Samriddhi College

[T.U. Affiliated]

Lokanthali-16, Bhaktapur

Lab Report on

Artificial Intelligence

Submitted By

Name: Nabin Timalsina
Roll No: 14
Year/ Semester: 2080/ 4th Semester

Submitted To

Department of Bachelor of Science in CSIT

INDEX

Name of Student: Nabin Timalsina **Subject: Artificial Intelligence**

S.N	Title	Date	Signature	Remarks
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				

LAB:1 Different AI Agents (Snake Game)

An AI agent is a *software program that can interact with its surroundings, gather information, and use that information to complete tasks on its own to achieve goals set by humans.*

How do AI Agents Work?

AI agents follow a structured process to perceive, analyze, decide, and act within their environment. Here's an overview of how AI agents operate:

1. Collecting Information (Perceiving the Environment)

AI agents gather information from their surroundings through various means:

- Sensors: For example, a self-driving car uses cameras and radar to detect objects.
- User Input: Chatbots read text or listen to voice commands.
- Databases & Documents: Virtual assistants search records or knowledge bases for relevant data.

2. Processing Information & Making Decisions

After gathering data, AI agents analyze it and decide what to do next. Some agents rely on pre-set rules, while others utilize machine learning to predict the best course of action. Advanced agents may also use retrieval-augmented generation (RAG) to access external databases for more accurate responses.

3. Taking Action (Performing Tasks)

Once an agent makes a decision, it performs the required task, such as:

- Answering a customer query in a chatbot.
- Controlling a device, like a smart assistant turning off lights.
- Running automated tasks, such as processing orders on an online store.

4. Learning & Improving Over Time

Some AI agents can learn from past experiences to improve their responses. This self-learning process, often referred to as reinforcement learning, allows agents to refine their behavior over time. For example, a recommendation system on a streaming platform learns users' preferences and suggests content accordingly.

Types of Agents

1. Simple Reflex Agents

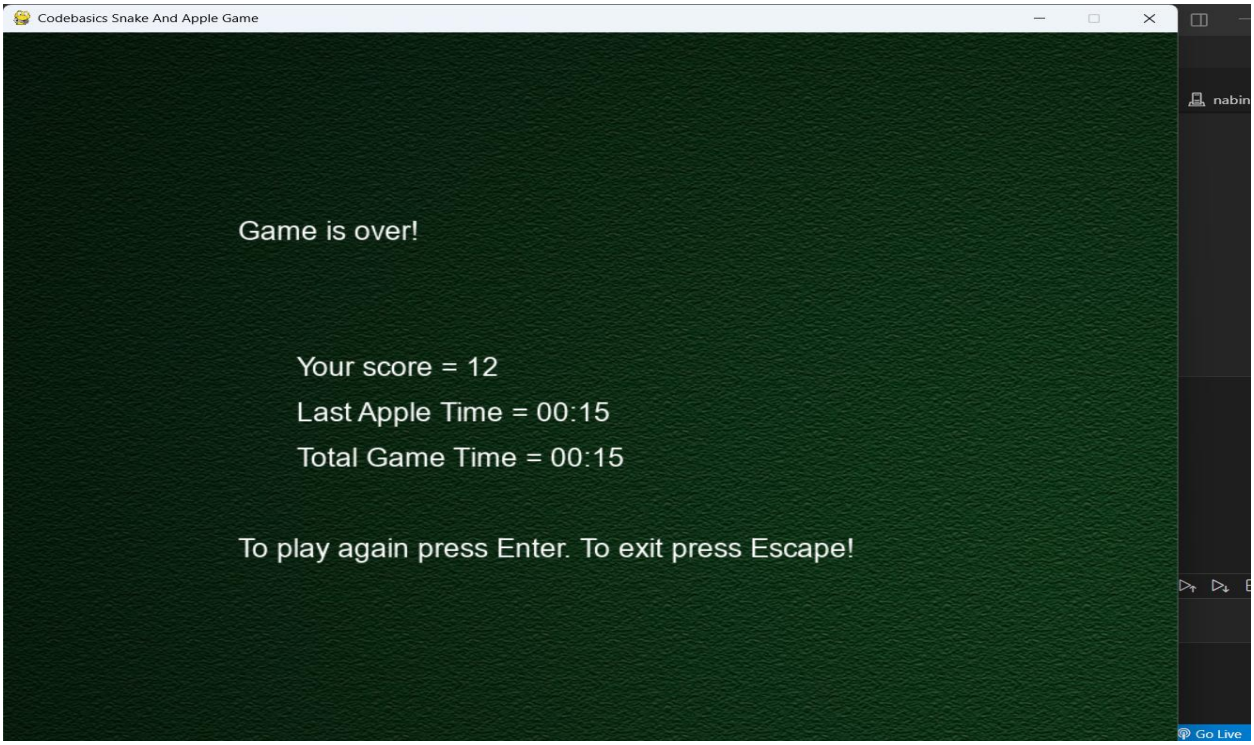
Simple reflex agents act solely based on the current percept and, percept history (record of past perceptions) is ignored by these agents. Agent function is defined by condition-action rules.

