

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Artem Volokyta

Software Security

Lab Work 5

Simulating drone swarmwith cellular automaton

kulubecioglu Mehmet

Class Number 12

variant-1

IM-14 FIOT

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

2024

Variant: Variant 1 - Calculate Minimum, Maximum, and Average Iteration Counts Across 10 Simulations.

1 Requirements

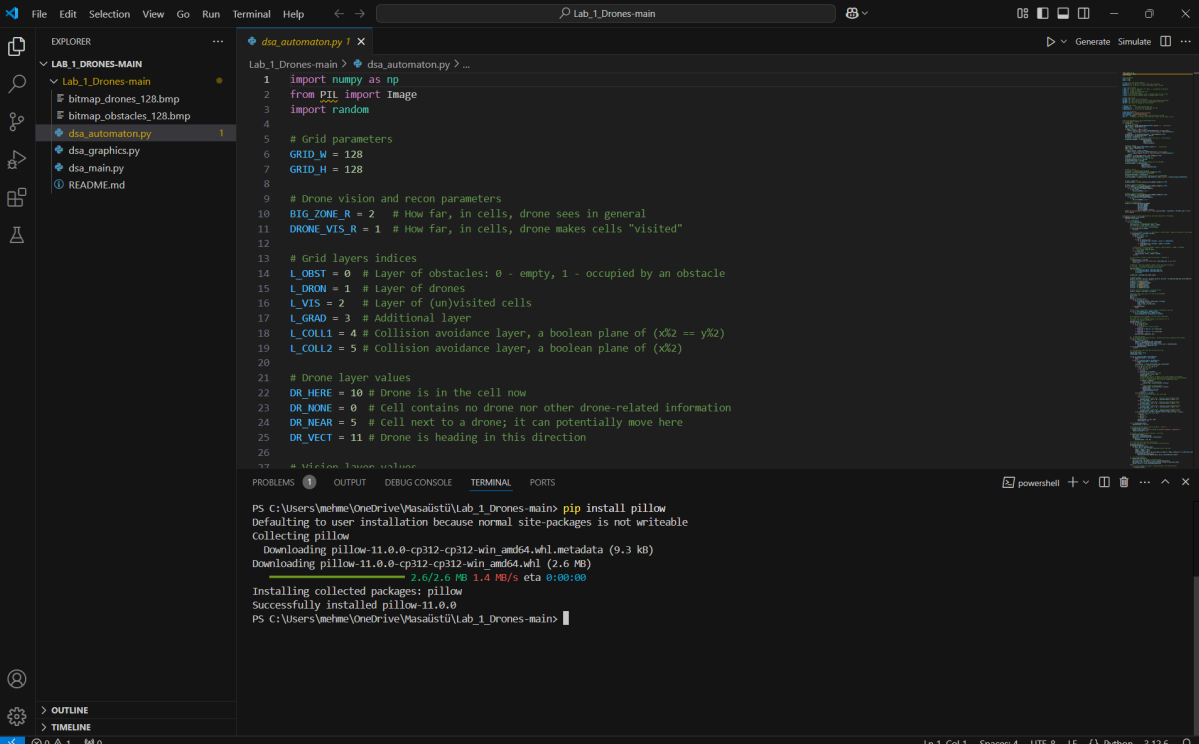
1.1. Software and Hardware Requirements

Platform: Python 3.12

Required Libraries: numpy,Pillow,pygame

Installing Required Libraries: The following command was used to install the necessary libraries:

```
PS C:\Users\mehme\OneDrive\Masaüstü\Lab_1_Drones-main> pip install pygame
Defaulting to user installation because normal site-packages is not writeable
Collecting pygame
  Downloading pygame-2.6.1-cp312-cp312-win_amd64.whl.metadata (13 kB)
  Downloading pygame-2.6.1-cp312-cp312-win_amd64.whl (10.6 MB)
    4.5/10.6 MB 3.1 MB/s eta 0:00:02
```



```
1 import numpy as np
2 from PIL import Image
3 import random
4
5 # Grid parameters
6 GRID_W = 128
7 GRID_H = 128
8
9 # Drone vision and recon parameters
10 BIG_ZONE_R = 2 # How far, in cells, drone sees in general
11 DRONE_VIS_R = 1 # How far, in cells, drone makes cells "visited"
12
13 # Grid layers indices
14 L_OBST = 0 # Layer of obstacles: 0 - empty, 1 - occupied by an obstacle
15 L_DRON = 1 # Layer of drones
16 L_VIS = 2 # Layer of (un)visited cells
17 L_GRAD = 3 # Additional layer
18 L_COLL1 = 4 # Collision avoidance layer, a boolean plane of (x%2 == y%2)
