



OBJECT-ORIENTED PROGRAMMING

FINAL PROJECT

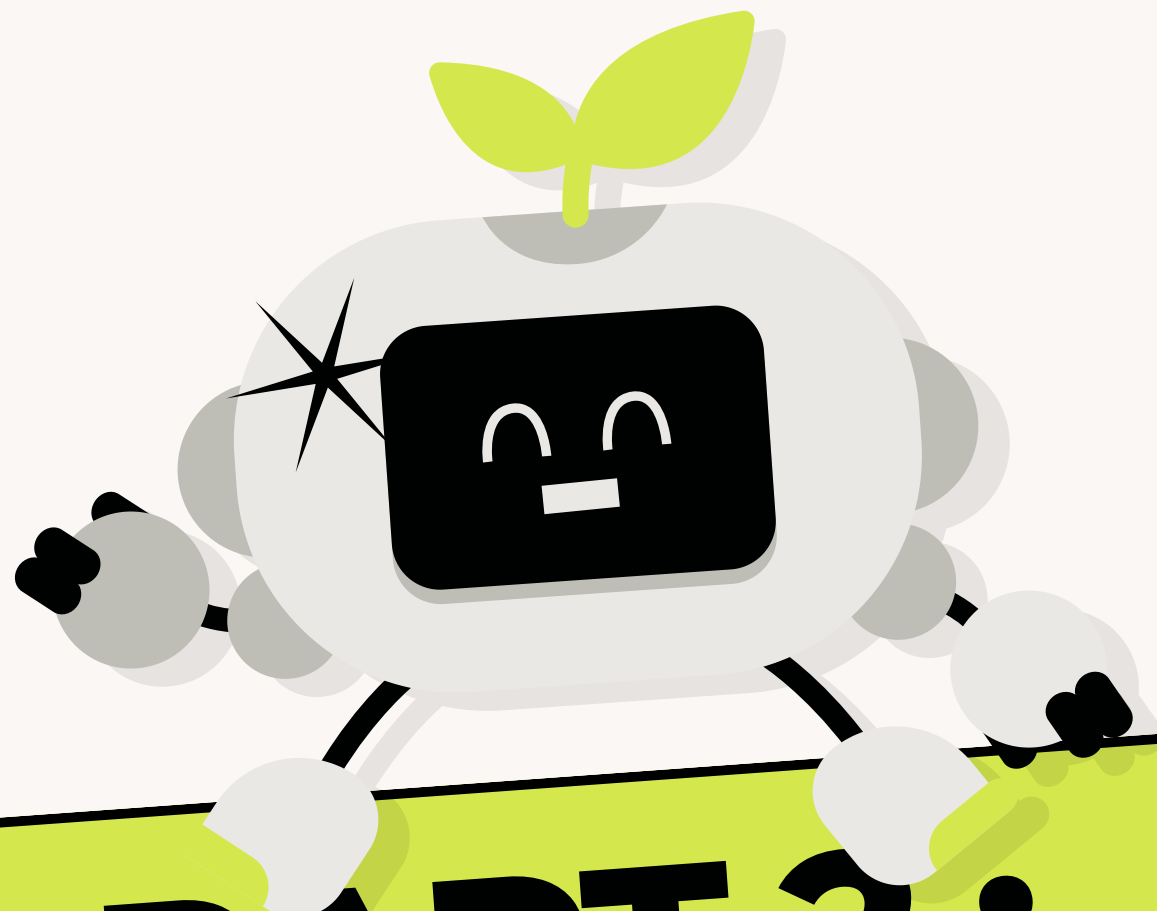
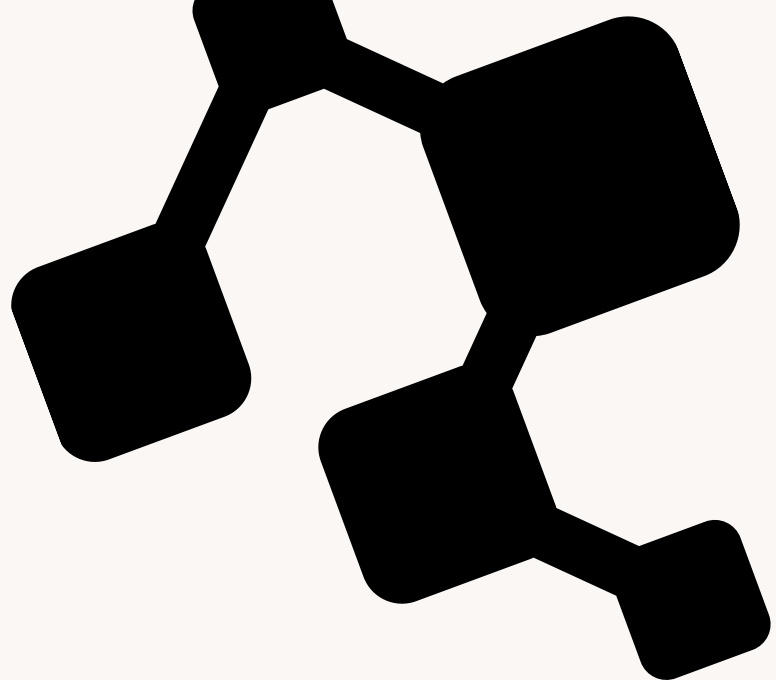


