Task 2 – Find the closest neighbor

In this task you will learn concept of path planning, a technique used for find the shortest path between a source and destination. Path planning ensures that navigation is done in least time and in most optimized way, saving energy and providing a optimized way of the doing task.

Please find the *task2_main.py* file in the folder named *Task_Description*. Modify the *task2_main.py* to accomplish the problem description.

Given:

A set of test images, each containing

1. 10x10 grid, making 100 **squares**
2. *Obstacles* marked as **black square**
3. *Objects* defined by three features, viz. **Shape, Size and Color**

test_images are given as image files in the folder test_images

Figure 1 shows a sample test images. The squares are identified by the coordinate (x,y) where x is the column and y is the row to which the square belongs. Each square can be empty or have an Obstacle or have an Object.

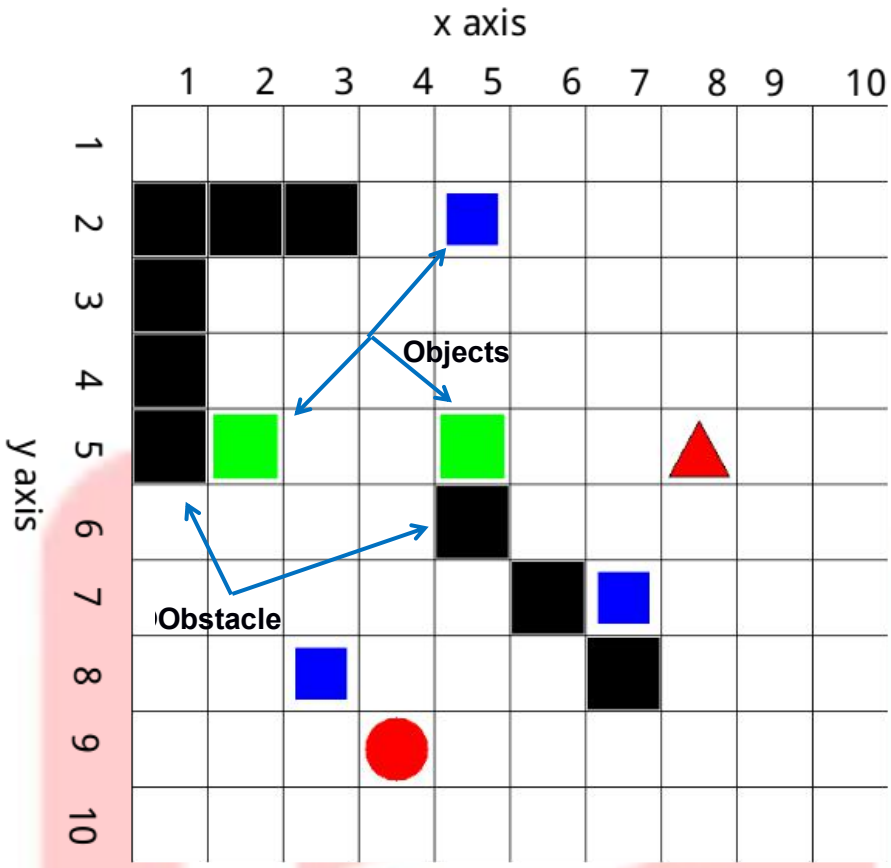

Figure 1: test_image

Problem Description:

For a given image, write a python program to solve the following:

1. Find the coordinates of occupied grid

Your code should return a *python list* having 'n' *python tuples, where 'n' denotes number of occupied grid in* test image. Grid is to be considered occupied if either grid has an *Obstacle* or an *Object*. Each tuple has two elements, first element is the x-coordinate of an *Obstacle/Object* and second element is the y-coordinate of an *Obstacle*.

Example: Considering Figure 1, the output should be:

[(1, 2), (1, 3), (1, 4), (1, 5), (2, 2), (2, 5), (3, 2), (3, 8), (4, 9), (5, 2), (5, 5), (5, 6), (6, 7), (7, 7), (7, 8), (8, 5)]

Note: List should be sorted according to x-coordinate followed by y-coordinate

2. For each object in the test_images, find a matching object which is **nearest** to it. *Object* is said to be nearest to another *Object*, if **length of path traversed** between two objects is **smallest**. Traversal is done by moving either **horizontally** or **vertically**. The length of the path is determined by the **number of moves** made during traversal.

Your code should return a *python* dictionary. Format for creating dictionary is as follows:

- Key for dictionary is a tuple - (x,y) coordinate of an **Object**
- first element of dictionary is a tuple - (x,y) coordinate of an **object nearest to it**
- second element is a **list of tuples** having (x,y) coordinate of all grids traversed i.e all route path
- third element of dictionary should be **number of moves** taken for traversal

For example, as shown in Figure 1, there is a green rectangle present at coordinate (2,5). The nearest matching object is present at coordinate (5,5).

- We have key for dictionary as **(2,5)** - the green rectangle
- First element of dictionary is **(5,5)** - the green rectangle closest to (2,5)
- Second element of dictionary - path of traversal. First grid traversed is (3,5) second grid traversed is (4,5). Output will be list of tuple **[(3,5), (4,5)]**
- Number of moves made are **three(03)**
  - (2,5) -> (3,5)
  - (3,5) -> (4,5)
  - (4,5) -> (5,5)

Dictionary structure* -- {(2,5): [(5,5), [(3,5), (4,5)], 3]}

Similarly, when we consider green rectangle present at coordinate (5,5). The nearest matching object is present at coordinate (2,5).

