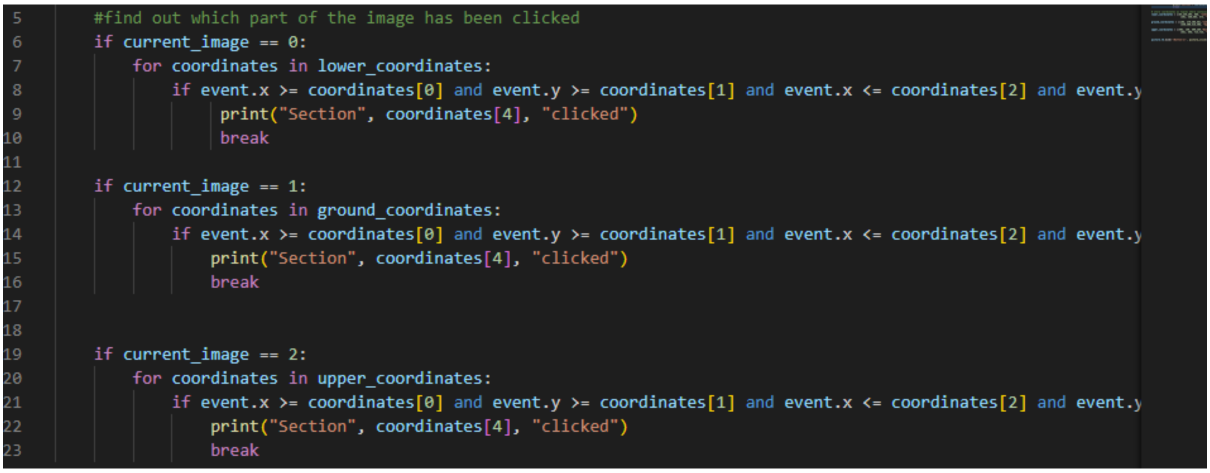


```python
# store coordinates as tuples where containing x1, y1, x2, y2, name
lower_coordinates = [(65,146,294, 588, "John lewis lower graound"),
                       (391, 360,503, 573, "h&m")]

ground_coordinates = [(108, 279,308,661,"John lewis ground level"),
                       (340,300,416,408, "Apple")]

upper_coordinates = [(502, 169, 609,289,"Nondos"),
                       (551, 596, 714,726, "leon")]
```

Each floor is represented by lower, ground, and upper coordinates which store a list of tupls where first 2 coordinates represent the top-left and last 2 coordinates represents the bottom-right corners of the rectangle position.
I use tuple for each object because array only can store single datatype and list are mutable(the value can be change after created). Tuples are immutable, which means their values cannot be changed once they are created. In this case, I do not want the coordinates or labels of the rectangular regions to be changed after they are defined(static). Also, tuples are generally more memory-efficient than lists.
I use list for each floor because list is dynamic data structure, which means I will be able add new value to after they created.

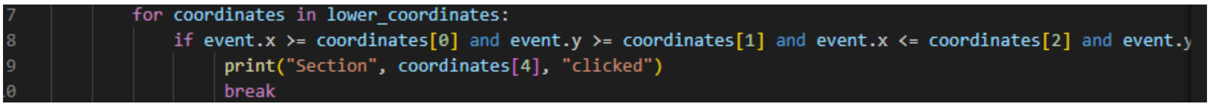
I use 2 object for each floor for test purposes.

To output the stored value and find out which part of the image has been clicked, I will use a for loop inside the picture_click function which unpacks the values of the tuple and checks if the coordinates of the mouse click event is within the current rectangular region by comparing them to the coordinates of the top-left and bottom-right corners of the rectangle. If the mouse clicked within the current rectangle, the name of the object will be print out. Here is the code I write:



```
    #find out which part of the image has been clicked
    if current_image == 0:
        for coordinates in lower_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y
                print("Section", coordinates[4], "clicked")
                break

    if current_image == 1:
        for coordinates in ground_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y
                print("Section", coordinates[4], "clicked")
                break


    if current_image == 2:
        for coordinates in upper_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y
                print("Section", coordinates[4], "clicked")
                break
```

This code checks the value of the current_image and iterate over each tuple in each floor using a “for” loop. Checking if the current is mouse clicked occurred within the rectangular region defined by the current tuple.



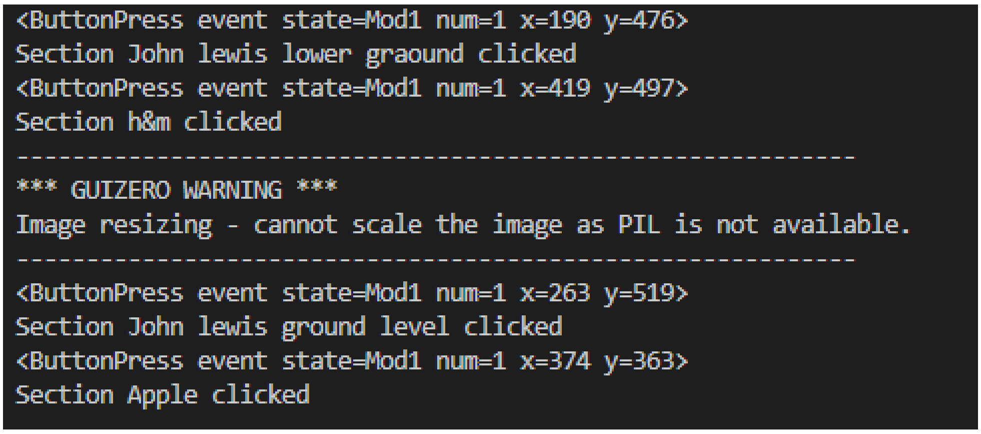
```
        for coordinates in lower_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y
                print("Section", coordinates[4], "clicked")
                break
```

The first line inside the loop unpacks the integer values representing the (x1, y1, x2, y2) coordinates of the current region and the name of the coordinates from the tuple.

The second line then checks if the x and y coordinates within the clicked region by comparing them to (coordinates [0], coordinates [1], coordinates[2], and coordinates[3].

If the mouse button is clicked within the region the third line print out the name of the object to the terminal that the object is clicked.

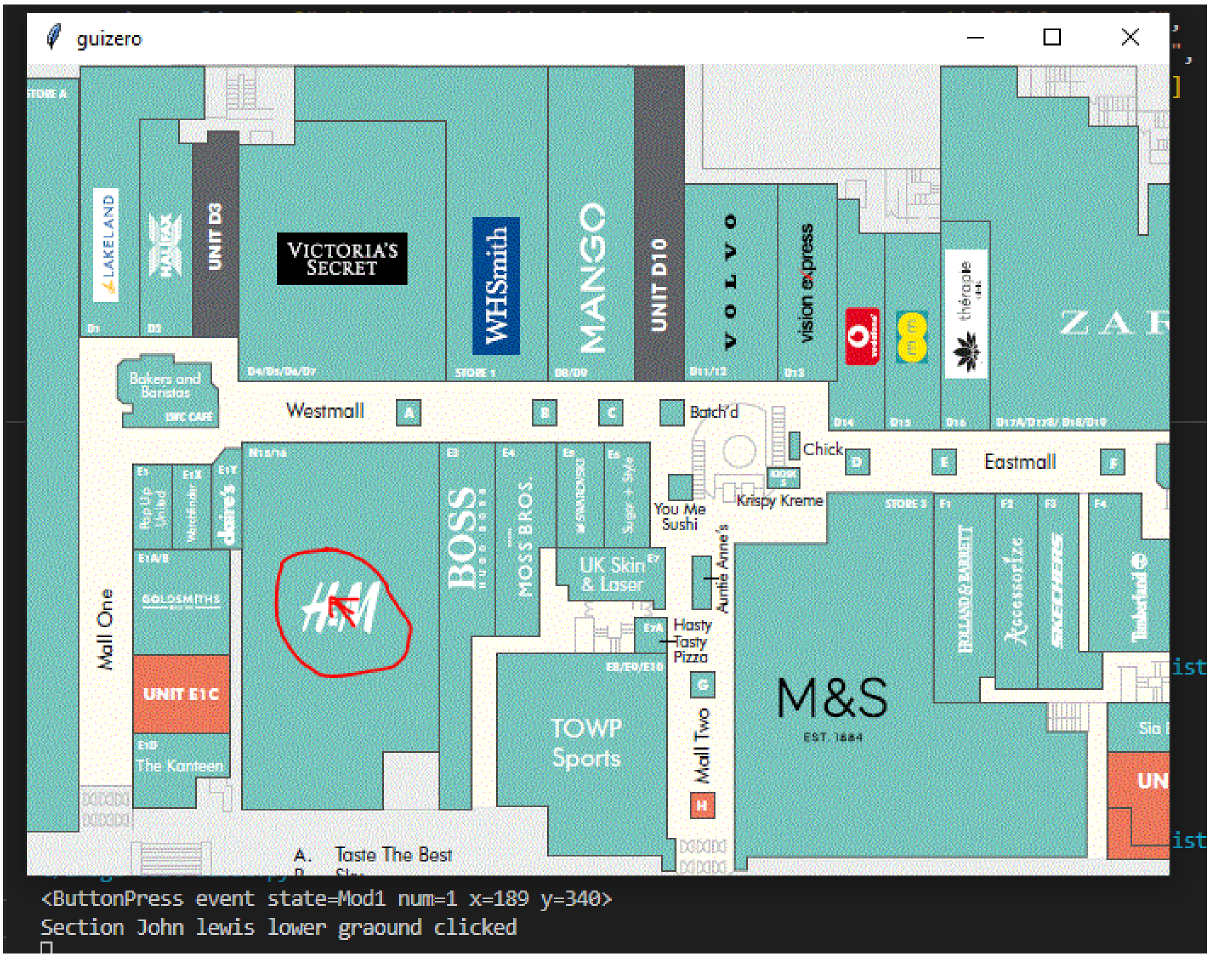
The “break” is used to exit the loop as soon as the matching region is found, and it is not necessary to check the other regions once matched. Here is the result this generate:



```
<ButtonPress event state=Mod1 num=1 x=190 y=476>
Section John lewis lower graound clicked
<ButtonPress event state=Mod1 num=1 x=419 y=497>
Section h&m clicked
------------------------------------------------------------
*** GUIZERO WARNING ***
Image resizing - cannot scale the image as PIL is not available.
------------------------------------------------------------
<ButtonPress event state=Mod1 num=1 x=263 y=519>
Section John lewis ground level clicked
<ButtonPress event state=Mod1 num=1 x=374 y=363>
Section Apple clicked
```

Although the code generates expected result, I find an issue.