A condition-action rule maps a state (condition) to an action.

- If the condition is true, the associated action is performed.
- If the condition is false, no action is taken.

Source code of Simple reflex agent in snake game:

```
# --- Simple AI Agent ---
    def simple_agent(self):
        head_x, head_y = self.snake.x[0], self.snake.y[0]
        apple_x, apple_y = self.apple.x, self.apple.y

        # Move horizontally toward apple if not aligned
        if head_x < apple_x and self.snake.direction != 'left':
            self.snake.move_right()
        elif head_x > apple_x and self.snake.direction != 'right':
            self.snake.move_left()
        # If aligned horizontally, move vertically toward apple
        elif head_y < apple_y and self.snake.direction != 'up':
            self.snake.move_down()
        elif head_y > apple_y and self.snake.direction != 'down':
            self.snake.move_up()
        # If on apple, no direction change needed
```



2. Model-Based Reflex Agents

Model-based reflex agents finds a rule whose condition matches the current situation or percept. It uses a model of the world to handle situations where the environment is only partially observable.

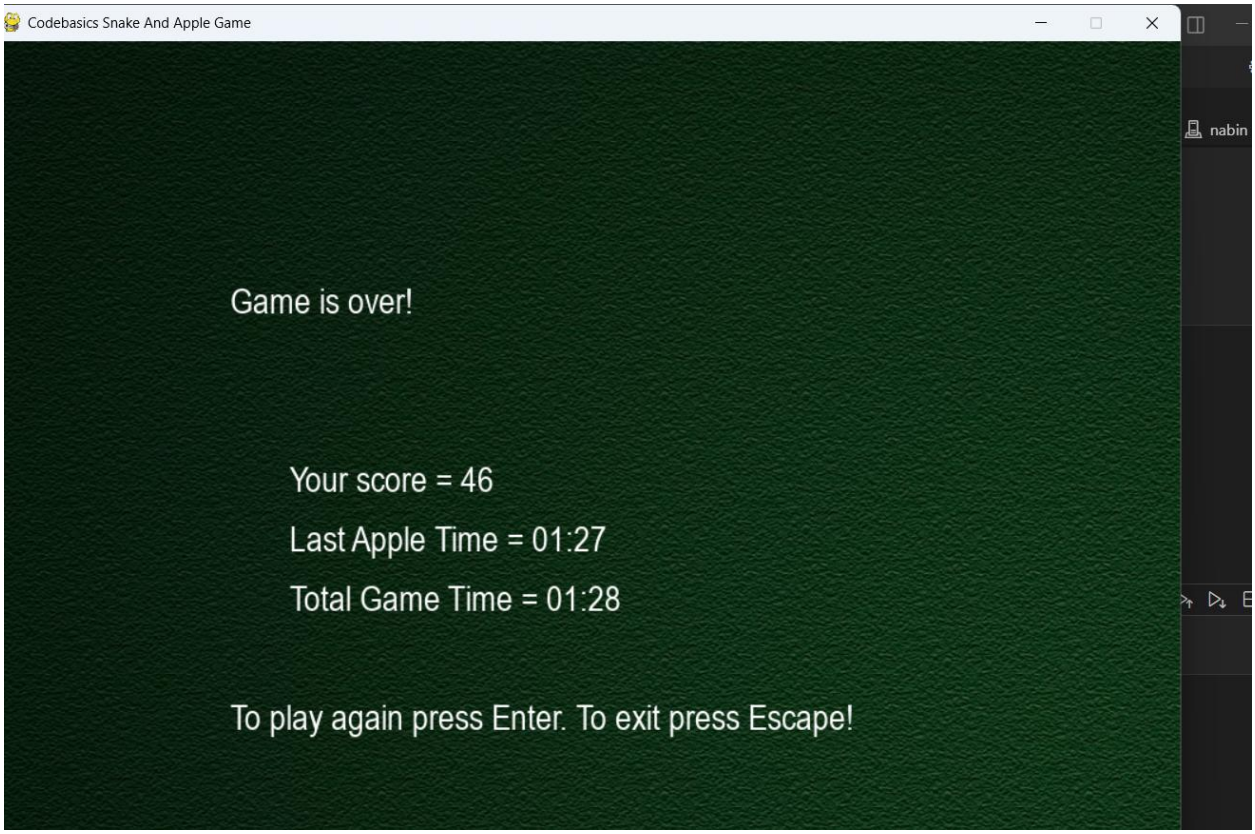
- The agent tracks its internal state, which is adjusted based on each new percept.
- The internal state depends on the percept history (the history of what the agent has perceived so far).

The agent stores the current state internally, maintaining a structure that represents the parts of the world that cannot be directly seen or perceived. The process of updating the agent's state requires information about:

- How the world evolves independently from the agent?
- How the agent's actions affect the world?

Source code of Model-Based reflex agent in snake game:

```
# MODEL-BASED AI AGENT:
    def model_based_agent(self):
        head_x, head_y = self.snake.x[0], self.snake.y[0]
        apple_x, apple_y = self.apple.x, self.apple.y
    # Define all possible moves and their corresponding direction changes
        possible_moves = {
            'left': (head_x - SIZE, head_y),
            'right': (head_x + SIZE, head_y),
            'up': (head_x, head_y - SIZE),
            'down': (head_x, head_y + SIZE)
        }
        # Prevent immediate reverse
        opposite = {'left': 'right', 'right': 'left', 'up': 'down', 'down':
'up'}
        # Remove opposite direction move to avoid U-turn
        if opposite[self.snake.direction] in possible_moves:
            del possible_moves[opposite[self.snake.direction]]
        # Filter safe moves (no wall or self collision)
        safe_moves = {}
        for direction, (x, y) in possible_moves.items():
            if not self._will_collide(x, y):
                safe_moves[direction] = (x, y)
        # If no safe moves (trapped), keep current direction (will cause game
over)
         if not safe_moves:
             return  # No safe move, do nothing
```



3. Goal-Based Agents

Goal-based agents make decisions based on their current distance from the goal and every action the agent aims to reduce the distance from goal. They can choose from multiple possibilities, selecting the one that best leads to the goal state.

