Evidence 1

Team Members:

Rafael Romo Muñoz (A01643137)
German Avelino Del Rio Guzman (A01641976)
José María Soto Valenzuela (A01254831)
César Alán Silva Ramos (A01252916)
Dilan Eduardo Ocampo Hernandez (A01634254)

Monterrey Institute of Technology

November 19, 2024

Summary of the System Design and Key Metrics

The proposed system is a simulation where agents interact with a shared environment to group scattered objects effectively. The system is structured around three core components: **agents**, the **environment**, and **metrics of success**.

1. Agent Model

Agents are autonomous entities implemented using the `Agent` class. Each agent operates independently within the grid environment and is defined by:

Properties

- **id:** A unique identifier for tracking individual agents.
- **x, y:** Coordinates denoting the agent's current position in the grid.
- **carrying:** A boolean that indicates whether the agent is holding an object, determining its ability to interact with the environment.

Capabilities

- **Movement:** Agents utilize the `move` method to update their position based on specified target coordinates. This forms the basis for navigating the grid and interacting with objects or groups.
- **Object Interaction:** Agents can pick up or deposit objects depending on their `carrying` state, contributing to the creation of object groups.

Agent Code Example

```
class Agent:
    def __init__(self, id, x, y):
        self.id = id
        self.x = x
        self.y = y
        self.carrying = False

    def move(self, new_x, new_y):
        self.x = new_x
        self.y = new_y
```

2. Environment Properties

The environment is modeled as a dynamic 2D grid (`GRID_SIZE` $\times$ `GRID_SIZE`) that serves as the shared workspace for agents. Objects are randomly distributed across the grid and initialized with a value of 1.

Object Placement

Objects are placed on unoccupied grid cells to prevent overlaps.

Environment Code Example

```python
# Initialize the grid
grid = np.full((GRID_SIZE, GRID_SIZE), '', dtype=str)

# Place objects on the grid
def place_objects(num_objects, grid):
    for _ in range(num_objects):
        while True:
            x, y = random.randint(0, GRID_SIZE - 1), random.randint(0,
                GRID_SIZE - 1)
            if grid[x, y] == '':
                grid[x, y] = '1'
                break
```

3. Metrics for Success

The success of the simulation is evaluated using:

Primary Metric

- **Completed Groups:** The number of object groups that reach the maximum size (MAX_GROUP_SIZE).

Secondary Metrics

- **Execution Time:** The total time taken for the simulation.
- **Efficiency:** Objects grouped per simulation step.

Metric Tracking and Reporting

Agents deposit carried objects onto grid cells, incrementing the cell's value. Once a cell reaches MAX_GROUP_SIZE, it is counted as a completed group.

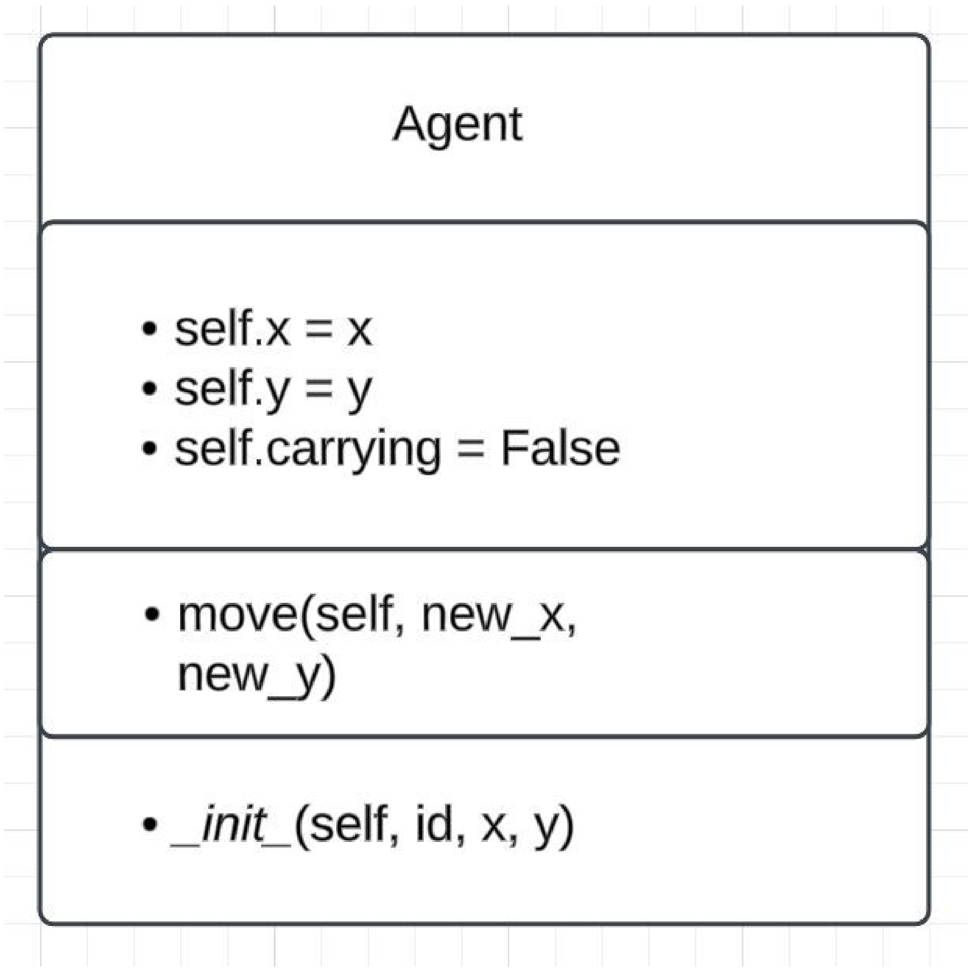
Metrics Code Example

```python
# Count completed groups
completed_groups = 0

def move_agents(agents, grid):
    global completed_groups
    for agent in agents:
        # Logic for object grouping
        if agent.carrying and grid[agent.x, agent.y].isdigit():
            current_value = int(grid[agent.x, agent.y])
            if current_value < MAX_GROUP_SIZE:
                grid[agent.x, agent.y] = str(current_value + 1)
                agent.carrying = False
            elif current_value == MAX_GROUP_SIZE:
                completed_groups += 1

# Results printed after the simulation
print(f"Completed groups: {completed_groups}")
print(f"Execution time: {execution_time:.2f} seconds")
```

Class diagram



4. Conclusion

In conclusion, the simulation presented offers an effective approach for modeling agents in a shared environment, allowing them to interact and group objects. The modular design enables scalability and adaptability, making it suitable for diverse scenarios.

To further improve agent efficiency, several alternative solutions can be explored:

- **Optimized Pathfinding Algorithms:** Incorporating advanced pathfinding algorithms, such as A* or Dijkstra's algorithm, could significantly reduce the time agents take to navigate the grid, improving their task completion rates.

- **Distributed Decision-Making:** Implementing a distributed decision-making system where agents communicate and share their state could allow them to coordinate better and work more efficiently, especially in crowded environments.
- **Learning-Based Approaches:** Agents could be enhanced with reinforcement learning, where they learn from their environment and improve their decision-making over time, adapting to changes and improving performance without explicit programming.
- **Parallel Processing:** By leveraging parallel computing techniques, tasks such as movement and object grouping can be handled simultaneously by multiple agents, reducing the overall execution time of the simulation.

These strategies can complement the existing system and help create a more efficient and scalable multi-agent simulation.