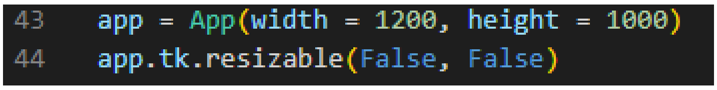
When the window resized, such as making smaller or bigger using double arrow pointer, the code generates completely different coordinates than before.



```
<ButtonPress event state=Mod1 num=1 x=189 y=340>
Section John lewis lower graound clicked
```

When the window is smaller, and “h&m” is clicked the program outputs “Section John lewis lower ground clicked”.

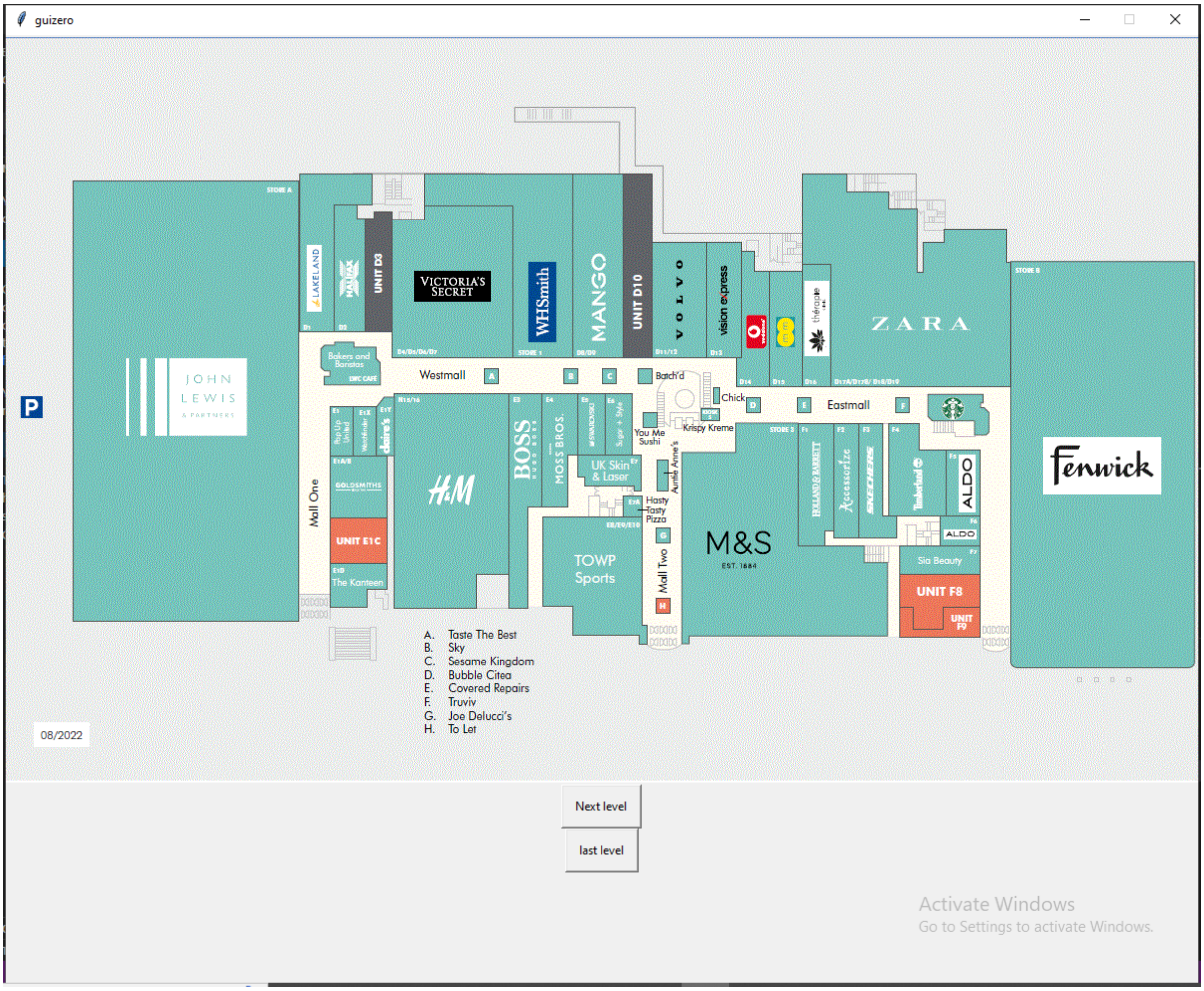
This is not effective as I do not want to change an object coordinate. I want the coordinates to be constant for each object. To do that I write following code:



```
app = App(width = 1200, height = 1000)
app.tk.resizable(False, False)
```

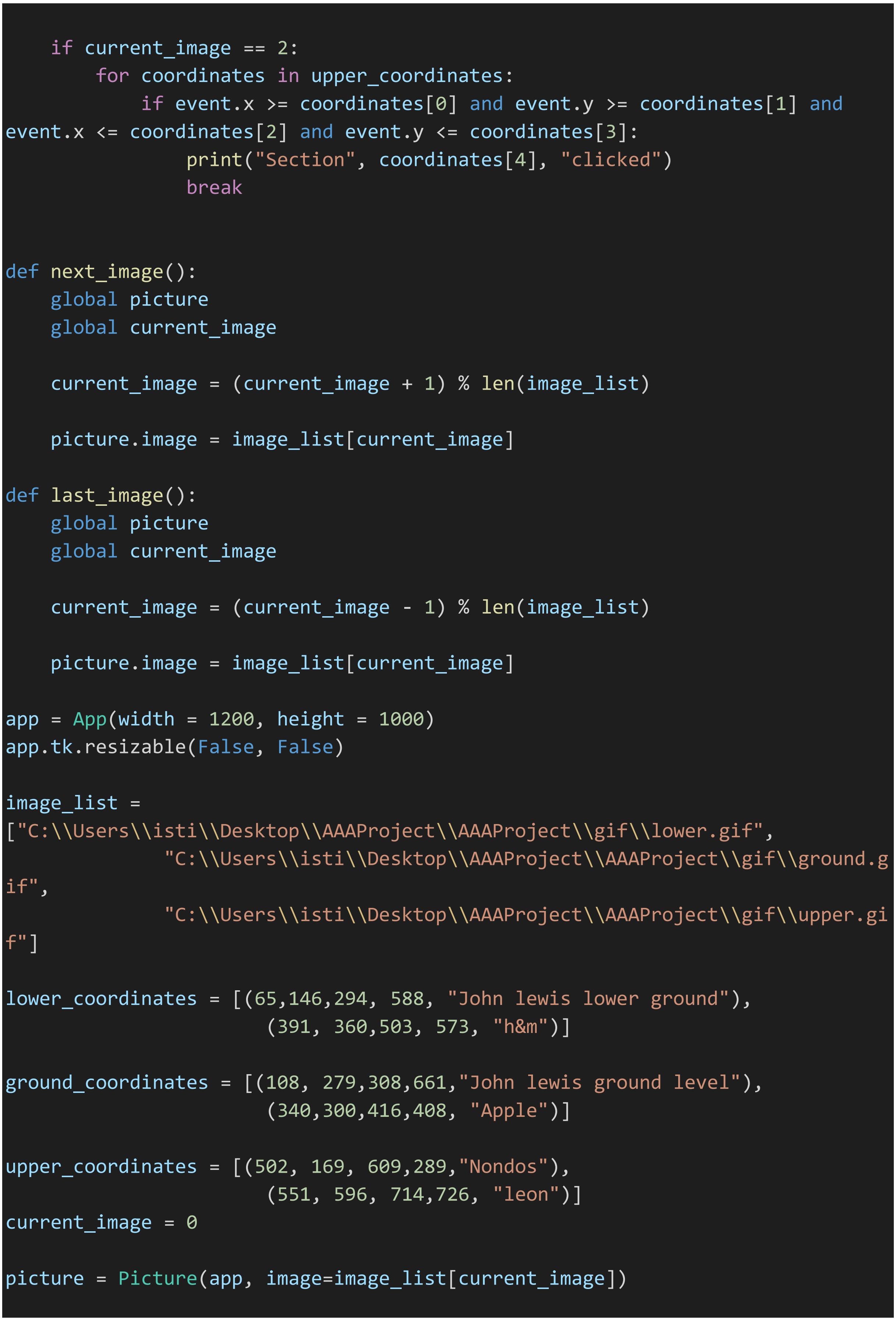
I use the “resizable” property from tkinter library which controls whether the user can resize the window by dragging its edges. I set the property of “resizable” to “false” for both width and height. So, the window cannot be resized.

It is useful for my case as I want a fixed window size to be sure that I get consistent coordinates for object. Here is the gui looks like:



Code in this stage

```
from guizero import App, Picture, PushButton, Window

def picture_click(event):
    print(event)

    #find out which part of the image has been clicked
    if current_image == 0:
        for coordinates in lower_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y <= coordinates[3]:
                 print("Section", coordinates[4], "clicked")
                 break

    if current_image == 1:
        for coordinates in ground_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y <= coordinates[3]:
                print("Section", coordinates[4], "clicked")
                break
```



```
    if current_image == 2:
        for coordinates in upper_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                print("Section", coordinates[4], "clicked")
                break


def next_image():
    global picture
    global current_image

    current_image = (current_image + 1) % len(image_list)

    picture.image = image_list[current_image]

def last_image():
    global picture
    global current_image

    current_image = (current_image - 1) % len(image_list)

    picture.image = image_list[current_image]

app = App(width = 1200, height = 1000)
app.tk.resizable(False, False)

image_list =
["C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\lower.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\ground.g
if",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\upper.gi
f"]

lower_coordinates = [(65,146,294, 588, "John lewis lower ground"),
                       (391, 360,503, 573, "h&m")]

ground_coordinates = [(108, 279,308,661,"John lewis ground level"),
                       (340,300,416,408, "Apple")]

upper_coordinates = [(502, 169, 609,289,"Nondos"),
                       (551, 596, 714,726, "leon")]
current_image = 0

picture = Picture(app, image=image_list[current_image])
```

```
picture.tk.bind('<Button-1>', picture_click)  # add an event handler for mouse clicks

#  add a button to change the picture
next_picture = PushButton(app, text="Next level", command=next_image)
last_picture = PushButton(app, text="last level", command=last_image)
# add route button
#route = PushButton(app, text="Route")
app.display()
```

WHAT HAS BEEN DONE

I have implemented the object coordinates for each floor and fix image resizing issue. The mouse click can be detect and compare by program, object name can be displayed. The coordinates are made consistent.

Main testing

Test No	Purpose	Data	Expected Results	Achieved?
1	see if the object position coordinates generate by mouse click	By clicking left mouse key outputs x1,x2 y1,y2	The terminal displays the output (x1, x2)(y1,y2) in first and second click	Yes When clicking: -Top left click generates (x1, x2) value. -Bottom left click generates (y1, y2 value.
2	Check the object display the name		The terminal should display the object name when clicked.	Yes Displays the name of the object when clicked.

SUCCESS CRITERIA MET

- Checking the object position by mouse click
- Finding the position of objects

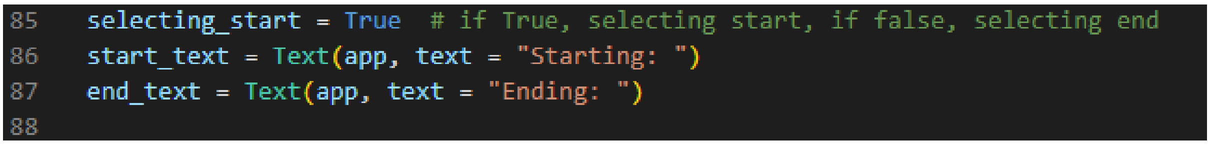
Iteration 3

In this iteration, I will focus on developing the following aspects of my success criteria:

-Creating label to display the starting and ending position.

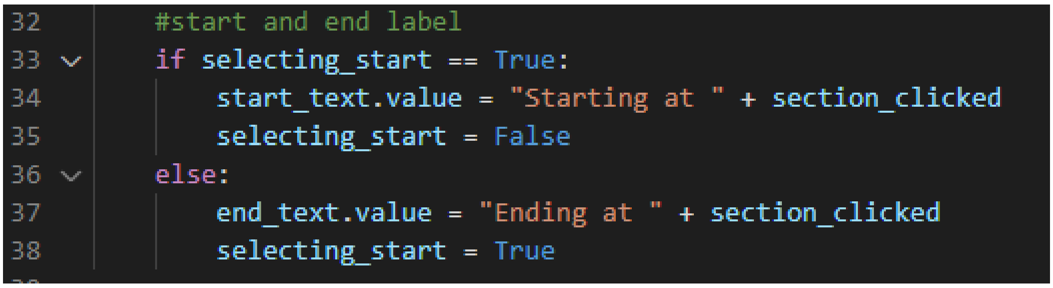
-Creating route button.

To do this I wrote this code:



```
selecting_start = True  # if True, selecting start, if false, selecting end
start_text = Text(app, text = "Starting: ")
end_text = Text(app, text = "Ending: ")
```

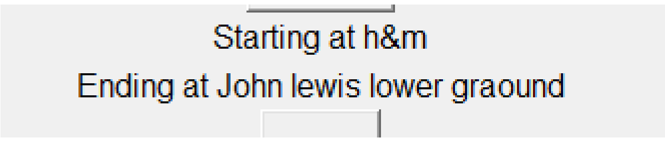
The variable selecting_start is a boolean variable that is set to True. It is currently processing of selecting a start point, as opposed to selecting an end point. The next two lines of code define two text objects using the Text() function from the guizero library. These text objects are being added to the app object, which is the main window of the GUI. The first text object is labeled "Starting: " and the second text object is labeled "Ending: ". These text objects is used to display information such as the starting and ending position to the user.



```
    #start and end label
    if selecting_start == True:
        start_text.value = "Starting at " + section_clicked
        selecting_start = False
    else:
        end_text.value = "Ending at " + section_clicked
        selecting_start = True
```

The code first checks the value of the boolean variable selecting_start. If it is True, then the code updates the text of the start text object to display the string "Starting at" concatenated with the value of the section clicked variable. It then sets selecting_start to False. If selecting_start is not False, then the code updates the text of the end text object to display the string "Ending at" concatenated with the value of section_clicked. It then sets selecting_start back to True. So, this code switches between selecting a starting point and an ending point for a section of map. Each time the user clicks on a section, the code switches between updating the start text and end text objects and toggling the value of selecting_start.

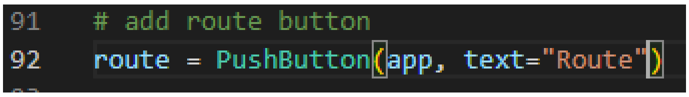
The code output is here:



```
Starting at h&m
Ending at John lewis lower graound
```

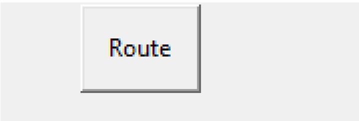
It generates output as I wanted and switches between the position when click.

To add the route button, I wrote this code:



```python
# add route button
route = PushButton(app, text="Route")
```

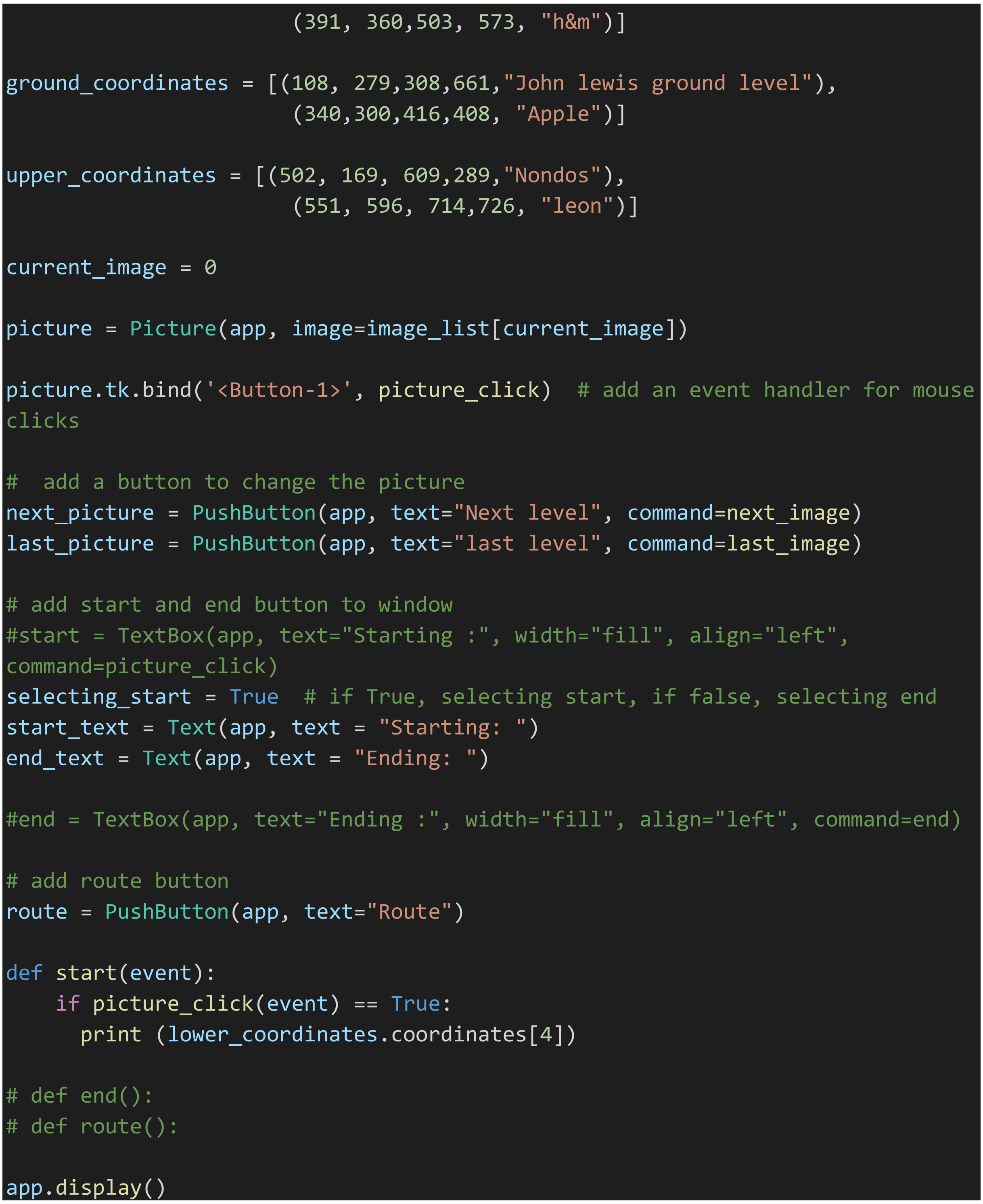
It is similar to other button. However, I did not add any command to it because that can be useful after implementing the algorithm. so I keep It simple and in current situation it only displays the button but functionally does not do anything which I will be adding when the algorithm is in place. Here is it appears:



Code in this stage

```python
from guizero import App, Picture, PushButton, Window, Text

def picture_click(event):
    global start_text, end_text
    global selecting_start

    print(event)

    section_clicked = ""
    #find out which part of the image has been clicked
    if current_image == 0:
        for coordinates in lower_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break

    if current_image == 1:
        for coordinates in ground_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
```

```
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break


    if current_image == 2:
        for coordinates in upper_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break
    #start and end label
    if selecting_start == True:
        start_text.value = "Starting at " + section_clicked
        selecting_start = False
    else:
        end_text.value = "Ending at " + section_clicked
        selecting_start = True

def next_image():
    global picture
    global current_image

    current_image = (current_image + 1) % len(image_list)

    picture.image = image_list[current_image]

def last_image():
    global picture
    global current_image

    current_image = (current_image - 1) % len(image_list)

    picture.image = image_list[current_image]

app = App(width = 1200, height = 1000)
app.tk.resizable(False, False)

image_list =
["C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\lower.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\ground.g
if",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\upper.gi
f"]

# store coordinates as tuples where containing x1, y1, x2, y2, name
lower_coordinates = [(65,146,294, 588, "John lewis lower graound"),
```



```
                       (391, 360,503, 573, "h&m")]

ground_coordinates = [(108, 279,308,661,"John lewis ground level"),
                       (340,300,416,408, "Apple")]

upper_coordinates = [(502, 169, 609,289,"Nondos"),
                       (551, 596, 714,726, "leon")]

current_image = 0

picture = Picture(app, image=image_list[current_image])

picture.tk.bind('<Button-1>', picture_click)  # add an event handler for mouse
clicks

#  add a button to change the picture
next_picture = PushButton(app, text="Next level", command=next_image)
last_picture = PushButton(app, text="last level", command=last_image)

# add start and end button to window
#start = TextBox(app, text="Starting :", width="fill", align="left",
command=picture_click)
selecting_start = True  # if True, selecting start, if false, selecting end
start_text = Text(app, text = "Starting: ")
end_text = Text(app, text = "Ending: ")

#end = TextBox(app, text="Ending :", width="fill", align="left", command=end)

# add route button
route = PushButton(app, text="Route")

def start(event):
    if picture_click(event) == True:
      print (lower_coordinates.coordinates[4])

# def end():
# def route():

app.display()
```

WHAT HAS BEEN DONE

I have implemented the labels to view starting and ending position easily. The mouse click switches the label value according to boolean value. The “Route” button is in place which will be useful later for implementing algorithm.

Main testing

Test No	Purpose	Data	Expected Results	Achieved?
1	Check if the name is displayed after mouse click	Clicking left mouse button ('<Button-1>')	The screen should display name accordingly to the key pressed. I.e. The ('<Button-1>') value display accordingly	YES. The starting and ending name is displayed properly.
2	See if the router button is displaying		The screen should display “Router” button.	Yes The “Router” button is displayed correctly.

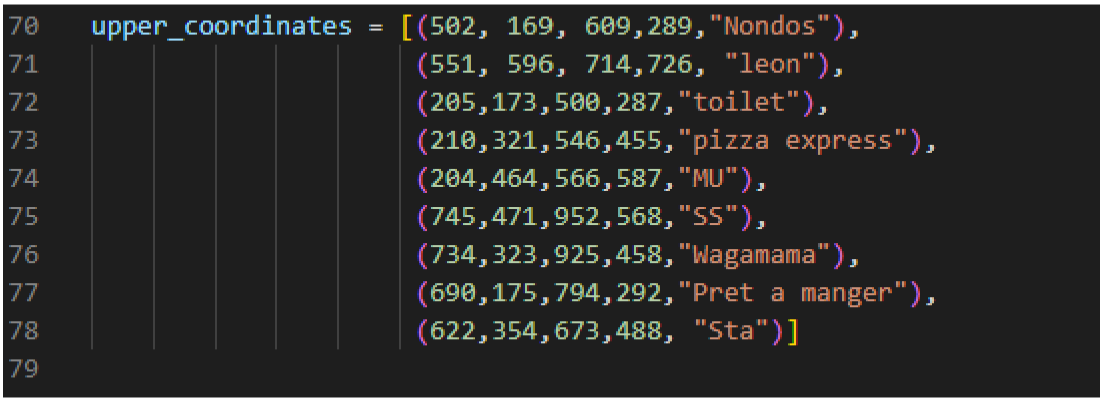
SUCCESS CRITERIA MET

- Users can view the starting and ending position.
- Router button to find the path between two positions.
- The map can display the destination.

Iteration 4

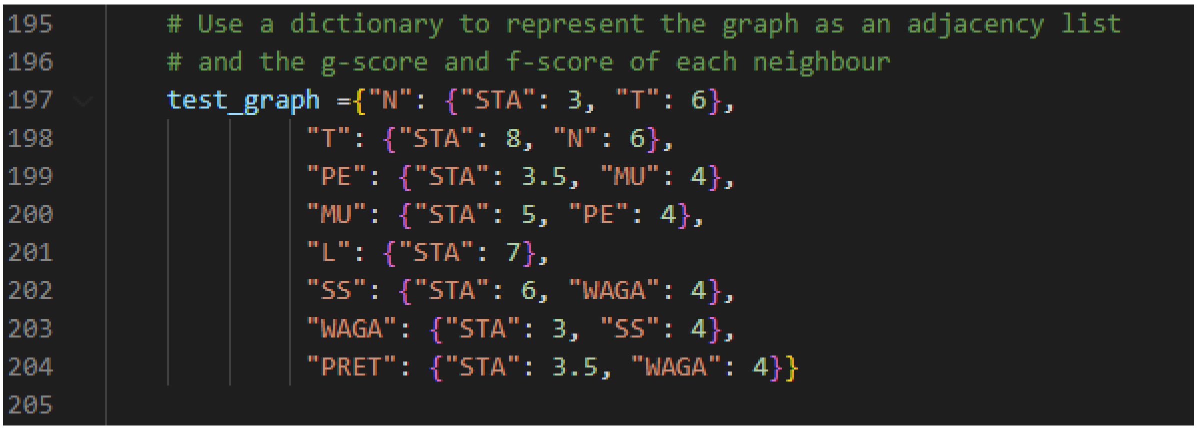
In this iteration, I will focus on developing the following aspects of my success criteria:
-linking the A* algorithm with the GUI

Because I am using the algorithm from Isaac computer science, I will only have to link it with gui and pass the inputs. I do need to create adjacency list for the map.
For initial test I select upper floor as it has less object compare to other 2 floor.
Here are the coordinates I am using for object:



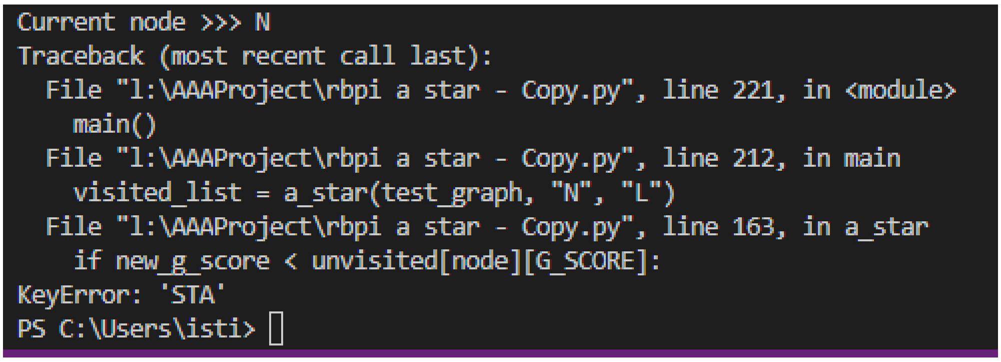
```
upper_coordinates = [(502, 169, 609,289,"Nondos"),
                     (551, 596, 714,726, "leon"),
                     (205,173,500,287,"toilet"),
                     (210,321,546,455,"pizza express"),
                     (204,464,566,587,"MU"),
                     (745,471,952,568,"SS"),
                     (734,323,925,458,"Wagamama"),
                     (690,175,794,292,"Pret a manger"),
                     (622,354,673,488, "Sta")]
```

I also build a adjacency list which the algorithm will be using to find the shortest path.



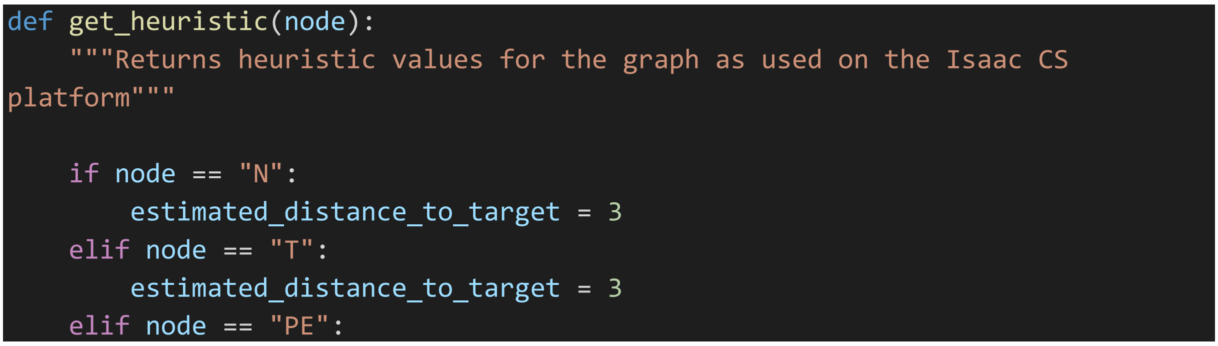
```
    # Use a dictionary to represent the graph as an adjacency list
    # and the g-score and f-score of each neighbour
    test_graph ={"N": {"STA": 3, "T": 6},
                 "T": {"STA": 8, "N": 6},
                 "PE": {"STA": 3.5, "MU": 4},
                 "MU": {"STA": 5, "PE": 4},
                 "L": {"STA": 7},
                 "SS": {"STA": 6, "WAGA": 4},
                 "WAGA": {"STA": 3, "SS": 4},
                 "PRET": {"STA": 3.5, "WAGA": 4}}
```

Which generates an error massage:

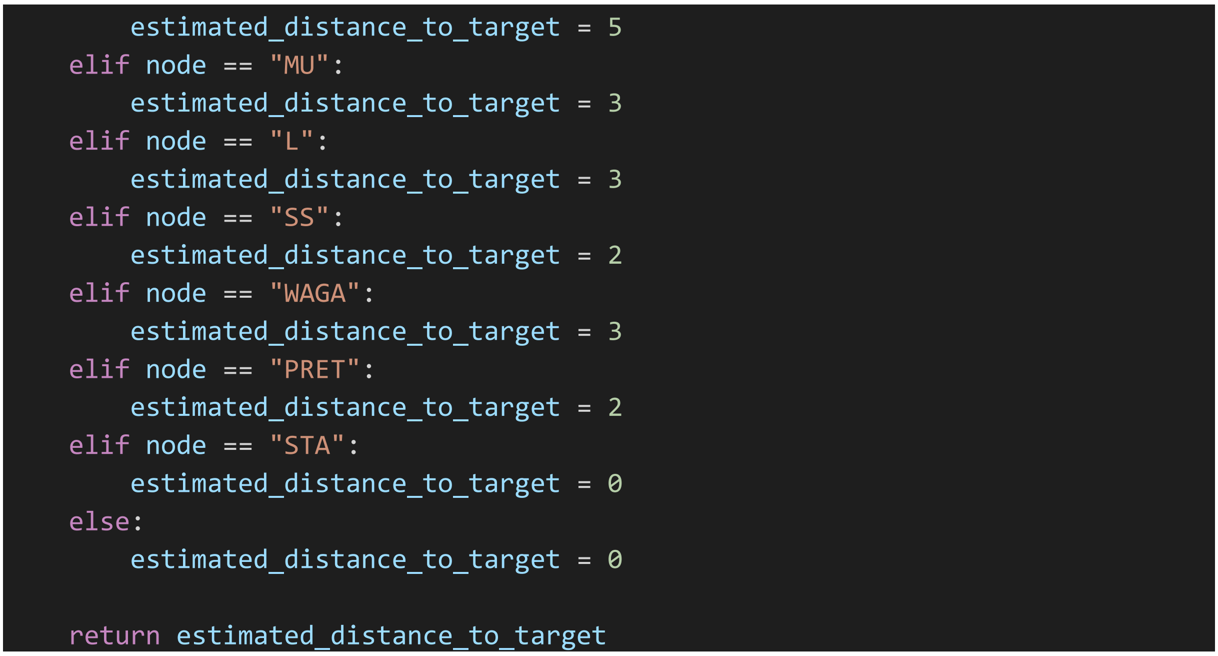


```
Current node >>> N
Traceback (most recent call last):
  File "l:\AAAProject\rbpi a star - Copy.py", line 221, in <module>
    main()
  File "l:\AAAProject\rbpi a star - Copy.py", line 212, in main
    visited_list = a_star(test_graph, "N", "L")
  File "l:\AAAProject\rbpi a star - Copy.py", line 163, in a_star
    if new_g_score < unvisited[node][G_SCORE]:
KeyError: 'STA'
PS C:\Users\isti> 
```

The issue I found is that the heuristic value of the algorithm is too high. In A*, the heuristic value cannot be overestimated. So I lower the value for each object:



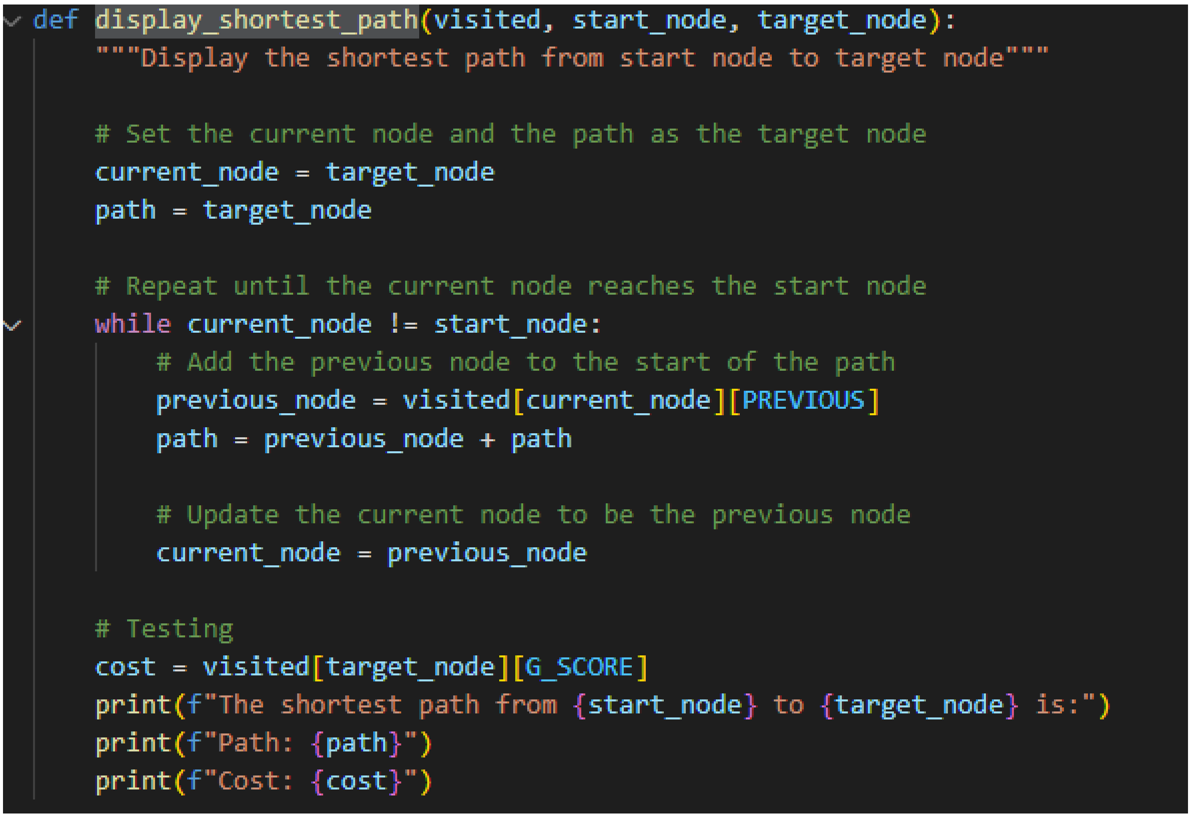
```
def get_heuristic(node):
    """Returns heuristic values for the graph as used on the Isaac CS 
platform"""

    if node == "N":
        estimated_distance_to_target = 3
    elif node == "T":
        estimated_distance_to_target = 3
    elif node == "PE":
```



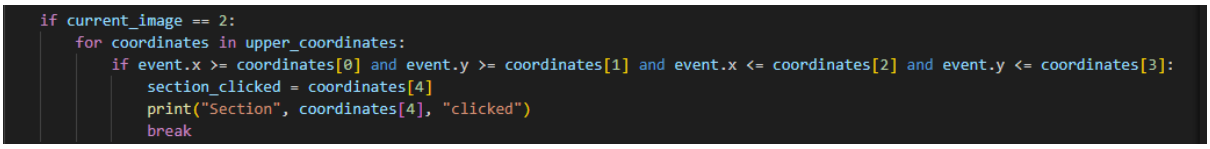
```
        estimated_distance_to_target = 5
    elif node == "MU":
        estimated_distance_to_target = 3
    elif node == "L":
        estimated_distance_to_target = 3
    elif node == "SS":
        estimated_distance_to_target = 2
    elif node == "WAGA":
        estimated_distance_to_target = 3
    elif node == "PRET":
        estimated_distance_to_target = 2
    elif node == "STA":
        estimated_distance_to_target = 0
    else:
        estimated_distance_to_target = 0

    return estimated_distance_to_target
```

although I lower the heuristic value the code still giving me same error as above.

To move on I pass the start_value and target_value variable to the gui label so the input value can be passed through algorithm.

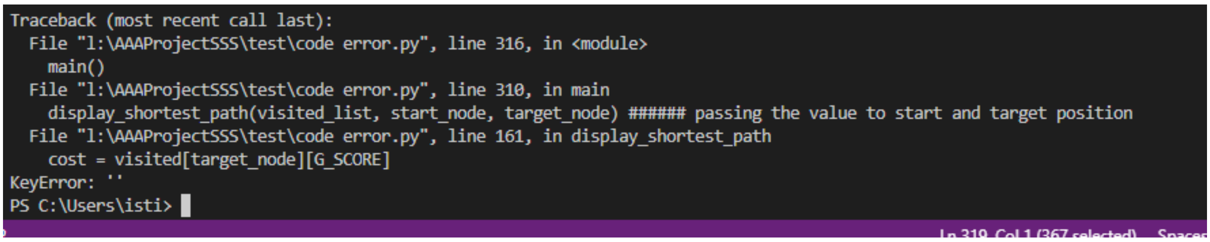


```
def display_shortest_path(visited, start_node, target_node):
    """Display the shortest path from start node to target node"""

    # Set the current node and the path as the target node
    current_node = target_node
    path = target_node

    # Repeat until the current node reaches the start node
    while current_node != start_node:
        # Add the previous node to the start of the path
        previous_node = visited[current_node][PREVIOUS]
        path = previous_node + path

        # Update the current node to be the previous node
        current_node = previous_node

    # Testing
    cost = visited[target_node][G_SCORE]
    print(f"The shortest path from {start_node} to {target_node} is:")
    print(f"Path: {path}")
    print(f"Cost: {cost}")
```



```
if current_image == 2:
    for coordinates in upper_coordinates:
        if event.x >= coordinates[0] and event.y >= coordinates[1] and event.x <= coordinates[2] and event.y <= coordinates[3]:
            section_clicked = coordinates[4]
            print("Section", coordinates[4], "clicked")
            break
```

I try to use the global variable for coordinates [4] which then pass through display_shortest_path function as parameter. But unable to do so. As the program cashes and giving error massage.



```
Traceback (most recent call last):
  File "l:\AAAProjectSSS\test\code error.py", line 316, in <module>
    main()
  File "l:\AAAProjectSSS\test\code error.py", line 310, in main
    display_shortest_path(visited_list, start_node, target_node) ###### passing the value to start and target position
  File "l:\AAAProjectSSS\test\code error.py", line 161, in display_shortest_path
    cost = visited[target_node][G_SCORE]
KeyError: ''
PS C:\Users\isti> 
```

This can be focused on further development.