B123245019 李佩萱

B123040055 松尾春輝

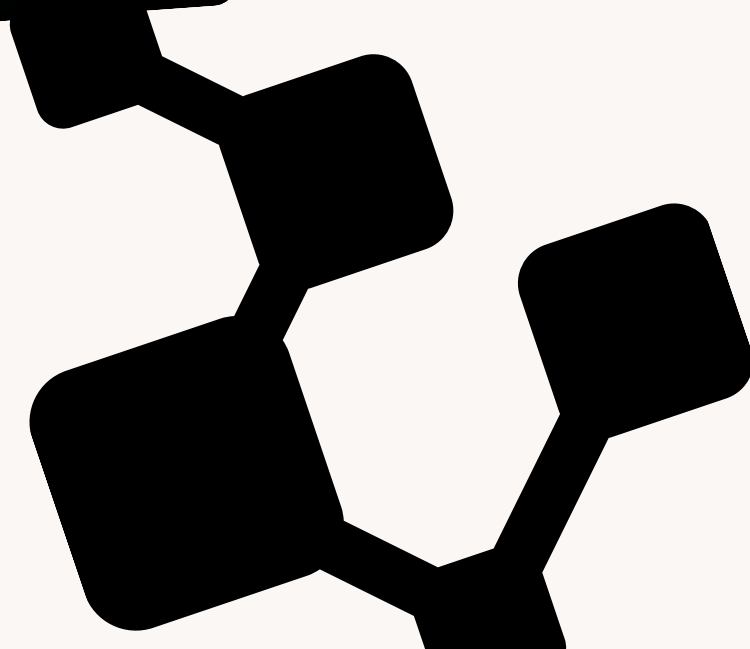
B112045001 湯昆璋



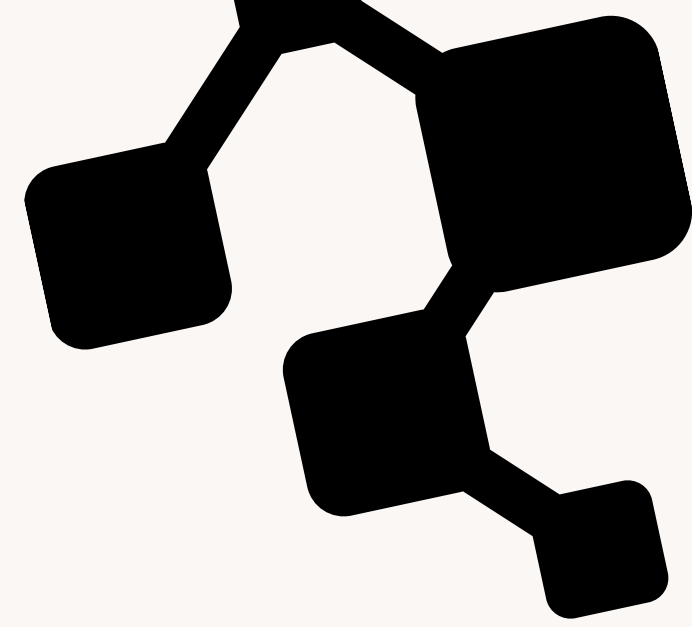


PART 2:

FROZEN LAKE



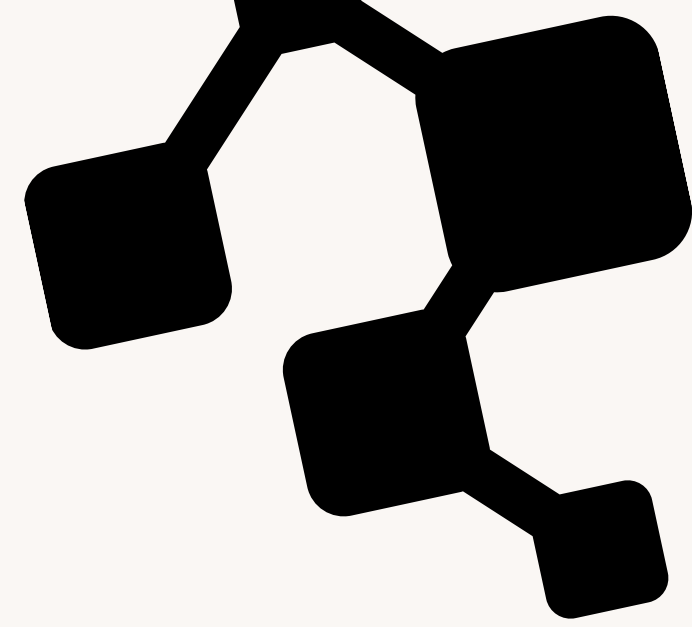
MODEL-BASED REINFORCEMENT LEARNING FOR FROZENLAKE



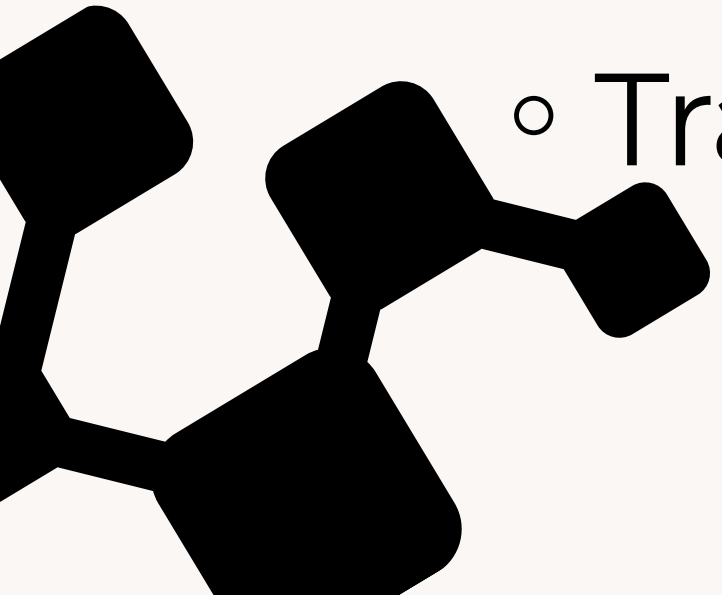
- In this project, we implemented a **Model-Based Reinforcement Learning agent** to solve the FrozenLake environment using Python and object-oriented design.
- The goal is to learn an environment model and improve the policy through **Value Iteration**.



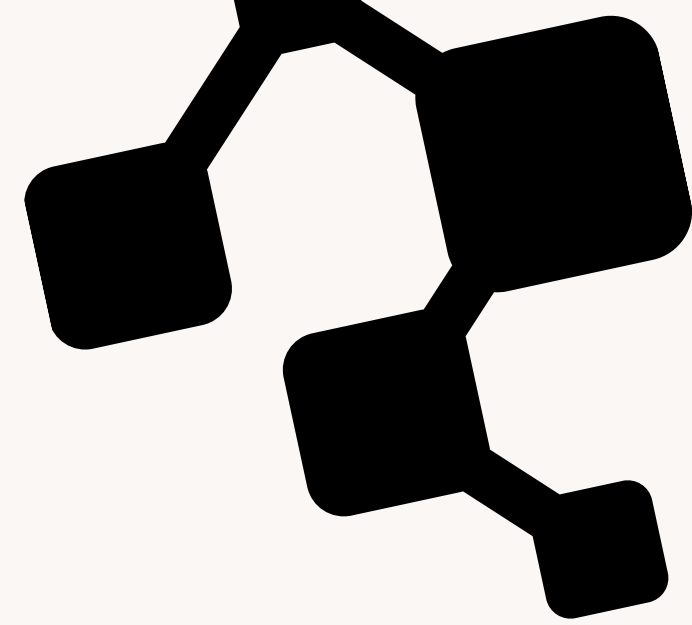
PROBLEM SETTING



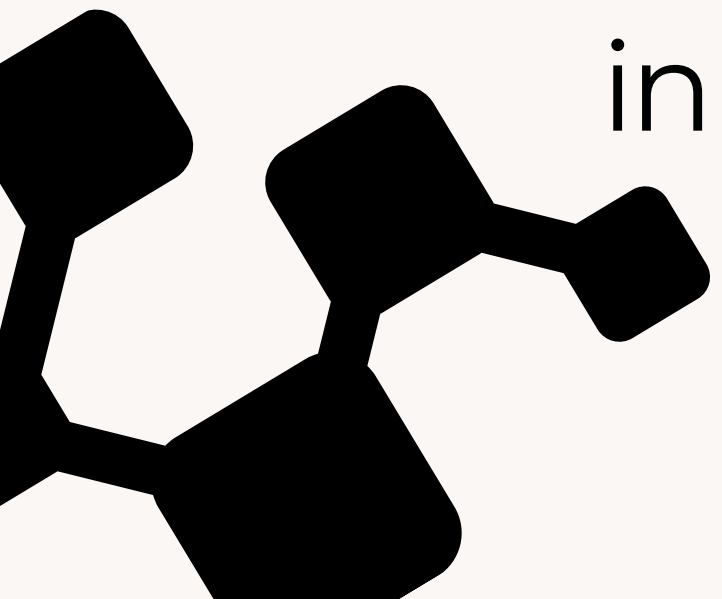
- FrozenLake has the following characteristics:
 - Number of actions: 4 (up, down, left, right)
 - Stochastic transitions (slippery surface)
 - Reward of 1 only when reaching the goal
 - Train case 15000, Test case 1000



WHY MODEL-BASED REINFORCEMENT LEARNING?



