# Hoover Service API Test Coverage & Analysis Overview

## Coverage Overview

The project is organized into distinct folders and files to ensure modularity, readability, and scalability for automated API testing.

### Cypress + Gherkin

- This is the core directory where all Cypress-related tests and configurations are stored.
- It contains subdirectories for API feature files, step definitions, custom commands, and API Helpers.
- The use of Gherkin allows the test scenarios to be written in a human-readable format, making it easier for both technical and non-technical stakeholders to understand the test cases.

### Feature Files

- This folder holds the Gherkin feature files, which are split into two based on functionality:
- hooverDirtCleaner.feature: Contains scenarios that test the API's ability to remove dirt patches in different patterns.
- hooverMovement.feature: Focuses on testing the hoover's movement behavior across the room.
- By splitting the feature files into two, it helps isolate concerns and test features independently, making the tests easier to maintain and scale.
- Each feature file is broken down into happy paths, edge cases, and error scenarios

### Step Files

- This folder contains the step definitions corresponding to the steps described in the Gherkin feature files.
- hoover.js: Implements the behavior for each step. This separation keeps the feature files clean and focuses on defining test logic rather than implementation details.
- Gherkin's **Given-When-Then** structure clearly defines the preconditions, actions, and expected outcomes for each test, ensuring clarity and consistency.

# Analysis

## Passing Feature: Hoover Navigation

The hooverMovement.feature has passed all test scenarios, demonstrating that the API handles the hoover's movement functionality as expected.

**Passing Tests (14 total):**

- Hoover can move in all directions (north, south, east, west).
- Hoover can navigate through every coordinate: The hoover successfully covers all positions within the room's boundaries.

**Boundary and Wall Handling:**

- Hoover moves in valid direction after hitting the wall: The hoover navigates away from walls as expected.
- Hoover skids in place when moving into a wall: The hoover does not move beyond walls but skids in place.
- Hoover rapidly skids against a boundary: The hoover correctly handles rapid movements against a boundary without errors.
- Hoover can start on a boundary and move in valid directions: Starting near a wall does not prevent valid movements away from it.

**Large Instruction Sets:**

- The API can handle large sets of movement instructions without errors, demonstrating its ability to manage complex navigation tasks.

**Error Handling:**

- The API correctly raises errors when provided with invalid instructions, invalid room dimensions, and invalid starting positions.

**No Movement Instructions:**

- When no movement instructions are provided, the hoover stays in place, showing the API behaves correctly with empty input.

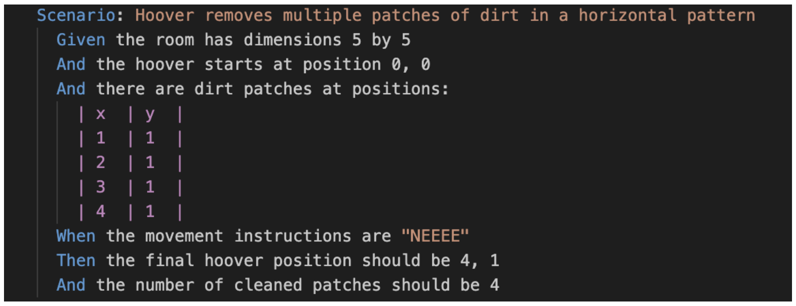
**Analysis:**
The movement tests are well-implemented and the API correctly handles all scenarios involving hoover navigation. The system deals effectively with both valid and invalid inputs and performs as expected in edge cases, such as hitting walls or handling large instruction sets. The navigation functionality is robust and passes all tests smoothly.

## Failing Feature: Hoover Cleaning

The hooverDirtCleaner.feature shows several issues related to dirt patch cleaning. While the API can remove a single dirt patch and clean every position in the room under certain conditions, it fails in more complex scenarios.

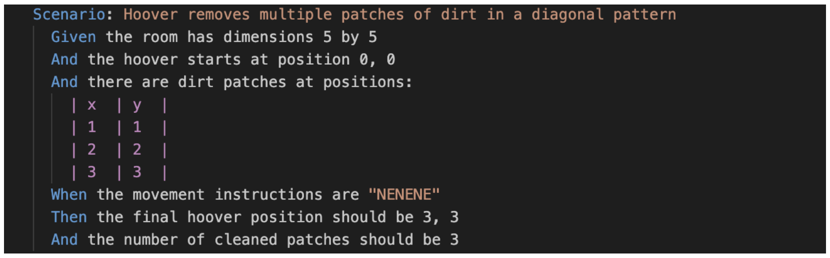
**Failing Tests (7 total)):**

1. Hoover removes multiple patches of dirt in a horizontal pattern:
    - **Error:** The test expected 4 patches to be cleaned, but the API reported cleaning 5 patches, indicating an overcount.
      - **Note:** *Seems to be returning the number of navigation movements/inputs.*



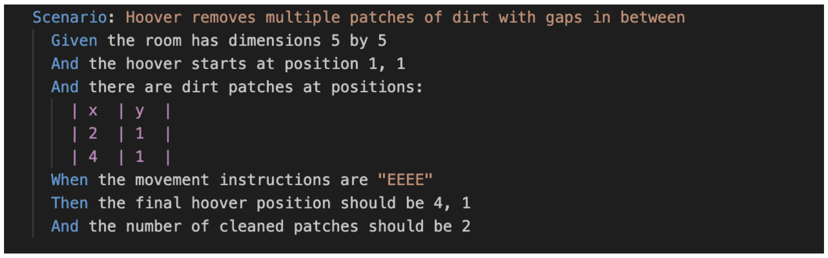
```
Scenario: Hoover removes multiple patches of dirt in a horizontal pattern
  Given the room has dimensions 5 by 5
  And the hoover starts at position 0, 0
  And there are dirt patches at positions:
    | x  | y  |
    | 1  | 1  |
    | 2  | 1  |
    | 3  | 1  |
    | 4  | 1  |
  When the movement instructions are "NEEEE"
  Then the final hoover position should be 4, 1
  And the number of cleaned patches should be 4
```

2. Hoover removes multiple patches of dirt in a diagonal pattern:
    - **Error:** The test expected 3 patches to be cleaned, but the API reported 6, suggesting incorrect patch counting in diagonal movements.



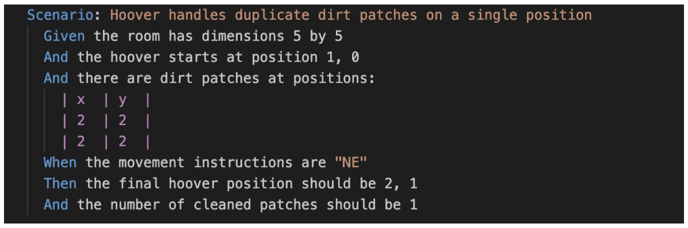
```
Scenario: Hoover removes multiple patches of dirt in a diagonal pattern
  Given the room has dimensions 5 by 5
  And the hoover starts at position 0, 0
  And there are dirt patches at positions:
    | x  | y  |
    | 1  | 1  |
    | 2  | 2  |
    | 3  | 3  |
  When the movement instructions are "NENENE"
  Then the final hoover position should be 3, 3
  And the number of cleaned patches should be 3
```

3. Hoover removes multiple patches of dirt with gaps in between:
    - **Error:** The test expected 2 patches to be cleaned, but the API reported 3, indicating issues when the hoover moves over non-adjacent patches.



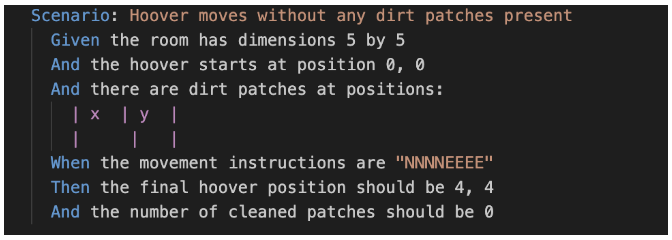
```
Scenario: Hoover removes multiple patches of dirt with gaps in between
  Given the room has dimensions 5 by 5
  And the hoover starts at position 1, 1
  And there are dirt patches at positions:
    | x  | y  |
    | 2  | 1  |
    | 4  | 1  |
  When the movement instructions are "EEEE"
  Then the final hoover position should be 4, 1
  And the number of cleaned patches should be 2
```

4. Hoover handles duplicate dirt patches on a single position:
    - **Error:** The test expected 1 patch to be cleaned, but the API reported 2, suggesting the hoover is cleaning duplicate patches more than once.



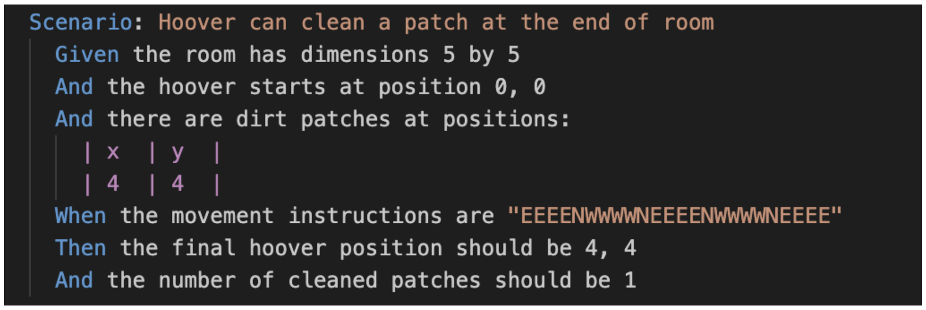
```
Scenario: Hoover handles duplicate dirt patches on a single position
  Given the room has dimensions 5 by 5
  And the hoover starts at position 1, 0
  And there are dirt patches at positions:
    | x  | y  |
    | 2  | 2  |
    | 2  | 2  |
  When the movement instructions are "NE"
  Then the final hoover position should be 2, 1
  And the number of cleaned patches should be 1
```

5. Hoover moves without any dirt patches present:
   - **Error:** The test expected 0 patches to be cleaned, but the API reported 8 patches, highlighting a major issue where the hoover counts non-existent Patches.



```
Scenario: Hoover moves without any dirt patches present
  Given the room has dimensions 5 by 5
  And the hoover starts at position 0, 0
  And there are dirt patches at positions:
    | x  | y  |
    |    |    |
  When the movement instructions are "NNNNEEEE"
  Then the final hoover position should be 4, 4
  And the number of cleaned patches should be 0
```

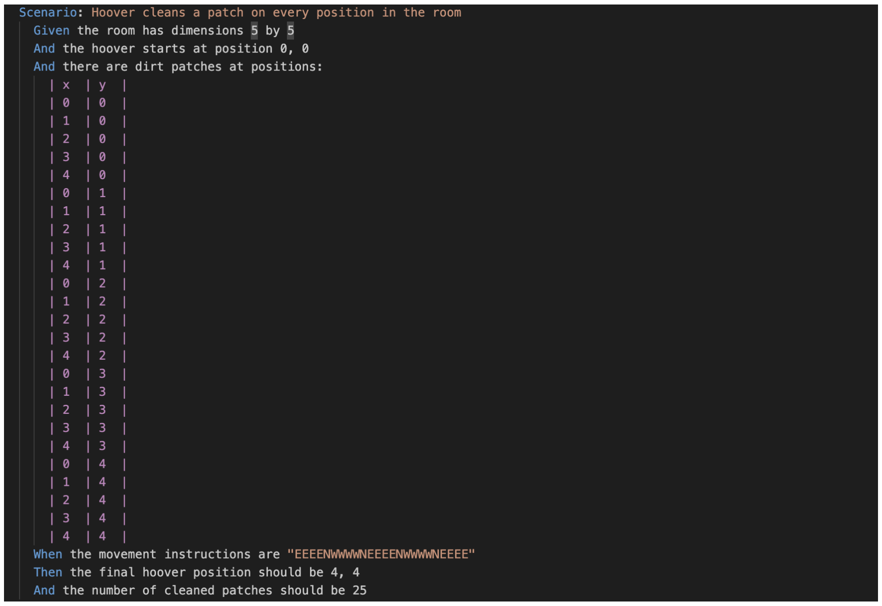
6. Hoover can clean a patch at the end of the room:
   - **Error:** The test expected 1 patch to be cleaned, but the API reported 24 patches, indicating a significant overcount.
     - **Note:** The patch counter seems to match the number of navigation movements instead of actual cleaned patches.



```
Scenario: Hoover can clean a patch at the end of room
  Given the room has dimensions 5 by 5
  And the hoover starts at position 0, 0
  And there are dirt patches at positions:
    | x  | y  |
    | 4  | 4  |
  When the movement instructions are "EEEENWWWWNEEEENWWWWNEEEE"
  Then the final hoover position should be 4, 4
  And the number of cleaned patches should be 1
```

7. Hoover cleans a patch on every position in the room:
   - **Error:** Expected 1 patch to be cleaned, but the API returned that 24 patches were cleaned.

- **Note:** The patch counter seems to match the number of navigation movements instead of actual cleaned patches.



```
Scenario: Hoover cleans a patch on every position in the room
  Given the room has dimensions 5 by 5
  And the hoover starts at position 0, 0
  And there are dirt patches at positions:
    | x  | y  |
    | 0  | 0  |
    | 1  | 0  |
    | 2  | 0  |
    | 3  | 0  |
    | 4  | 0  |
    | 0  | 1  |
    | 1  | 1  |
    | 2  | 1  |
    | 3  | 1  |
    | 4  | 1  |
    | 0  | 2  |
    | 1  | 2  |
    | 2  | 2  |
    | 3  | 2  |
    | 4  | 2  |
    | 0  | 3  |
    | 1  | 3  |
    | 2  | 3  |
    | 3  | 3  |
    | 4  | 3  |
    | 0  | 4  |
    | 1  | 4  |
    | 2  | 4  |
    | 3  | 4  |
    | 4  | 4  |
  When the movement instructions are "EEEENWWWWNEEEENWWWWNEEEE"
  Then the final hoover position should be 4, 4
  And the number of cleaned patches should be 25
```

**Passing Tests (3 total):**

- Hoover removes a single patch of dirt: The API successfully handles a simple case where only one patch is cleaned.
- Hoover starts on a dirt patch: When the hoover begins on a dirt patch, it cleans it. ( May be a flaky pass, requiring further analysis to confirm )
- Hoover raises an error with invalid patch coordinates: The API correctly raises an error when invalid patch coordinates are provided.

**Analysis:**
The cleaning feature has difficulty managing multiple dirt patches, particularly when they are spaced apart or arranged in more complex patterns like horizontal or diagonal lines. It also tends to overcount patches, especially when there's no dirt or when patches are located near the edges of the room. The API needs improvements in accurately counting patches and handling edge cases, such as duplicate or missing patches.

**Conclusion:**

**Hoover Movement Feature:** All movement tests passed successfully, demonstrating that the API's navigation capabilities are well-implemented, handling both regular and edge cases effectively.

**Hoover Cleaning Feature:** The API's patch-cleaning function faces challenges with overcounting and handling more intricate patch layouts. Simple scenarios, like clearing a single patch, work as expected, but it falters when managing multiple patches, especially when they’re spaced out or include duplicates and gaps.

**Next Steps:**

**Refactor patch counting:** Review and adjust the logic responsible for counting cleaned patches. Focus on scenarios with multiple patches and ensure that patches are only counted once when the hoover moves over them. Ensure that the counter doesn’t increment incorrectly when no patches are present.

**Enhance handling of complex patch layouts:** Ensure the hoover navigates and cleans patches correctly across horizontal, diagonal, and spaced-out patterns without miscounting.