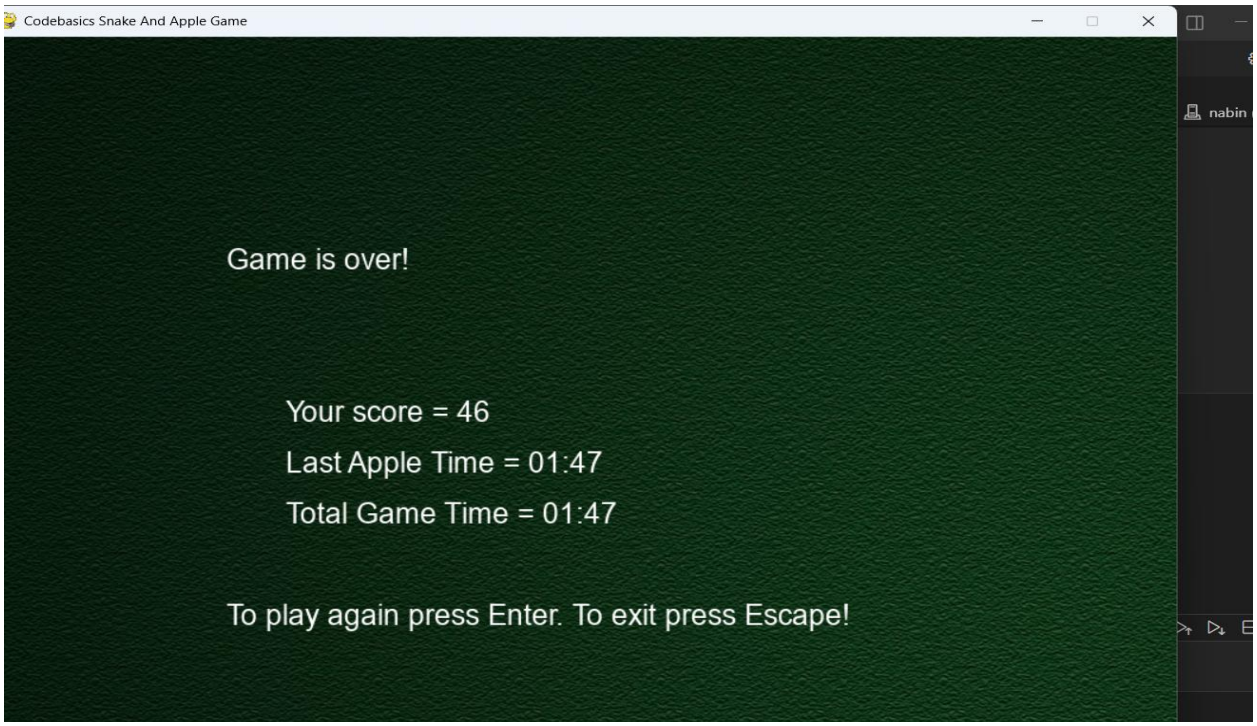
- Knowledge that supports the agent's decisions is represented explicitly, meaning it's clear and structured. It can also be modified, allowing for adaptability.
- The ability to modify the knowledge makes these agents more flexible in different environments or situations.
-

Source code of Goal-Based agent in snake game:

```
# Goal-Based AI Agent
  def goal_based_agent(self):
      head_x, head_y = self.snake.x[0], self.snake.y[0]
      apple_x, apple_y = self.apple.x, self.apple.y
```

```
possible_moves = []
for direction in ['left', 'right', 'up', 'down']:
    if direction == 'left' and self.snake.direction == 'right':
        continue
    if direction == 'right' and self.snake.direction == 'left':
        continue
    if direction == 'up' and self.snake.direction == 'down':
        continue
    if direction == 'down' and self.snake.direction == 'up':
        continue
    if self._is_move_safe(direction):
        next_x, next_y = head_x, head_y
        if direction == 'left':
            next_x -= SIZE
        elif direction == 'right':
            next_x += SIZE
        elif direction == 'up':
            next_y -= SIZE
        elif direction == 'down':
            next_y += SIZE
        dist = self._distance(next_x, next_y, apple_x, apple_y)
        possible_moves.append((dist, direction))
if possible_moves:
    # Choose move with minimum distance to apple
    possible_moves.sort(key=lambda x: x[0])
    best_move = possible_moves[0][1]

    if best_move == 'left':
```

```
                self.snake.move_left()
            elif best_move == 'right':
                self.snake.move_right()
            elif best_move == 'up':
                self.snake.move_up()
            elif best_move == 'down':
                self.snake.move_down()
        else:
            # No safe moves? Just continue current direction (may cause
collision)
            Pass
```