1. Transition probabilities and rewards can be learned explicitly
2. Value Iteration can be performed using the learned model
3. Stable and near-optimal policies can be obtained in small MDPs



OOP DESIGN (UML)

- **Main Class: ModelBasedAgent**

- Central class that manages the entire learning process
- Encapsulates data and behaviors related to learning and planning

- **Key Attributes**

- q_table: Q-value table
- transition_counts: State transition counts
- model_prob: Transition probabilities
- model_rewards: Average rewards
- V: State value function

OOP DESIGN (UML)

```
class ModelBasedAgent:
    def __init__(self, n_states, n_actions, gamma=0.99, epsilon=1.0, epsilon_decay=0.00015, min_epsilon=0.0):
        self.n_states = n_states
        self.n_actions = n_actions
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_decay = epsilon_decay
        self.min_epsilon = min_epsilon

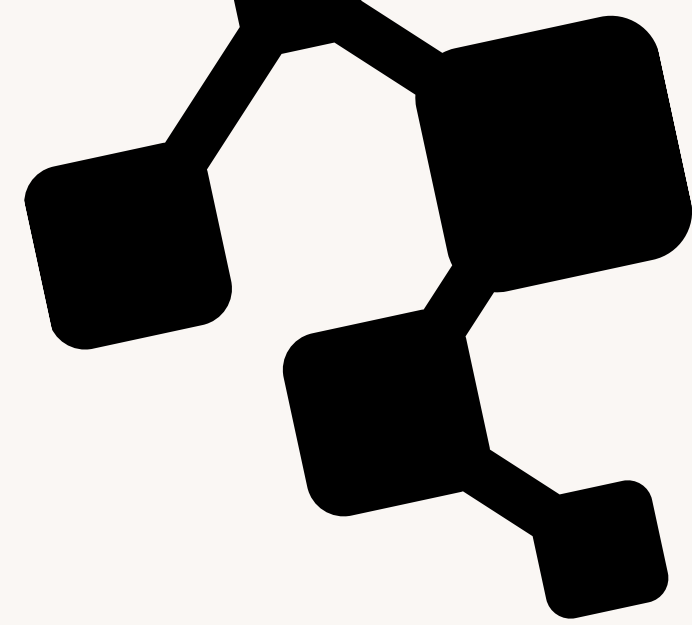
        # Q-table for action values
        self.q_table = np.zeros((n_states, n_actions))
        # Model-based components
        self.transition_counts = np.zeros((n_states, n_actions, n_states))
        self.reward_sums = np.zeros((n_states, n_actions, n_states))
        self.model_prob = np.zeros((n_states, n_actions, n_states))
        self.model_rewards = np.zeros((n_states, n_actions, n_states))
        self.V = np.zeros(n_states)
```

OOP DESIGN (UML)

Result

By using OOP, the reinforcement learning agent's logic (both data and processes) is organized into a self-contained module called `ModelBasedAgent`. This improves code reusability (the same agent can be used in the `evaluate_agent` function), maintainability, and overall understandability

LEARNING FLOW

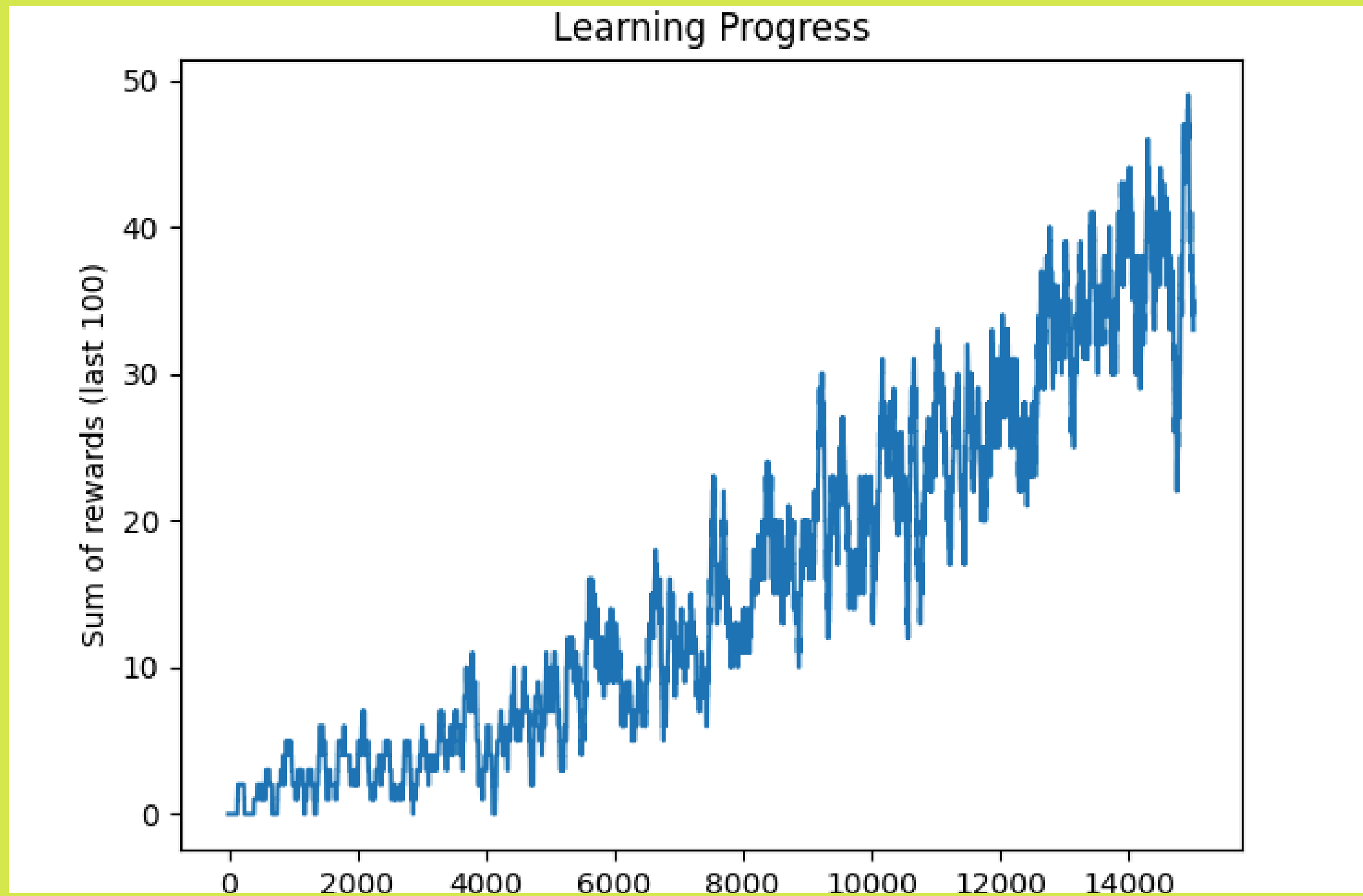


Learning Process

1. Select an action using ϵ -greedy strategy
2. Observe next state and reward from the environment
3. Update Q-values using Q-learning
4. Store transition counts and rewards in the model
5. Perform Value Iteration after sufficient experience



