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.

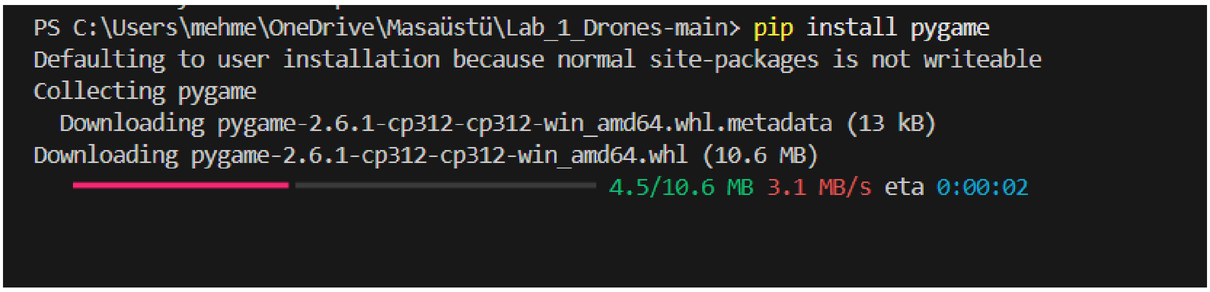
1 Requirements

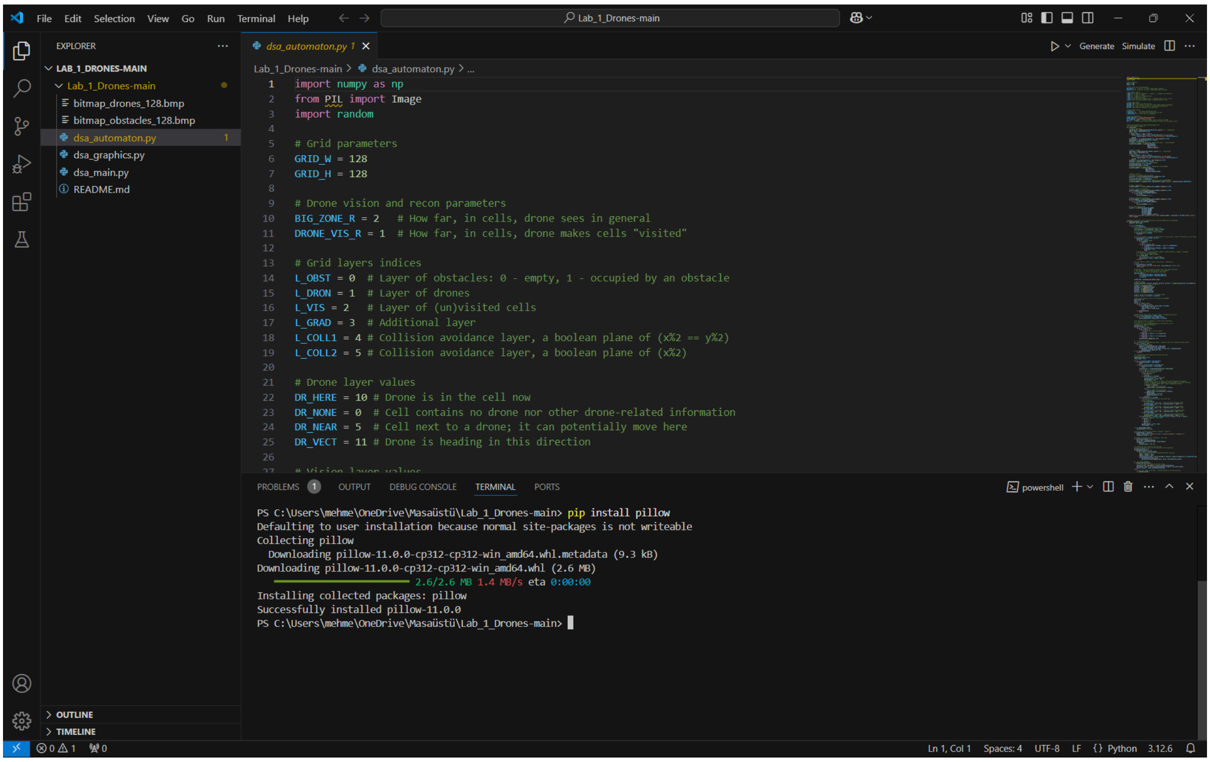
1.1. Software and Hardware Requirements

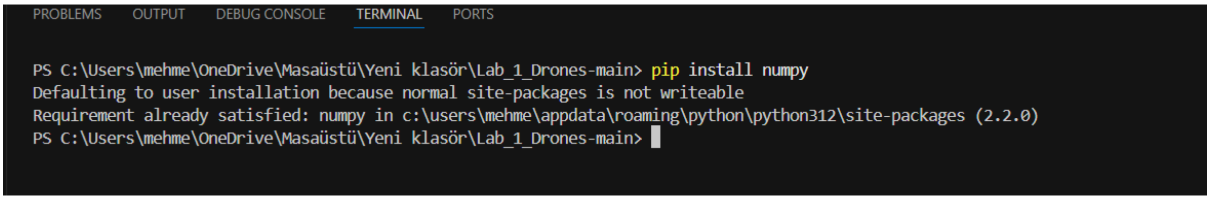
Platform: Python 3.12

Required Libraries: numpy,Pillow,pygame

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







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

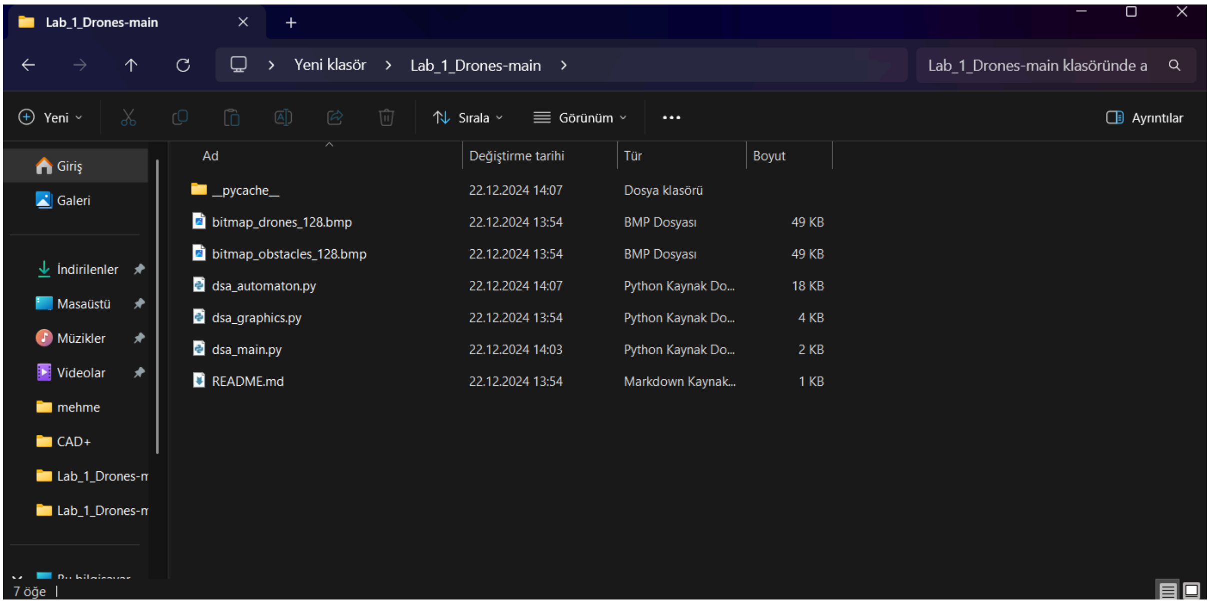
1.2. Required Files

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:**

```python
# Grid init parameters
BITMAP_OBSTACLES = 'bitmap_obstacles_128.bmp'
BITMAP_DRONES = 'bitmap_drones_128.bmp'
CELL_TYPE = np.int8  # Daha küçük integer tipi, negatif değerleri destekler
PAD = 2  # Padding eklenmesi

# Grid initialization as a multi-layered Numpy array
def init_grid():
    # Layer 0: obstacles:
    img_obstacles = Image.open(BITMAP_OBSTACLES).convert('L')  # Monochrome
    img_w, img_h = img_obstacles.size
    if img_w < GRID_W or img_h < GRID_H:
        print(f'Obstacle layer bitmap ({BITMAP_OBSTACLES}) not big enough: '
              f'{img_w}x{img_h} instead of required minimum of {GRID_W}x{GRID_H}')
