University of Puerto Rico - Mayaguez
Department of Electrical and Computer Engineering
ICOM5015 Inteligencia Artificial
Dr. J. Fernando Vega Riveros
21 de Febrero de 2025

Agents
Programming Assignment Chapter 2

ALEJANDRO J RODRIGUEZ-BURGOS - ICOM Undergrad
Miguel A. Maldonado Maldonado - ICOM Undergrad

Abstract

In this programming assignment, we explored agent behavior in partially observable environments by implementing three different agent models: simple reflex agents, simple reflex agents with randomized functions, and state-based simple reflex agents. Simple reflex agents operate based on immediate environmental stimuli using predefined condition-action rules. We tested these agents in two distinct environments: a two-location world and a more complex grid world with obstacles, where agents needed to navigate and clean dirt while maximizing efficiency.

Our experiments demonstrated that while simple reflex agents can perform well in straightforward environments, they can become inefficient in constrained spaces, such as hallways, where they may get stuck in repetitive movement patterns. Introducing randomness to agent decision-making helped mitigate this issue in some cases but also introduced inefficiencies. By incorporating state memory, the agents improved their navigation, avoiding previously visited locations and escaping loops more effectively. The results highlight the limitations of purely reactive agents and the benefits of adding state awareness for improved rationality in dynamic environments.

Introduction

In this programming assignment, we explored agent behavior in partially observable environments by implementing and analyzing three models: simple reflex agents, simple reflex agents with randomized functions, and state-based reflex agents. “AI agents operate independently, making decisions without constant human supervision.” As stated in [1] Understanding how these agents function in different environments is essential to the study of artificial intelligence.

As further explained in [2], “Simple reflex agents are fundamental constructs in artificial intelligence (AI) that perform on a simple precept: they make choices primarily based on immediate environmental stimuli. Unlike more complex AI systems that rely on learning algorithms or neural networks, simple reflex agents depend on a set of predefined rules or conditional action pairs.” Studying agent behavior provides insights into how AI systems operate under different conditions and levels of environmental awareness.

In this project we used partially observable, dynamic environments. The environments used varied in complexity, one with only two locations and another where the agent could move in four directions while navigating around obstacles. The goal of this project was to observe how different agent models perform and to analyze how increasing an agent’s knowledge about its environment affects its efficiency. We will be applying PEAS for agents, agent types, environment types and rationality in this study of agents.

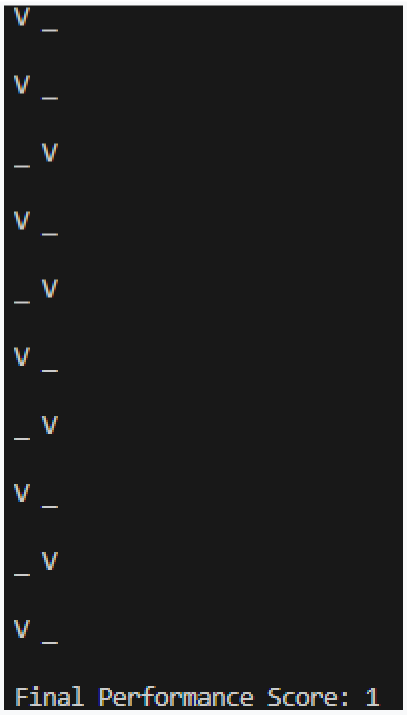
Information and Methods

The simple reflex simulates a vacuum cleaner agent interacting with a 2D grid environment. The environment is randomly populated with dirt and obstacles, and the agent starts in the left corner. The agent follows a simple reflex based strategy: it checks its current position for dirt, and if it finds any, it cleans the spot. If the spot is not dirty, the agent decides to move left or right, selecting a direction based on which would be a valid choice. The agent's behavior is governed by a performance score, where cleaning dirt increases the score and moving decreases it. The agent performs these actions over a series of steps, with the environment displayed after each step to show the agent's position and progress. The simplicity of the agent's decision making based on immediate perceptions works well in basic scenarios but could struggle in more complex environments with many obstacles, as the agent may waste time moving back and forth in confined spaces.

We then expand to four directions the agent can move to and add obstacles the agent cannot go through. A simple reflex agent would be perfectly rational for this environment. This is due to the fact that a simple reflex agent would be able to sense if its closest positions in all four directions are dirt spots, obstacles, bounds or empty spots. Although it does not have memory, it is able to efficiently traverse the map, sensing dirt as it gets close and moving towards it, while avoiding crashing into the wall.

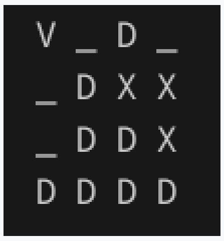
Now, can a simple reflex agent with a randomized agent function outperform this simple reflex agent? Yes. Given a hallway effect environment, our agent might end up wasting its time by moving back and forth infinitely. For reference, we have figure 1.a being the initial state of our agent and figure 1.b being the back and forth movement of our agent. However, we can still design environments where these simple reflex agents with randomized functions struggle. In maze-like environments or simply environments with a lot of obstacles, these agents might still get stuck. Our example of this is figure 2, where our randomized agent spent a lot of unnecessary time trying to get out of the hallway.

To help combat this, we can add a state to our agent. This means adding a memory system, so our agent can know where it has been and where it has not been, so then our agent can prioritize going where it has not been yet and escape hallways like those in the previous figure. State simple reflex agents can still get stuck as shown in figure 3 and 4, however from our experiments it proves to get out slightly quicker than randomized simple reflex agents.



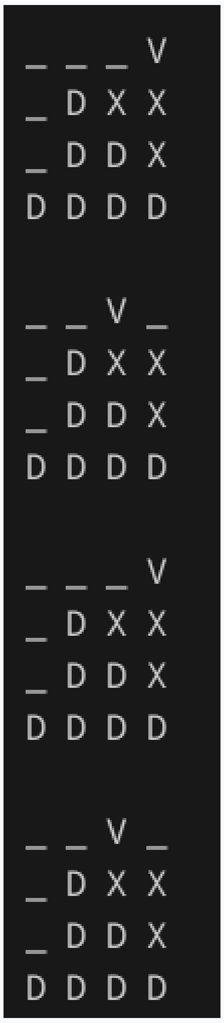
```
V _

V _

_ V

V _

_ V

V _

_ V

V _

_ V

V _

Final Performance Score: 1
```