LEARNING PROGRESS



CODE REVIEW



ENVIRONMENT MODEL

```
self.transition_counts[state, action, next_state] += 1  
self.model_prob[state, action, :] = counts / total
```

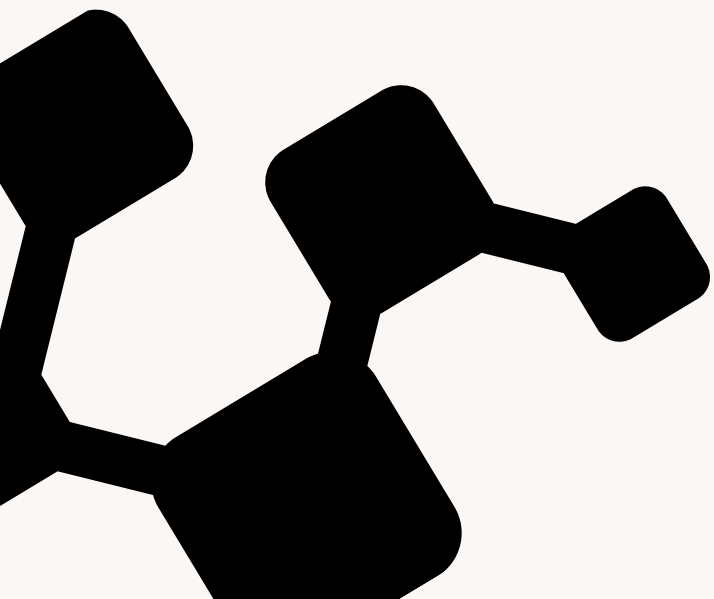
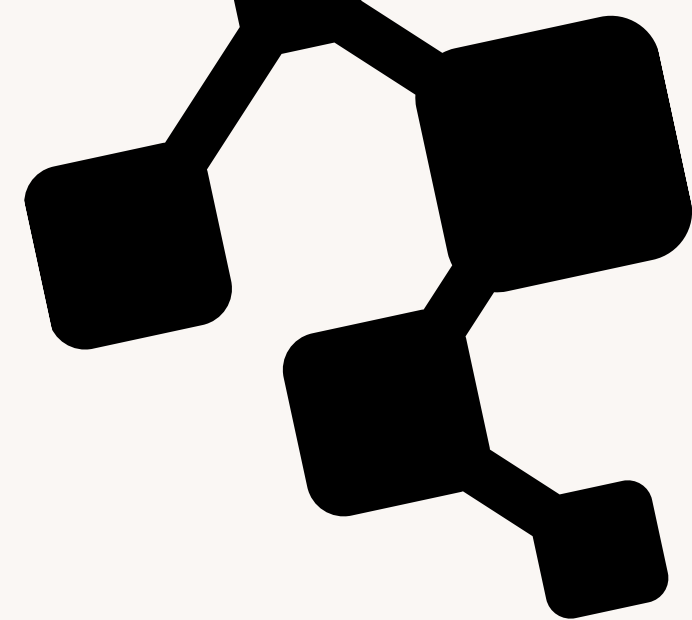
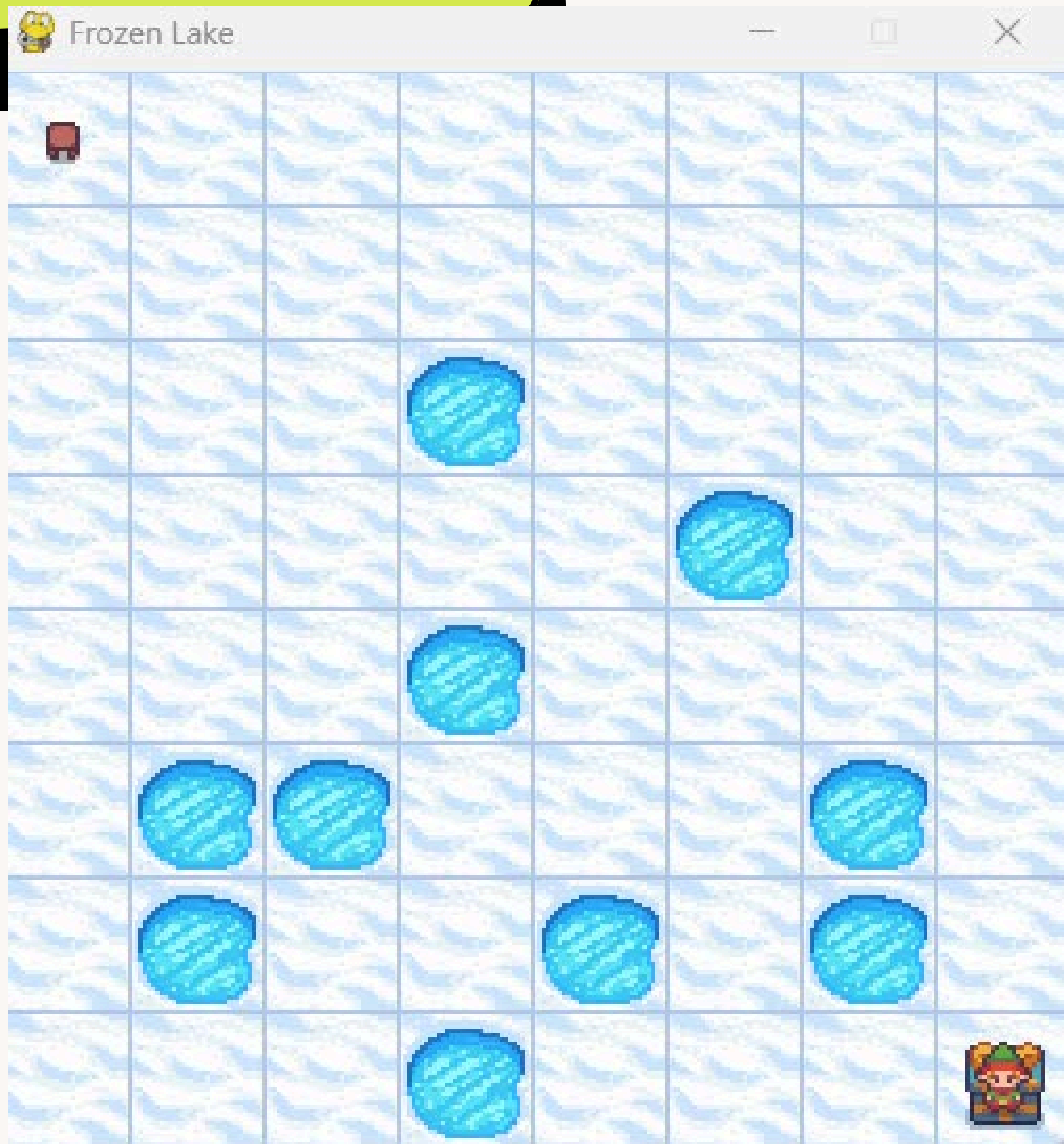
→ The agent counts state transitions and calculates transition probabilities $P(s' | s, a)$.

VALUE ITERATION

$$Q[s, a] = \sum P(s' | s, a) * (R + \gamma V(s'))$$

→ The Bellman expectation equation is directly implemented.