        exit(0)
    np_obstacles = (np.array(img_obstacles) < 128).astype(CELL_TYPE)
    np_obstacles = np_obstacles[:GRID_W, :GRID_H]
    obst_mask = np_obstacles == 1
    np_obstacles_padded = np.pad(np_obstacles, pad_width=PAD, mode='constant', constant_values=1)

    # Layer 1: drones
    img_drones = Image.open(BITMAP_DRONES).convert('L')  # Monochrome
    img_w, img_h = img_drones.size
    if img_w < GRID_W or img_h < GRID_H:
        print(f'Drone layer bitmap ({BITMAP_OBSTACLES}) not big enough: '
              f'{img_w}x{img_h} instead of required minimum of {GRID_W}x{GRID_H}')
        exit(0)
    np_drones = (np.array(img_drones) < 128).astype(CELL_TYPE)
    np_drones = np_drones[:GRID_W, :GRID_H]
    np_drones[np_drones == 1] = DR_HERE
    np_drones[obst_mask] = DR_NONE
    np_drones_padded = np.pad(np_drones, pad_width=PAD, mode='constant', constant_values=DR_NONE)

    # Layer 2: visits
    np_visits = np.zeros_like(np_obstacles).astype(CELL_TYPE)
    np_visits[obst_mask] = V_UNREACHABLE
    np_visits_padded = np.pad(np_visits, pad_width=PAD, mode='constant', constant_values=V_UNREACHABLE)
```