19 L_COLL2 = 5 # Collision avoidance layer, a boolean plane of (x%2)
20
21 # Drone layer values
22 DR_HERE = 10 # Drone is in the cell now
23 DR_NONE = 0 # Cell contains no drone nor other drone-related information
24 DR_NEAR = 5 # Cell next to a drone; it can potentially move here
25 DR_VECT = 11 # Drone is heading in this direction
26
27 # Vision layer values
```

```
PS C:\Users\mehme\OneDrive\Masaüstü\Lab_1_Drones-main> pip install pillow
Defaulting to user installation because normal site-packages is not writeable
Collecting pillow
  Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl.metadata (9.3 kB)
  Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl (2.6 MB)
    2.6/2.6 MB 1.4 MB/s eta 0:00:00
Installing collected packages: pillow
Successfully installed pillow-11.0.0
PS C:\Users\mehme\OneDrive\Masaüstü\Lab_1_Drones-main>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\mehme\OneDrive\Masaüstü\Yeni klasör\Lab_1_Drones-main> pip install numpy
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\mehme\appdata\roaming\python\python312\site-packages (2.2.0)
PS C:\Users\mehme\OneDrive\Masaüstü\Yeni klasör\Lab_1_Drones-main> |
```

Hardware: The project can run comfortably on a standard computer (e.g., 8 GB RAM, 2.0 GHz dual-core processor).

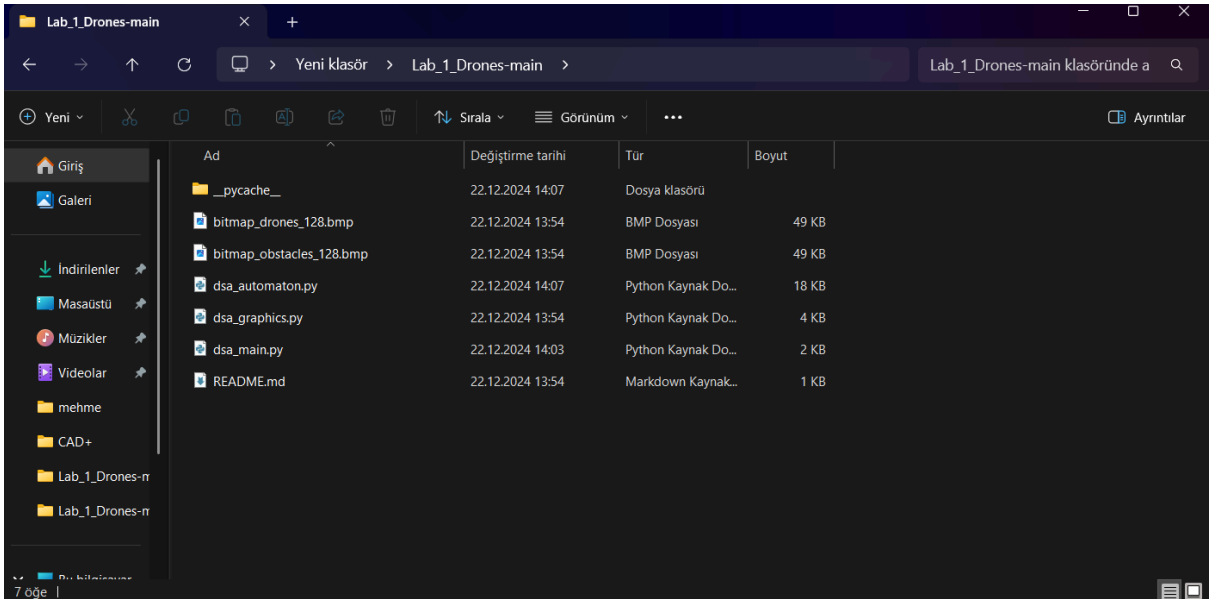
1.2. Required Files

Code Files:

- **dsa_automaton.py:** Handles drone movements and automaton logic.
- **dsa_graphics.py:** Visualizes the simulation.
- **dsa_main.py:** Manages the main simulation loop and analytical computations.