RESULTS & DEMO



RESULTS & DEMO



Success Rate: 16.37% (2455 / 15000 episodes)
Evaluation Success Rate: 76.30% (763 / 1000 episodes)

Success Rate: 16.19% (2428 / 15000 episodes)
Evaluation Success Rate: 74.90% (749 / 1000 episodes)

Success Rate: 15.87% (2380 / 15000 episodes)
Evaluation Success Rate: 75.80% (758 / 1000 episodes)

Success Rate: 16.11% (2416 / 15000 episodes)
Evaluation Success Rate: 75.80% (758 / 1000 episodes)

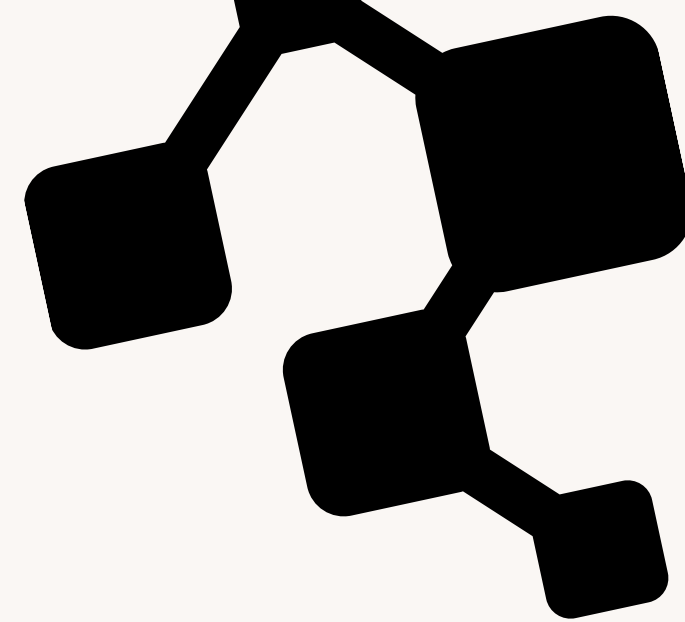
Success Rate: 15.85% (2378 / 15000 episodes)
Evaluation Success Rate: 75.00% (750 / 1000 episodes)



PART 3: CHOOSE YOUR OWN ADVENTURE

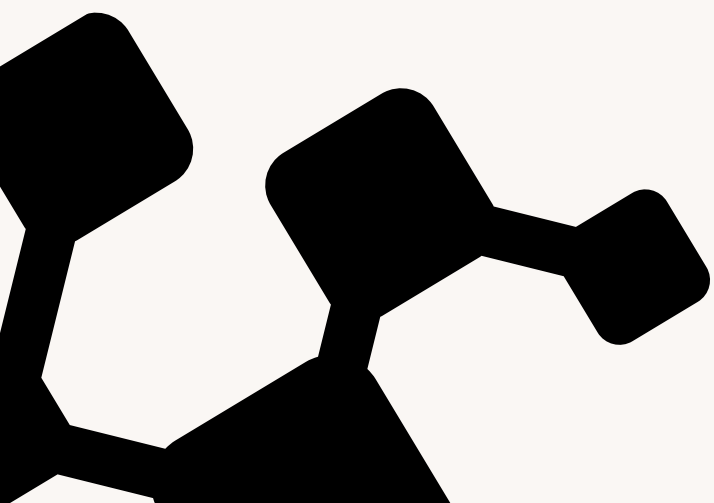
WAREHOUSE ROBOT

OVERALL SYSTEM OVERVIEW

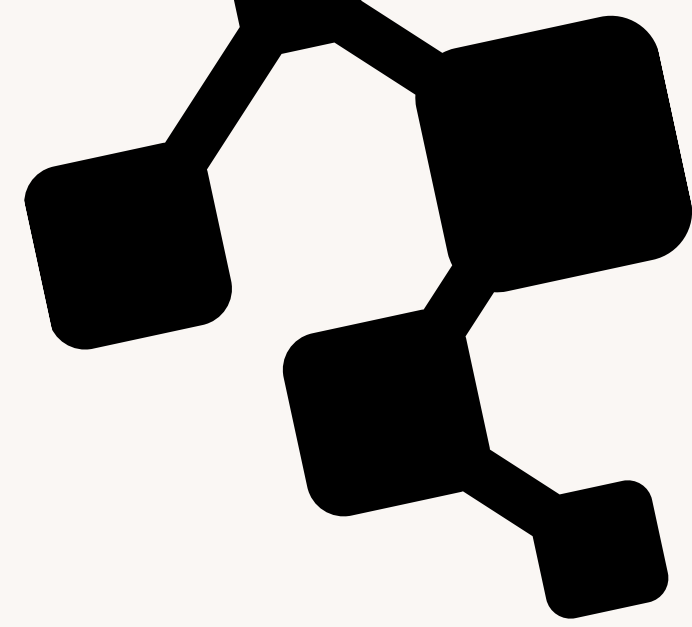


Clear Role Separation

- Environment (**warehouse_env.py**): Defines the physical rules of the warehouse and interactions between objects.
- Agent (**agent.py**): Defines the learning logic (Q-Learning) for navigating and deciding actions within the warehouse.
- Main Logic (**train_final.py, demo.py**): Connects the environment and the agent, executing training and demonstrations.



CORE FILES AND OOP PRINCIPLES

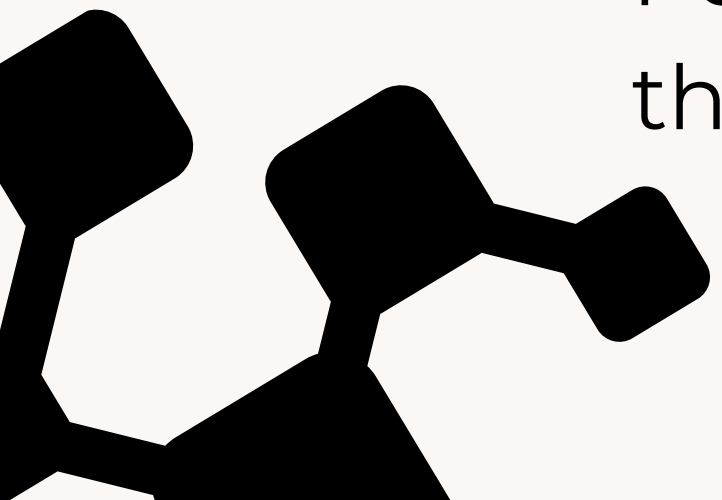


1. **agent.py** (Learning)

- Abstraction / Inheritance: Defines the Agent class as a base class.
- Encapsulation: QLearningAgent internally maintains the Q-table.

2. **warehouse_env.py** (World)

- Abstraction / Inheritance: Defines the GameObject class.
- Polymorphism: Wall and Package exhibit different behaviors through the same interact() method.