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

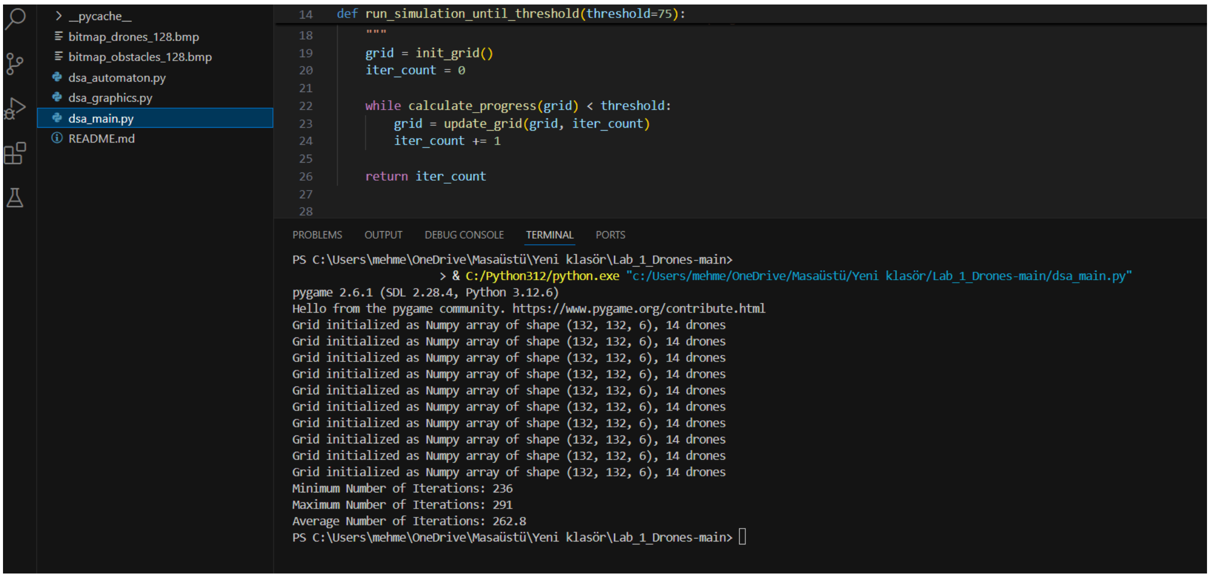
```python
import pygame
from dsa_graphics import init_pygame, draw_grid, observer, FPS
from dsa_automaton import init_grid, update_grid, GRID_W, GRID_H


def calculate_progress(grid):
    """Calculates the percentage of the operational area that has been visited."""
    visit_layer = grid[:, :, 2]  # Visit layer (L_VIS)
    visited_cells = (visit_layer == 1).sum()  # Number of visited cells
    total_cells = (visit_layer != -1).sum()  # Total number of cells excluding obstacles
    return (visited_cells / total_cells) * 100


def run_simulation_until_threshold(threshold=75):
    """
    Runs the simulation until the specified progress threshold (75%) is reached.
    Returns the number of iterations to achieve the progress.
    """
    grid = init_grid()
    iter_count = 0

    while calculate_progress(grid) < threshold:
        grid = update_grid(grid, iter_count)
        iter_count += 1

    return iter_count


def run_multiple_simulations(simulation_count=10, threshold=75):
    """
    Runs multiple simulations and calculates statistics.
    Measures the number of iterations in each simulation and prints the results to the terminal.
    """
    results = []
```

```python

    for _ in range(simulation_count):
        steps = run_simulation_until_threshold(threshold)
        results.append(steps)

    print("Minimum Number of Iterations:", min(results))
    print("Maximum Number of Iterations:", max(results))
    print("Average Number of Iterations:", sum(results) / len(results))


if __name__ == '__main__':
    run_multiple_simulations()
```

3. Results and Evaluation

3.1. Simulation Output

- **The project produced the following output in the terminal:**



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.