Linnæus University
Sweden

The Robotic Lawnmower Project

Author: Yaryna Mryhlotska, Sofiia-Mariia Denysiak
Supervisor: Björn Lindenberg
University: Linnaeus University
Subject: Introduction to programming
Tutoring group: NGMAT+
Date of submission: 01.11.2024

Table of Contents

1 Introduction

The **Robotic Lawnmower Project** is a programming-based simulation developed for the course 1DV501 – Introduction to Programming. The project models the behavior of a robotic lawnmower as it navigates a lawn, allowing analysis of its path, efficiency, and coverage over time. With interactive visualization options, the project provides insights into how different parameters and settings affect the mower's performance.
This simulation, implemented in Python, includes the following components:

- **File Handling and Ground Mapping**: The program reads map data from CSV files, identifying elements such as lawn areas ('L'), obstacles ('O'), and start positions ('S').
- **Dynamic Visualization Options**: A user-friendly menu allows real-time visualization of multiple aspects of the mower's operation, including:
    - Ground map visualization, showing lawn and obstacles.
    - Path visualization, displaying the mower's movement from the start position.
    - Lawn area percentage, depicting how much of the lawn is cut versus uncut.
    - Cut vs. uncut lawn chart, which gives a detailed look at the mower's coverage.
- **Pathfinding with Randomized Obstacle Avoidance**: The mower randomly navigates the lawn while adjusting its path to avoid obstacles, mimicking real-world robotic mowing.
- **Adjustable Operational Parameters**: Users can specify the mower's **speed** and **working hours** to simulate different mowing durations and coverage levels. This directly influences the number of steps the mower takes, offering flexibility for various scenarios.

Through interactive visual feedback and adjustable settings, the project allows for in-depth exploration of autonomous lawn mowing, highlighting how settings and lawn layout impact overall efficiency and coverage.

2 Grade E

2.1 Ground Maps

In this section, we visualize and analyze two ground maps, small.csv and my_map.csv, representing different lawn configurations. The maps, stored in CSV format, contain symbols representing various ground elements:

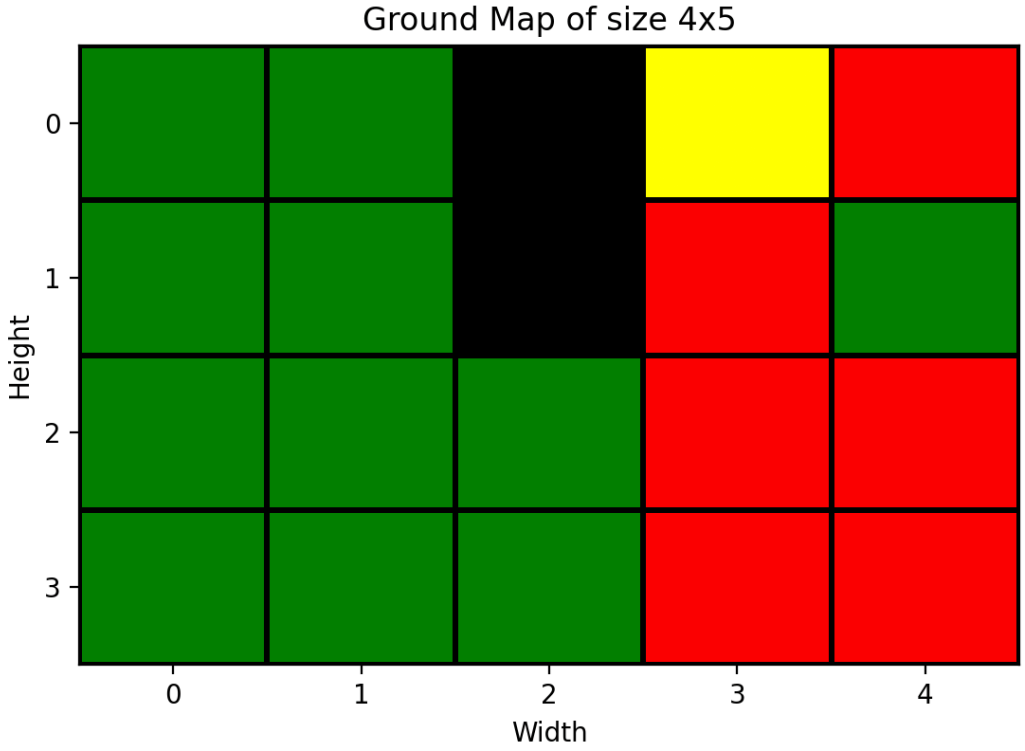
- **'L'** – Lawn area.
- **'O'** – Obstacles.
- **'S'** – Start position.
- **'C'** – Cut lawn, updated during mowing to track coverage.

Ground Map Analysis

1. **Map Visualization**
   The plot_ground_map function uses the matplotlib library to visualize each map. Different colors indicate lawn, obstacles, and the start position, offering a clear layout view and aiding in predicting mower movement. Here's a code snippet for map visualization:

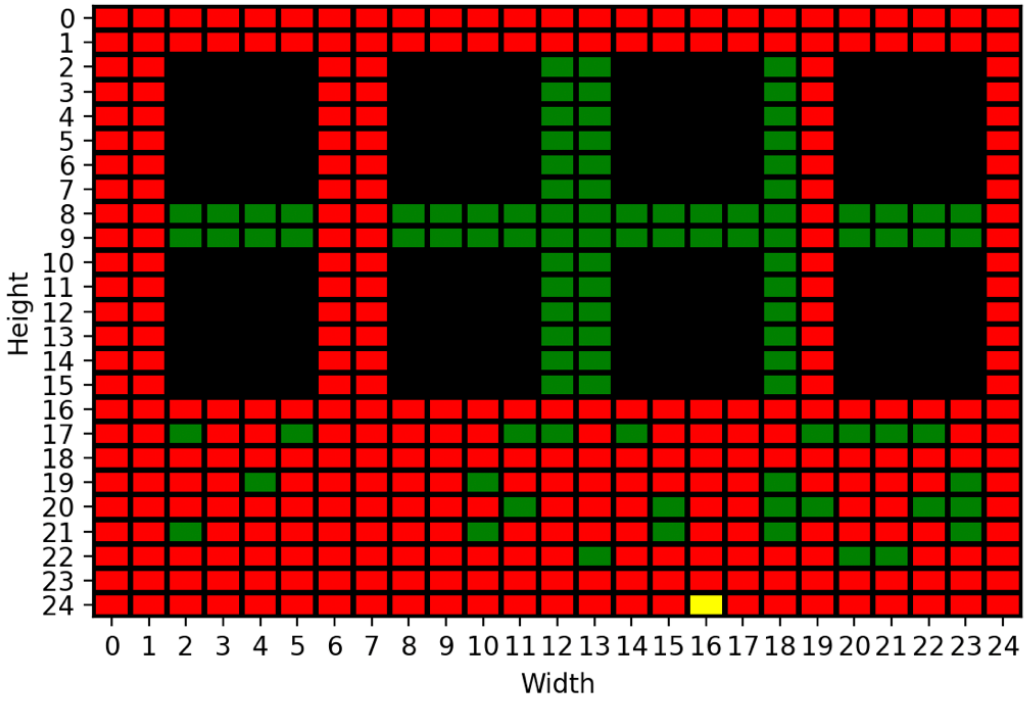
```
def plot_ground_map(ground_map, total_area, lawn_area, lawn_percentage):
    color_map = {'L': 0, 'O': 1, 'S': 2, 'C': 3, 'E': 2}
    visual_map = [[color_map[cell] for cell in row] for row in ground_map]
    rows = len(visual_map)
    cols = len(visual_map[0])
    col_map = ListedColormap(['green', 'black', 'yellow', 'red'], 'indexed')

    plt.figure()
    plt.pcolormesh(visual_map, edgecolors='k', linewidth=2, cmap=col_map)
    ax = plt.gca()
    ax.set_xticks([x + 0.5 for x in range(cols)])
    ax.set_yticks([y + 0.5 for y in range(rows)])
    ax.set_xticklabels(range(0, cols))
    ax.set_yticklabels(range(0, rows))

    plt.title(f"Ground Map of size {rows}x{cols}")
    plt.xlabel("Width")
    plt.ylabel("Height")
    plt.gca().invert_yaxis()
    plt.subplots_adjust(bottom=0.2)
    plt.figtext(0.5, 0.05, f"Total area: {total_area} m² | Lawn area: {lawn_area} m² |
Lawn percentage: {lawn_percentage:.2f}%", ha='center', va='top', fontsize=10,
bbox=dict(facecolor='white', alpha=0.7))
```