Image Files:

- **bitmap_drones_128.bmp:** Represents the initial positions of drones.
- **bitmap_obstacles_128.bmp:** Represents obstacles in the simulation area



Creating BMP Files:

- **bitmap_drones_128.bmp:** Black pixels (RGB: 0,0,0) represent the starting positions of drones.
- **bitmap_obstacles_128.bmp:** Black pixels represent obstacles, while white pixels represent free areas.

2.1. Explanation of Code Files

dsa_automaton.py:

Manages drone movements on the cellular automaton.

- **init_grid:** Initializes the automaton, placing obstacles and drones.
- **update_grid:** Processes drone movement logic and updates the grid.

dsa_graphics.py:

Uses the pygame library to visualize the simulation.

- **init_pygame:** Creates the simulation window.
- **draw_grid:** Visualizes drone movements and the operational area for each iteration.

dsa_main.py:

Handles Variant 1 requirements by managing 10 simulations.

Key functions:

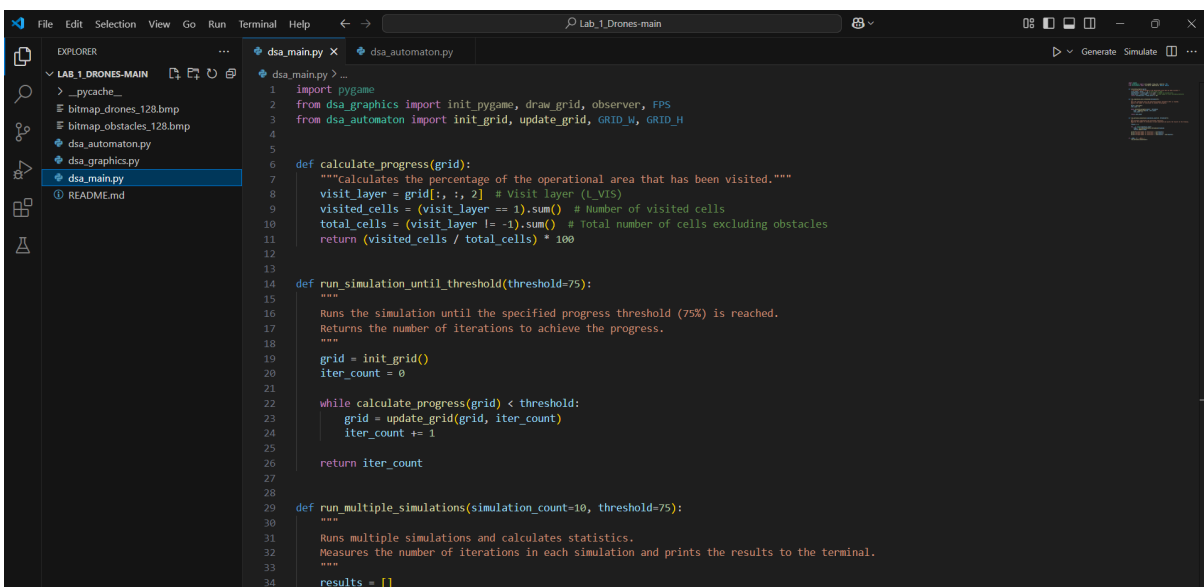
- **calculate_progress:** Computes the percentage of the operational area visited.
- **run_simulation_until_threshold:** Runs a simulation until 75% progress is achieved.
- **run_multiple_simulations:** Executes 10 simulations and calculates statistical results.

