Bio-Hazard Cleaning Agent - Project Documentation

Table of Contents

Problem Statement

Objective

Develop an intelligent autonomous agent that navigates a 2D grid environment to clean bio-hazardous waste while avoiding human encounters. The agent must:

1. **Random Navigation**: Move through the grid randomly while respecting boundaries and constraints
2. **Human Avoidance**: When encountering a human, identify and move toward the nearest bio-hazard object to avoid the human
3. **Waste Collection**: Collect and clean bio-hazard cells encountered during navigation
4. **Path Tracking**: Maintain a record of all visited positions and collected waste

Constraints

- Grid size: Configurable (default 100×100)
- Bio-hazard cells: Randomly placed (default 1000)
- Human positions: Randomly placed (configurable)
- Inaccessible areas: Predefined obstacles and grid boundaries
- Agent must not revisit cells
- Agent must stop when no valid moves remain

Key Metrics

- **Human Encounters**: Count of times agent encounters humans
- **Alternative Paths Selected**: Count of times agent avoids human by moving toward nearest bio-hazard
- **Objects Collected**: Total bio-hazard cells cleaned

Data Structures

1. Environment Grid

Type: 2D NumPy Array `(size, size)` with integer values

Cell Values:

- `0` = Clean area (accessible, no hazards)
- `1` = Bio-hazard (accessible, needs cleaning)

- `2` = Inaccessible (walls, obstacles, boundaries)
- `3` = Human (position of human to avoid)

Initialization:

```
self.grid = np.zeros((size, size), dtype=int)
```

Grid Layout (100×100):

- Entire border (row 0, row 99, col 0, col 99) = inaccessible
- Additional internal obstacles placed in Create_inaccessible_areas()

2. Agent State

Type: `Agent` class

Attributes:

- `current_position`: Tuple (row, col) - current grid location
- `active`: Boolean - agent active status
- `stop_reason`: String - reason for stopping
- `visited_positions`: Set - set of all visited coordinates
- `path`: List - ordered list of positions from start to current
- `steps_taken`: Integer - total moves made
- `waste_collected`: Integer - total bio-hazards cleaned
- `human_encounters`: Integer - count of human encounters
- `alternative_paths_used`: Integer - count of avoidance maneuvers

3. Action Space

Type: Dictionary in `Action` class

Available Actions:

```
{
    "MOVE_UP": (-1, 0),
    "MOVE_DOWN": (1, 0),
    "MOVE_LEFT": (0, -1),
    "MOVE_RIGHT": (0, 1)
}
```

Each action defines row and column deltas for movement.

4. Movement Validator

Type: `Movement` class

Method: `is_move_valid(current_pos, next_pos, visited_set)`

Returns: Boolean

Validation Checks:

1. Next position is inside grid boundaries
2. Next position is accessible (not inaccessible/wall)
3. Next position has not been visited
4. Next position differs from current position

5. Random Movement

Type: `Random` class

Key Method: `perform_random_move()`

Logic:

1. Shuffle all available actions randomly
2. For each action:
    - Calculate new position
    - If new position has human:
        - Find nearest bio-hazard (Manhattan distance)
        - Generate candidate steps toward that bio-hazard
        - Select first valid candidate move
        - Increment `human_encounters` and `alternative_paths_used`
    - Else if new position is valid:
        - Move to new position
        - If bio-hazard exists, clean it

Note: a short adjacency check now pre-scans neighbor cells for humans and delegates avoidance work to a small `human_avoidance.py` module for clearer separation of concerns.

Algorithms

This section describes the core algorithms used by the agent, with concise pseudocode and complexity notes.

1. Movement Validation (Movement.is_move_valid)

Purpose: ensure a single-step candidate is legal before moving.

Pseudocode:

```
function is_move_valid(current, next, visited):
  if next is outside grid: return False
  if grid[next] == INACCESSIBLE: return False
  if next in visited: return False
  if next == current: return False
  return True
```

Complexity: O(1) per candidate (bounds and cell-value checks, set membership).