4. Utility-Based Agents

Utility-based agents are designed to make decisions that optimize their performance by evaluating the preferences (or utilities) for each possible state. These agents assess multiple alternatives and choose the one that maximizes their utility, which is a measure of how desirable or "happy" a state is for the agent.

- Achieving the goal is not always sufficient; for example, the agent might prefer a quicker, safer, or cheaper way to reach a destination.
- The utility function is essential for capturing this concept, mapping each state to a real number that reflects the agent's happiness or satisfaction with that state.

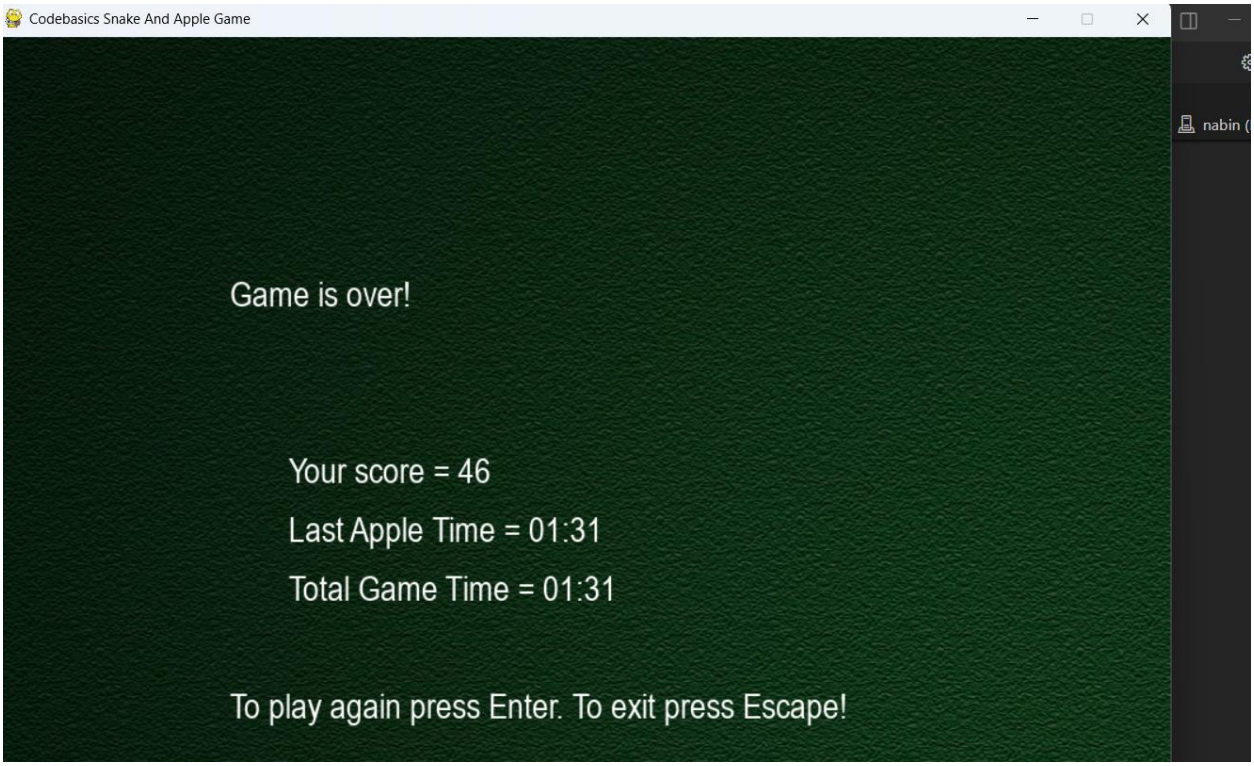
Since the world is often uncertain, utility-based agents choose actions that maximize expected utility, ensuring they make the most favorable decision under uncertain conditions.