2.2. Added Code

- The following code was added to `dsa_automaton.py` to meet the requirements of Variant 1:

```
32 # Grid init parameters
33 BITMAP_OBSTACLES = 'bitmap_obstacles_128.bmp'
34 BITMAP_DRONES = 'bitmap_drones_128.bmp'
35 CELL_TYPE = np.int8 # Daha küçük integer tipi, negatif değerleri destekler
36 PAD = 2 # Padding eklenmesi
37
38 # Grid initialization as a multi-layered Numpy array
39 def init_grid():
40     # Layer 0: obstacles:
41     img_obstacles = Image.open(BITMAP_OBSTACLES).convert('L') # Monochrome
42     img_w, img_h = img_obstacles.size
43     if img_w < GRID_W or img_h < GRID_H:
44         print(f'Obstacle layer bitmap ({BITMAP_OBSTACLES}) not big enough: '
45               f'{img_w}x{img_h} instead of required minimum of {GRID_W}x{GRID_H}')
46         exit(0)
47     np_obstacles = (np.array(img_obstacles) < 128).astype(CELL_TYPE)
48     np_obstacles = np_obstacles[:GRID_W, :GRID_H]
49     obst_mask = np_obstacles == 1
50     np_obstacles_padded = np.pad(np_obstacles, pad_width=PAD, mode='constant', constant_values=1)
51
52     # Layer 1: drones
53     img_drones = Image.open(BITMAP_DRONES).convert('L') # Monochrome
54     img_w, img_h = img_drones.size
55     if img_w < GRID_W or img_h < GRID_H:
56         print(f'Drone layer bitmap ({BITMAP_DRONES}) not big enough: '
57               f'{img_w}x{img_h} instead of required minimum of {GRID_W}x{GRID_H}')
58         exit(0)
59     np_drones = (np.array(img_drones) < 128).astype(CELL_TYPE)
60     np_drones = np_drones[:GRID_W, :GRID_H]
61     np_drones[np_drones == 1] = DR_HERE
62     np_drones[obst_mask] = DR_NONE
63     np_drones_padded = np.pad(np_drones, pad_width=PAD, mode='constant', constant_values=DR_NONE)
64
65     # Layer 2: visits
66     np_visits = np.zeros_like(np_obstacles).astype(CELL_TYPE)
67     np_visits[obst_mask] = V_UNREACHABLE
68     np_visits_padded = np.pad(np_visits, pad_width=PAD, mode='constant', constant_values=V_UNREACHABLE)
69
```