ABSTRACTION



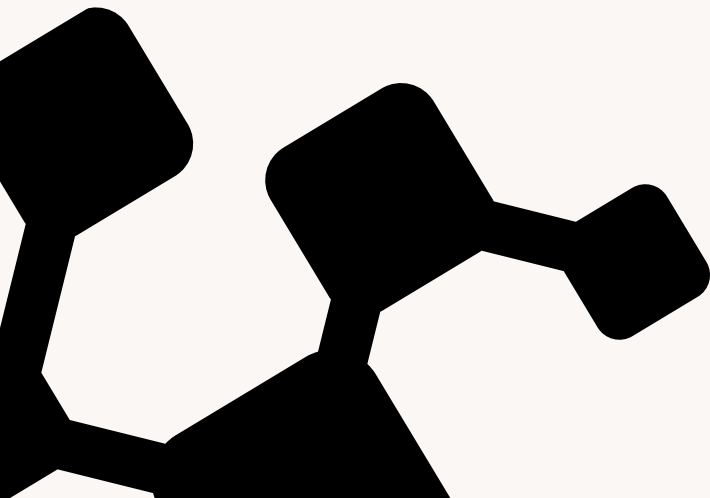
Agent abstract base class (agent.py)

```
class Agent(ABC):
    @abstractmethod
    def select_action(self, observation, training=True):
        pass

    @abstractmethod
    def learn(self, state, action, reward, next_state, done):
        pass
```

GameObject abstract base class (warehouse_env.py)

```
class GameObject(ABC):
    @abstractmethod
    def interact(self, agent):
        pass
```



INHERITANCE

GameObject

- |— Floor
- |— Wall
- |— Package
- |— Robot

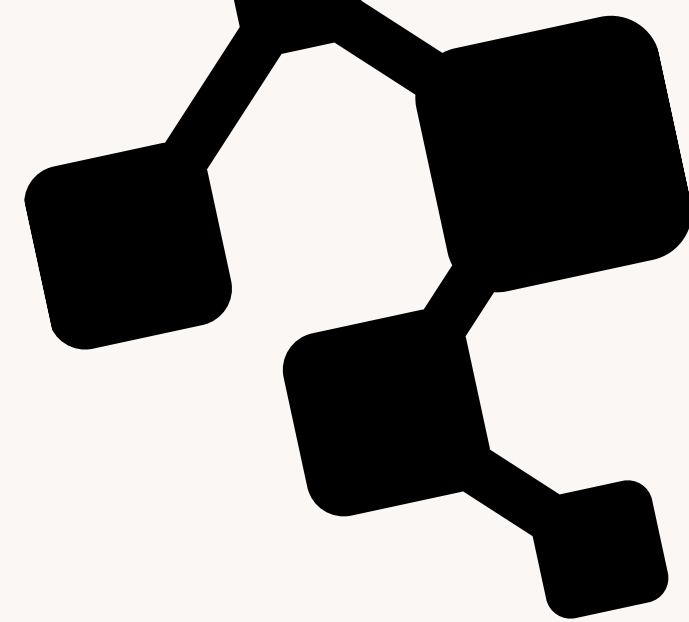
Agent

- |— RandomAgent
- |— QLearningAgent

```
class Floor(GameObject):  
    def interact(self, agent):  
        return True, -0.01, False
```

```
class QLearningAgent(Agent):  
    def select_action(self, observation, training=True):
```

POLYMORPHISM



Polymorphism in interact()

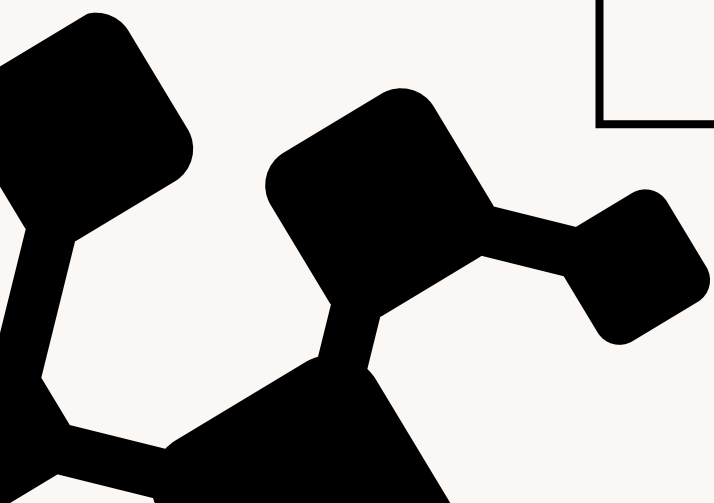
```
can_move, reward, done = obj.interact(agent)
```

Behavior per object

class	Return	Mean
Floor	(True, -0.01, False)	Movement OK/Small penalty
Wall	(False, -0.5, False)	Unable to move/big penalty
Packege	(True, 100.0, True)	Goal/end

