

HLD document-Advanced Topics in Programming C++

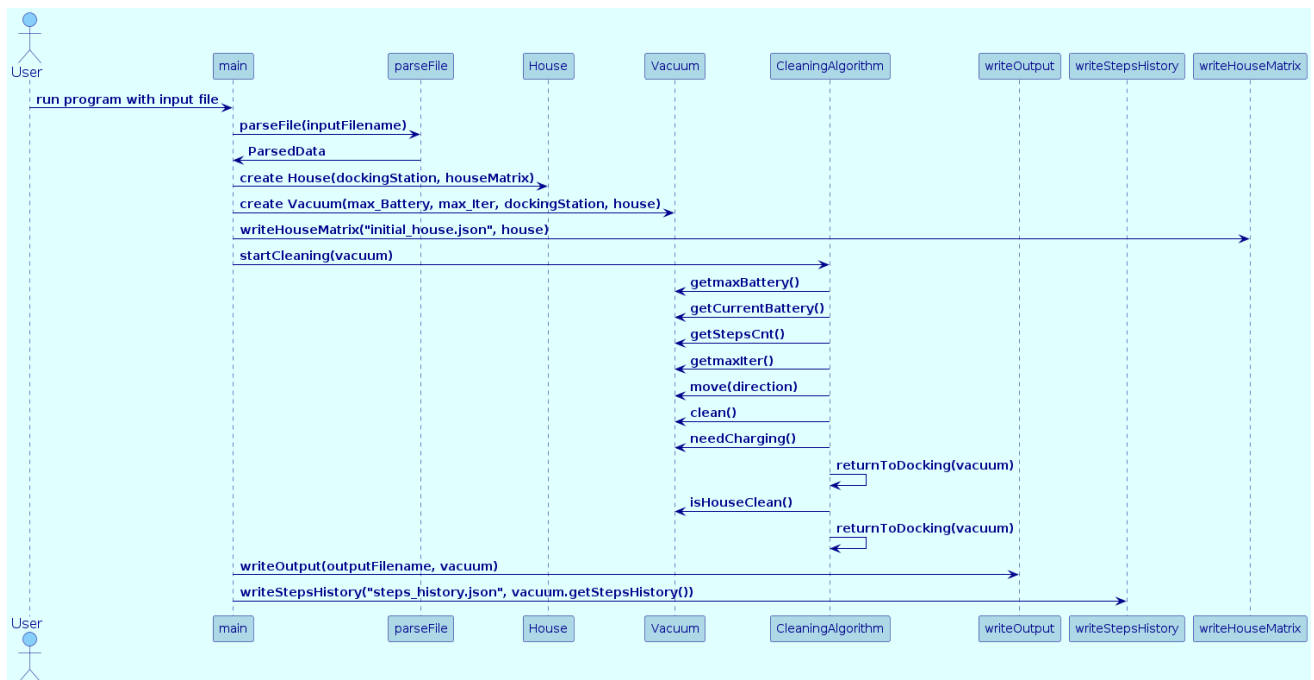
Assignment 1

Odai Agbaria 212609440, Mohammad Mahamid:322776063

1.UML class diagram:



2.UML sequence diagram:



3.Explanations of the design considerations and alternatives:

Before we write our consideration and the alternatives we considered, the most important thing is to explain how the house input file should be. The house input file should follow a specific format to ensure that the vacuum cleaner can correctly interpret and navigate the environment. The file is structured as follows:

1. House Matrix:

- The first part of the file contains the matrix representing the house layout. Each line corresponds to a row in the house matrix, each row has integers(-1 till 9) **separated by spaces**.
- The matrix elements are integers, where each number represents a specific type of cell:
 - -1 indicates a wall.
 - 0 indicates an empty space.
 - Any positive integer (1-9) indicates the amount of dirt in that cell.
- Each row in the matrix can have a different number of columns, but the matrix will be padded with -1 (walls) to ensure uniform dimensions(that's important, since ' ' spaces are not considered zeros, as you suggested in the document, if there is a line which has less numbers(columns) than the other then it is padded with walls).

