Task 1B - Dynamic Obstacle Detection

Note: Go through this file only after you have gone through the tutorials provided in *resources.pdf* document in the *1. Theory* folder.

Please find the ***task_1b.py*** and ***robot-server.c*** file in the *codes* folder.

Modify the ***task_1b.py*** and ***robot-server.c*** to accomplish the following:

Given:

A set of five images, each contain a maze image.

- These images are provided in ***task_1b_images*** folder. An example image of maze **maze00.jpg** is shown in Figure 1.

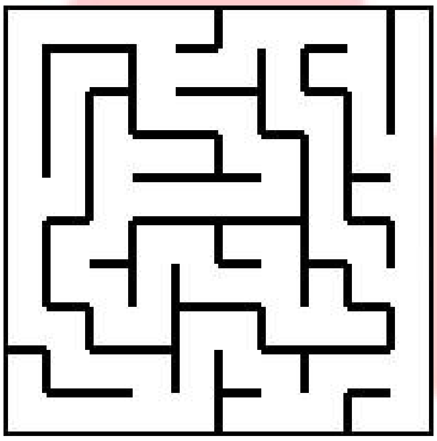


Figure 1: Example maze00.jpg

Problem Statement:

Given such a maze, the problem is to navigate from a START location in the maze to an END location avoiding the dynamic obstacles in the path.

In this task, there are two entities communicating to each other through Socket Programming.

Two entities:

Python client program: ***task_1b.py***

C server program: ***robot-server.c***

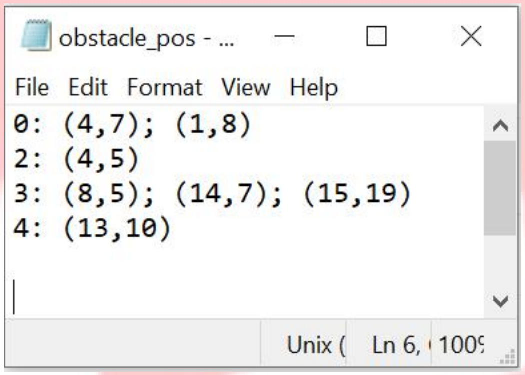
Server will create socket communication and wait for client to connect. Client will connect to the server and send the shortest path found in the maze image.

Positions of dynamic obstacles are stored in ***obstacle_pos.txt*** (for e.g., maze00.jpg has two obstacles at (4,7) and (1,8), maze01.jpg has no obstacle and so on as shown in Figure 2).

Server then checks whether any obstacle is present or not for particular image. It then sends co-ordinate of each obstacle one at a time to the client.

Client marks the obstacle co-ordinate in maze image and computes new shortest path.

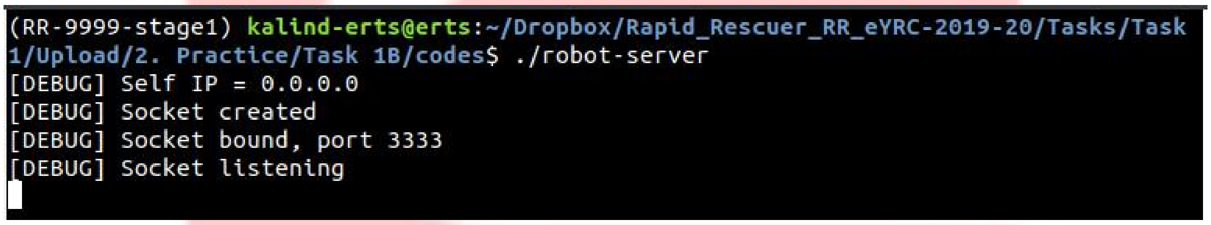
This process continues until final path is computed avoiding all obstacles and repeat the same for all maze images.



```
0: (4,7); (1,8)
2: (4,5)
3: (8,5); (14,7); (15,19)
4: (13,10)
```

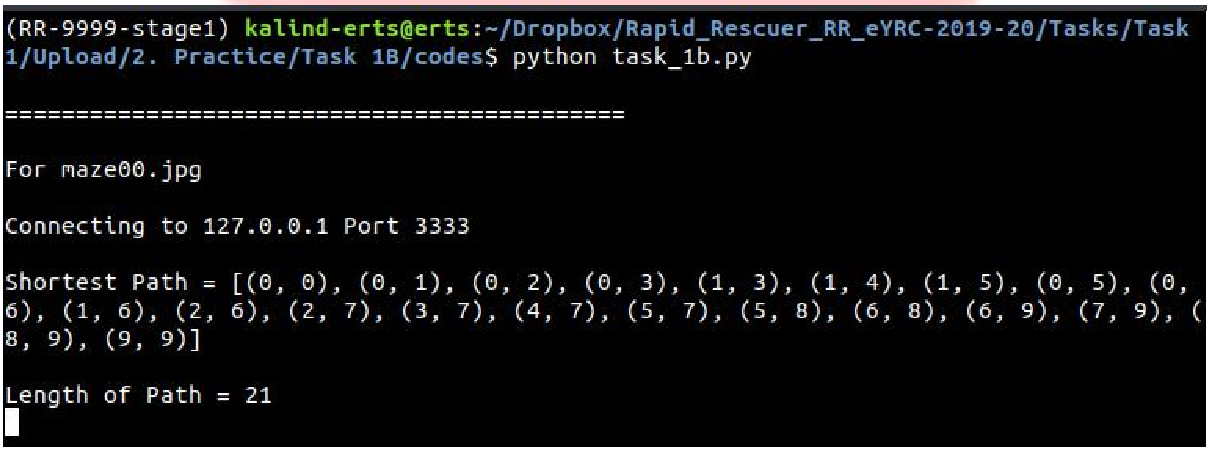
Figure 2: Obstacle position

For maze00.jpg image, following Figures show the communication between client and server.



```
(RR-9999-stage1) kalind-erts@erts:~/Dropbox/Rapid_Rescuer_RR_eYRC-2019-20/Tasks/Task
1/Upload/2. Practice/Task 1B/codes$ ./robot-server
[DEBUG] Self IP = 0.0.0.0
[DEBUG] Socket created
[DEBUG] Socket bound, port 3333
[DEBUG] Socket listening
```

Figure 3: Server first creates connection and waits for client



```
(RR-9999-stage1) kalind-erts@erts:~/Dropbox/Rapid_Rescuer_RR_eYRC-2019-20/Tasks/Task
1/Upload/2. Practice/Task 1B/codes$ python task_1b.py

============================================

For maze00.jpg

Connecting to 127.0.0.1 Port 3333

Shortest Path = [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0,
6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (
8, 9), (9, 9)]

Length of Path = 21
```