No type checking of object!

```
obj.interact(agent)
```



ENCAPSULATION



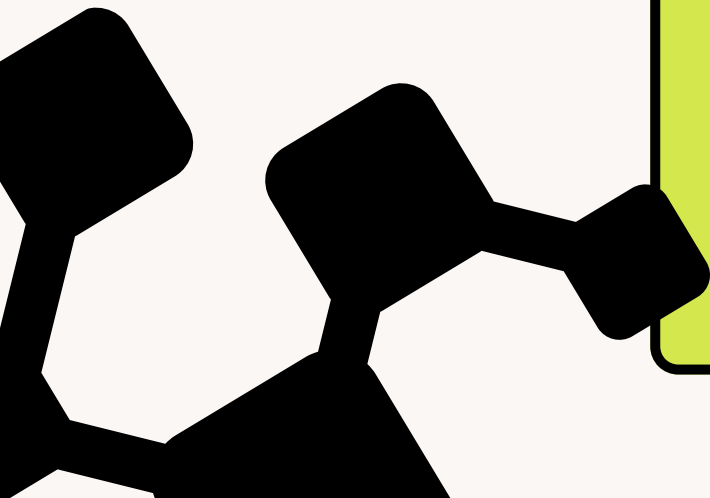
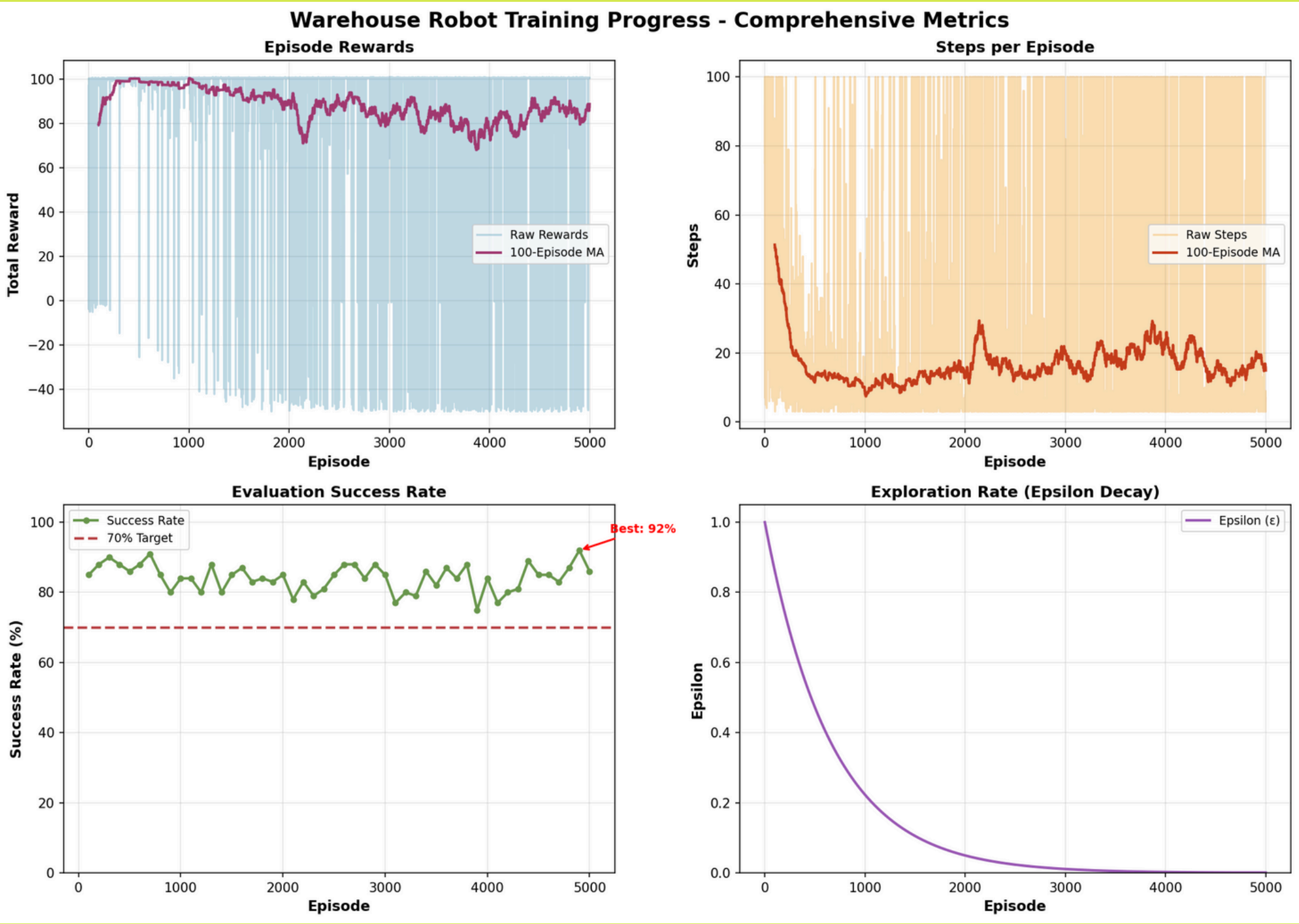
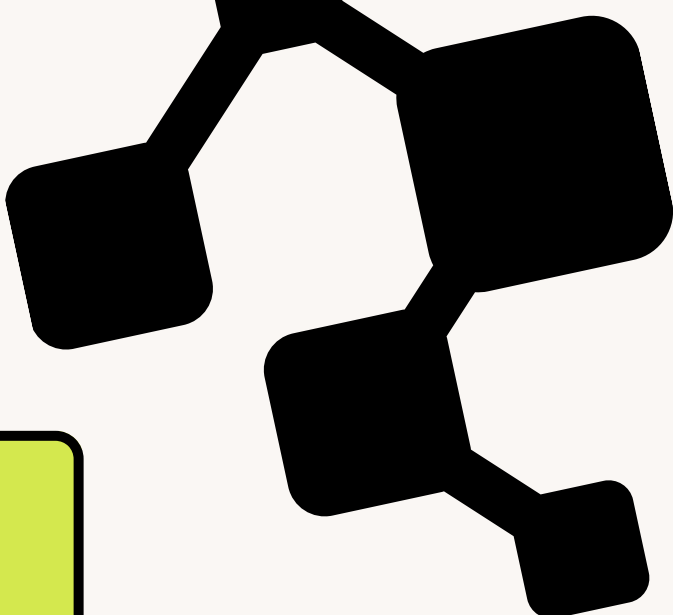
INTERNAL DATA (PROTECTED STATE)

- Q_TABLE
- EPSILON
- LEARNING_RATE
- DISCOUNT_FACTOR

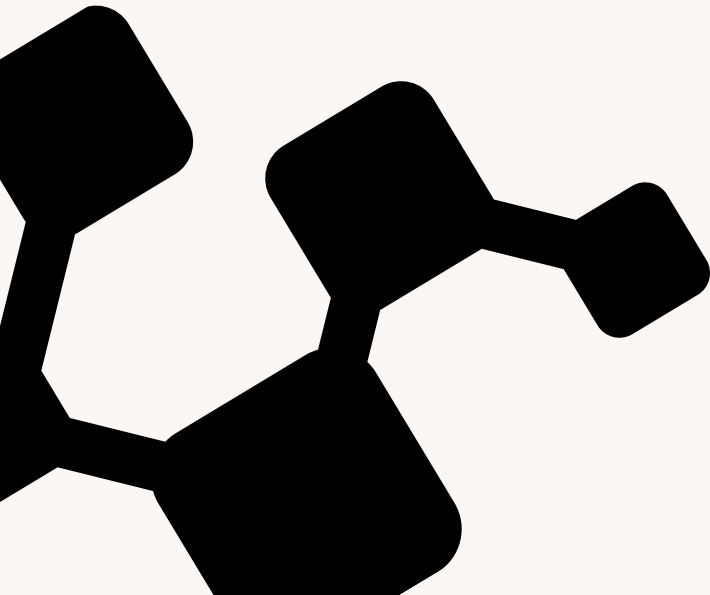
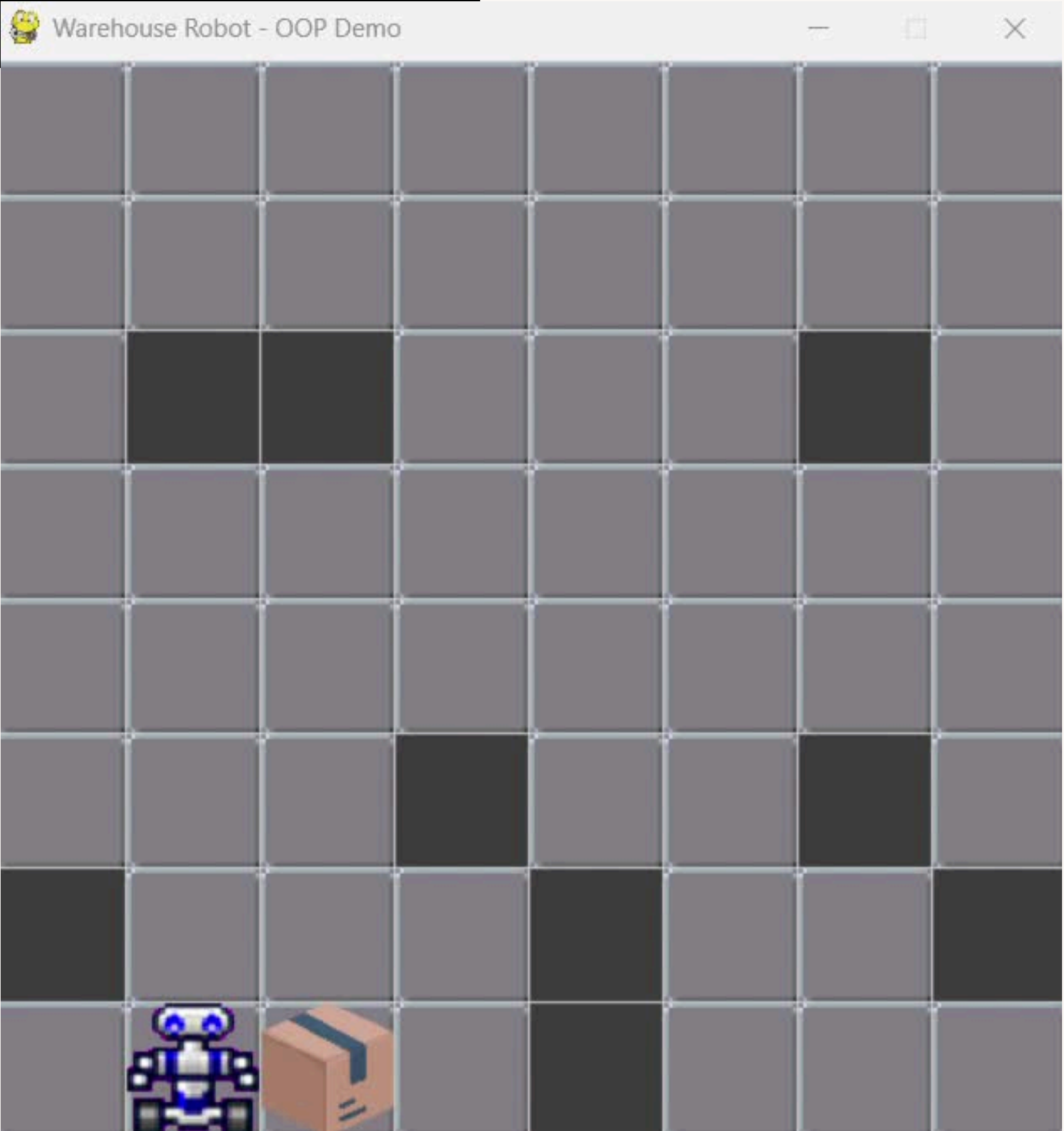
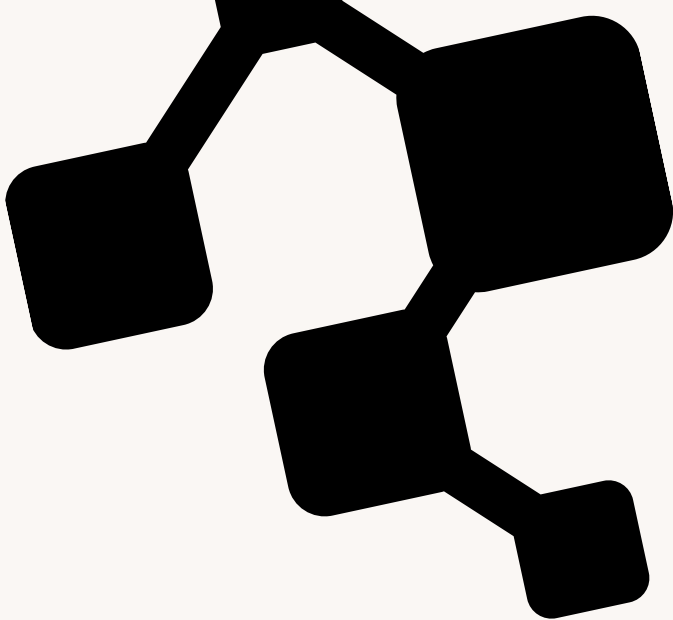
PUBLIC API (EXTERNAL INTERFACE)