Source code of Utility-Based agent in snake game:

```
# --- Utility-Based AI Agent ---
    def utility_agent(self):
        head_x, head_y = self.snake.x[0], self.snake.y[0]
        apple_x, apple_y = self.apple.x, self.apple.y

        directions = ['left', 'right', 'up', 'down']
        best_utility = -float('inf')
        best_move = None

        for direction in directions:
            if self.snake.direction == 'left' and direction == 'right':
                continue  # no U-turns
            if self.snake.direction == 'right' and direction == 'left':
                continue
            if self.snake.direction == 'up' and direction == 'down':
                continue
```

```
            if self.snake.direction == 'down' and direction == 'up':
                continue

            # Calculate next head position if we take this direction
            if direction == 'left':
                nx = head_x - SIZE
                ny = head_y
            elif direction == 'right':
                nx = head_x + SIZE
                ny = head_y
            elif direction == 'up':
                nx = head_x
                ny = head_y - SIZE
            else:  # down
                nx = head_x
                ny = head_y + SIZE

            # Check for collisions if move is taken
            if self.check_wall_collision_pos(nx, ny) or
self.self_collision_pos(nx, ny):
                utility = -1000  # very bad move
            else:
                # Utility = - distance to apple + length bonus
                dist = math.sqrt((apple_x - nx)**2 + (apple_y - ny)**2)
                utility = -dist + (self.snake.length * 10)

            if utility > best_utility:
                best_utility = utility
                best_move = direction
```

```
# Execute best move
if best_move == 'left':
    self.snake.move_left()
elif best_move == 'right':
    self.snake.move_right()
elif best_move == 'up':
    self.snake.move_up()
elif best_move == 'down':
    self.snake.move_down()
```