Figure 4: Client communicated first shortest path

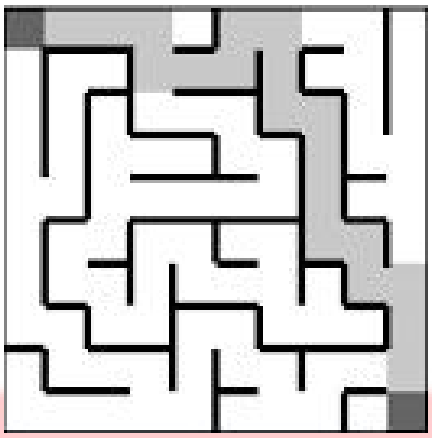
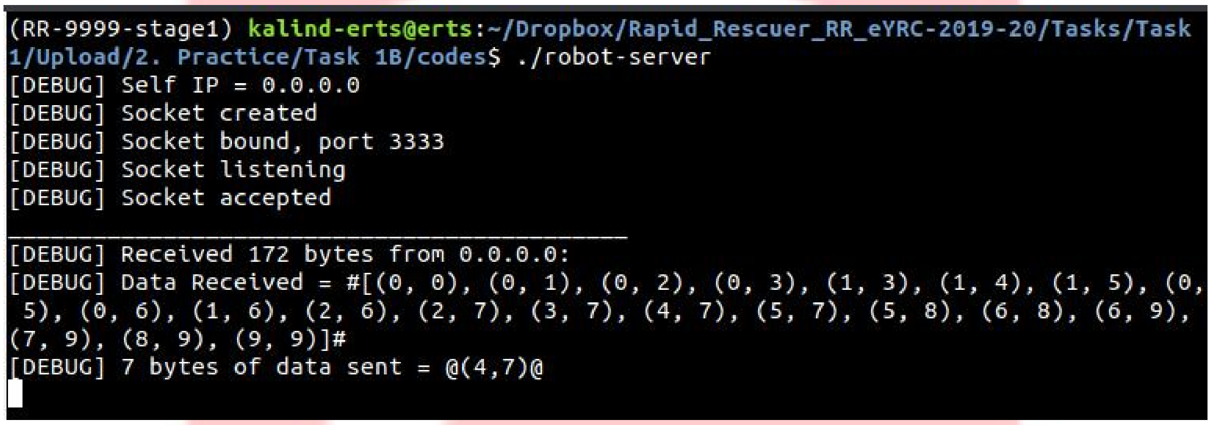
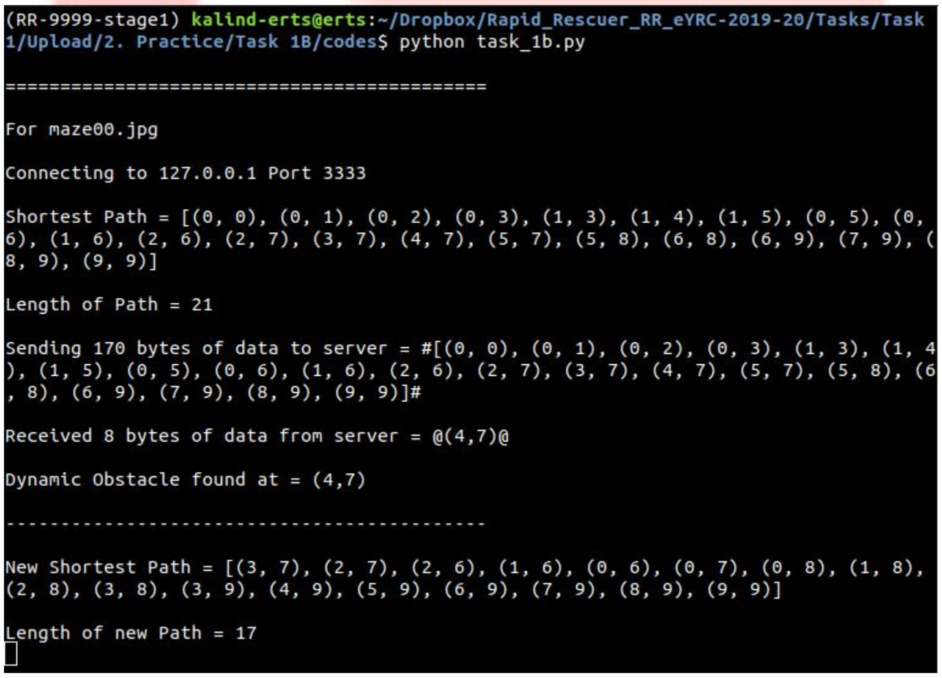


Figure 5: First shortest path



```
(RR-9999-stage1) kalind-erts@erts:~/Dropbox/Rapid_Rescuer_RR_eYRC-2019-20/Tasks/Task
1/Upload/2. Practice/Task 1B/codes$ ./robot-server
[DEBUG] Self IP = 0.0.0.0
[DEBUG] Socket created
[DEBUG] Socket bound, port 3333
[DEBUG] Socket listening
[DEBUG] Socket accepted
____________________________________________
[DEBUG] Received 172 bytes from 0.0.0.0:
[DEBUG] Data Received = #[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0,
 5), (0, 6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9),
(7, 9), (8, 9), (9, 9)]#
[DEBUG] 7 bytes of data sent = @(4,7)@
```