2. Docking Station:

- After the house matrix, there is a line specifying the coordinates of the docking station. The coordinates are given as two integers separated by a space, representing the row and column indices, respectively.

- The docking station must be placed on a non-wall cell.
- 3. **Maximum Battery:** The next line specifies the maximum battery capacity of the vacuum cleaner.
- 4. **Maximum Iterations:** The final line specifies the maximum number of iterations (steps) the vacuum cleaner can perform during the cleaning process.

You can take a look at the input files which are given as examples and visualize the result to see it better.

Design Considerations:

Class Structure:


- **House Class:** Represents the environment in which the vacuum cleaner operates. It contains the matrix of the house and the docking station position. It has getter and setter methods to access and modify these attributes.
- **Vacuum Class:** Represents the vacuum cleaner. It maintains the current position, battery level, steps history, and interaction with the house. It includes methods like Move, clean, needCharging, and charge. The CleaningAlgorithm interacts with these functions to make decisions based on the vacuum's sensors (needCharging, getStepsCnt,...).
- **CleaningAlgorithm Class:** Contains the cleaning logic, including how the vacuum cleaner moves and when it decides to return to the docking station. In this class the algorithm determines the next step, which is a random move except for the return to docking.


Use of STL Containers:

- Utilized std::vector for the house matrix to provide dynamic sizing and easy access.
- Used std::tuple to represent positions as pairs of integers.
- Employed std::queue to manage the history of steps and directions taken by the vacuum cleaner.

Error Handling:

- Implemented robust error handling in the parseFile function to ensure input validation.
- Used standard output streams (std::cerr) to log errors and return codes to indicate failure.

 **JSON Serialization (for Bonus):** Provided two functions to serialize the initial house matrix and steps history into JSON format for visualization.

 **Visualization (for Bonus):** Created a Python script using Pygame to visualize the vacuum cleaner's movement and house cleaning process, Included the option to visualize the cleaning process step by step.

I will provide a bonus.txt to explain these additional features.

Alternatives Considered:

Data Structures:

- Considered using a 2D array instead of `std::vector`, but `std::vector` provides better flexibility and safety in modern C++.
- Considered using `std::vector` instead of `std::queue` for the steps history but chose `std::queue` as it is more compatible with the concept of step history.

🚦 Algorithm Complexity:

- considered BFS/DFS for pathfinding and cleaning but opted for a simpler random walk with checks for walls and charging to match the assignment requirements. More advanced algorithms will be implemented in the next assignments as you mentioned in the assignment.

🚦 Class Structure:

- Considered integrating the house matrix directly into the Vacuum class but decided to encapsulate it in a separate House class for better modularity and separation of concerns.

4.Explanations of the testing approach:

To ensure the correctness of the program, we implemented the following testing strategies:

1. Testing parseFile Function:

- Added print statements to verify the original matrix and the expanded matrix(after padding the walls).
- Ensured that the function correctly parses input files and handles edge cases such as invalid values (integer values smaller than -1 for example), out-of-bounds docking stations, and non-positive battery or negative iteration values.

2. Testing CleaningAlgorithm:

- Added print statements to track the vacuum's movement and house state.
- Verified that the vacuum interacts correctly with the house and performs cleaning and charging as expected.
- Used the visualization script to visually inspect the vacuum's behavior, ensuring it matches the expected outcomes.

3. Edge Cases:

- Tested scenarios with different house configurations, such as houses with no walls surrounded, varying amounts of dirt, docking station on a wall.
- Verified the vacuum's behaviour when starting at different positions, with varying battery levels, and with different maximum iteration limits.
- Ensured the program handles invalid inputs gracefully, such as integer values which are not in the range -1 to 9, non integer values, out-of-bounds docking stations, docking station on a wall, and negative battery (battery also cannot be zero) or iteration values.

Note: we decided that the max battery cannot be zero, it should be positive, the max iteration can be zero but then there are no steps made, and the visualization would show the vacuum in the docking station, and the “end of simulation” will appear immediately.