University of the People

CS4408 Artificial Inteligence

Unit 2 Written Assignment 2

Liang Xiao

**Programming Assignment: Rational Agent for Vacuum Cleaner**

**Part A: PEAS Framework**

1. **Performance Measure:**
   * Complete cleaning of all dirty spots with minimal actions (Russell & Norvig, 2020).
   * Minimize energy consumption (each move or suck costs 1 energy point).
   * Ensure the agent returns to the starting location (Location A).
   * Avoid overfilling the dirt bag (maximum capacity of 10). Efficiency is measured by how often the agent needs to return home.
2. **Environment:**
   * A 4x4 static grid labeled A to P, representing the vacuum environment.
   * Some squares contain dirt, and others are clean. The dirt distribution is initially unknown.
   * The agent starts at Location A (Russell & Norvig, 2020).
   * The environment is static, meaning no dynamic obstacles or real-time changes during operation.
3. **Actuators:**
   * Move in four directions: North, South, East, or West (no diagonal movement allowed).
   * Suck dirt from a location when detected.
   * Navigate back to the home position (Location A) to empty the dirt bag or conclude cleaning.
4. **Sensors:**
   * **Dirt Sensor:** Detects if the current location has dirt.
   * **Position Sensor:** Tracks the agent's current position within the grid.
   * **Bag Capacity Sensor:** Monitors the dirt bag's fill level and signals when it's full.
   * **Energy Sensor:** Keeps track of remaining energy to ensure the agent can complete tasks without getting stranded (Russell & Norvig, 2020).

**Part B: Pseudo-code Implementation**

# Initialize environment

environment = [['A', 'B', 'C', 'D'],

['E', 'F', 'G', 'H'],

['I', 'J', 'K', 'L'],

['M', 'N', 'O', 'P']]

# Initialize all locations as dirty

# Using a dictionary for efficient lookup and status tracking

dirt\_status = {location: 'Dirty' for row in environment for location in row}

# Initialize agent parameters

current\_location = 'A'

energy = 100

bag\_capacity = 0

max\_bag\_capacity = 10

# Function to decide action based on current status

def decide\_action(location):

if dirt\_status[location] == 'Dirty':

return 'Suck'

elif bag\_capacity >= max\_bag\_capacity or not enough\_energy\_to\_continue():

return 'Go Home'

elif all(value == 'Clean' for value in dirt\_status.values()):

return 'Go Home'

else:

return 'Move'

# Function to determine if enough energy is available to continue cleaning

def not\_enough\_energy\_to\_continue():

# Simplistic check: Ensure agent has enough energy to return to 'A'

distance\_to\_home = calculate\_distance(current\_location, 'A')

return energy <= distance\_to\_home

# Function to calculate Manhattan distance (simplified for grid)

def calculate\_distance(start, end):

start\_index = ord(start) - ord('A')

end\_index = ord(end) - ord('A')

return abs(start\_index // 4 - end\_index // 4) + abs(start\_index % 4 - end\_index % 4)

# Function to move in the environment

def move\_direction(location):

# Enhanced movement logic: prioritize adjacent dirty tiles for efficiency

adjacent\_tiles = get\_adjacent\_tiles(location)

for tile in adjacent\_tiles:

if dirt\_status[tile] == 'Dirty':

return tile

# If no adjacent dirt, follow a systematic search pattern (e.g., row by row)

return next\_unvisited\_tile()

# Function to navigate back home

def navigate\_home(current\_location):

# Simple pathfinding logic to return to A (can be improved with A\* for larger grids)

path = []

while current\_location != 'A':

if current\_location in ['B', 'C', 'D']:

path.append('Move West')

current\_location = chr(ord(current\_location) - 1)

elif current\_location in ['E', 'F', 'G', 'H']:

path.append('Move North')

current\_location = chr(ord(current\_location) - 4)

else:

path.append('Move West')

current\_location = chr(ord(current\_location) - 1)

return path

# Function to check if goal state is achieved

def check\_goal\_state():

return all(status == 'Clean' for status in dirt\_status.values()) and current\_location == 'A'

# Main execution loop

while energy > 0:

action = decide\_action(current\_location)

if action == 'Suck':

dirt\_status[current\_location] = 'Clean'

bag\_capacity += 1

energy -= 1

elif action == 'Go Home':

path\_to\_home = navigate\_home(current\_location)

for step in path\_to\_home:

energy -= 1

current\_location = 'A'

bag\_capacity = 0

elif action == 'Move':

next\_tile = move\_direction(current\_location)

current\_location = next\_tile

energy -= 1

if check\_goal\_state():

break

**Part C: Summary Reflection**

Through this assignment, I gained a deeper understanding of how rational agents operate in a controlled environment (Russell & Norvig, 2020). Designing the PEAS framework clarified the key components that influence agent behavior, and developing pseudo-code highlighted the importance of efficient decision-making to conserve energy and manage resources like bag capacity.

I believe the algorithm should be able to achieve the goal within the provided 100 energy points, as long as the movement logic is optimized, and unnecessary actions are minimized. The added logic to ensure the agent has enough energy to return home helps prevent potential failures. However, if the static environment becomes larger, the algorithm will need enhancements such as more sophisticated pathfinding techniques like the A\* algorithm and dynamic energy management strategies (LaValle, 2006).

In a real-world scenario, smart vacuum cleaners must navigate complex environments with obstacles like furniture, humans, and pets. To handle these, the agent would require additional sensors (e.g., proximity sensors, cameras) and advanced algorithms for obstacle detection and avoidance (Thrun et al., 2005). Furthermore, incorporating machine learning could help the agent adapt to different home layouts over time. The vacuum cleaner would also benefit from real-time mapping techniques (SLAM) to dynamically adjust its route based on environmental changes.

Overall, this exercise has reinforced the importance of balancing simplicity with functionality in agent design while being mindful of real-world complexities. The careful consideration of energy management, obstacle detection, and scalability will be essential in designing robust intelligent agents.

**References**

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT Press.