Figure 6: First shortest path received and obstacle position sent



```
(RR-9999-stage1) kalind-erts@erts:~/Dropbox/Rapid_Rescuer_RR_eYRC-2019-20/Tasks/Task
1/Upload/2. Practice/Task 1B/codes$ python task_1b.py

============================================

For maze00.jpg

Connecting to 127.0.0.1 Port 3333

Shortest Path = [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0,
6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (
8, 9), (9, 9)]

Length of Path = 21

Sending 170 bytes of data to server = #[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4
), (1, 5), (0, 5), (0, 6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6
, 8), (6, 9), (7, 9), (8, 9), (9, 9)]#

Received 8 bytes of data from server = @(4,7)@

Dynamic Obstacle found at = (4,7)

--------------------------------------------

New Shortest Path = [(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 8), (1, 8),
(2, 8), (3, 8), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]

Length of new Path = 17
```

Figure 7: Client found new path based on obstacle provided by server

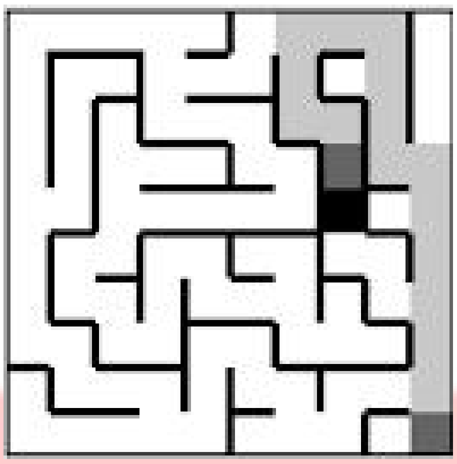
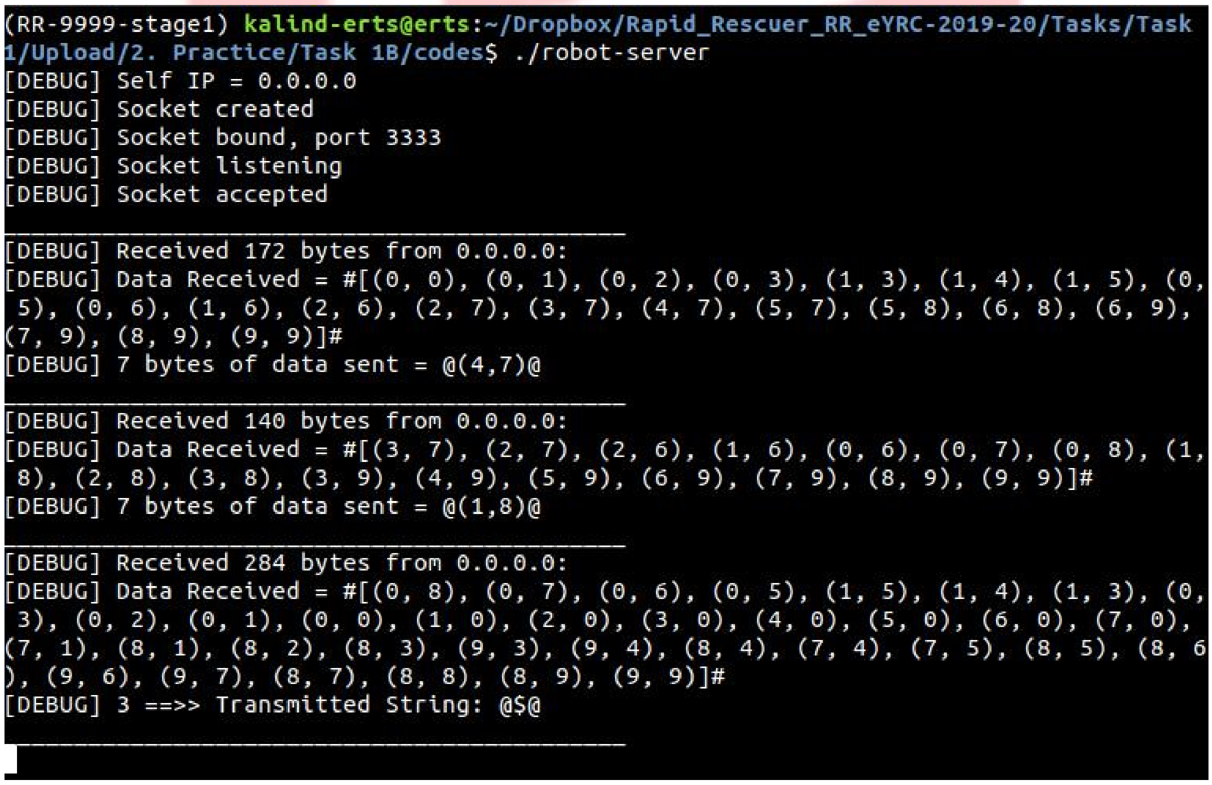
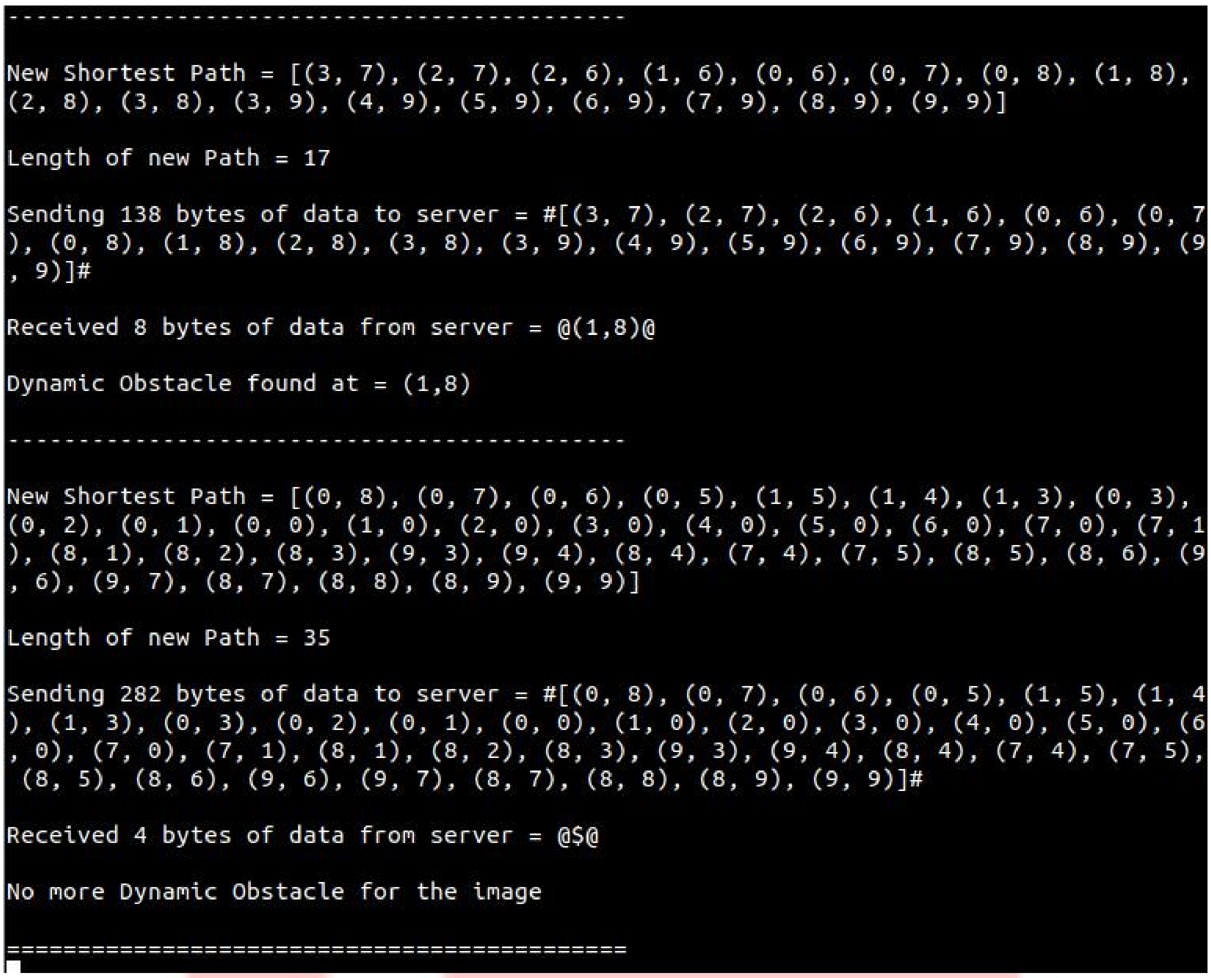


Figure 8: New path with first obstacle

As shown above for obstacle (4,7), similar communication will take place for second and last dynamic obstacle (1,8).

Once server encounters that there are no more obstacles for the image, it sends “$” to the client as shown in Figure 9 and Figure 10.


```
(RR-9999-stage1) kalind-erts@erts:~/Dropbox/Rapid_Rescuer_RR_eYRC-2019-20/Tasks/Task 1/Upload/2. Practice/Task 1B/codes$ ./robot-server
[DEBUG] Self IP = 0.0.0.0
[DEBUG] Socket created
[DEBUG] Socket bound, port 3333
[DEBUG] Socket listening
[DEBUG] Socket accepted
____________________________________________
[DEBUG] Received 172 bytes from 0.0.0.0:
[DEBUG] Data Received = #[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0, 6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (8, 9), (9, 9)]#
[DEBUG] 7 bytes of data sent = @(4,7)@
____________________________________________
[DEBUG] Received 140 bytes from 0.0.0.0:
[DEBUG] Data Received = #[(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]#
[DEBUG] 7 bytes of data sent = @(1,8)@
____________________________________________
[DEBUG] Received 284 bytes from 0.0.0.0:
[DEBUG] Data Received = #[(0, 8), (0, 7), (0, 6), (0, 5), (1, 5), (1, 4), (1, 3), (0, 3), (0, 2), (0, 1), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (9, 4), (8, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9, 6), (9, 7), (8, 7), (8, 8), (8, 9), (9, 9)]#
[DEBUG] 3 ==>> Transmitted String: @$@
____________________________________________
```
Figure 9: Sending “$”



```
--------------------------------------------

New Shortest Path = [(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 8), (1, 8),
(2, 8), (3, 8), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]

Length of new Path = 17

Sending 138 bytes of data to server = #[(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7
), (0, 8), (1, 8), (2, 8), (3, 8), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9
, 9)]#

Received 8 bytes of data from server = @(1,8)@

Dynamic Obstacle found at = (1,8)

--------------------------------------------

New Shortest Path = [(0, 8), (0, 7), (0, 6), (0, 5), (1, 5), (1, 4), (1, 3), (0, 3),
(0, 2), (0, 1), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1
), (8, 1), (8, 2), (8, 3), (9, 3), (9, 4), (8, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9
, 6), (9, 7), (8, 7), (8, 8), (8, 9), (9, 9)]

Length of new Path = 35

Sending 282 bytes of data to server = #[(0, 8), (0, 7), (0, 6), (0, 5), (1, 5), (1, 4
), (1, 3), (0, 3), (0, 2), (0, 1), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6
, 0), (7, 0), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (9, 4), (8, 4), (7, 4), (7, 5),
 (8, 5), (8, 6), (9, 6), (9, 7), (8, 7), (8, 8), (8, 9), (9, 9)]#

Received 4 bytes of data from server = @$@

No more Dynamic Obstacle for the image

============================================
```