Main testing

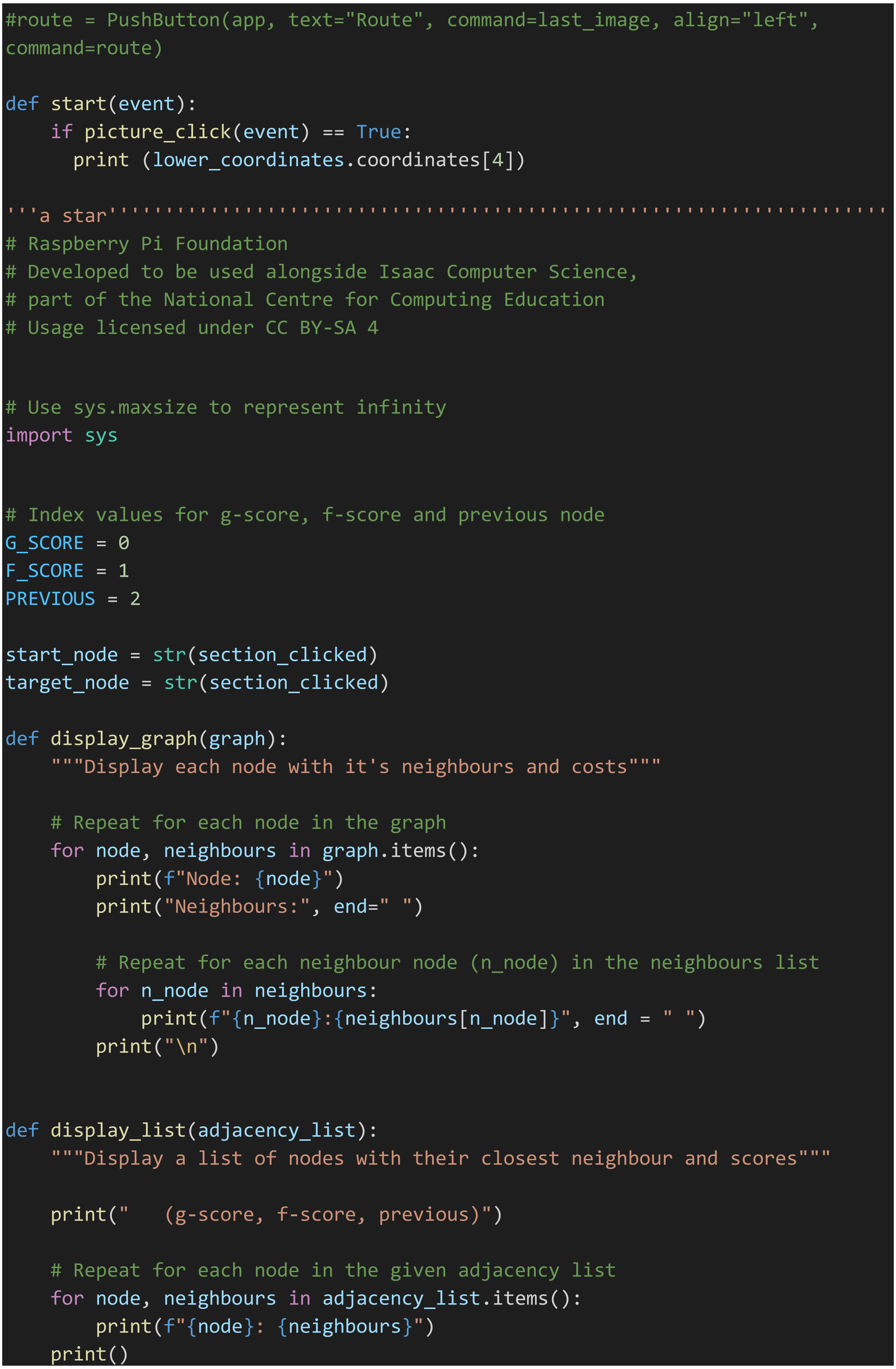
Test No	Purpose	Data	Expected Results	Achieved?
1	see if the gui is passing the input to algorithm	coordinates [4]	The terminal will displays the value of the coordinates [4]	No When running it generates error massage.
2	Check the shortest path has been found		The terminal should display the shortest path between start and end.	No When running it generates error massage.

SUCCESS CRITERIA MET

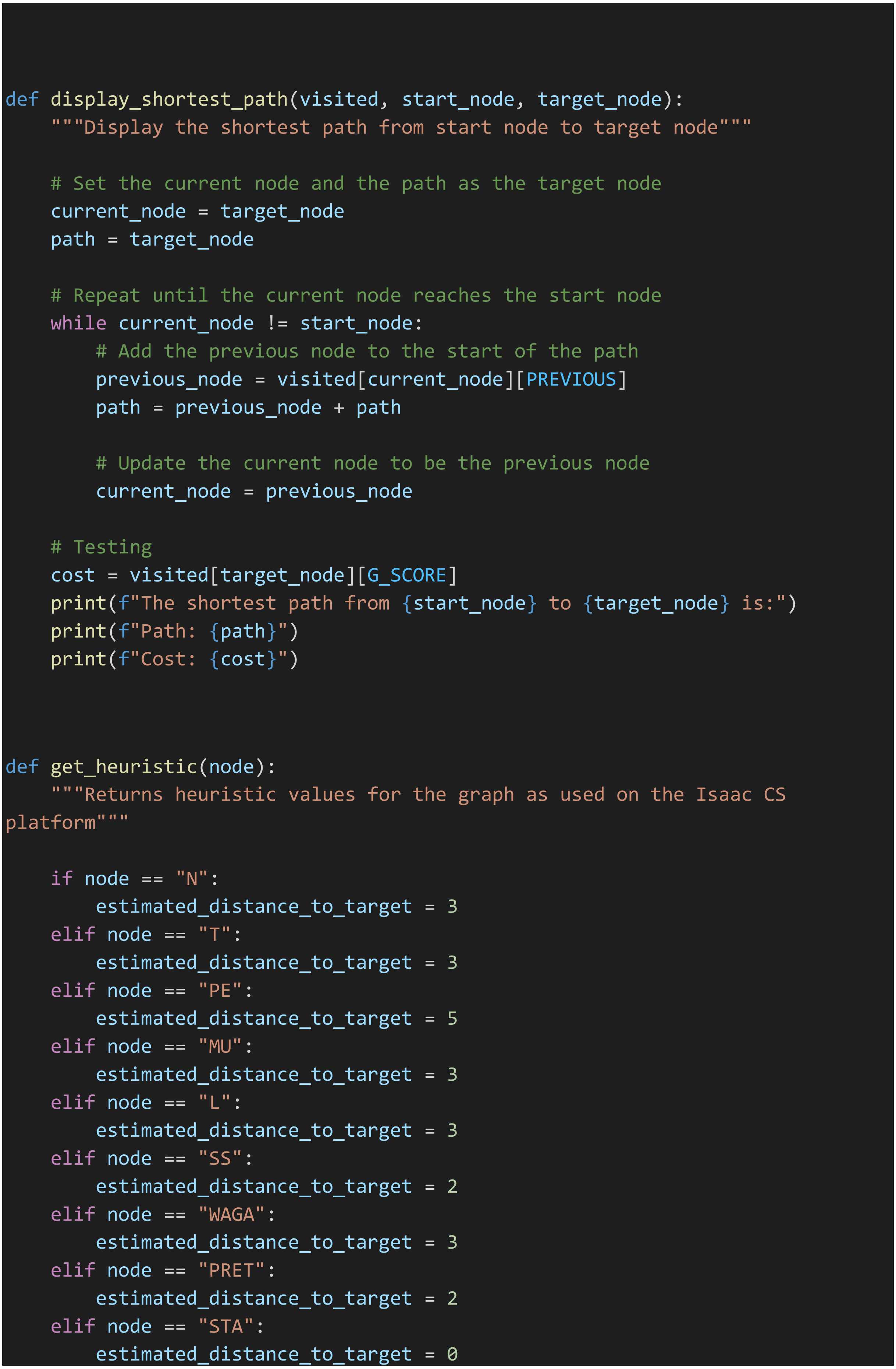
No SUCCESS CRITERIA is met in this iteration as the code produce error massage.

Full code

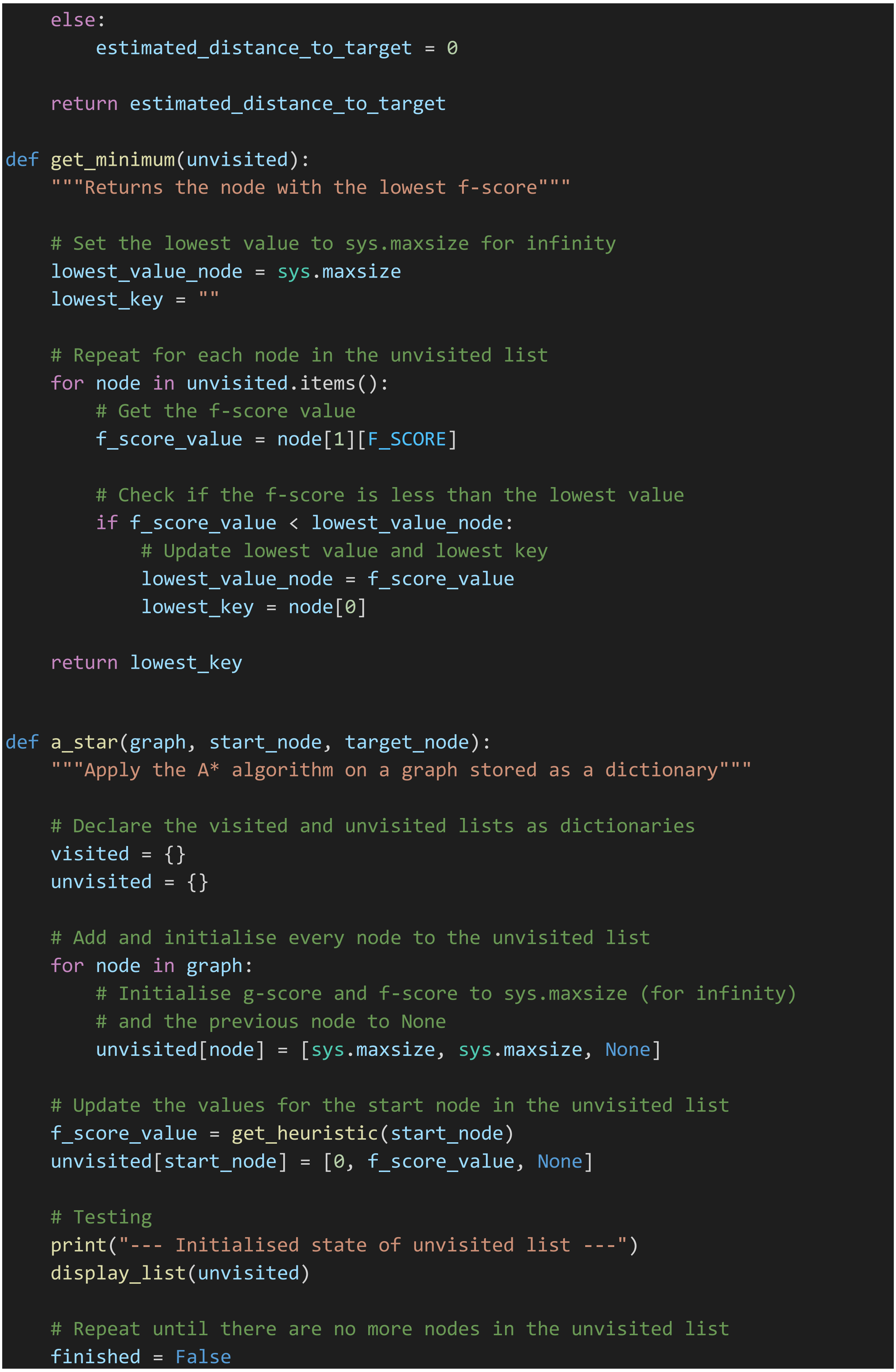
```
from guizero import App, Picture, PushButton, Window, Text
section_clicked = ""
def picture_click(event):
    global start_text, end_text
    global selecting_start
    global section_clicked ################using section_clicked to pass onto
start_node and target_node'''

    print(event)


    #find out which part of the image has been clicked
    if current_image == 0:
        for coordinates in lower_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break

    if current_image == 1:
        for coordinates in ground_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break


    if current_image == 2:
        for coordinates in upper_coordinates:
            if event.x >= coordinates[0] and event.y >= coordinates[1] and
event.x <= coordinates[2] and event.y <= coordinates[3]:
                section_clicked = coordinates[4]
                print("Section", coordinates[4], "clicked")
                break
    #start and end label
    if selecting_start == True:
        start_text.value = "Starting at " + section_clicked
        selecting_start = False
    else:
        end_text.value = "Ending at " + section_clicked
        selecting_start = True

def next_image():
    global picture
    global current_image

    current_image = (current_image + 1) % len(image_list)

    picture.image = image_list[current_image]

def last_image():
    global picture
    global current_image

    current_image = (current_image - 1) % len(image_list)

    picture.image = image_list[current_image]

app = App(width = 1200, height = 1000)
app.tk.resizable(False, False)

image_list =
["C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\lower.gif",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\ground.g
if",
              "C:\\Users\\isti\\Desktop\\AAAProject\\AAAProject\\gif\\upper.gi
f"]

# store coordinates as tuples where containing x1, y1, x2, y2, name
lower_coordinates = [(65,146,294, 588, "John lewis lower graound"),
                       (391, 360,503, 573, "h&m")]

ground_coordinates = [(108, 279,308,661,"John lewis ground level"),
                       (340,300,416,408, "Apple")]

upper_coordinates = [(502, 169, 609,289,"Nondos"),
                       (551, 596, 714,726, "leon")]

current_image = 0

picture = Picture(app, image=image_list[current_image])

picture.tk.bind('<Button-1>', picture_click)  # add an event handler for mouse
clicks

#  add a button to change the picture
next_picture = PushButton(app, text="Next level", command=next_image)
last_picture = PushButton(app, text="last level", command=last_image)

# add start and end button to window
#start = TextBox(app, text="Starting :", width="fill", align="left",
command=picture_click)
selecting_start = True  # if True, selecting start, if false, selecting end
start_text = Text(app, text = "Starting: ")
end_text = Text(app, text = "Ending: ")

#end = TextBox(app, text="Ending :", width="fill", align="left", command=end)

# add route button
```