2. Random Move Selection (Random.perform_random_move)

Purpose: choose a random legal move while respecting human avoidance.

Pseudocode:

```
function perform_random_move():
  actions = shuffled(all_actions)
  // quick neighbor-scan for humans
  for action in actions:
    neigh = current + delta(action)
    if env.is_human(neigh):
      if handle_human_encounter(agent, env, movement_validator):
        return True
  // normal attempt loop
  for action in actions:
    new = current + delta(action)
    if env.is_human(new):
      if handle_human_encounter(agent, env, movement_validator):
        return True
      continue
    if is_move_valid(current, new, visited):
      update_position(new)
      if env.is_bio_hazard(new): clean and collect
      return True
  stop("No valid moves")
  return False
```

Complexity: tries at most 4 actions; each validation is O(1). Overall O(1) per move attempt (ignoring avoidance nearest-search cost).

3. Human Avoidance (human_avoidance.handle_human_encounter)

Purpose: when a proposed move would confront a human, pick an alternative one-step move oriented toward the nearest bio-hazard.

Pseudocode:

```
function handle_human_encounter(agent, env, validator):
  agent.human_encounters += 1
  bio_coords = env.get_bio_hazard_coordinates()
  if bio_coords is empty: return False
  nearest = argmin_b manhattan_distance(agent.pos, b)
  sdr = sign(nearest.row - agent.row)
  sdc = sign(nearest.col - agent.col)
  candidates = []
  if sdr != 0: candidates.append((agent.row + sdr, agent.col))
  if sdc != 0: candidates.append((agent.row, agent.col + sdc))
```

```
    if sdr != 0 and sdc != 0: candidates.append((agent.row + sdr, agent.col + sdc))
    for cand in candidates: // order: row-only, col-only, diagonal
      if validator.is_move_valid(agent.pos, cand, agent.visited_positions):
        agent.update_position(cand)
        agent.alternative_paths_used += 1
        if env.is_bio_hazard(cand): env.clean_cell(cand); agent.collect_waste()
        return True
    return False
```

Complexity:

- Nearest search: O(H) where H is number of bio-hazard cells (linear scan using Manhattan distance).
- Candidate checks: up to 3, each O(1).

Notes and trade-offs:

- Greedy, single-step approach: the avoidance routine only selects one-step alternatives (row-only, column-only, diagonal) toward the chosen target. It does not perform multi-step pathfinding. This keeps CPU cost low and logic simple but can fail in tightly constrained mazes.
- If no bio-hazards exist or all candidates are invalid (visited/inaccessible/occupied), avoidance returns False and the normal random-move loop continues.
- For larger environments or high-density hazards, consider replacing the linear nearest search with a spatial index or performing multi-step planning (BFS/A*) toward the target.

Input and Output Examples

Example 1: Single Simulation Run

Input Parameters:

```
Environment size: 100×100
Initial bio-hazards: 1000
Initial humans: 50
Agent start position: Random clean cell
Max steps: Unlimited (until no valid moves)
```

Execution:

```
Initialize Environment
Place 1000 bio-hazards randomly
Place 50 humans randomly
Agent starts at random position
Perform moves until stopped
```

Output:

```
```

```
Stop reason: No valid moves
Total steps taken: 287
Total waste collected: 45
Human encounters: 12
Alternative paths used: 8
Path length: 287
```

Example 2: 100-Run Batch Testing

Input Parameters:

```
Number of simulations: 100
Environment size: 30×30
Bio-hazards per run: 200
Humans per run: 30
Max steps per run: 1000
```

Note: `Main.py` runs this batch by default (100 runs) and prints aggregated totals for encounters, avoidance actions, and objects collected.

Aggregated Output:

```
-- Number of human encountered: 236
-- Number of nearest path selected and avoided: 206
-- Number of object collected: 1400
```

Interpretation:

- Total humans encountered across 100 runs: 236
- Total avoidance maneuvers performed: 206
- Total bio-hazards cleaned: 1400 (average 14 per run)

Example 3: Movement Validation Sequence

Scenario: Agent at (10, 10) in 20×20 grid

Input Moves:

```
1. Try move UP to (9, 10)
2. Try move LEFT to (10, 9)
3. Try move DOWN to (11, 10)
4. Try revisit UP to (9, 10)
```

Validation Output:

```
Move 1: Valid ✓ → Position (9, 10) added to path
Move 2: Valid ✓ → Position (10, 9) added to path
Move 3: Valid ✓ → Position (11, 10) added to path
Move 4: Invalid ✗ → Already visited (9, 10)
```

Example 4: Human Avoidance in Action

Scenario: Agent at (15, 15), human at (14, 15), nearest bio at (13, 14)

Input:

```
Agent position: (15, 15)
Next random move leads to: (14, 15) - HUMAN!
Bio-hazard coordinates: [(13, 14), (17, 18), (10, 10)]
Nearest bio: (13, 14) - distance 3
```

Processing:

```
Increment human_encounters += 1
Calculate direction toward (13, 14):
  - Row direction: sign(-2) = -1 (move up)
  - Col direction: sign(-1) = -1 (move left)
Generate candidates:
  1. (14, 15) - row move only
  2. (15, 14) - col move only
  3. (14, 14) - diagonal move
Test candidate 1: (14, 15) = human there, invalid
Test candidate 2: (15, 14) = accessible, valid ✓
```

Output:

```
Move to: (15, 14)
Increment alternative_paths_used += 1
If (15, 14) has bio-hazard:
  -> Clean it
  -> Increment waste_collected += 1
```

Key Features & Innovations

1. **Intelligent Avoidance**

When detecting a human, the agent doesn't just stop—it actively moves toward the nearest bio-hazard to continue mission objectives while avoiding the human.

The avoidance logic is implemented in `human_avoidance.py`; `Random.perform_random_move()` delegates to it and also performs a quick neighbor check to improve detection.

2. **Efficient Distance Metric**

Uses Manhattan distance to find nearest bio-hazard: `|r1 - r2| + |c1 - c2|`

3. **Multi-Directional Movement**

Candidate generation includes row-only, column-only, and diagonal moves to maximize movement options.

4. **Comprehensive Tracking**

Records:

- Complete path history
- Visited cell set (prevents cycles)
- Human encounter count
- Avoidance maneuver count
- Waste collection count

5. **Robust Validation**

Movement validation checks:

- Grid boundaries
- Obstacle avoidance
- Cycle prevention
- Self-collision prevention

Performance Metrics

Typical Single Run (100×100 grid, 1000 bio-hazards)

- **Steps taken**: 200-400
- **Waste collected**: 20-80 bio-hazards
- **Collection rate**: 2-20% of total hazards
- **Average steps per collection**: 3-8 steps

100-Run Batch (30×30 grid, 200 bio-hazards, 30 humans)

- **Total human encounters**: 200-300
- **Total avoidance maneuvers**: 150-250
- **Total waste collected**: 1200-1500 (12-15 per run)
- **Success rate**: 100% (all runs complete)

Dependencies

- **numpy**: Grid operations and coordinate arrays

- **pytest**: Testing framework (optional for running tests)
- **Python 3.8+**: Base language

Notes for PDF Conversion

This markdown file is optimized for PDF conversion using tools like:

- **Pandoc**: `pandoc PROJECT_DOCUMENTATION.md -o PROJECT_DOCUMENTATION.pdf`
- **Markdown to PDF**: Online converters (markdowntopdf.com)
- **VS Code Extensions**: Markdown PDF extension

Formatting features:

- Clear table of contents with links
- Structured sections with headers
- Code blocks with syntax highlighting
- Tables for data organization
- Proper spacing for readability

Document Version: 1.0
Last Updated: February 24, 2026
Project Status: Complete & Tested (41/41 tests passing)