small.csv Visualization:



Total area: 20 m² | Lawn area: 17 m² | Lawn percentage: 85.00%

my_map.csv Visualization:



Total area: 625 m² | Lawn area: 432 m² | Lawn percentage: 69.12%

2. **Area Calculations**

To calculate the total and lawn areas, the calculate_map_area function iterates through the map, counting all cells (total area) and those marked as 'L' (lawn area). Here's the code for this calculation:

```
def calculate_map_area(ground_map):
    total_area = 0
    lawn_area = 0

    for row in ground_map:
        for cell in row:
            total_area += 1
            if cell == 'L':
                lawn_area += 1

    lawn_percentage = (lawn_area / total_area) * 100 if total_area > 0 else 0

    print(f"Total map area: {total_area} m²")
    print(f"Lawn area: {lawn_area} m²")
    print(f"Lawn area percentage: {lawn_percentage:.2f}%")

    return total_area, lawn_area, lawn_percentage
```

Results for small.csv

- Total Area: 20 m²
- Lawn Area: 17 m²
- Lawn Percentage: 85%

```
Total area: 20 m² | Lawn area: 17 m² | Lawn percentage: 85.00%
```

Results for my_map.csv

- Total Area: 625 m²
- Lawn Area: 432 m²
- Lawn Percentage: 69,12%

```
Total area: 625 m² | Lawn area: 432 m² | Lawn percentage: 69.12%
```

3. **Description of Scenario (my_map.csv)**

The my_map.csv script depicts a lawn between a series of buildings that act as obstacles, creating a challenging environment to test the lawnmower's navigation capabilities. The obstacles are arranged as eight 4x6 blocks, each representing a house. This arrangement resembles a residential area with closely spaced houses, providing a test of the mower's ability to maneuver efficiently around large structured obstacles.

2.2 Coordinate Map

The **coordinate map** coord [X][Y] is generated from the ground_map, mapping each coordinate (X, Y) to a specific ground type: *Obstacle*, *Lawn*, or *Start*. This mapping allows our mower to distinguish between mowable areas, obstacles, and the starting point, enabling efficient and safe navigation across the lawn.

Coordinate Map Computation
To create the coordinate map, we iterate over each cell in ground_map to classify its content based on character values:

- 'O' represents an obstacle,
- 'L' represents lawn,
- 'S' represents the starting point.

We use the find_start_position function to locate the mower's initial position by searching for the 'S' character:

```
def find_start_position(ground_map):
    for y, row in enumerate(ground_map):
        for x, cell in enumerate(row):
            if cell == 'S':
                return x, y
    return None
```

This function:

1. Loops through each row and each cell in ground_map using enumerate to retrieve both the coordinates and the cell content.
2. When cell == 'S', it identifies (x, y) as the mower's starting point and returns these coordinates.

3. If no start position is found, it returns None.

Using this approach, we identify the exact coordinates where the mower begins mowing.

Checking if a Position is Outside the Lawn

In the function mow_lawn, the mower moves by calculating a new position (new_x, new_y) based on its current position (x, y) and the current direction. This function uses a set of conditions to check if (new_x, new_y) falls outside the lawn boundaries or encounters an obstacle. Here is the relevant code segment from mow_lawn:

```
def mow_lawn(ground_map, x, y, direction=None):
    ...
    while True:
        new_x, new_y = x + direction[0], y + direction[1]

        if not (0 <= new_x < len(ground_map[0]) and 0 <= new_y < len(ground_map)) or
ground_map[new_y][new_x] == 'O':
            direction = random_direction(exclude_direction=direction)
            continue
        ...
```

This section:

1. Calculates the New Position: new_x and new_y are determined by adding the directional offsets to the current coordinates.

2. Boundary Check: The condition 0 <= new_x < len(ground_map[0]) ensures new_x is within the lawn's width, and 0 <= new_y < len(ground_map) verifies new_y is within the lawn's height.

3. Obstacle Check: If ground_map[new_y][new_x] == 'O', it confirms that the mower is trying to move into an obstacle cell.

If any of these conditions fail (i.e., if (new_x, new_y) is outside the lawn or hits an obstacle), a new direction is selected using random_direction(exclude_direction=direction). The process then continues until a valid mowing path is identified.

Once a valid position is found, the mower:

Marks the Cell as Mowed: It checks if the current cell ground_map[y][x] is lawn ('L'), start ('S'), or an already mowed area ('E') and then marks it as cut ('C').

Updates Position: It moves to (new_x, new_y) and marks this new cell as visited ('E'), indicating that the mower has passed through this cell.

```
    ...

    if ground_map[y][x] in ['L', 'S', 'E']:
        ground_map[y][x] = 'C'

    x, y = new_x, new_y
    ground_map[y][x] = 'E'

    return x, y, direction
```

This logic ensures the mower stays within the lawn area, avoids obstacles, and continues mowing only within designated, mowable spaces.

2.3 Trace

In this project, the **trace** represents the mower's sequence of steps across the lawn as it moves from one position (xi, yi) to the next position (xi+1, yi+1). This sequence is created by simulating each movement step-by-step, adjusting directions when obstacles or lawn boundaries are encountered.

Step-by-Step Movement

Each step is computed within the mow_lawn function. The function determines the mower's next position (new_x, new_y) based on a given direction vector. Here's how it works:

- Initial Movement Direction: If no specific direction is provided, the mower starts by selecting a random direction using the random_direction() function.

- Calculating the Next Position: Given the current position (x, y), the mower calculates the next position (new_x, new_y) by adding the direction vector's components to the current coordinates. For example, if the current direction is (1, 0), the mower moves one step to the right, setting new_x = x + 1 and new_y = y.

- Handling Obstacle Bounces: If the mower encounters an obstacle ('O') or moves outside the lawn boundaries, it recalculates its direction using random_direction(exclude_direction=direction), which ensures it does not immediately reverse its path. The while loop in mow_lawn checks that the next position is valid (not an obstacle or out of bounds) before moving. If invalid, the mower chooses a new direction and tries again until it finds a valid move.

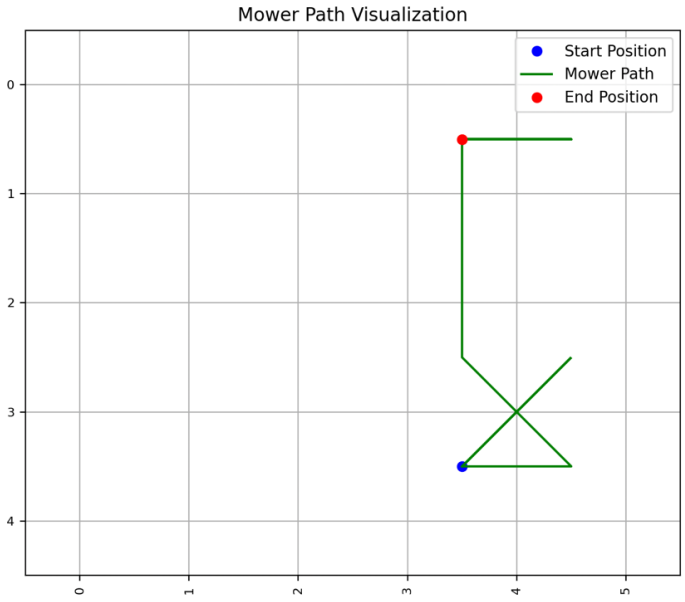
Here's a code snippet from the mow_lawn function illustrating these steps:

```
def mow_lawn(ground_map, x, y, direction=None):
    if direction is None:
        direction = random_direction()

    while True:
        new_x, new_y = x + direction[0], y + direction[1]

        if not (0 <= new_x < len(ground_map[0]) and 0 <= new_y < len(ground_map)) or
ground_map[new_y][new_x] == 'O':
            direction = random_direction(exclude_direction=direction)
            continue

        if ground_map[y][x] in ['L', 'S', 'E']:
            ground_map[y][x] = 'C'

        x, y = new_x, new_y
        ground_map[y][x] = 'E'

        return x, y, direction
```

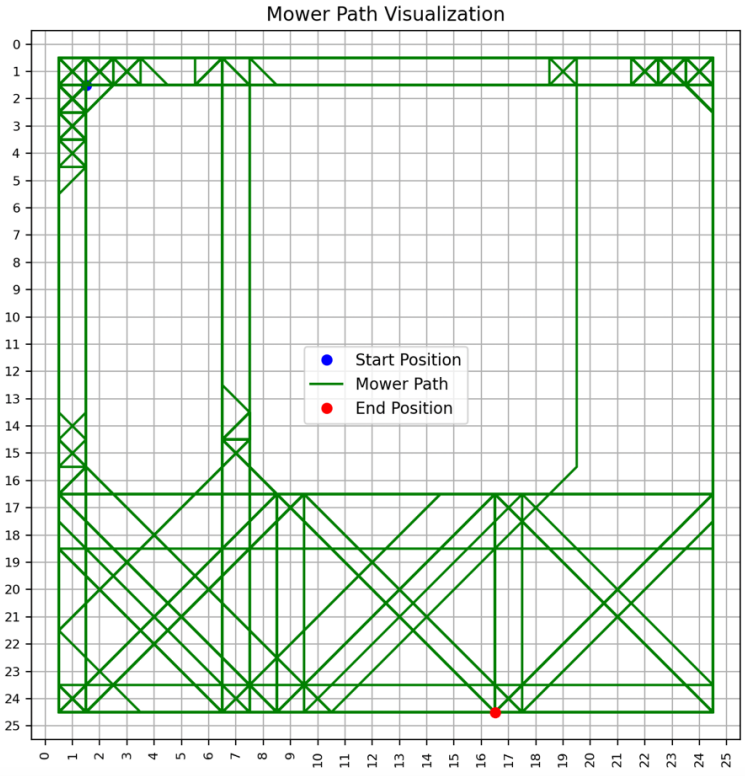
Visualizing the Trace

To better understand the mower's path, we used matplotlib to create visualizations of the trace for two maps: small.csv and my_map.csv. The trace plot shows the mower's start point, end point, and its movement path across the lawn, indicating where obstacles caused direction changes.

- small.csv Trace Visualization



- my_map.csv Trace Visualization



These trace plots help visualize the mower's step-by-step path and demonstrate how it responds to obstacles, giving insights into its coverage efficiency and movement pattern

3 Grade C

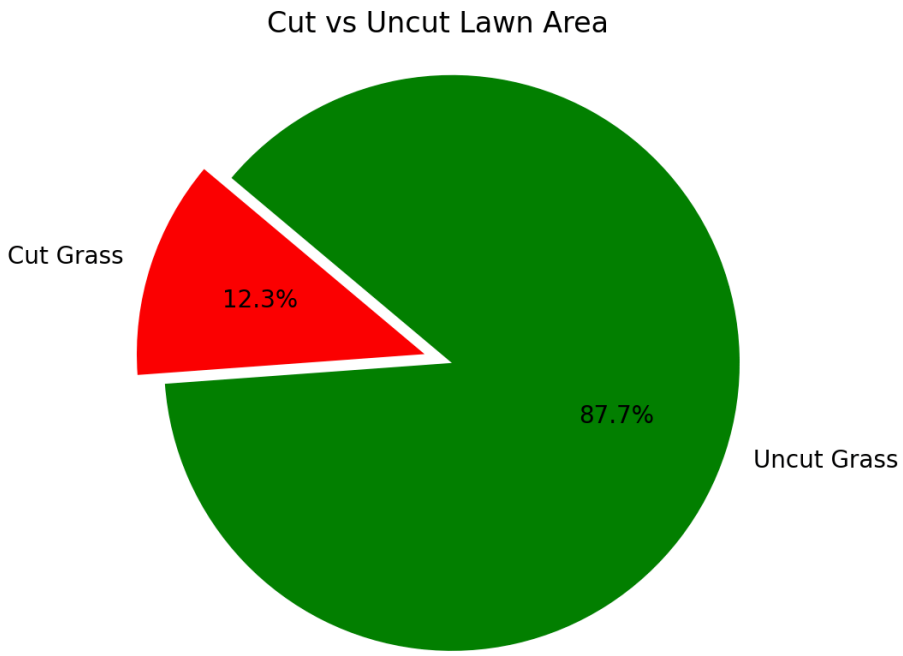
3.1 Coverage

To calculate coverage, we divide each square meter of the area into a finer $N \times N$ grid, where N represents the level of detail in our analysis. For example, for this report, we use a 5 × 5 grid (N = 5) to determine coverage after 2 hours for two land maps: small.csv and my_map.csv.

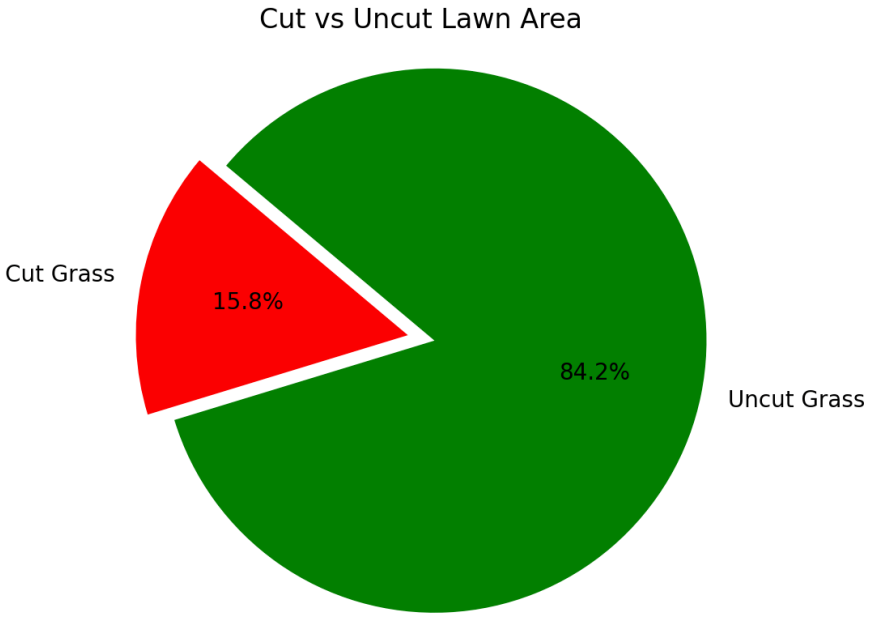
```
def main():
        ...
    N = 5
    speed = 0.3
    work_hours = 2
    total_time_seconds = work_hours * 3600
    max_steps = int(total_time_seconds * speed)
        ...
```

• What is the coverage after 2 hours for the ground maps small.csv and my_map.csv using a 5 x 5 grid?

- Cut and Uncut Lawn Area Chart for detailed Ground Map small.csv:

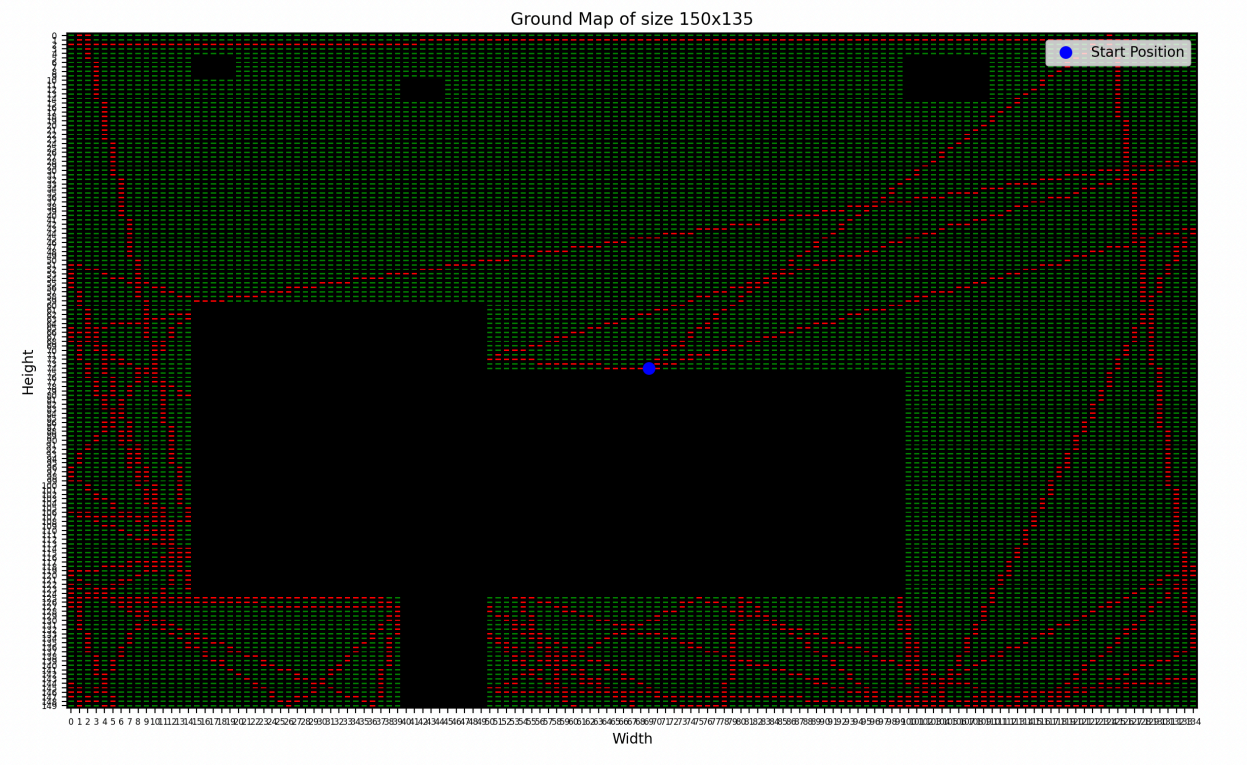


- Cut and Uncut Lawn Area Chart for detailed Ground Map my_map.csv:

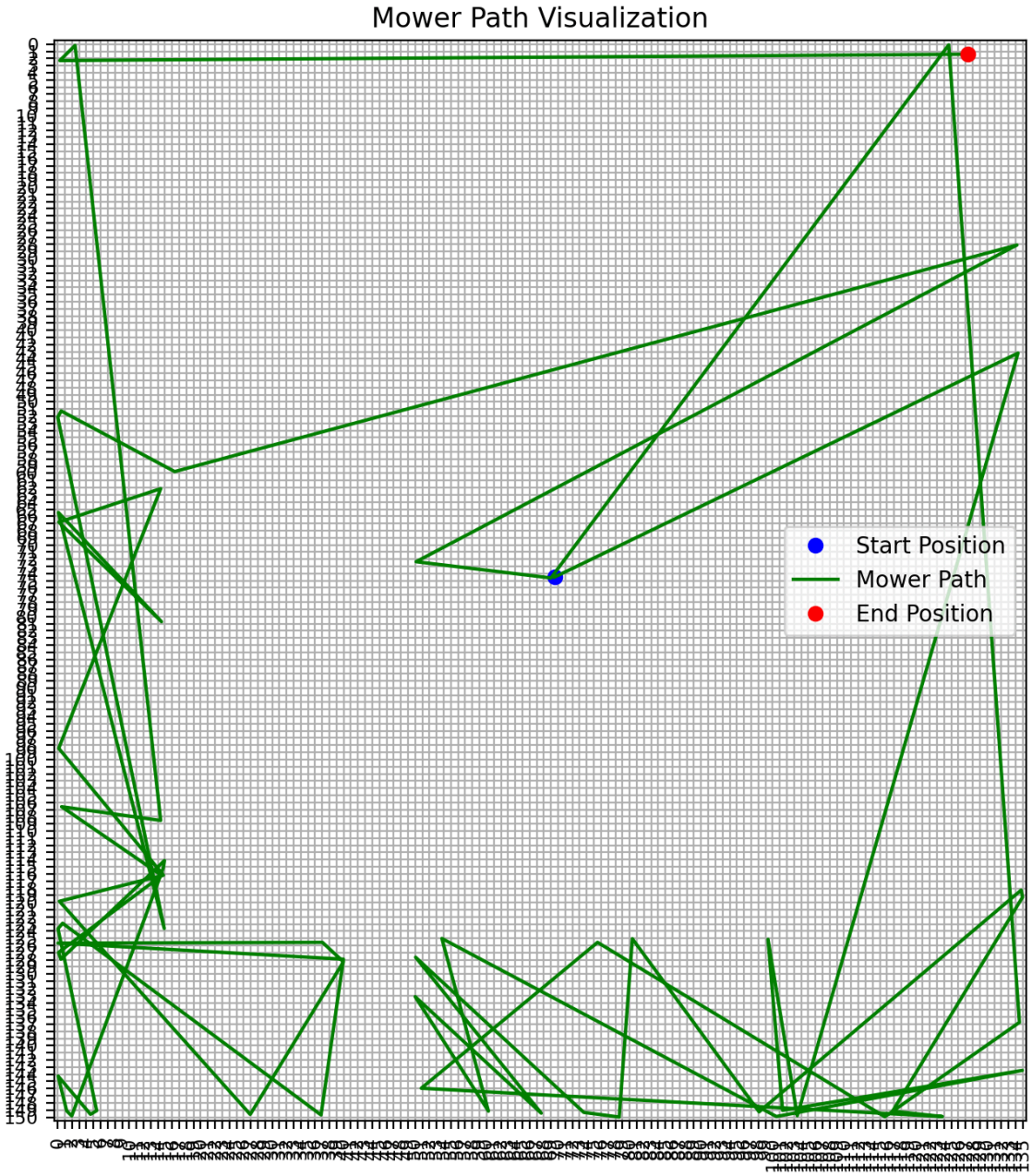


Results after 2 hours:

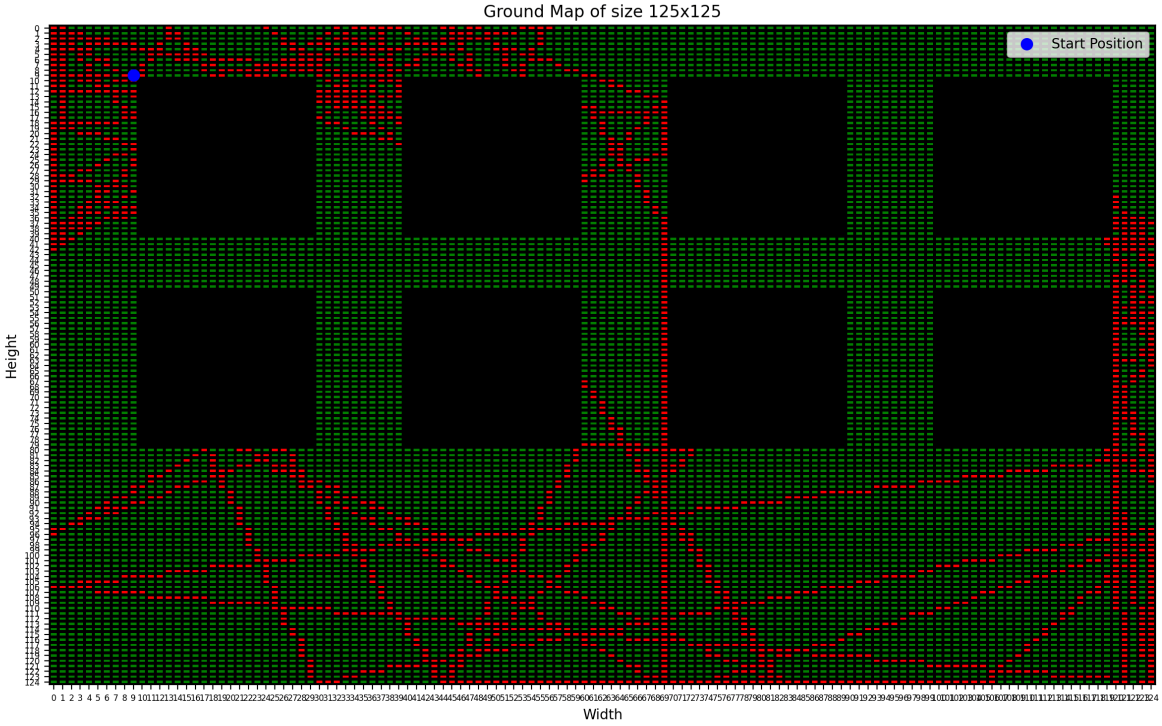
- Detailed Ground Map Visualization small.csv:



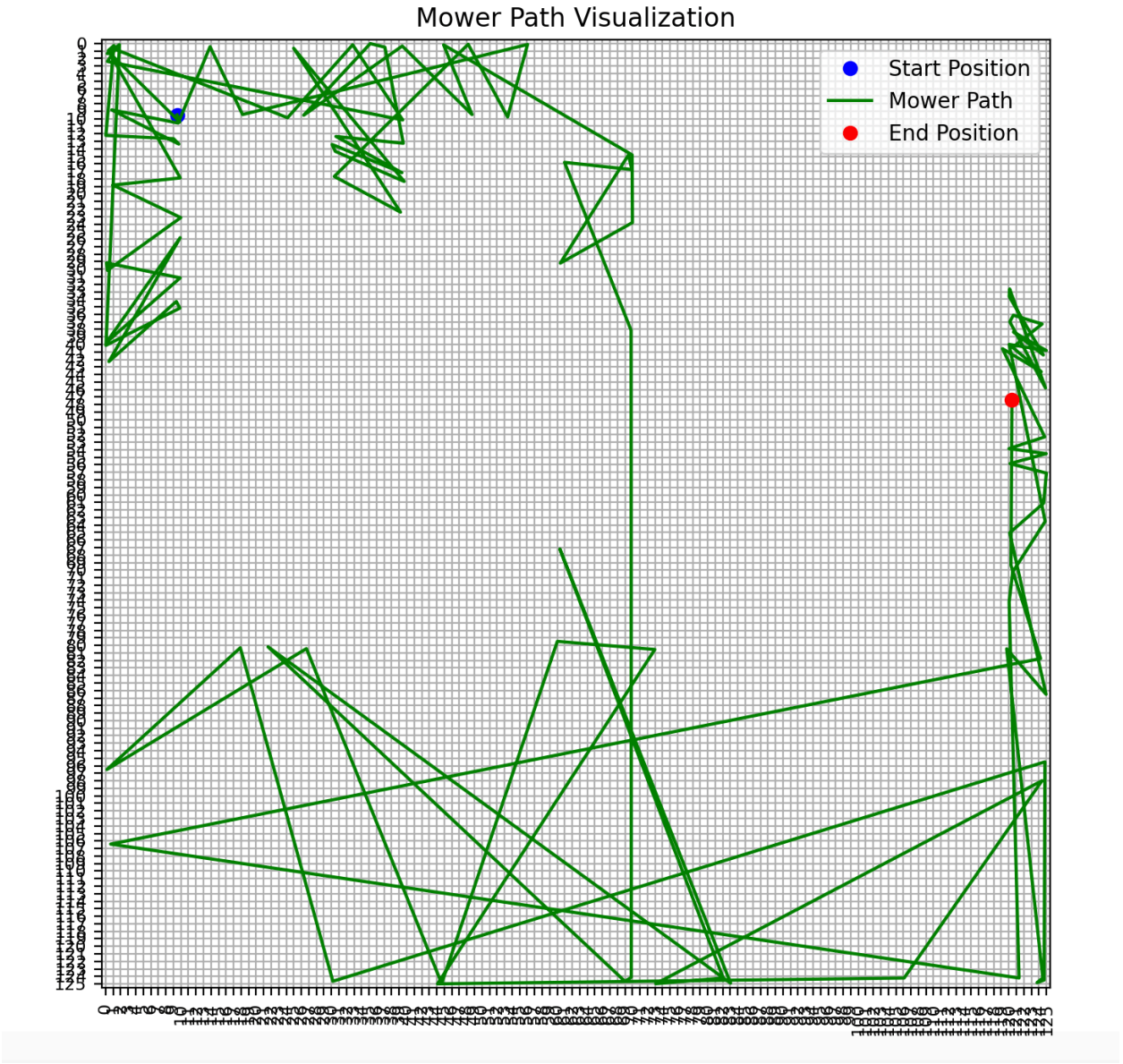
- Detailed Mower Path Visualization small.csv:



- Detailed Ground Map Visualization: my_map.csv:



- Detailed Mower Path Visualization: my_map.csv:



• How do the two parameters Δt and N influence the coverage?