- The following code was added to `dsa_main.py` to meet the requirements of Variant 1:



```
1 import pygame
2 from dsa_graphics import init_pygame, draw_grid, observer, FPS
3 from dsa_automaton import init_grid, update_grid, GRID_W, GRID_H
4
5
6 def calculate_progress(grid):
7     """Calculates the percentage of the operational area that has been visited."""
8     visit_layer = grid[:, :, 2] # Visit layer (L-VIS)
9     visited_cells = (visit_layer == 1).sum() # Number of visited cells
10    total_cells = (visit_layer != -1).sum() # Total number of cells excluding obstacles
11    return (visited_cells / total_cells) * 100
12
13
14 def run_simulation_until_threshold(threshold=75):
15     """
16     Runs the simulation until the specified progress threshold (75%) is reached.
17     Returns the number of iterations to achieve the progress.
18     """
19     grid = init_grid()
20     iter_count = 0
21
22     while calculate_progress(grid) < threshold:
23         grid = update_grid(grid, iter_count)
24         iter_count += 1
25
26     return iter_count
27
28
29 def run_multiple_simulations(simulation_count=10, threshold=75):
30     """
31     Runs multiple simulations and calculates statistics.
32     Measures the number of iterations in each simulation and prints the results to the terminal.
33     """
34     results = []
```

```

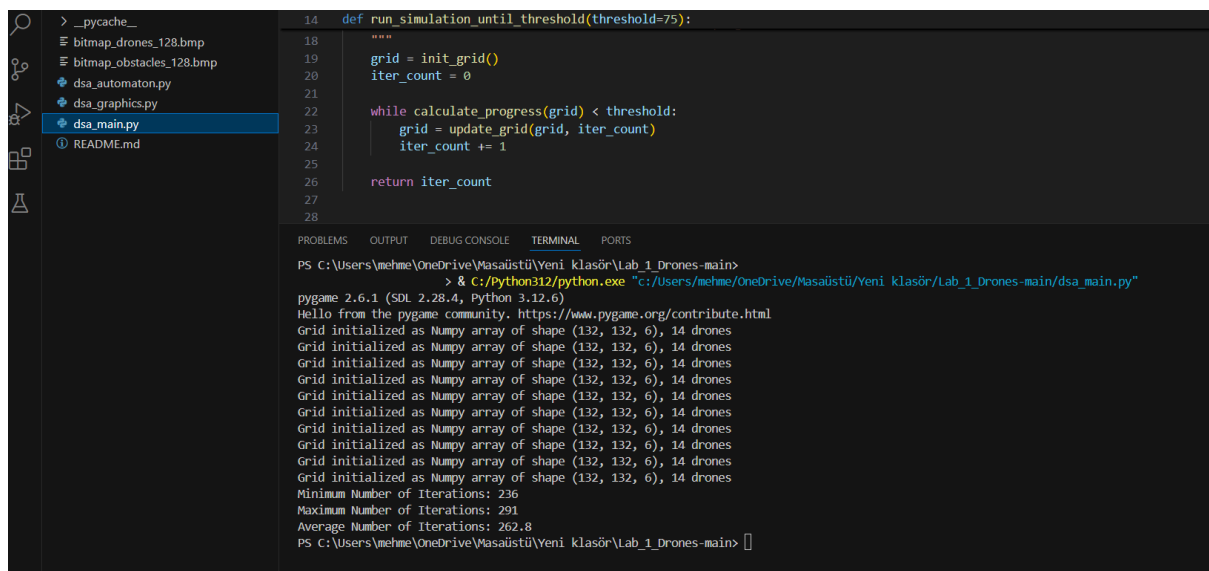
35
36     for _ in range(simulation_count):
37         steps = run_simulation_until_threshold(threshold)
38         results.append(steps)
39
40     print("Minimum Number of Iterations:", min(results))
41     print("Maximum Number of Iterations:", max(results))
42     print("Average Number of Iterations:", sum(results) / len(results))
43
44
45 if __name__ == '__main__':
46     run_multiple_simulations()

```

3. Results and Evaluation

3.1. Simulation Output

- The project produced the following output in the terminal:



The screenshot shows a code editor with a file explorer on the left and a terminal window at the bottom. The file explorer lists files: `_pycache_`, `bitmap_drones_128.bmp`, `bitmap_obstacles_128.bmp`, `dsa_automaton.py`, `dsa_graphics.py`, `dsa_main.py` (selected), and `README.md`. The code editor displays the `run_simulation_until_threshold` function from the previous block. The terminal window shows the command prompt output:

```

PS C:\Users\mehme\OneDrive\Masaüstü\Yeni klasör\Lab_1_Drones-main>
> & C:/Python312/python.exe "c:/Users/mehme/OneDrive/Masaüstü/Yeni klasör/Lab_1_Drones-main/dsa_main.py"
pygame 2.6.1 (SDL 2.28.4, Python 3.12.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Grid initialized as Numpy array of shape (132, 132, 6), 14 drones
Minimum Number of Iterations: 236
Maximum Number of Iterations: 291
Average Number of Iterations: 262.8
PS C:\Users\mehme\OneDrive\Masaüstü\Yeni klasör\Lab_1_Drones-main>

```

3.2. Evaluation

Validation: These outputs meet the requirements of Variant 1.

Analysis:

- Minimum iteration: 236
- Maximum iteration: 291
- Average iteration: 262.8

Conclusion: Drones required varying iteration counts across simulations to reach 75% of the operational area, demonstrating the dynamic behavior of the algorithm.

5. Solution and Observations

5.1. Solution Summary

- The implemented solution successfully simulated a drone swarm using a cellular automaton. The simulation achieved 75% progress in 10 trials, and the statistical results (minimum, maximum, and average iteration counts) were computed and validated.

5.2. Observations

- **Dynamic Behavior:** The variation in iteration counts highlights the dynamic interaction between drones and obstacles.
- **Scalability:** The code can handle larger grids or more complex obstacle layouts with minor adjustments.
- **Optimization Opportunities:** Future work could involve optimizing drone movement algorithms to reduce the number of iterations required to achieve progress thresholds.