Figure 1



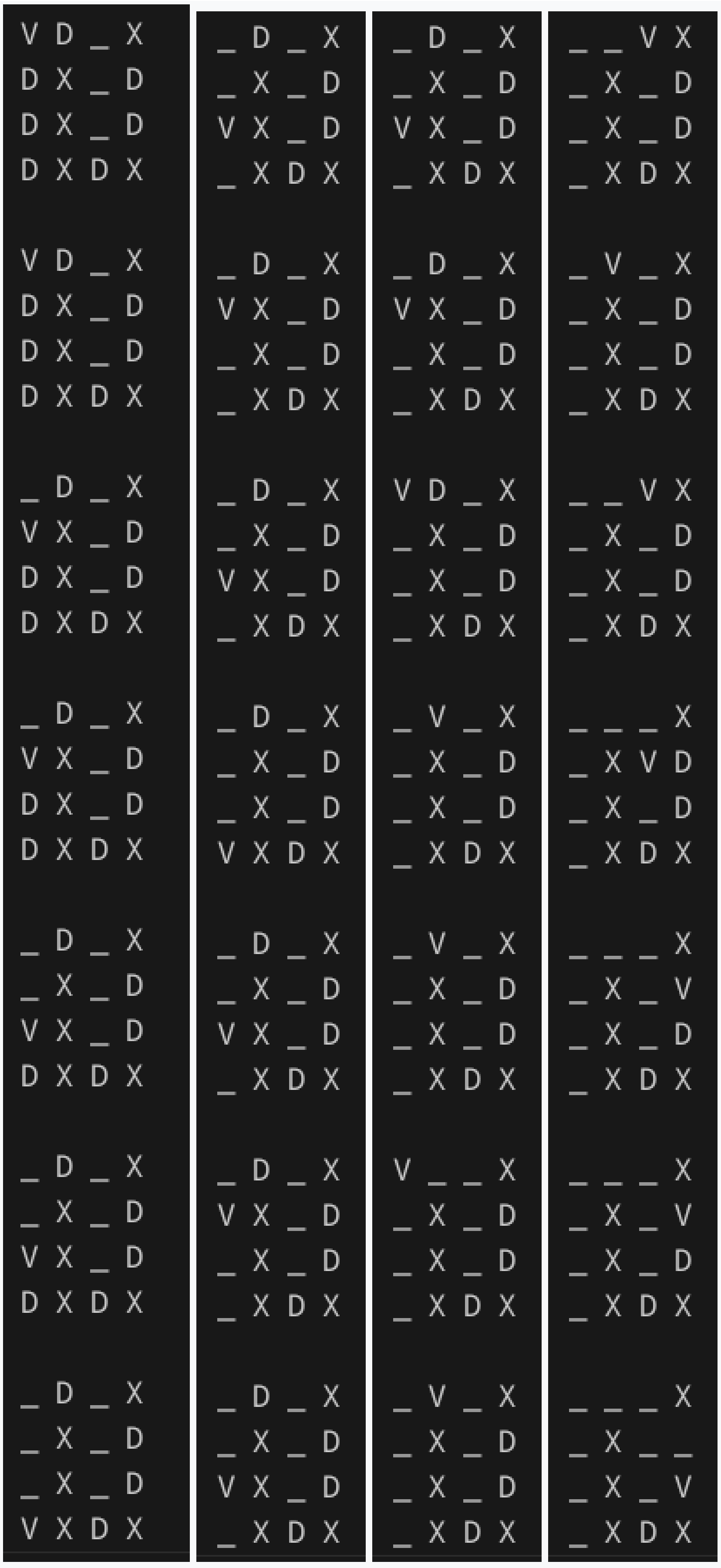
```
V _ D _
_ D X X
_ D D X
D D D D
```

figure 2.a



```
_ _ _ V
_ D X X
_ D D X
D D D D

_ _ V _
_ D X X
_ D D X
D D D D

_ _ _ V
_ D X X
_ D D X
D D D D

_ _ V _
_ D X X
_ D D X
D D D D
```

