Design Considerations:

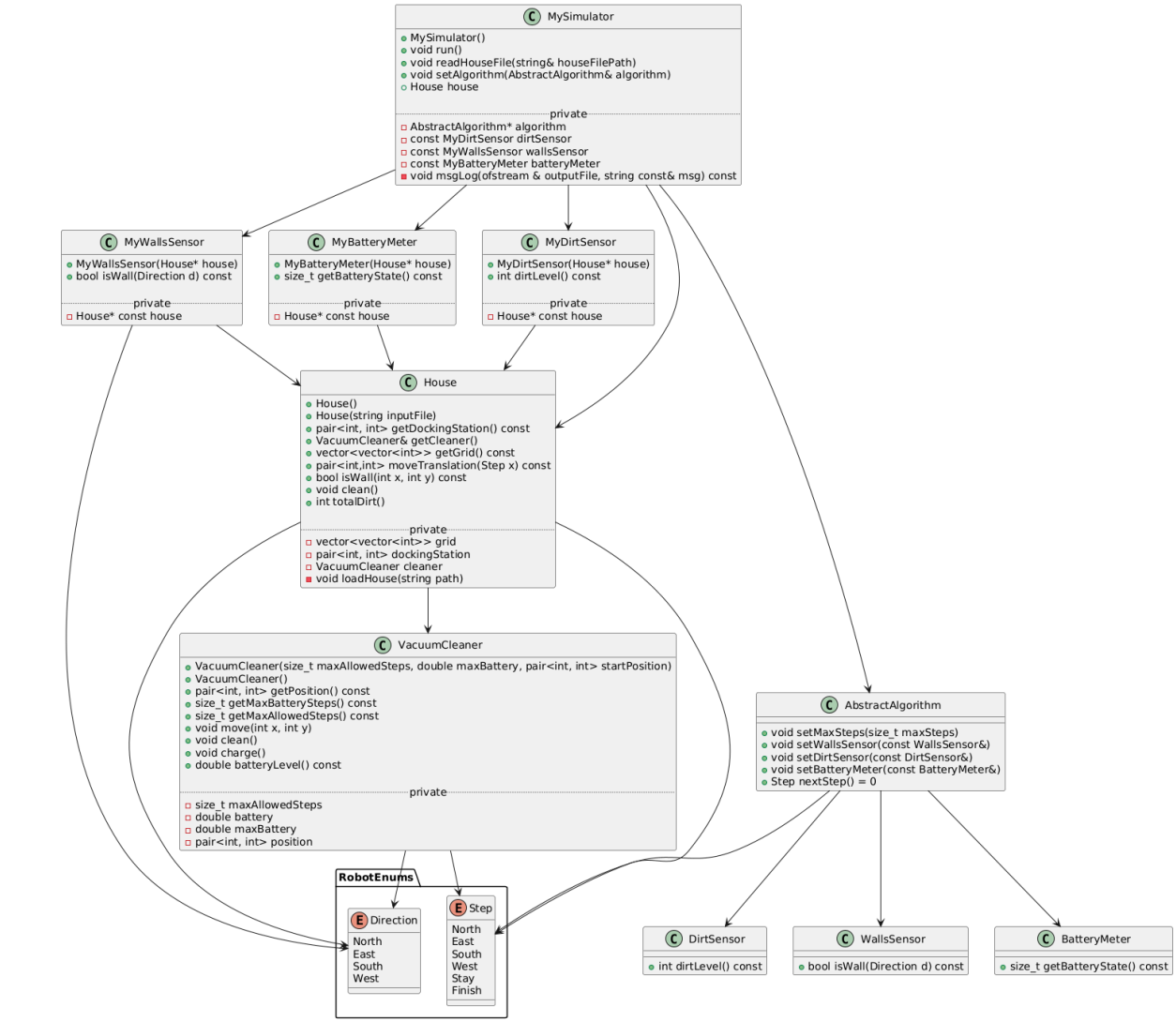
Given Constraints:

- **Skeleton Header Files**: Header (.h) skeleton files for various sensors were provided.
- **Restrictions**: The skeleton files imposed certain restrictions, such as the inability to use modern C++ features like smart pointers (unique_ptr, shared_ptr).

Implementation Strategy:

1. **Creating Implementation Headers and Source Files**:
    - For each provided sensor header file, a corresponding implementation header file was created, inheriting from the provided skeleton header file.
    - Additionally, a source file was created to implement the functions declared in the implementation header file.
2. **Encapsulation**:
    - All attributes within the classes are set to `private` to ensure encapsulation.
    - Getter and setter methods are used to access and modify private attributes, providing controlled access and ensuring data integrity.
3. **Const-Correctness**:
    - Attributes that are not meant to be modified are marked as `const`.
    - Non-modifying member functions are also marked as `const` to indicate that they do not alter the object's state.
4. **Use of Raw Pointers**:
    - Due to the constraints imposed by the skeleton files, raw pointers are used instead of modern C++ smart pointers.
    - This decision ensures compatibility with the provided skeleton files, even though it involves more manual memory management.
5. **Algorithm:**
    - Our algorithm is kind of dfs-like, exploring the house, cleaning when seeing dirt and returning to charge when needed.
    - We decided to make decision based on this priority:
      (Known Dirt Level > 0) > (Unknown non wall) > (Known Clean Position)
      We chose this priority because it made sense and had good results on a few tests we ran.
6. **Testing:**
    - We tested edge cases locally to ensure we are aligned with the submission guidelines
    - Created function to handle .txt file correctly.

UML Class Diagram:



UML Sequence Diagram: