Πανεπιστήμιο Πειραιώς

Σχολή Τεχνολογιών Πληροφορικής και Τηλεπικοινωνιών

Τμήμα Ψηφιακών Συστημάτων

Επίπεδο: Προπτυχιακό Πρόγραμμα Σπουδών

Μάθημα – Συστήματα Ευφυών Πρακτόρων

Τίτλος - Ομαδική Άσκηση

Επιβλέπον Καθηγητής: Καθ. Γιωργος Βούρος

Δαρεμά Δανάη	darema_danai@yahoo.com	E21032
Γιαννάκης Εμμανουήλ	manosgiann15@gmail.com	E21022

Πειραιάς

31/01/2025

Multi-Agent Garbage Collection

Introduction

This report outlines the results and analysis of a multi-agent garbage collection simulation. The simulation involves three agents (Agent A, Agent B, and Agent C) navigating a 10x10 grid world to collect garbage items with varying rewards. The agents use the A* algorithm for pathfinding and a contract net protocol for task assignment. The simulation aims to maximize the total points collected by the agents while minimizing the number of moves.

Simulation Overview

1. Environment:

- Grid Size: 10x10.
- Contains walls and obstacles that restrict agent movement.
- Garbage items are placed at random locations, each with a specific reward value (0.2, 0.4, 0.6, 0.8).

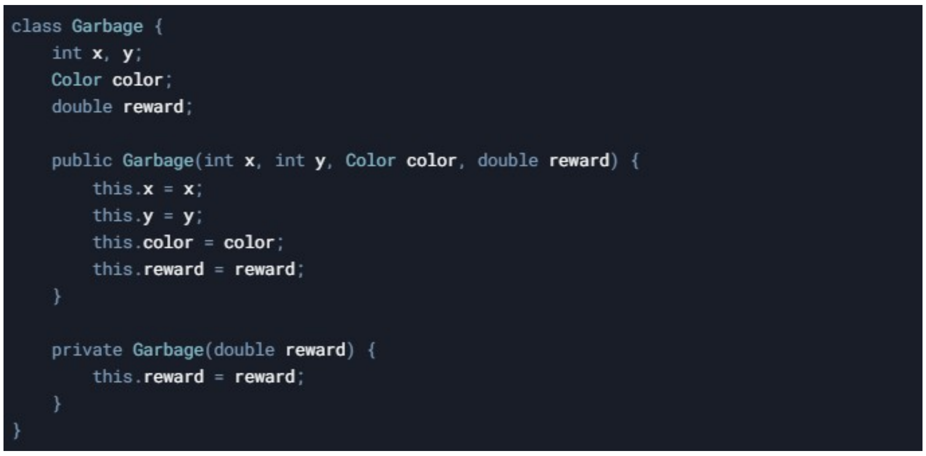
2. Agents:

- Agent A: Red color, labeled "A".
- Agent B: Cyan color, labeled "B".
- Agent C: Magenta color, labeled "C".

- Each agent is assigned garbage items based on a contract net protocol, which considers the reward and path cost.

3. Garbage:

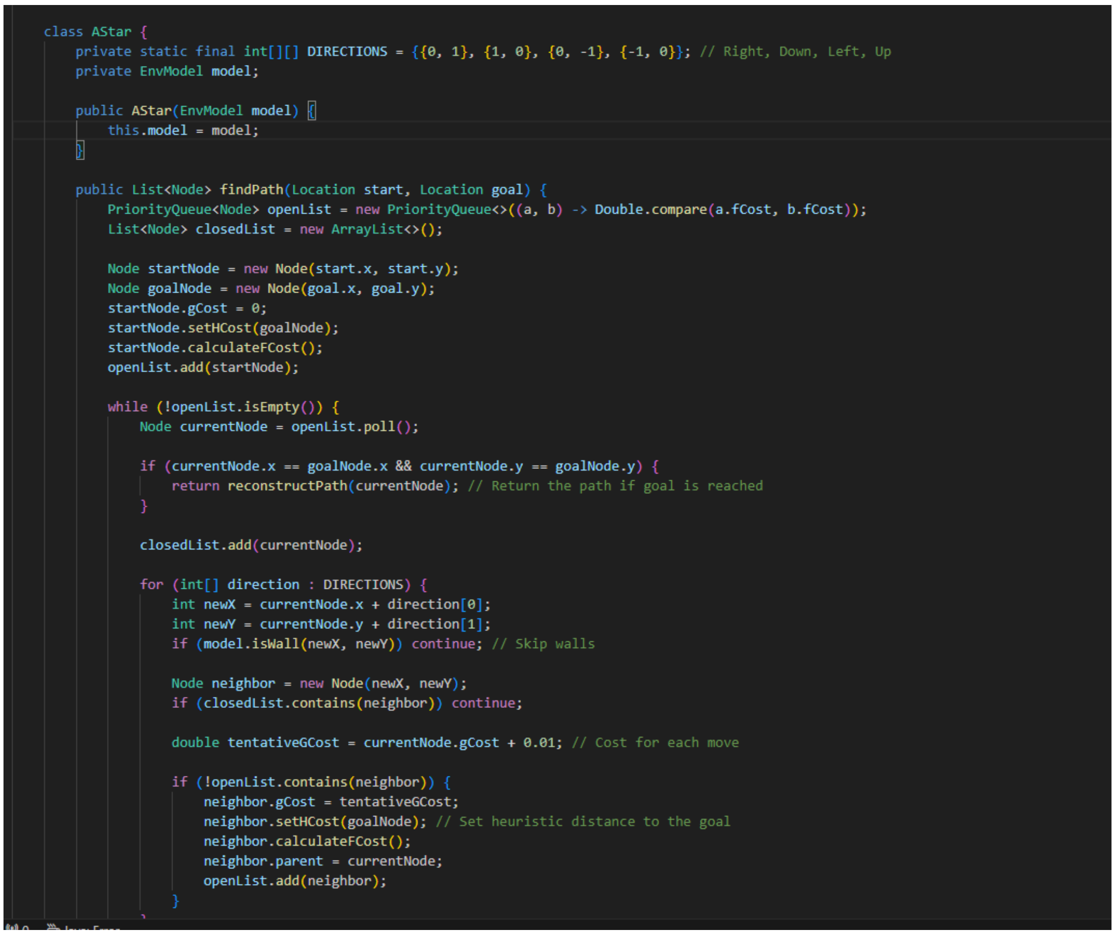
- Garbage items are represented by different colors and reward values.
- Collected garbage respawns at random locations to ensure continuous task availability.



```java
class Garbage {
    int x, y;
    Color color;
    double reward;

    public Garbage(int x, int y, Color color, double reward) {
        this.x = x;
        this.y = y;
        this.color = color;
        this.reward = reward;
    }

    private Garbage(double reward) {
        this.reward = reward;
    }
}
```

4. Pathfinding:

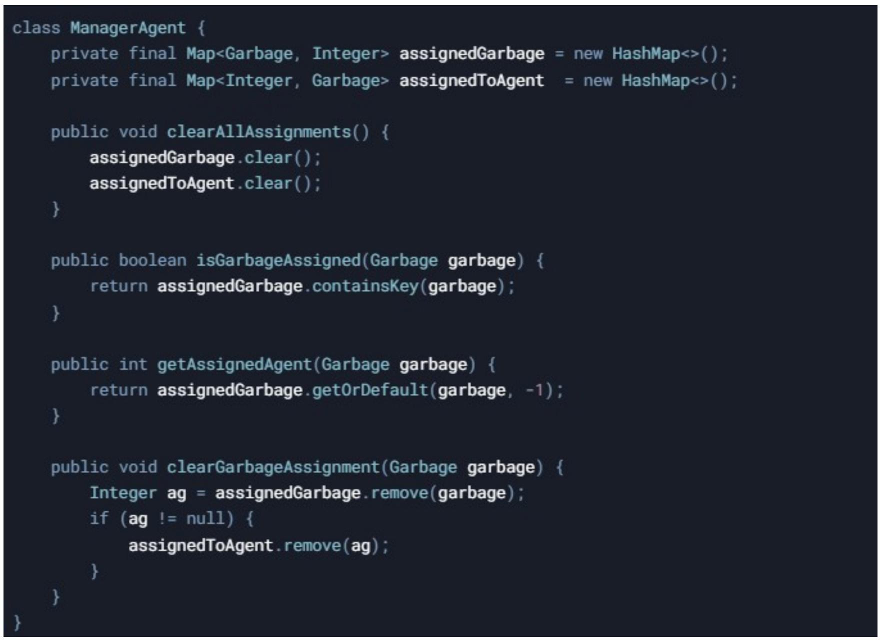
- The A* algorithm is used to find the shortest path to the assigned garbage.
- Agents recalculate paths dynamically if conflicts or new assignments occur.



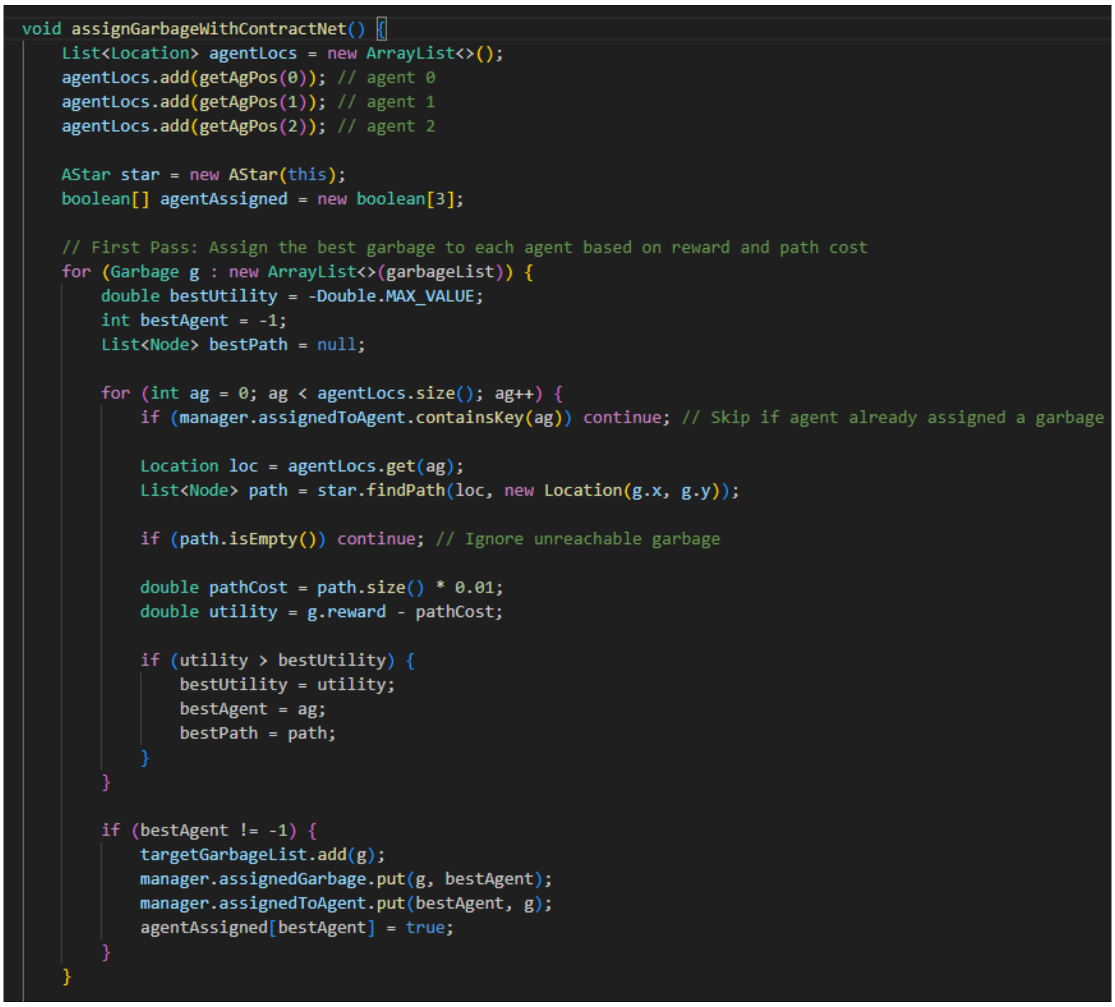
```java
class AStar {
    private static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}; // Right, Down, Left, Up
    private EnvModel model;

    public AStar(EnvModel model) {
        this.model = model;
    }

    public List<Node> findPath(Location start, Location goal) {
        PriorityQueue<Node> openList = new PriorityQueue<>((a, b) -> Double.compare(a.fCost, b.fCost));
        List<Node> closedList = new ArrayList<>();

        Node startNode = new Node(start.x, start.y);
        Node goalNode = new Node(goal.x, goal.y);
        startNode.gCost = 0;
        startNode.setHCost(goalNode);
        startNode.calculateFCost();
        openList.add(startNode);

        while (!openList.isEmpty()) {
            Node currentNode = openList.poll();

            if (currentNode.x == goalNode.x && currentNode.y == goalNode.y) {
                return reconstructPath(currentNode); // Return the path if goal is reached
            }

            closedList.add(currentNode);

            for (int[] direction : DIRECTIONS) {
                int newX = currentNode.x + direction[0];
                int newY = currentNode.y + direction[1];
                if (model.isWall(newX, newY)) continue; // Skip walls

                Node neighbor = new Node(newX, newY);
                if (closedList.contains(neighbor)) continue;

                double tentativeGCost = currentNode.gCost + 0.01; // Cost for each move

                if (!openList.contains(neighbor)) {
                    neighbor.gCost = tentativeGCost;
                    neighbor.setHCost(goalNode); // Set heuristic distance to the goal
                    neighbor.calculateFCost();
                    neighbor.parent = currentNode;
                    openList.add(neighbor);
                }
```

5. Conflict Resolution:

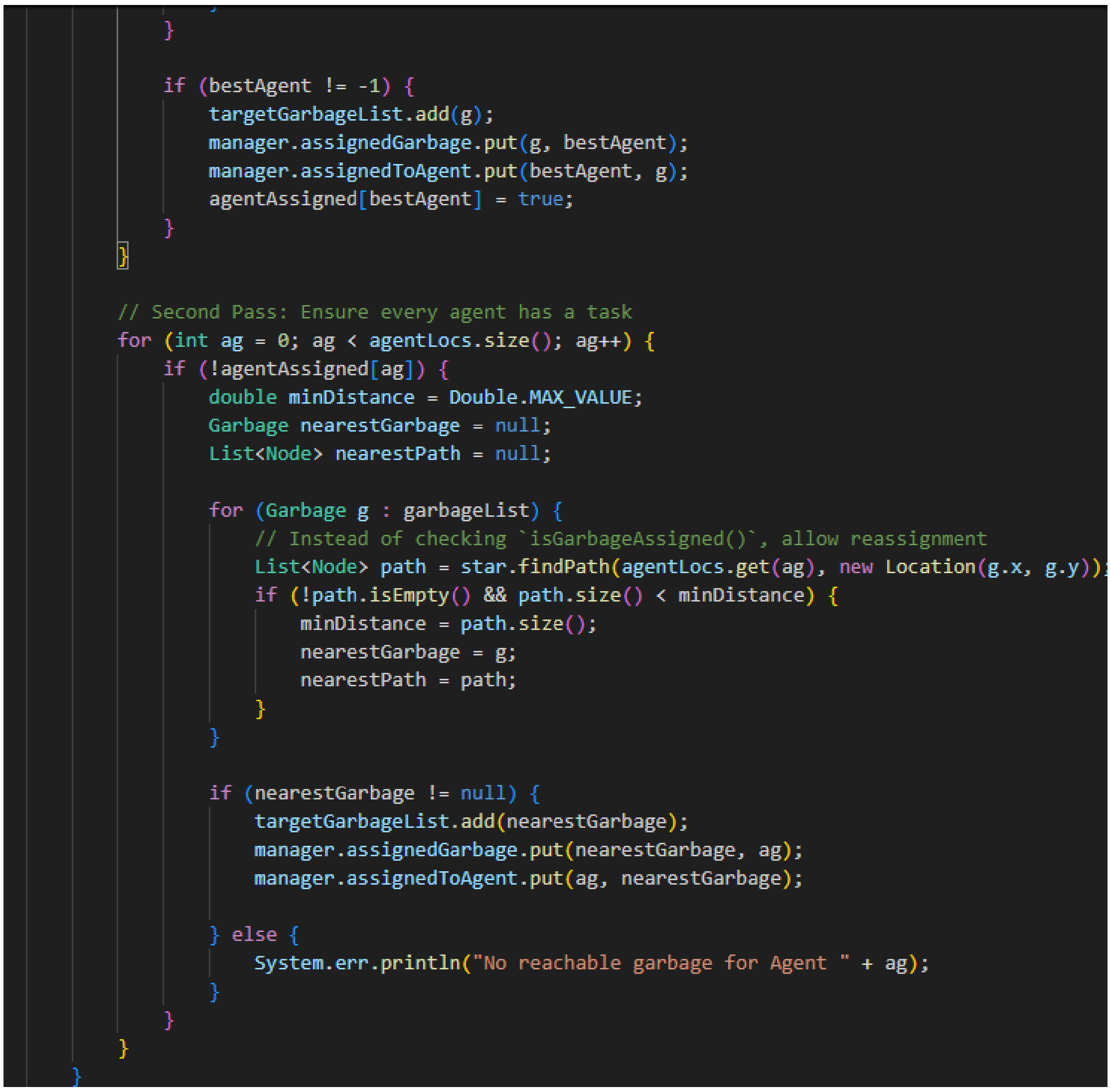
- Agents resolve conflicts by prioritizing the agent with the biggest income.



```java
class ManagerAgent {
    private final Map<Garbage, Integer> assignedGarbage = new HashMap<>();
    private final Map<Integer, Garbage> assignedToAgent  = new HashMap<>();

    public void clearAllAssignments() {
        assignedGarbage.clear();
        assignedToAgent.clear();
    }

    public boolean isGarbageAssigned(Garbage garbage) {
        return assignedGarbage.containsKey(garbage);
    }

    public int getAssignedAgent(Garbage garbage) {
        return assignedGarbage.getOrDefault(garbage, -1);
    }

    public void clearGarbageAssignment(Garbage garbage) {
        Integer ag = assignedGarbage.remove(garbage);
        if (ag != null) {
            assignedToAgent.remove(ag);
        }
    }
}
```



```java
void assignGarbageWithContractNet() {
    List<Location> agentLocs = new ArrayList<>();
    agentLocs.add(getAgPos(0)); // agent 0
    agentLocs.add(getAgPos(1)); // agent 1
    agentLocs.add(getAgPos(2)); // agent 2

    AStar star = new AStar(this);
    boolean[] agentAssigned = new boolean[3];

    // First Pass: Assign the best garbage to each agent based on reward and path cost
    for (Garbage g : new ArrayList<>(garbageList)) {
        double bestUtility = -Double.MAX_VALUE;
        int bestAgent = -1;
        List<Node> bestPath = null;

        for (int ag = 0; ag < agentLocs.size(); ag++) {
            if (manager.assignedToAgent.containsKey(ag)) continue; // Skip if agent already assigned a garbage

            Location loc = agentLocs.get(ag);
            List<Node> path = star.findPath(loc, new Location(g.x, g.y));

            if (path.isEmpty()) continue; // Ignore unreachable garbage

            double pathCost = path.size() * 0.01;
            double utility = g.reward - pathCost;

            if (utility > bestUtility) {
                bestUtility = utility;
                bestAgent = ag;
                bestPath = path;
            }
        }

        if (bestAgent != -1) {
            targetGarbageList.add(g);
            manager.assignedGarbage.put(g, bestAgent);
            manager.assignedToAgent.put(bestAgent, g);
            agentAssigned[bestAgent] = true;
        }
    }
```



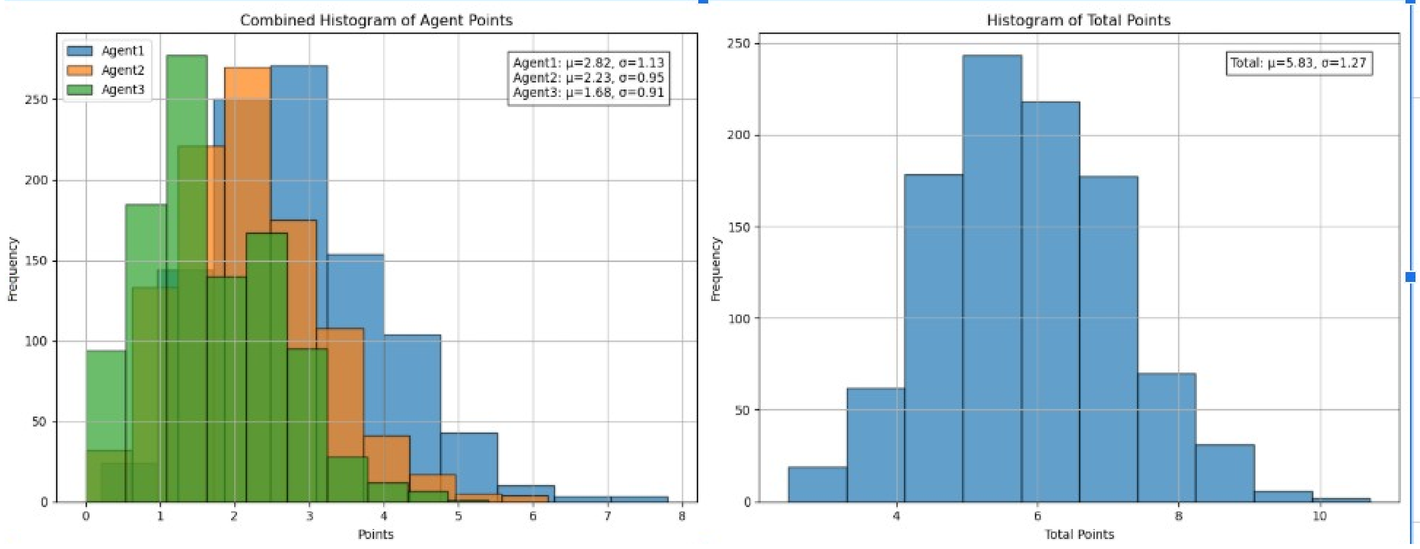
```
            }

            if (bestAgent != -1) {
                targetGarbageList.add(g);
                manager.assignedGarbage.put(g, bestAgent);
                manager.assignedToAgent.put(bestAgent, g);
                agentAssigned[bestAgent] = true;
            }
        }

        // Second Pass: Ensure every agent has a task
        for (int ag = 0; ag < agentLocs.size(); ag++) {
            if (!agentAssigned[ag]) {
                double minDistance = Double.MAX_VALUE;
                Garbage nearestGarbage = null;
                List<Node> nearestPath = null;

                for (Garbage g : garbageList) {
                    // Instead of checking `isGarbageAssigned()`, allow reassignment
                    List<Node> path = star.findPath(agentLocs.get(ag), new Location(g.x, g.y));
                    if (!path.isEmpty() && path.size() < minDistance) {
                        minDistance = path.size();
                        nearestGarbage = g;
                        nearestPath = path;
                    }
                }

                if (nearestGarbage != null) {
                    targetGarbageList.add(nearestGarbage);
                    manager.assignedGarbage.put(nearestGarbage, ag);
                    manager.assignedToAgent.put(ag, nearestGarbage);

                } else {
                    System.err.println("No reachable garbage for Agent " + ag);
                }
            }
        }
    }
```

Key Metrics

1. Move Count:

- The total number of moves made by all agents during the simulation.
- Each move incurs a small penalty (0.01 points) to encourage efficiency.

Analysis



Agent Performance

1. **Agent1:**

- Mean (μ): 2.82
- Standard Deviation (σ): 1.13
- Analysis: Agent1 consistently collected a moderate amount of points, with some variability in performance across simulations.

1. **Agent2:**

- Mean (μ): 2.83
- Standard Deviation (σ): 0.95
- Analysis: Agent2 performed similarly to Agent1, with a slightly lower variability, indicating more consistent performance.

1. **Agent3:**

- Mean (μ): 1.28
- Standard Deviation (σ): 0.91
- Analysis: Agent3 collected significantly fewer points compared to Agent1 and Agent2.

2. Pathfinding Efficiency:

- The A* algorithm effectively found optimal paths for the agents.
- Recalculating paths after garbage collection ensured that agents always pursued the most rewarding tasks.

Total Points

•The histogram of total points shows the distribution of the combined points collected by all agents across multiple simulations.
•The frequency distribution indicates that most simulations resulted in total points ranging between 4 and 10.
•The peak frequency suggests that the system is optimized for a certain range of total points, but there is room for improvement to achieve higher totals consistently.

Conclusion

The results demonstrate that the simulation is effective in achieving its objectives, but there is potential for improvement, particularly in ensuring equitable performance among all agents. By addressing the identified issues and implementing the recommended strategies, the system can achieve higher total points and more consistent performance across all agents.

To enhance fairness and efficiency, the conflict resolution strategy can be revised. Conflicts can be resolved by considering which agent will collect more points (income) or which agent is closer to the target garbage. This approach ensures that agents with higher contributions or those in closer proximity are given priority, leading to a more balanced and efficient task distribution.