1. Parameter N - coverage resolution

The code has a function generate_detailed_map, which generates a detailed map according to the given value of N. This function divides each square meter into N×N cells, which increases the accuracy of coverage measurement.

```
def generate_detailed_map(ground_map, N):
    detailed_map = []

    for row in ground_map:
        expanded_row = []
        for cell in row:
            expanded_cell = [cell] * N
```

```
            expanded_row.extend(expanded_cell)
        for _ in range(N):
            detailed_map.append(expanded_row.copy())

    return detailed_map
```

Thus, changing N affects the detail of the map and allows you to measure coverage with greater or lesser accuracy, depending on the chosen value of N.

2. Parameter Δt - time interval between steps

Our code does not have a parameter Δt that would regulate the time interval between the steps of the mower's movement. But our code does have speed and work that determine how long the mower works.

We also have a function that stops the mower when all the grass is mowed.

```
    ...
    speed = 0.3
    work_hours = 2
    total_time_seconds = work_hours * 3600
    max_steps = int(total_time_seconds * speed)

    for _ in range(max_steps):
        if not is_lawn_left(ground_map):
            print("All grass has been cut.")
            break
        new_x, new_y, direction = mow_lawn(ground_map, start_x, start_y, direction)

       ...
```

• What can you say about the effective cutting width? What parameters influence it? Motivate your answers.

The effective cutting width refers to the width of the area that the robot covers in each pass. This width depends on multiple factors:

- Grid Size (N): When using an N×N grid, the cutting width effectively becomes narrower with finer grid sizes, as more precision is required to mark coverage for each cell within the grid. A larger N results in a more detailed map, showing the mower’s coverage with higher accuracy, which can effectively reduce the coverage width per step due to the finer granularity.

- Movement Direction and Step Size: If the robot changes direction frequently or takes smaller steps, the effective width per pass can be narrower as well, especially if it follows a more random path or is instructed to cover irregular sections of the map.

- Robot's Path Consistency: If the robot follows a straight, systematic path (like a lawnmower pattern), the effective width will be closer to the actual width of the mower blade. In cases where the robot moves randomly or re-cuts areas, the effective width may vary, reducing overall efficiency.

The effective cutting width is influenced by grid resolution (parameter N), the robot's movement strategy, and any changes in direction. A larger grid size and systematic path generally maximize the effective width, improving efficiency.

• What is a reasonable workload (hours of work per day) for the robot to handle ground map small.csv?

The reasonable workload for the robot, in terms of hours of work per day, depends on various factors, including:

Size of the Lawn: The total area that needs to be mowed directly influences the time required. Larger areas will naturally require more time to cover.

Mowing Speed: The speed at which the robot operates affects how quickly it can mow the lawn. Faster speeds may allow for more area to be covered in less time.

Terrain Complexity: The presence of obstacles (e.g., trees, flower beds) and varying terrain types can slow down the mowing process, as the robot may need to navigate around these challenges.

Battery Life: The duration for which the robot can operate on a single charge will limit the amount of time it can work each day.

Grass Growth and Density: The thickness and height of the grass can impact mowing efficiency. Denser or taller grass may require more time and effort to cut effectively.

Weather Conditions: Rain or wet ground may prevent the robot from mowing efficiently, potentially reducing the effective working hours.

Considering these factors, a reasonable workload might range from a few hours to a full day, depending on the specific conditions of the lawn as outlined in the ground map (small.csv).

4 Key features we believe are necessary

To improve the user experience and provide clear insights into the lawn mowing process, we have added new visualizations and a user-friendly main menu. These features make it easier for users to interact with the program and understand complex data. The main menu acts as a central hub, allowing users to quickly move between different visualizations, each designed to provide specific information about the mowing process. By combining these visual tools with an easy-to-use interface, we want to help users make better decisions based on real-time data, ultimately improving their lawn care efforts.

4.1 Visualization of lawn data

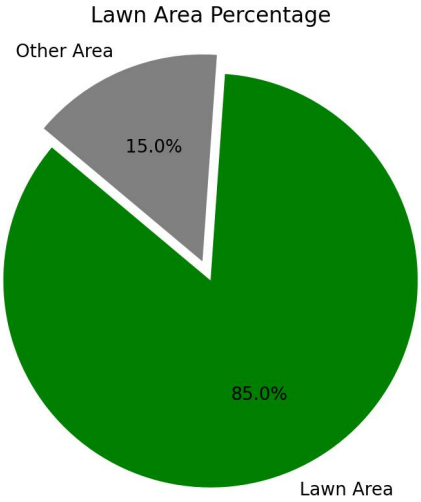
As part of our project, we have added diagrams to better visualize the data related to the lawn mowing process. These visualizations provide a clear and informative representation of our results, making it easier to understand and analyze the performance of our machine. Below are the details of the diagrams we implemented:

Lawn Area Percentage Chart

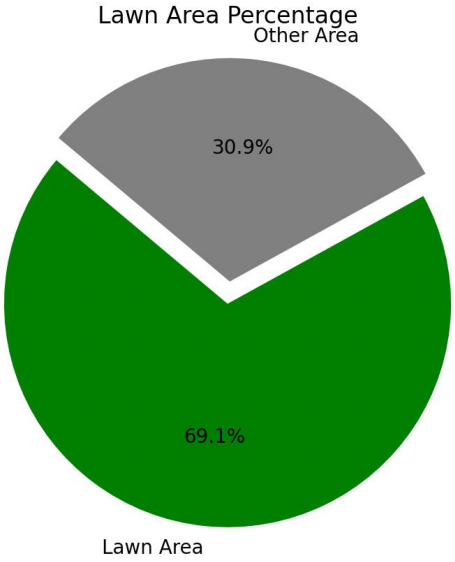
We created a pie chart that illustrates the percentage of the lawn area compared to other areas. This chart allows us to quickly see how much of the total area is dedicated to the lawn. It is essential for assessing the scope of the mowing task and ensuring that we allocate the right resources.

```
def plot_lawn_percentage_chart(lawn_percentage):
    plt.figure()
    labels = 'Lawn Area', 'Other Area'
    sizes = [lawn_percentage, 100 - lawn_percentage]
    colors = ['green', 'gray']
    explode = (0.1, 0)

    plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)
    plt.title("Lawn Area Percentage")
    plt.axis('equal')
```

- small.csv Percentage Chart



- my_map.csv Percentage Chart



Cut vs. Uncut Grass Chart

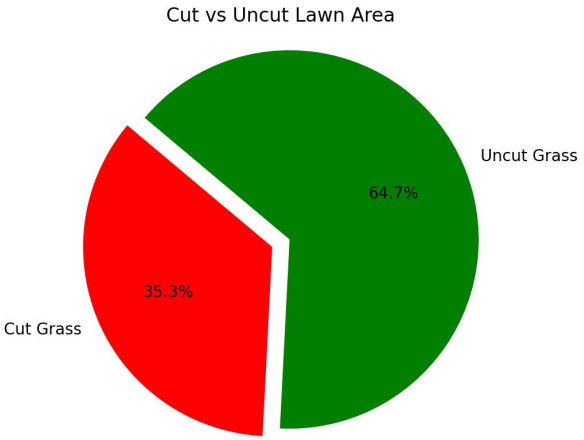
Additionally, we implemented a chart to compare the cut grass versus uncut grass. This pie chart shows the percentage of the lawn that has been successfully cut and the portion that remains uncut. This visualization is crucial for evaluating the effectiveness of the mowing process and identifying areas that need more attention.

```
def plot_cut_uncut_chart(ground_map):
    total_lawn_cells = 0
    cut_cells = 0

    for row in ground_map:
        for cell in row:
            if cell == 'L' or cell == 'C':
                total_lawn_cells += 1
                if cell == 'C':
                    cut_cells += 1
```

```
    cut_percentage = (cut_cells / total_lawn_cells) * 100 if total_lawn_cells > 0 else
0
    uncut_percentage = 100 - cut_percentage

    plt.figure()
    labels = 'Cut Grass', 'Uncut Grass'
    sizes = [cut_percentage, uncut_percentage]
    colors = ['red', 'green']
    explode = (0.1, 0)

    plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)
    plt.title("Cut vs Uncut Lawn Area")
    plt.axis('equal')
```

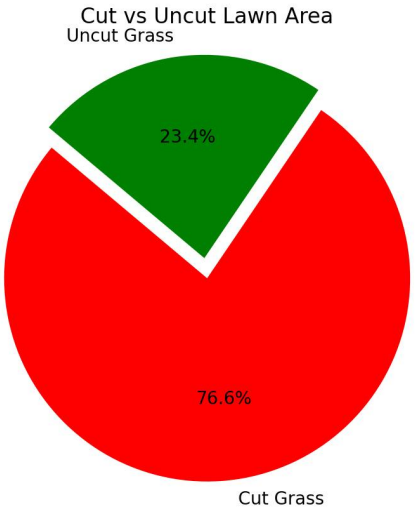
- small.csv Cut Uncut Chart

The operation of a lawn mower with a speed of 0.01, which worked for 0.3 hours



- my_map.csv Cut Uncut Chart

The operation of a lawn mower with a speed of 0.4, which worked for 13 hours



Importance of the Diagrams

These diagrams play a significant role in our analysis as they provide immediate visual feedback about the mowing process. They help us:

- Assess Coverage: Understand how much of the lawn area has been covered versus what remains.

- Evaluate Efficiency: Determine the effectiveness of our mowing strategy.

- Inform Decision-Making: Make data-driven decisions on how to improve future mowing operations.

4.2 Main menu functionality

In our project, we have implemented a main menu that allows users to interact with various visualizations related to the lawn mowing process. This menu is designed to enhance user experience by providing easy access to key features and analyses of the lawn area. Below is a description of the menu and its components, highlighting their purpose and importance.

Purpose of the Main Menu

The main menu serves as a central hub for users to choose different visualizations and analyses of the lawn data. By organizing these options in a clear and concise manner, we aim to make it easy for users to navigate through the program and access valuable insights about the mowing process.

Menu Options

1. **Ground Map Visualization**:

This option displays the layout of the ground map, providing a visual representation of the areas to be mowed. It helps users understand the initial conditions of the lawn, including obstacles and designated mowing areas.

2. **Mower Path Visualization**:

Users can visualize the path taken by the mower during its operation. This feature helps illustrate how effectively the mower covers the designated area and allows for analysis of its movement patterns.

3. **Lawn Area Percentage Chart**:

This chart shows the proportion of the total area that is lawn versus other areas. It provides a quick reference for understanding the scale of the mowing task and the distribution of different land types.

4. **Cut and Uncut Lawn Area Chart**:

This visualization compares the areas that have been cut with those that remain uncut. It is crucial for assessing the effectiveness of the mowing operation and identifying any areas that may need additional attention.

5. Detailed Ground Map Visualization

This function displays an expanded version of the ground map, dividing each square meter into a smaller N x N grid. It provides a detailed view of the terrain, highlighting obstacles, lawn areas, and the starting position.

6. Detailed Mower Path Visualization

This function visualizes the mower's path on the detailed map, also divided into an N x N grid. It illustrates the mower's movement through different terrains, helping users assess its effectiveness and coverage within the more granular layout.

7. **Detailed Cut and Uncut Lawn Area Chart**

This function generates a chart comparing the areas of the detailed map that have been cut versus those that remain uncut. It allows users to evaluate the mowing operation's effectiveness and identify specific areas needing additional attention, benefiting from the increased detail of the grid division.

0. **Exit**:

This option allows users to exit the program gracefully, ensuring that they can easily conclude their session.

Importance of the Menu

The main menu is an essential component of our project for several reasons:

- **User-Friendly**: It simplifies navigation, allowing users to access different features without needing to understand the underlying code.
- **Interactive**: The menu encourages user engagement by offering real-time visual feedback based on their choices, enhancing the overall experience.
- **Data-Driven Decisions**: By providing various visualizations, the menu helps users make informed decisions about the mowing process and future improvements.

Code Snippet

Here is a brief overview of the main function that creates and manages the menu:

```
def main():

    # … other menu parts…

while True:
        print("\nChoose a visualization to display:")
        print("1 - Ground Map Visualization")
        print("2 - Mower Path Visualization")
        print("3 - Lawn Area Percentage Chart")
        print("4 - Cut and Uncut Lawn Area Chart")
        print("5 - Detailed Ground Map Visualization")
        print("6 - Detailed Mower Path Visualization")
        print("0 - Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            plot_ground_map(ground_map, total_area, lawn_area, lawn_percentage)
            plt.show()
        elif choice == '2':
            move_visualization(path, initial_x, initial_y, ground_map, start_x,
start_y)
```

```
            plt.show()
        elif choice == '3':
            plot_lawn_percentage_chart(lawn_percentage)
            plt.show()
        elif choice == '4':
            plot_cut_uncut_chart(ground_map)
            plt.show()
        elif choice == '5':
            plot_ground_detailed_map(detailed_map, det_start_position)
            plt.show()
        elif choice == '6':
            move_visualization(det_path, det_initial_x, det_initial_y, detailed_map,
det_start_x, det_start_y)
            plt.show()
        elif choice == '7':
            plot_cut_uncut_chart(detailed_map)
            plt.show()

        elif choice == '0':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")
```

Below is a visual representation of how the menu works:

```
Choose a visualization to display:
1 - Ground Map Visualization
2 - Mower Path Visualization
3 - Lawn Area Percentage Chart
4 - Cut and Uncut Lawn Area Chart
5 - Detailed: Ground Map Visualization
6 - Detailed: Mower Path Visualization
7 - Detailed: Cut and Uncut Lawn Area Chart
0 - Exit
Enter your choice:
```