Figure 10: No more obstacle indication from server for maze00.jpg

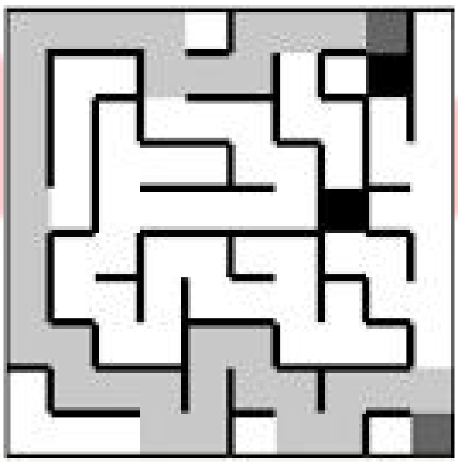


Figure 11: Final shortest path

Nomenclature to be followed:

"$": indicates no more obstacles for an image in ***obstacle_pos.txt***

#<data from client to server>#

> **(for e.g.,** #[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0, 6), (1, 6), (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (8, 9), (9, 9)]#**)**

@<data from server to client>@

> **(for e.g.,** @(4,7)@, @$@**)**

A "snippet" of outline code for client side is given in ***task_1b.py*** file and for server side in ***robot-server.c***.

For ***task_1b.py,***

- Teams modify the following functions:
  - *connect_to_server()*
  - *send_to_receive_from_server()*
  - *find_new_path()*
- Teams **should not modify** the following functions:
  - *main()*
- Make sure you run **task_1b.py** using the conda environment created in Task 0.
- **Note:** Input and output arguments should not be modified at any cost else task will not be evaluated.

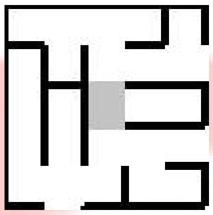
For ***robot-server.c,***

- Teams modify the following functions:
  - *socket_create()*
  - *receive_from_send_to_client()*
- Teams **should not modify** the following functions:
  - *main()*
- **Note:** Input and output arguments should not be modified at any cost else task will not be evaluated.

- To generate executable file, type in Terminal “**gcc robot-server.c -o robot-server**” and to run this executable file type “**./robot-server**”

Instructions:

- Teams are **not allowed** to import any **library/module** related to **Image Processing** (apart from the ones installed in **Task 0**) in ***task_1b.py*** file. **If found so, it will lead to disqualification**.
- Do not edit the **main** function in ***task_1b.py*** file. Teams have to modify **only** the above mentioned functions.
- You may use **colourCell()** function of image_enhancer module with input arguments **img, row, column, colourVal** and it returns **img.** For e.g., **image_enhancer.colourCell(img,2,2,200)** as shown below.