figure2 .b



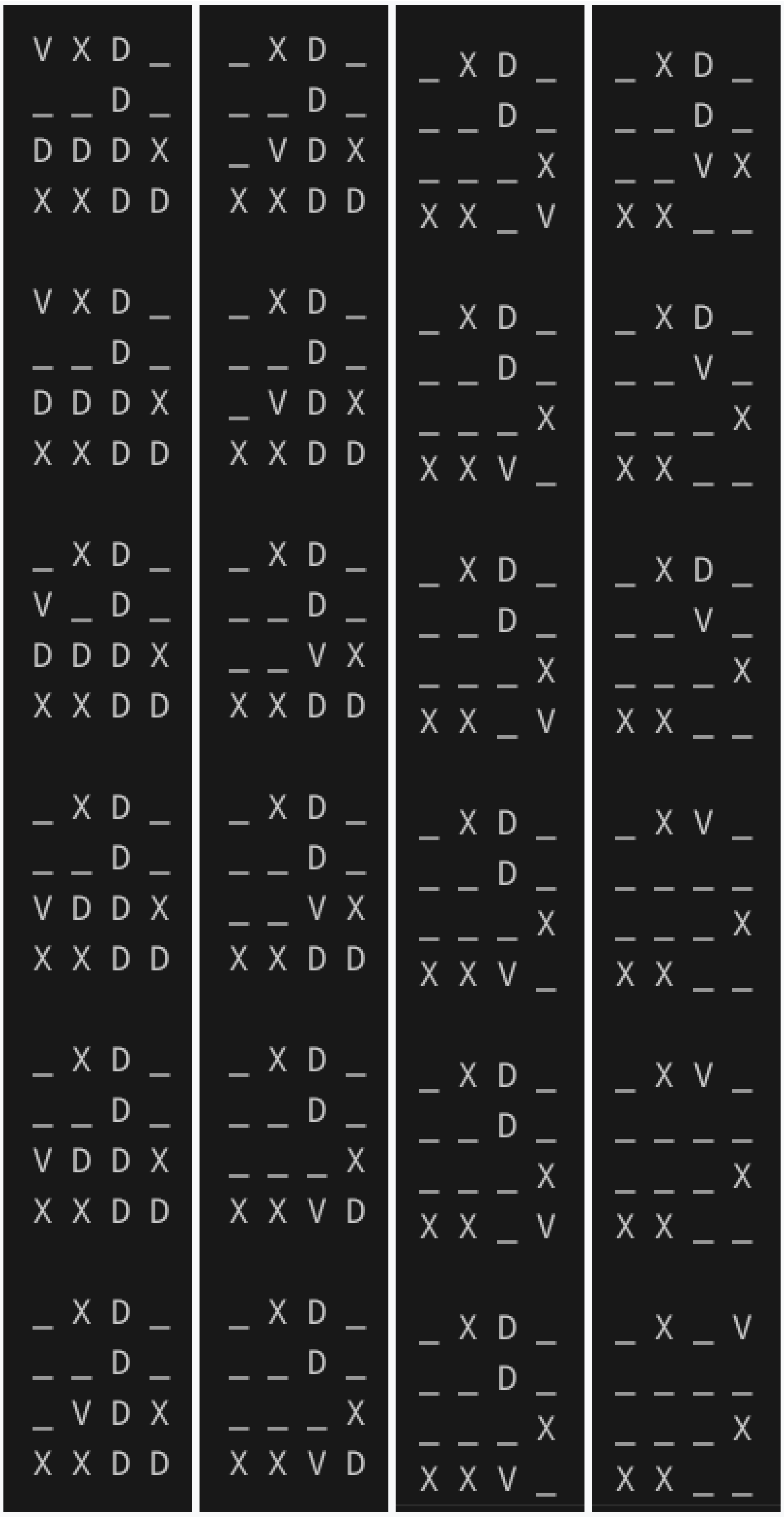
```
V D _ X
D X _ D
D X _ D
D X D X

V D _ X
D X _ D
D X _ D
D X D X

_ D _ X
V X _ D
D X _ D
D X D X

_ D _ X
V X _ D
D X _ D
D X D X

_ D _ X
_ X _ D
V X _ D
D X D X

_ D _ X
_ X _ D
V X _ D
D X D X

_ D _ X
_ X _ D
_ X _ D
V X D X
```

```
_ D _ X
_ X _ D
V X _ D
_ X D X

_ D _ X
V X _ D
_ X _ D
_ X D X

_ D _ X
_ X _ D
V X _ D
_ X D X

_ D _ X
_ X _ D
_ X _ D
V X D X

_ D _ X
_ X _ D
V X _ D
_ X D X

_ D _ X
V X _ D
_ X _ D
_ X D X

_ D _ X
_ X _ D
V X _ D
_ X D X
```

```
_ D _ X
_ X _ D
V X _ D
_ X D X

_ D _ X
V X _ D
_ X _ D
_ X D X

V D _ X
_ X _ D
_ X _ D
_ X D X

_ V _ X
_ X _ D
_ X _ D
_ X D X

_ V _ X
_ X _ D
_ X _ D
_ X D X

V _ _ X
_ X _ D
_ X _ D
_ X D X

_ V _ X
_ X _ D
_ X _ D
_ X D X
```

```
_ _ V X
_ X _ D
_ X _ D
_ X D X

_ V _ X
_ X _ D
_ X _ D
_ X D X

_ _ V X
_ X _ D
_ X _ D
_ X D X

_ _ _ X
_ X V D
_ X _ D
_ X D X

_ _ _ X
_ X _ V
_ X _ D
_ X D X

_ _ _ X
_ X _ V
_ X _ D
_ X D X

_ _ _ X
_ X _ _
_ X _ V
_ X D X
```

figure 3



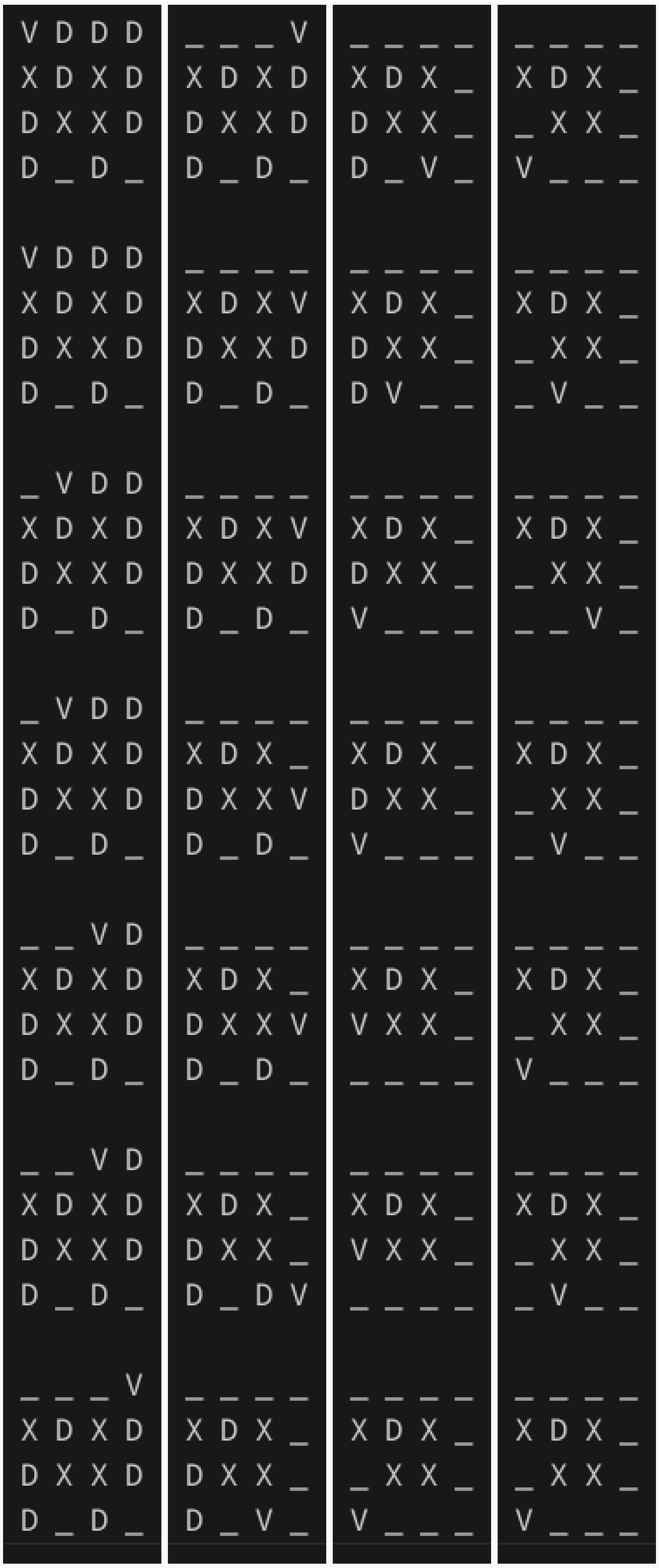
```
V X D _
_ _ D _
D D D X
X X D D

V X D _
_ _ D _
D D D X
X X D D

_ X D _
V _ D _
D D D X
X X D D

_ X D _
_ _ D _
V D D X
X X D D

_ X D _
_ _ D _
V D D X
X X D D

_ X D _
_ _ D _
_ V D X
X X D D
```

```
_ X D _
_ _ D _
_ V D X
X X D D

_ X D _
_ _ D _
_ V D X
X X D D

_ X D _
_ _ D _
_ _ V X
X X D D

_ X D _
_ _ D _
_ _ V X
X X D D

_ X D _
_ _ D _
_ _ _ X
X X V D

_ X D _
_ _ D _
_ _ _ X
X X V D
```

```
_ X D _
_ _ D _
_ _ _ X
X X _ V

_ X D _
_ _ D _
_ _ _ X
X X V _

_ X D _
_ _ D _
_ _ _ X
X X _ V

_ X D _
_ _ D _
_ _ _ X
X X V _

_ X D _
_ _ D _
_ _ _ X
X X _ V

_ X D _
_ _ D _
_ _ _ X
X X V _
```

```
_ X D _
_ _ D _
_ _ V X
X X _ _

_ X D _
_ _ V _
_ _ _ X
X X _ _

_ X D _
_ _ V _
_ _ _ X
X X _ _

_ X V _
_ _ _ _
_ _ _ X
X X _ _

_ X V _
_ _ _ _
_ _ _ X
X X _ _

_ X _ V
_ _ _ _
_ _ _ X
X X _ _
```

figure 4



```
V D D D
X D X D
D X X D
D _ D _

V D D D
X D X D
D X X D
D _ D _

_ V D D
X D X D
D X X D
D _ D _

_ V D D
X D X D
D X X D
D _ D _

_ _ V D
X D X D
D X X D
D _ D _

_ _ V D
X D X D
D X X D
D _ D _

_ _ _ V
X D X D
D X X D
D _ D _
```

```
_ _ _ V
X D X D
D X X D
D _ D _

_ _ _ _
X D X V
D X X D
D _ D _

_ _ _ _
X D X V
D X X D
D _ D _

_ _ _ _
X D X _
D X X V
D _ D _

_ _ _ _
X D X _
D X X V
D _ D _

_ _ _ _
X D X _
D X X _
D _ D V

_ _ _ _
X D X _
D X X _
D _ V _
```

```
_ _ _ _
X D X _
D X X _
D _ V _

_ _ _ _
X D X _
D X X _
D V _ _

_ _ _ _
X D X _
D X X _
V _ _ _

_ _ _ _
X D X _
D X X _
V _ _ _

_ _ _ _
X D X _
V X X _
_ _ _ _

_ _ _ _
X D X _
V X X _
_ _ _ _

_ _ _ _
X D X _
_ X X _
V _ _ _
```

```
_ _ _ _
X D X _
_ X X _
V _ _ _

_ _ _ _
X D X _
_ X X _
_ V _ _

_ _ _ _
X D X _
_ X X _
_ _ V _

_ _ _ _
X D X _
_ X X _
_ V _ _

_ _ _ _
X D X _
_ X X _
V _ _ _

_ _ _ _
X D X _
_ X X _
_ V _ _

_ _ _ _
X D X _
_ X X _
V _ _ _
```

Figura 5

Exercise 2.11

```
import random

# Environment Class: Defines the environment where the agent operates
class Environment:
    def __init__(self, width=2, height=2, dirt_probability=0.5,
obstacle_probability=0.2):
        self.width = width
        self.height = height

        # Random dirt placement
        self.grid = [[random.random() < dirt_probability for _ in
range(width)] for _ in range(height)]

        # Random obstacle placement (True represents an obstacle)
        self.obstacles = [[random.random() < obstacle_probability for _
in range(width)] for _ in range(height)]
        self.agent_position = (0, 0)

    def is_dirty(self, x, y):
        return self.grid[y][x]

    def is_obstacle(self, x, y):
        return self.obstacles[y][x]

    def clean(self, x, y):
        self.grid[y][x] = False

    def move_agent(self, new_x, new_y):
        if 0 <= new_x < self.width and 0 <= new_y < self.height:
            self.agent_position = (new_x, new_y)

    def display(self):
        for y, row in enumerate(self.grid):
            for x, cell in enumerate(row):
                if (x, y) == self.agent_position:
                    print("V", end=" ")  # Agent's position
                elif self.is_obstacle(x, y):
                    print("X", end=" ")  # Obstacle
                else:
                    print("D" if cell else "_", end=" ")  # Dirt or
empty space
            print()
        print()

# Vacuum Cleaner Agent Class: Defines the agent's behavior
class VacuumCleaner:
    def __init__(self, environment):
        self.env = environment
        self.x, self.y = self.env.agent_position
        self.performance = 0

    def perceive(self):
        return self.env.is_dirty(self.x, self.y)

    def clean(self):
        if self.perceive():
            self.env.clean(self.x, self.y)
            self.performance += 10  # Gain performance points for
cleaning

    def move(self, direction):
        dx, dy = direction
        new_x, new_y = self.x + dx, self.y + dy
        if 0 <= new_x < self.env.width and not
self.env.is_obstacle(new_x, new_y):
            self.env.move_agent(new_x, new_y)
            self.x, self.y = new_x, new_y
            self.performance -= 1  # Cost for moving

    def act(self):
        if self.perceive():
            self.clean()
        else:
            self.move(self.check_surr())

    def check_surr(self):
        options = [(1, 0), (-1, 0)]  # Only move Right or Left
        valid_options = [move for move in options if 0 <= self.x +
move[0] < self.env.width and not self.env.is_obstacle(self.x + move[0],
self.y)]
        return random.choice(valid_options) if valid_options else (0,
0)  # Stay in place if no valid move
```

```
    def run(self, steps=10):
        for _ in range(steps):
            self.env.display()
            self.act()
        print("Final Performance Score:", self.performance)

if __name__ == "__main__":
    env = Environment(width=2, height=1, dirt_probability=0.3,
obstacle_probability=0)
    vacuum = VacuumCleaner(env)
    vacuum.run(steps=10)
```

Code 1

Exercise 2.14

```
import random
from collections import namedtuple

class Environment:
   def __init__(self, width, height, dirt_probability ,
obstacle_probability):
       self.width = width
       self.height = height
       self.grid = [[random.random() < dirt_probability for _ in
range(width)] for _ in range(height)]
       self.obstacles = [[random.random() < obstacle_probability for _
in range(width)] for _ in range(height)]
       self.agent_position = (0, 0)

   def is_dirty(self, x, y):
       return self.grid[y][x]

   def is_obstacle(self, x, y):
       return self.obstacles[y][x]

   def clean(self, x, y):
       self.grid[y][x] = False
```

```python
    def move_agent(self, new_x, new_y):
        if 0 <= new_x < self.width and 0 <= new_y < self.height:
            self.agent_position = (new_x, new_y)

    def display(self):

        for y, row in enumerate(self.grid):
            for x, cell in enumerate(row):
                if (x, y) == self.agent_position:
                    print("V", end=" ")
                elif self.is_obstacle(x, y):
                    print("X", end=" ")
                else:
                    print("D" if cell else "_", end=" ")
            print()
        print()


class VacuumCleaner:
    def __init__(self, environment):
        self.env = environment
        self.x, self.y = self.env.agent_position
        self.performance = 0

    def perceive(self):
        return self.env.is_dirty(self.x, self.y)

    def clean(self):
        if self.perceive():
            self.env.clean(self.x, self.y)
            self.performance += 10

    def move(self, direction):
        dx, dy = direction
        new_x, new_y = self.x + dx, self.y + dy
        if 0 <= new_x < self.env.width and 0 <= new_y < self.env.height
and not self.env.is_obstacle(new_x, new_y):
            self.env.move_agent(new_x, new_y)
            self.x, self.y = new_x, new_y
            self.performance -= 1
    def act(self):
        if self.perceive():
            self.clean()
        else:
            self.move(self.check_surr())

    def check_surr(self):
        options = [(1, 0), (-1, 0), (0, 1), (0, -1)]
        valid_options = []  # list for valid moves

        # checking for valid moves
        for dx, dy in options:
            new_x, new_y = self.x + dx, self.y + dy
            if (0 <= new_x < self.env.width and 0 <= new_y <
self.env.height and
                    not self.env.is_obstacle(new_x, new_y)):
                valid_options.append((dx, dy))

        # dirt check
        for dx, dy in valid_options:
            if self.env.is_dirty(self.x + dx, self.y + dy):
                return (dx, dy)

        # choose a random valid move
        if valid_options:
            return valid_options[0]
        else:
            return (0, 0)

    def run(self, steps):
        for _ in range(steps):
            self.env.display()
            self.act()
        print("Final Performance Score:", self.performance)

if __name__ == "__main__":
    width=4
    height=4
    env = Environment(width, height, dirt_probability=0.5,
obstacle_probability=0.25)
    vacuum = VacuumCleaner(env)
    vacuum.run(steps=width * height*2)
```

Code 2 - Simple Reflex Agent in a 2D environment with obstacles.

```
class VacuumCleaner:
   def __init__(self, environment):
       self.env = environment
       self.x, self.y = self.env.agent_position
       self.performance = 0

   def perceive(self):
       return self.env.is_dirty(self.x, self.y)

   def clean(self):
       if self.perceive():
           self.env.clean(self.x, self.y)
           self.performance += 10

   def move(self, direction):
       dx, dy = direction
       new_x, new_y = self.x + dx, self.y + dy
       if 0 <= new_x < self.env.width and 0 <= new_y < self.env.height
and not self.env.is_obstacle(new_x, new_y):
           self.env.move_agent(new_x, new_y)
           self.x, self.y = new_x, new_y
           self.performance -= 1

   def act(self):
       if self.perceive():
           self.clean()
       else:
           self.move(self.check_surr())

   def check_surr(self):
       options = [(1, 0), (-1, 0), (0, 1), (0, -1)]
       valid_options = []

       for dx, dy in options:
           new_x, new_y = self.x + dx, self.y + dy
           if (0 <= new_x < self.env.width and 0 <= new_y <
self.env.height and
                   not self.env.is_obstacle(new_x, new_y)):
               valid_options.append((dx, dy))

       dirty_options = []
```

```python
        for dx, dy in valid_options:
            if self.env.is_dirty(self.x + dx, self.y + dy):
                dirty_options.append((dx, dy))

        if dirty_options:
            # Randomly choose from dirty options
            return random.choice(dirty_options)

        if valid_options:
            # Randomly choose from valid options
            return random.choice(valid_options)
        else:
            return (0, 0)

    def run(self, steps):
        for _ in range(steps):
            self.env.display()
            self.act()
        print("Final Performance Score:", self.performance)
```