- AGENT.SELECT_ACTION(OBSERVATION, TRAINING=TRUE)
 - AGENT.LEARN(STATE, ACTION, REWARD, NEXT_STATE, DONE)
 - AGENT.SAVE(FILEPATH)
 - AGENT.LOAD(FILEPATH)
- 

LEARNING PROGRESS



RESULTS & DEMO



RESULTS & DEMO

Episode 1/5

Starting position: (5, 5)

Package position: (6, 0)

Initial distance: 6

✓ SUCCESS! Reached package in 35 steps

Total reward: 100.26

Episode 3/5

Starting position: (5, 0)

Package position: (4, 2)

Initial distance: 3

✓ SUCCESS! Reached package in 18 steps

Total reward: 100.13

Episode 5/5

Starting position: (7, 2)

Package position: (5, 5)

Initial distance: 5

✓ SUCCESS! Reached package in 42 steps

Total reward: 98.13

Episode 2/5

Starting position: (5, 4)

Package position: (1, 4)

Initial distance: 4

X FAILED - Timeout after 200 steps

Total reward: -2.00

Episode 4/5

Starting position: (5, 7)

Package position: (0, 3)

Initial distance: 9

X FAILED - Timeout after 200 steps

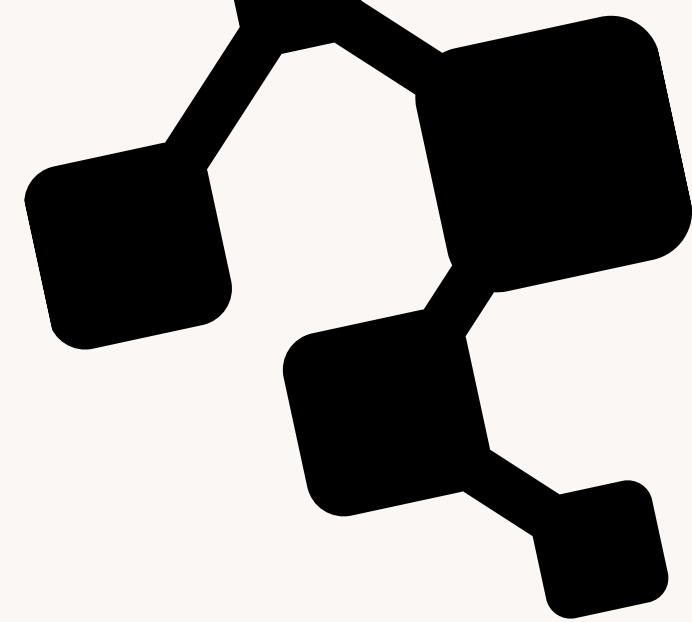
Total reward: -50.40

Success Rate: 3/5 (60.0%)

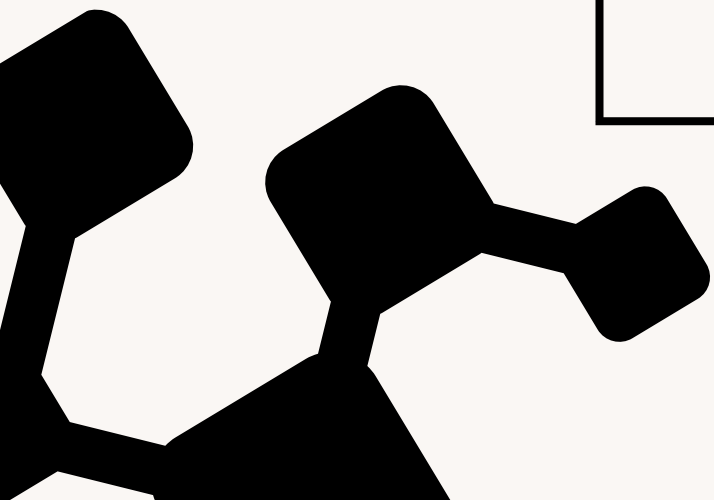
Average Reward: 49.22

Average Steps: 99.0

CONTRIBUTION TABLE



member	work
李佩萱	part2/ppt/report
松尾春輝	part2&part3 main code /ppt
湯昆璋	part2/part3



THANK YOU