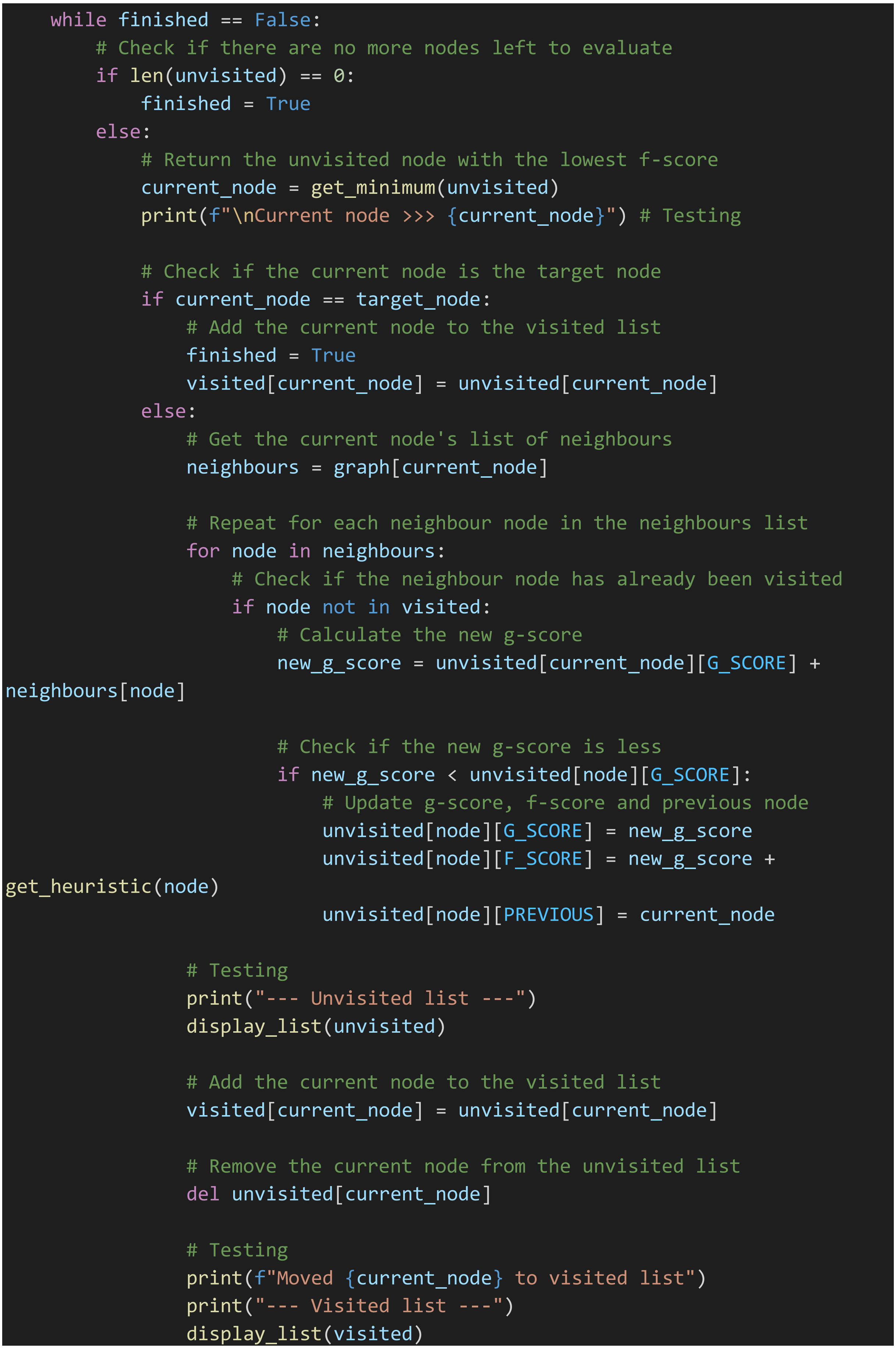
```
#route = PushButton(app, text="Route", command=last_image, align="left",
command=route)

def start(event):
    if picture_click(event) == True:
      print (lower_coordinates.coordinates[4])

'''a star'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# Raspberry Pi Foundation
# Developed to be used alongside Isaac Computer Science,
# part of the National Centre for Computing Education
# Usage licensed under CC BY-SA 4


# Use sys.maxsize to represent infinity
import sys


# Index values for g-score, f-score and previous node
G_SCORE = 0
F_SCORE = 1
PREVIOUS = 2

start_node = str(section_clicked)
target_node = str(section_clicked)

def display_graph(graph):
    """Display each node with it's neighbours and costs"""

    # Repeat for each node in the graph
    for node, neighbours in graph.items():
        print(f"Node: {node}")
        print("Neighbours:", end=" ")

        # Repeat for each neighbour node (n_node) in the neighbours list
        for n_node in neighbours:
            print(f"{n_node}:{neighbours[n_node]}", end = " ")
        print("\n")


def display_list(adjacency_list):
    """Display a list of nodes with their closest neighbour and scores"""

    print("   (g-score, f-score, previous)")

    # Repeat for each node in the given adjacency list
    for node, neighbours in adjacency_list.items():
        print(f"{node}: {neighbours}")
    print()
```



```
def display_shortest_path(visited, start_node, target_node):
    """Display the shortest path from start node to target node"""

    # Set the current node and the path as the target node
    current_node = target_node
    path = target_node

    # Repeat until the current node reaches the start node
    while current_node != start_node:
        # Add the previous node to the start of the path
        previous_node = visited[current_node][PREVIOUS]
        path = previous_node + path

        # Update the current node to be the previous node
        current_node = previous_node

    # Testing
    cost = visited[target_node][G_SCORE]
    print(f"The shortest path from {start_node} to {target_node} is:")
    print(f"Path: {path}")
    print(f"Cost: {cost}")



def get_heuristic(node):
    """Returns heuristic values for the graph as used on the Isaac CS 
platform"""

    if node == "N":
        estimated_distance_to_target = 3
    elif node == "T":
        estimated_distance_to_target = 3
    elif node == "PE":
        estimated_distance_to_target = 5
    elif node == "MU":
        estimated_distance_to_target = 3
    elif node == "L":
        estimated_distance_to_target = 3
    elif node == "SS":
        estimated_distance_to_target = 2
    elif node == "WAGA":
        estimated_distance_to_target = 3
    elif node == "PRET":
        estimated_distance_to_target = 2
    elif node == "STA":
        estimated_distance_to_target = 0
```