Code 3 - Simple Reflex VacuumCleaner Class with randomized functions

```python
class VacuumCleaner:
    def __init__(self, environment):
        self.env = environment
        self.x, self.y = self.env.agent_position
        self.performance = 0
        self.visited = set()

    def perceive(self):
        return self.env.is_dirty(self.x, self.y)

    def clean(self):
        if self.perceive():
            self.env.clean(self.x, self.y)
            self.performance += 10

    def move(self, direction):
        dx, dy = direction
        new_x, new_y = self.x + dx, self.y + dy
        if 0 <= new_x < self.env.width and 0 <= new_y < self.env.height and not self.env.is_obstacle(new_x, new_y):
            self.env.move_agent(new_x, new_y)
            self.x, self.y = new_x, new_y
            self.performance -= 1
            self.visited.add((self.x,self.y))

    def act(self):
        if self.perceive():
            self.clean()
        else:
            self.move(self.check_surr())

    def check_surr(self):
        options = [(1, 0), (-1, 0), (0, 1), (0, -1)]
        valid_options = []

        for dx, dy in options:
            new_x, new_y = self.x + dx, self.y + dy
            if (0 <= new_x < self.env.width and 0 <= new_y <
self.env.height and
                    not self.env.is_obstacle(new_x, new_y)):
                valid_options.append((dx, dy))

        for dx, dy in valid_options:
            if self.env.is_dirty(self.x + dx, self.y + dy):
                return (dx, dy)

                # Use visited set to avoid revisiting
        unvisited_options = [(dx, dy) for dx, dy in valid_options if
(self.x + dx, self.y + dy) not in self.visited]

        if unvisited_options:
            return random.choice(unvisited_options)
        elif valid_options:
            # Randomly choose from valid options
            return random.choice(valid_options)
        else:
            return (0, 0)

    def run(self, steps):
        for _ in range(steps):
            self.env.display()
            self.act()
        print("Final Performance Score:", self.performance)
```

Code 4 - Vacuum Cleaner class with states implemented

Conclusion

We conclude that this research highlights the strengths and limitations of various agent models in partially observable, dynamic environments. While simple reflex agents perform well in basic scenarios, they become inefficient in complex environments, such as hallways or maze like grids, where they may get stuck in repetitive movement patterns. Introducing randomness in the agents decision making provides some flexibility, but it also leads to inefficiencies, especially in environments with many obstacles. By adding state memory, the agent can better avoid previously visited locations, improving its navigation and ability to escape loops. These findings demonstrate that although reactive agents can function effectively in simple environments, incorporating memory and state awareness enhances their performance and rationality, allowing them to better adapt to dynamic and complex situations.

Future Work

Our observations indicate that even state-based reflex agents struggle with solving complex problems. To improve efficiency and adaptability, future work should focus on studying and testing more advanced agent models. Exploring agents with learning capabilities, could provide valuable insights into optimizing decision-making in dynamic environments.

Work Distribution:

Simple reflex agent and base code was done by Alejandro J Rodríguez Burgos
Randomized simple reflex agent and state simple state agent was done by Miguel A. Maldonado Maldonado
Report Prepared by Both
Slides Prepared by Both
Video Prepared by Both

Appendix:

[1]V. Chugani, “Understanding AI Agents: The Future of Autonomous Systems,” *Datacamp.com*, Jan. 13, 2025. https://www.datacamp.com/blog/ai-agents (accessed Mar. 07, 2025).

[2]“Simple Reflex Agents in AI,” *GeeksforGeeks*, May 14, 2024. https://www.geeksforgeeks.org/simple-reflex-agents-in-ai/

[3] A. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 2021. [Online]. Available: https://doi.org/10.1093/mind/LIX.236.433 (accessed Mar. 05, 2025)