User Manual for Optimal Coverage Area Project Source Code

Author: Ozan Oytun Karakaya

Required parameters that should be specified according to run case, in the beginning lines of the Python program, **source_code.py**. You can see them just below the necessary imports, starting from *line number 8*.

source_code.py > ...

```
1   import random
2   import math
3   import heapq
4
5   # Parameters of the program is given below, please specify the values of the parameters before running the program
6   # according to the requirements of the case
7
8   width = 100 # horizontal length of the area in number of (1x1) unit squares
9   height = 100 # vertical length of the area in number of (1x1) unit squares
10  sense_radius = 10 # sensing radius of sensors, same for all sensors, any whole number
11  max_num_of_sensors = 75 # maximum number of sensors to be placed in the area
12  min_sep_length = 1 # minimum seperation length between sensors, any multiplier with sensing radius
13  sensor_placed_ids = [] # ids of the unit squares where sensors are placed
14  covered_ids = set() # ids of the unit squares covered by sensors
15  rock_density = 0.2 # density of rocks in the area, black id number over whole grid area
16  max_number_of_black_ids = width*height*rock_density # maximum number of black ids to be placed in the area
17  max_rock_length = sense_radius # maximum length for an obstacle is bounded with sensing radius of sensors
18  rock_number = 10 # specifies the number of starting points for obstacles inside obstacle creation loop
19
20  # The code below is the implementation of the algorithm that places sensors in the area
```

In case the in-line comments may not be clear, explanations for variables are below.

- **width**: Horizontal length of the rectangular area to be deployed with sensors, expressed in terms of 1x1 unit squares
- **height**: Vertical length of the rectangular area to be deployed with sensors, expressed in terms of 1x1 unit squares
- **sense_radius**: Sensing radius of all identical sensors, expressed in terms of 1 unit length in the area
- **max_num_of_sensors**: Maximum number of sensors that can be placed, limits the execution of the greedy algorithm
- **covered_ids** = this is not a variable that should be changed, keeps the IDs of covered squares for the algorithm
- **rock_density**: Density of the obstacles in the area should be specified, if it will be generated randomly in the program. Value should be expressed between [0,1]. Not recommended to go above 0.5.
- **max_number_of_black_ids**: Maximum number of obstacle IDs, calculated as multiplying the density with the area.
- **rock_number**: Number of starting positions in a single for loop when generating random obstacles, not an important variable. Please note that, it shouldn't be larger than the **max_number_of_black_ids** variable.

There are definitions of auxiliary functions which are used in the program under this initialization section. Before running the main algorithm of the program which determines the places where sensors are placed, there is a section for generating random obstacles and initializing it to the **black_ids** variable, which holds the IDs of the obstacle IDs. User can proceed as wished according to the 2 scenarios below;

1. Using Random Obstacle Generation for the specified obstacle density
2. Giving the obstacle IDs manually for a specific area

First scenario doesn't require any further configuration other than specifying the **rock_density** variable for generating random obstacles algorithm.

As for the second scenario, the **black_ids** variable should be assigned to the list of the IDs which are black, in *line number 245*. Please note that this variable must have the data type of **list** data type of Python.

```
241   # Here random generated obstacles are assigned to the black_ids list
242   black_ids = random_black_ids
243
244   # Please provide the wanted black_id list here if you want to use a specific obstacle configuration under this line
245   # black_ids = []
246   # Please note that black_ids should be specified as a list data type
247
248   # Random obstacle generation inside the area ends here
```

You can comment line 242, then uncomment line 245 and assign the values in manual configuration.

Lastly, there are few print statements for showing sensor placements and coverage. You can add more print statements to see the state of any variable.

```
# Print the results, you can print other results you want to see here at the end of the program
print("Coverage: ", calculate_coverage())
print("Number of sensors placed: ", sensor_placed_ids.__len__())
print("Sensor placed ids: ", sensor_placed_ids)
```