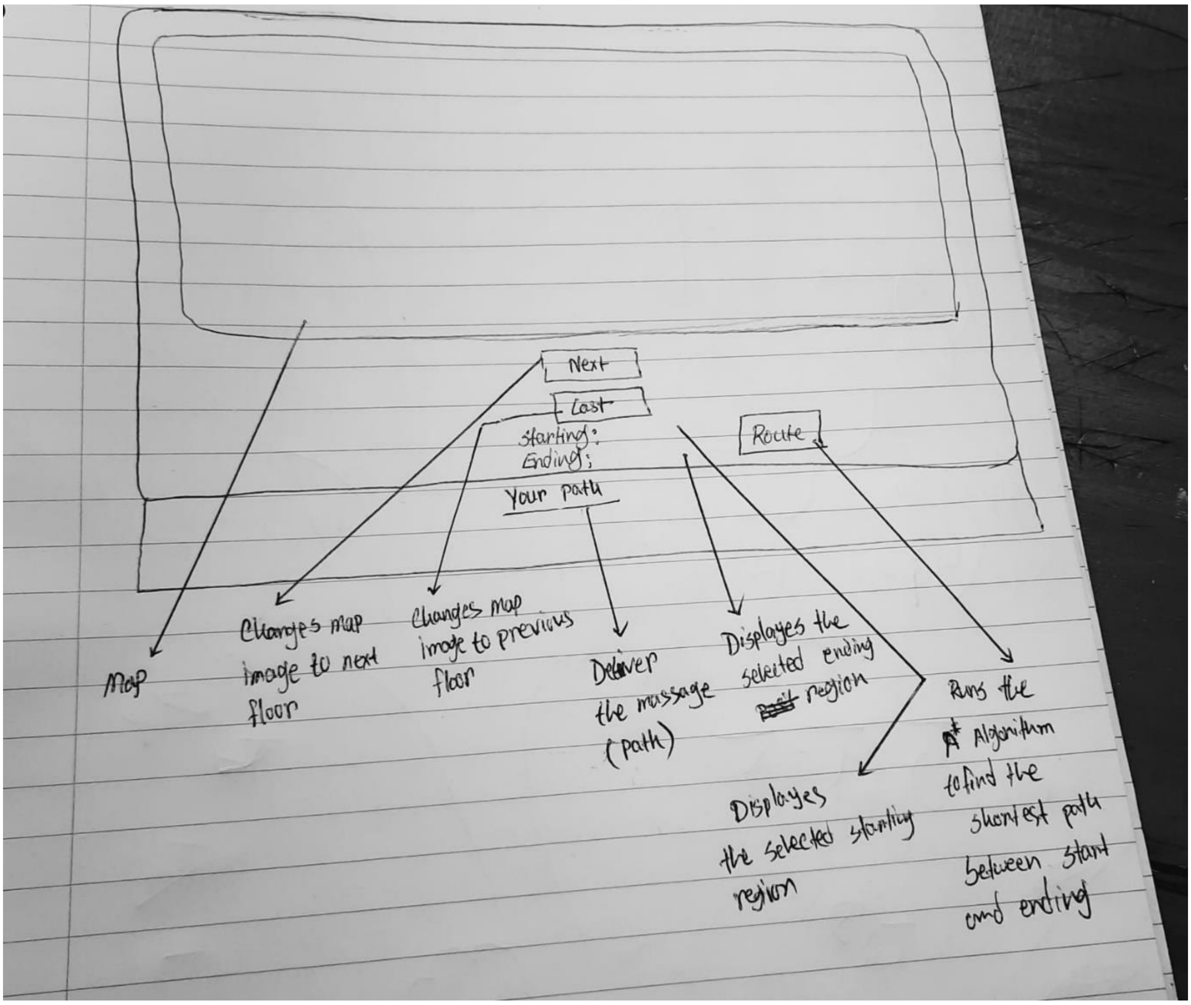


```
    else:
        estimated_distance_to_target = 0

    return estimated_distance_to_target

def get_minimum(unvisited):
    """Returns the node with the lowest f-score"""

    # Set the lowest value to sys.maxsize for infinity
    lowest_value_node = sys.maxsize
    lowest_key = ""

    # Repeat for each node in the unvisited list
    for node in unvisited.items():
        # Get the f-score value
        f_score_value = node[1][F_SCORE]

        # Check if the f-score is less than the lowest value
        if f_score_value < lowest_value_node:
            # Update lowest value and lowest key
            lowest_value_node = f_score_value
            lowest_key = node[0]

    return lowest_key


def a_star(graph, start_node, target_node):
    """Apply the A* algorithm on a graph stored as a dictionary"""

    # Declare the visited and unvisited lists as dictionaries
    visited = {}
    unvisited = {}

    # Add and initialise every node to the unvisited list
    for node in graph:
        # Initialise g-score and f-score to sys.maxsize (for infinity)
        # and the previous node to None
        unvisited[node] = [sys.maxsize, sys.maxsize, None]

    # Update the values for the start node in the unvisited list
    f_score_value = get_heuristic(start_node)
    unvisited[start_node] = [0, f_score_value, None]

    # Testing
    print("--- Initialised state of unvisited list ---")
    display_list(unvisited)

    # Repeat until there are no more nodes in the unvisited list
    finished = False
```



```
    while finished == False:
        # Check if there are no more nodes left to evaluate
        if len(unvisited) == 0:
            finished = True
        else:
            # Return the unvisited node with the lowest f-score
            current_node = get_minimum(unvisited)
            print(f"\nCurrent node >>> {current_node}") # Testing

            # Check if the current node is the target node
            if current_node == target_node:
                # Add the current node to the visited list
                finished = True
                visited[current_node] = unvisited[current_node]
            else:
                # Get the current node's list of neighbours
                neighbours = graph[current_node]

                # Repeat for each neighbour node in the neighbours list
                for node in neighbours:
                    # Check if the neighbour node has already been visited
                    if node not in visited:
                        # Calculate the new g-score
                        new_g_score = unvisited[current_node][G_SCORE] +
neighbours[node]

                        # Check if the new g-score is less
                        if new_g_score < unvisited[node][G_SCORE]:
                            # Update g-score, f-score and previous node
                            unvisited[node][G_SCORE] = new_g_score
                            unvisited[node][F_SCORE] = new_g_score +
get_heuristic(node)
                            unvisited[node][PREVIOUS] = current_node

                # Testing
                print("--- Unvisited list ---")
                display_list(unvisited)

                # Add the current node to the visited list
                visited[current_node] = unvisited[current_node]

                # Remove the current node from the unvisited list
                del unvisited[current_node]

                # Testing
                print(f"Moved {current_node} to visited list")
                print("--- Visited list ---")
                display_list(visited)
    # Testing
    print("--- Visited list ---")
    display_list(visited)

    # Return the final visited list
    return visited


def main():
    """Perform the A* algorithm on the test data"""

    # Use a dictionary to represent the graph as an adjacency list
    # and the g-score and f-score of each neighbour
    test_graph ={"N": {"STA": 3, "T": 6},
             "T": {"STA": 3, "N": 6},
             "PE": {"STA": 3, "MU": 4},
             "MU": {"STA": 5, "PE": 4},
             "L": {"STA": 4},
             "SS": {"STA": 6, "WAGA": 4},
             "WAGA": {"STA": 3, "SS": 4},
             "PRET": {"STA": 3.5, "WAGA": 4}}


    print("### A* algorithm ###")
    print("\nDisplay graph: \n")
    display_graph(test_graph)

    # Perform the A* algorithm between nodes A and F
    visited_list = a_star(test_graph, "B", "F")

    # Display the shortest path using the visited list
    display_shortest_path(visited_list, start_node, target_node) ######
passing the value to start and target position


# This code will run if this file is executed directly
# (i.e. not called by another program)
if __name__ == '__main__':
    main()
'''a star'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
app.display()
```

Evaluation


Next
Last
Starting:
Ending:
Your path
Route
Map
Changes map image to next floor
Changes map image to previous floor
Deliver the massage (path)
Displayes the selected ending region
Displayes the selected starting region
Runs the A* Algorithm to find the shortest path between start and ending

This is the map menu which will be displayed when user starts the program. The design is simple and consist of all the features at one window. It shows user the map image and has button to change between floors. User can pick the desire starting and ending position by mouse click. As soon as any starting or ending position is selected it will display on the screen. Once the starting and ending position is selected, the “Route” button will run the a* algorithm to find the shortest path of the route. The path will display on “Your path” section.

<u>Success Criteria</u>

Criteria	**Completed?**
Simple main menu	Yes
Search tab and button	No
Displaying the map	Yes

Maps and objects placed correctly	Yes
checking the object position by mouse click	Yes
Finding the position of objects	Yes
The images can be change by using button.	Yes
Users can view the starting and ending position	Yes
Users can search for a type of product (shoes, clothes etc.)	No
User can choose which mood to use (easy, moderate, quick)	No
The map can display the destination	Yes
The map can navigate to the destination	Yes
The map shows the time of the journey	No
Router button to find the path between two positions.	Yes

Review of Stakeholders Feedback

The feedback on the map was mostly positive, with some minor bugs needing to be fixed. The design was generally well-received, but some stakeholders prefer more colours. The buttons are easily noticeable to inexperienced users, so the user experience is somewhat quick. Some stakeholders prefer larger button and colours but to keep it simple I would not make any changes. Overall, the game was well-received, and while there are some fixes to be made, it is suitable for most users.

Limitations

-Specific location: In current situation the map can only be function in one location.
-Portability: The map is not very portable. And user have to use it in a place where the device is available.
-Multiuser input- The system is not designed to take multiple user input at once. It is not ideal for the system take multiple inputs at once, this might crash the program. Essentially meaning that my map is only suitable for one individual use at once.
- Offline system: The map cannot be use on internet in this stage because there is no socket and database.

How to overcome them

To overcome these limitations, here is the possible steps I could take:

To make my map system more versatile, I can allow the user to input their location. This will enable the map to function in different locations and improve its usefulness. I can consider developing a mobile application or making it accessible through a web browser. This will allow users to access the map from any device with an internet connection and improve the portability of the system. I can implement a system for handling concurrent user inputs. This can involve using multithreading, multiprocessing, or other techniques to manage and prioritise user requests, to allow multiple users to use the map system simultaneously without crashing the program. To make the map system accessible online, I can implement this in a website. This will allow users to access the map from all devices.

Maintenance

The map system currently has limitations with portability, and data storage. To address these limitations, the map can be made more portable through development of a web or mobile version, updates can be added to provide new features, and data can be stored in a cloud service or external server for multiple users. The code is modular, which makes it easy for future developers to implement new features and test them independently.

Further Development

The algorithm can be linked to the gui. Further developments for the map can be done by using the cloud for storage, implementing a settings menu, adding a shop feature for product, and creating timer modes according to distance.

These developments would primarily depend on the functionality of the map. The timer is the most suitable and reliable feature to add in the short term. On the other hand, it can be building a platform where user only provides picture of the map and other features will be generated by system like coordinates, names by AI recognition.

Reference

GitHub. (2023). *isaac-code-samples*. [online] Available at: https://github.com/isaaccomputerscience/isaac-code-samples/blob/main/pathfinding/a-star/python/a_star.py [Accessed 4 Apr. 2023].