- We have key for dictionary as **(5,5)** - the green rectangle
- First element of dictionary is **(2,5)** - the green rectangle closest to (5,5)
- Second element of dictionary - path of traversal. First grid traversed is (4,5) second grid traversed is (3,5). Output will be list of tuple **[(4,5), (3,5)]**
- Number of moves made are **three(03)**
  - (5,5) -> (4,5)
  - (4,5) -> (3,5)
  - (3,5) -> (2,5)

Dictionary structure* -- {(5,5): [(2,5), [(4,5), (3,5)], 3]}

* For illustration only.

Actual output dictionary will be single dictionary with all elements. More detailed example is given below in the form of expected output.

As shown in Figure 2, there are multiple path/route to reach from one object to another. Paths between objects are marked in green and red line. Red line shows the available route for path traversal whereas green line show shortest path available for traversal. For blue square in grid (3,8), there are two paths to reach blue square in grid (7,7), these paths take 9 or 10 moves. However, both these paths are not the shortest as we have another blue square in grid (5,2) which can be reached in only 8 moves.

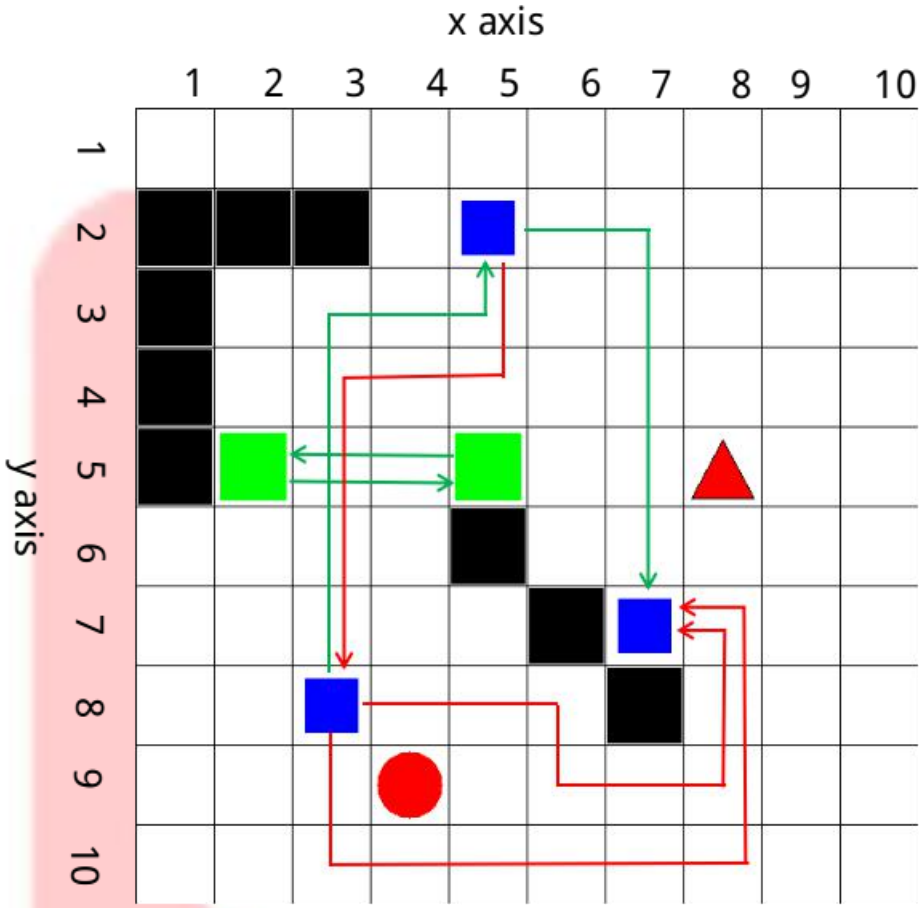

Figure 2: test_image

Example: Expected output (with reference to Figure 2)

{‘(2,5)’: [(5,5), [(3,5), (4,5)], 3], ‘(5,5)’: [(2,5), [(4,5), (3,5)], 3], ‘(3,8)’: [(5,2), [(3,7), (3,6), (3,5), (3,4), (3,3), (4,3), (5,3)], 8], ‘(5,2)’: [(7,7), [(6,2) ,(7,2), (7,3), (7,4), (7,5), (7,6)], 7] }

To do/Instructions:

1. Open *task2_main.py* in editor of your choice.

2. Code file has a function called *main()*. Do not change name of this function. It expects one argument, *image_filename*. This function returns a python lists, *occupied_grids* and a python dictionary *planned_path*. *occupied_grids* should store information about the grids that are occupied either by *Obstacles or by Objects* (Problem Description 1). *planned_path* should store *information about the*

shortest path (Problem Description 2). **Do not change names of any of the return variables.**

3. Add required variables, functions according to your requirement to get the desired output.

4. Follow the guidelines mentioned in *task2_main.py* to complete the above task.

5. Submission:
   a) Save **program files** in folder called Task_2
   b) Compress the folder into zip format. Upload the **Task_2.zip** on the portal.

Rules:

1. There can be multiple shortest path, any of the shortest path will be considered as correct response.

2. Teams are free to use any algorithm of there choice to find the shortest path. Mention the algorithm used as comments in the beginning of the program file.

3. In case there is no matching object found, return “NO MATCH” as first element of dictionary, empty list as second element and 0 as third element. For example as shown in Figure 2, there is no matching object for circle at (4,9).

4. In case there is a matching object but it cannot be reached due to grids being occupied, return “NO PATH” as first element of dictionary, empty list as second element and 0 as third element.

5. You need to write a **generic program**. Note that we have provided five (05) test_images for testing purposes. In addition y*our code will be tested on several **private test_images when you submit your code.***

6. Teams are encouraged to design their own test_images.

Happy Learning

All The Best!!!