5. Learning Agent

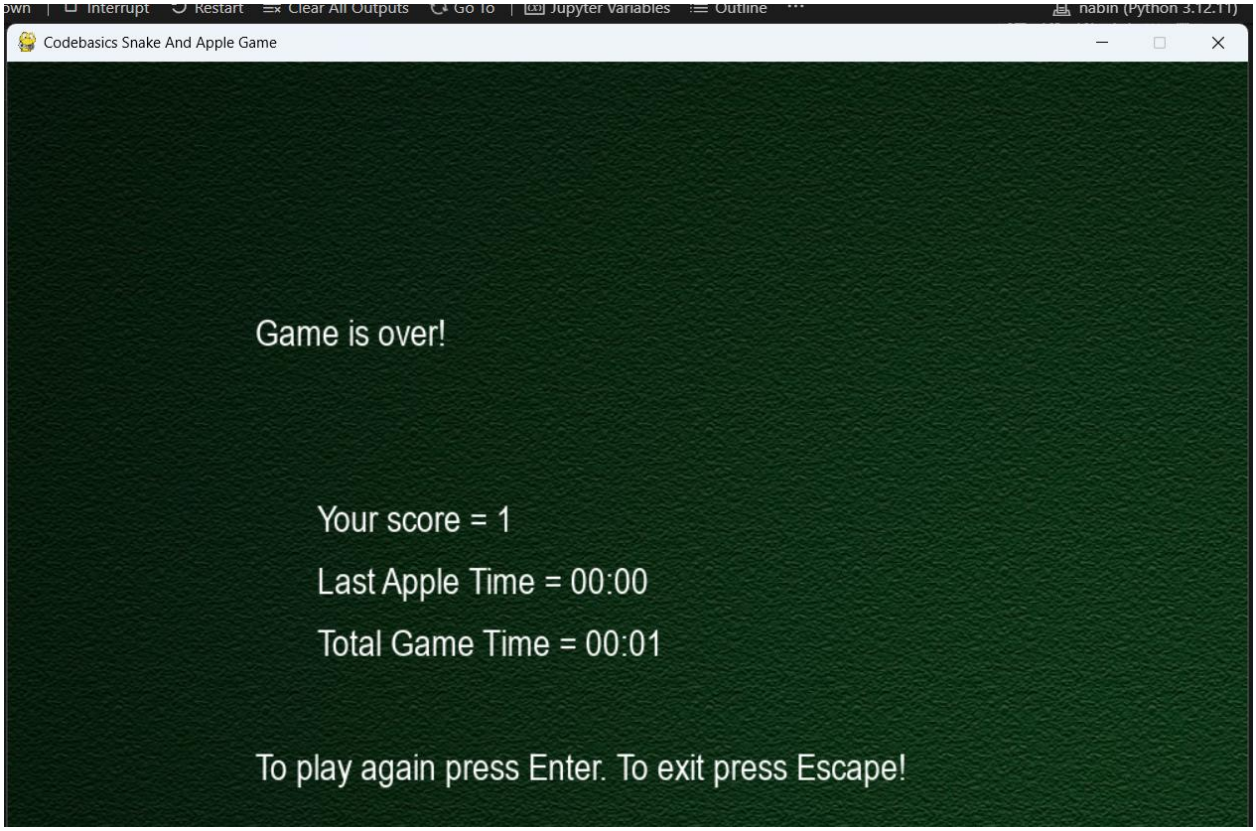
A learning agent in AI is the type of agent that can learn from its past experiences or it has learning capabilities. It starts to act with basic knowledge and then is able to act and adapt automatically through learning. A learning agent has mainly four conceptual components, which are:

1. **Learning element:** It is responsible for making improvements by learning from the environment.
2. **Critic:** The learning element takes feedback from critics which describes how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** It is responsible for selecting external action.
4. **Problem Generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Source code of Learning agent in snake game:

```
# --- Q-Learning Agent Implementation ---

    def get_state(self):

        head_x, head_y = self.snake.x[0], self.snake.y[0]

        apple_x, apple_y = self.apple.x, self.apple.y


        # Danger straight, right, left? Based on current direction

        danger_straight = self.danger_in_direction(self.snake.direction)

        danger_right =
self.danger_in_direction(self.turn_right(self.snake.direction))

        danger_left =
self.danger_in_direction(self.turn_left(self.snake.direction))


        # Apple relative location

        apple_left = apple_x < head_x

        apple_right = apple_x > head_x

        apple_up = apple_y < head_y
```

```
        apple_down = apple_y > head_y


        # Current direction
        dir_left = self.snake.direction == 'left'
        dir_right = self.snake.direction == 'right'
        dir_up = self.snake.direction == 'up'
        dir_down = self.snake.direction == 'down'


        # State is a tuple of these boolean flags (can be converted to a
string or tuple)
        state = (
            danger_straight,
            danger_right,
            danger_left,
            apple_left,
            apple_right,
            apple_up,
            apple_down,
            dir_left,
            dir_right,
            dir_up,
            dir_down
        )
        return state
```



Discussion

In this lab, we implemented and tested five different types of AI agents—Simple Reflex, Model-Based, Goal-Based, Utility-Based, and Learning-Based (Q-learning)—in the classic Snake game using Python and Pygame. Each agent had a unique way of perceiving the game environment and making movement decisions to maximize the score by eating apples while avoiding collisions.

Conclusion

In real-world applications, combining the strengths of these agents—like model-awareness with learning capability—can result in more efficient and robust AI systems. This lab provided a strong foundation in AI agent design, simulation, and evaluation in a controlled game environment.