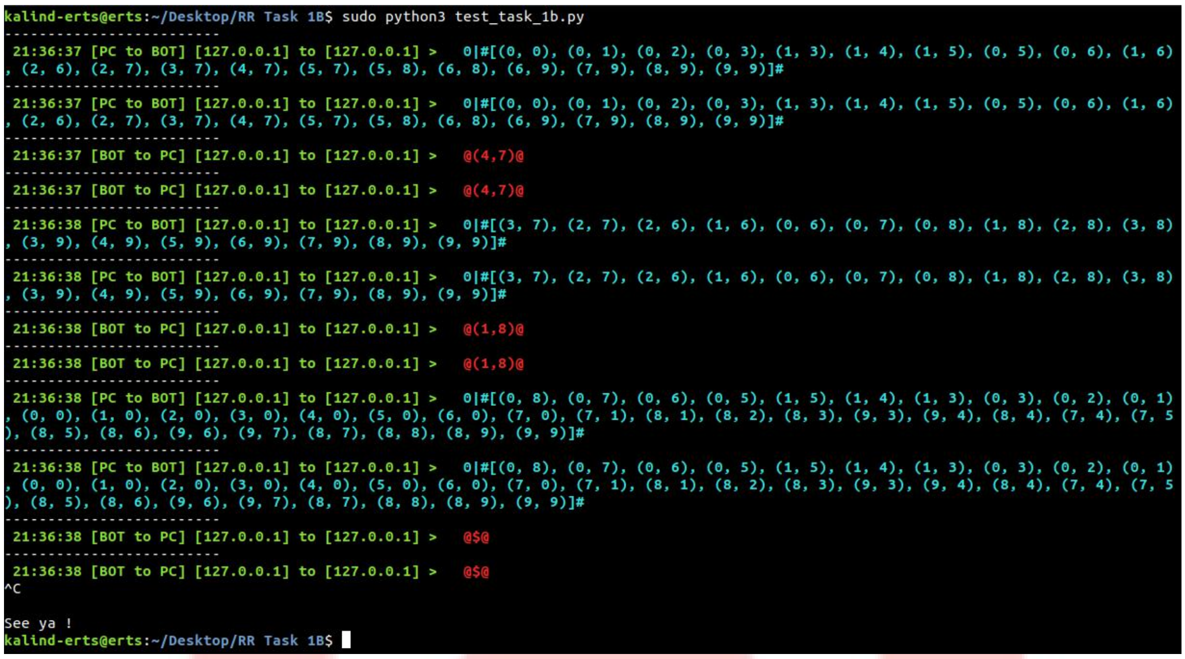


- The **main** function, by default displays output for the one image, maze00.jpg and asks the user whether to check the output for rest images.
- To check the output for all the test images, type “**y**” and press Enter.
- Teams can verify the output with the help of **output_1b.pdf** file given.
- Once done with the Task, first run **robot-server.c, task_1b.py** and **test_task_1b.py** in **separate three terminals**. It will show the output of your program on terminal and also generate **task_1b_output.txt** file in the same folder.

NOTE: While running **task_1b.py**, make sure conda environment is activated.

While running **test_task_1b.py**, make sure no conda environment is activated and run file using “**sudo python3 test_task_1b.py**” as shown in Figure 12.

Once communication is done, close the **test_task_1b.py** using Ctrl+C.



```
kalind-erts@erts:~/Desktop/RR Task 1B$ sudo python3 test_task_1b.py
-------------------------
 21:36:37 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0, 6), (1, 6)
, (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (8, 9), (9, 9)]#
-------------------------
 21:36:37 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (0, 5), (0, 6), (1, 6)
, (2, 6), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (6, 8), (6, 9), (7, 9), (8, 9), (9, 9)]#
-------------------------
 21:36:37 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @(4,7)@
-------------------------
 21:36:37 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @(4,7)@
-------------------------
 21:36:38 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8)
, (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]#
-------------------------
 21:36:38 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(3, 7), (2, 7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8)
, (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]#
-------------------------
 21:36:38 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @(1,8)@
-------------------------
 21:36:38 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @(1,8)@
-------------------------
 21:36:38 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(0, 8), (0, 7), (0, 6), (0, 5), (1, 5), (1, 4), (1, 3), (0, 3), (0, 2), (0, 1)
, (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (9, 4), (8, 4), (7, 4), (7, 5
), (8, 5), (8, 6), (9, 6), (9, 7), (8, 7), (8, 8), (8, 9), (9, 9)]#
-------------------------
 21:36:38 [PC to BOT] [127.0.0.1] to [127.0.0.1] >   0|#[(0, 8), (0, 7), (0, 6), (0, 5), (1, 5), (1, 4), (1, 3), (0, 3), (0, 2), (0, 1)
, (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (9, 4), (8, 4), (7, 4), (7, 5
), (8, 5), (8, 6), (9, 6), (9, 7), (8, 7), (8, 8), (8, 9), (9, 9)]#
-------------------------
 21:36:38 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @$@
-------------------------
 21:36:38 [BOT to PC] [127.0.0.1] to [127.0.0.1] >   @$@
^C

See ya !
kalind-erts@erts:~/Desktop/RR Task 1B$
```

Figure 12: Output for test_task_1b.py

Note: If you are not getting any output, check the following:

1. You have executed **robot-server.c** file before running **task_1b.py.**
2. You are running **task_1b.py** using the conda environment created in Task 0.
3. You are running **test_task_1b.py** without activating conda environment
4. **task_1a.py** and **image_enhancer.pyc** files are present in ../../Task 1A/codes folder (Do not change the name of these two files**)**
5. ***images*** folder location is unaltered (Do not change the name of folder or images provided in that folder)
6. Input and output arguments of defined functions are as specified (Do not change the name of functions)