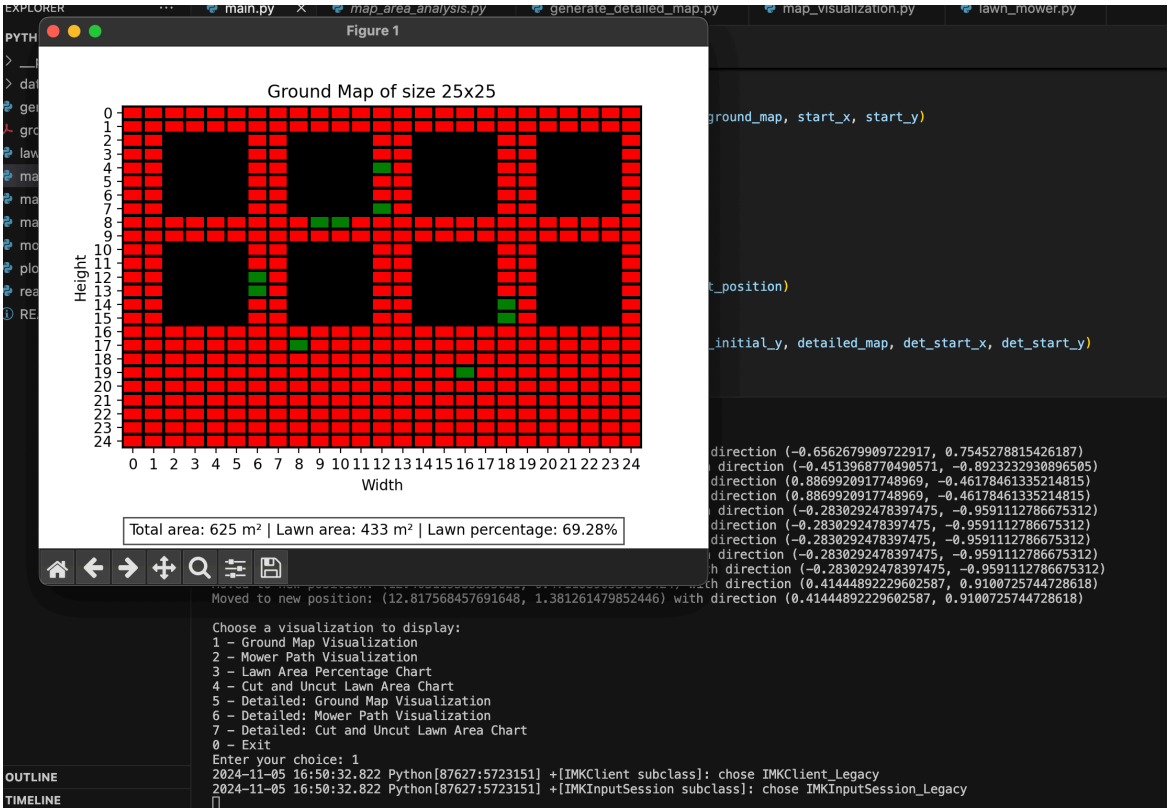


Figure 1
Ground Map of size 25x25
Height
Width
Total area: 625 m² | Lawn area: 433 m² | Lawn percentage: 69.28%
Choose a visualization to display:
1 - Ground Map Visualization
2 - Mower Path Visualization
3 - Lawn Area Percentage Chart
4 - Cut and Uncut Lawn Area Chart
5 - Detailed: Ground Map Visualization
6 - Detailed: Mower Path Visualization
7 - Detailed: Cut and Uncut Lawn Area Chart
0 - Exit
Enter your choice: 1
2024-11-05 16:50:32.822 Python[87627:5723151] +[IMKClient subclass]: chose IMKClient_Legacy
2024-11-05 16:50:32.822 Python[87627:5723151] +[IMKInputSession subclass]: chose IMKInputSession_Legacy

5 Project conclusions and lessons learned

5.1 Technical issues

• What were the major technical challenges as you see it? What parts were the hardest and most time consuming?

Throughout the project, we encountered several significant technical challenges:
- Algorithm Development: One of the most challenging aspects was devising the correct algorithm to ensure that the lawn mower did not change direction until it encountered an obstacle or reached the edge of the map. This required us to think critically about the logic and flow of the program to manage the mower's path effectively.
- Simultaneous Visualizations: We struggled with the implementation of displaying two visualizations concurrently. Finding a way to manage the rendering of both visual elements without compromising performance or clarity was a complex task that took considerable time and effort.
- Label Positioning on the Map Visualization: Another challenge involved adjusting the labels on the map visualization. We needed to ensure that the labels were appropriately aligned and did not overlap with other visual elements, which required fine-tuning of the positioning logic.

Overall, these challenges were not only time-consuming but also required us to engage in extensive troubleshooting and collaboration to find effective solutions.

• How could the results be improved if you were given a bit more time to complete the task.

If we had extra time to work on this project, we would make some important changes to improve our results. This includes:
- We would utilize the recommended method, which would help us achieve better results and make the process more efficient. By adopting this approach, we would improve the overall effectiveness of our project and enhance our final outcomes.
- We would pay more attention to the specific tasks listed in points A, B, and C.
- We would also add a function to determine how long the machine needs to operate until it has mowed the entire area of grass. This way, we can ensure complete coverage while tracking the total time required for the task.

5.2 Project issues

Describe how your team organized the work. How did you communicate? How often did you communicate?

Our team adopted a structured approach to organizing our work on the Robotic Lawnmower project. We recognized the importance of clear communication and collaboration to ensure that each aspect of the project was completed efficiently and effectively.To facilitate our communication, we utilized several tools:

- **Google Meet**: We held regular video calls via Google Meet to discuss progress, share updates, and brainstorm ideas. These meetings allowed us to connect face-to-face, fostering better collaboration and understanding among team members. We scheduled these meetings once-twice a day, but additional sessions were held as needed during critical phases of the project.
- **Telegram**: Also, for communication, we use Telegram. This platform enabled us to share quick updates, ask questions, and provide support to one another in real-time. We found Telegram's instant messaging capabilities to be particularly useful for resolving minor issues promptly without waiting for our next scheduled meeting.
- **GitHub**: We used GitHub for version control and to manage our codebase collaboratively. Each team member was responsible for pushing their changes to designated branches, which helped keep our code organized and allowed us to track modifications effectively. Regular commits ensured that we could easily revert to earlier versions if necessary, and GitHub's issue tracking feature helped us prioritize tasks and manage deadlines.

Our repository: https://github.com/yarynaamryhlotska/python_project.git

- **Library Study Sessions**: In addition to our virtual communication, we also organized in-person study sessions at the library. These gatherings provided a focused environment where we could work together on complex tasks, discuss specific challenges, and collaborate more effectively.

Overall, our communication strategy was multi-faceted, combining digital tools and in-person collaboration to enhance our productivity.

Describe which parts (or subtasks) of the project they were responsible for. Consider writing the report as a separate task. Try to identify main contributors and co-contributors.

In our team, we utilized pair programming throughout the project. This approach allowed us to collaboratively write code, where one member focused on coding while the other reviewed, provided feedback, and offered suggestions. As a result, it is challenging to specify which parts or subtasks were completed by individual members, as both contributors were involved in the development process simultaneously.

Both team members played equal roles as main contributors, actively participating in all aspects of the project. Our collaboration fostered a shared understanding of the code and improved overall code quality through continuous review.

Estimate hours spend each week (on average)

Over the course of six days, each team member dedicated approximately 26 hours to the project. On average, this equates to around 4.35 hours per person, per day, focused on planning, coding, troubleshooting, and meetings. This commitment allowed us to make steady progress, allocate time effectively across different stages of the project, and ensure we met our objectives within a limited timeframe.

Our team spent a total of 52 man-hours to complete the project.

What lessons have you learned?

Thorough Planning: We recognized the importance of detailed planning for project success. A well-defined project plan, including clear goals, milestones, tasks, timelines, and required resources, provides a roadmap that keeps the team focused and organized.

Adaptability and Flexibility: Projects often come with unexpected challenges, making it essential to remain flexible. We learned the importance of being open to strategic changes and adapting our approach to meet new challenges effectively.

Effective Communication: Open communication within the team is vital, especially in a small team of just two people where all project discussions and decisions directly impact our progress. Regular check-ins and updates between us ensured that we stayed aligned, could quickly address any issues, and moved forward efficiently with a shared understanding of the project goals.

What should you have done differently if you now were facing a similar project.

If faced with a similar project in the future, we would consider documenting specific tasks more clearly, even within the pair programming framework. This could involve assigning distinct roles for certain subtasks to better track contributions and facilitate a more detailed evaluation of individual performance. Additionally, setting aside time for retrospective discussions could help us identify areas for improvement throughout the project.