Benha University

Faculty of Computer Science & Artificial Intelligence

Treasure Hunt Game Using Reinforcement Learning

Team Members

Ahmed El-Sayed Ahmed

Mennatu-allah Nabil Ali

Ahmed Mahmoud Ismael

Mariam Yasser Hassan

Nosayba Rafet Abdel ghany

Supervised By :-

Eng.Mariam Mahmoud

Date: April 2025

1. Introduction

In this project, we have developed a custom gym environment to simulate a treasure hunt game. The environment is grid-based, and an agent navigates through it, aiming to collect treasures while avoiding traps and enemies. The agent is trained using two different reinforcement learning algorithms—Policy Gradient and Deep Q-Learning Network (DQN). The objective is to compare the performance and effectiveness of both algorithms in training the agent to achieve its goal efficiently.

2. Project Overview

-The Treasure Hunt environment is designed as a grid where the agent must perform various actions to achieve the following objectives:

- Collect treasures scattered across the grid.
- Avoid traps and enemies that can reduce its score or end the episode.

-The environment includes various challenges and conditions:

- Multiple penalties for mistakes like stepping into traps, getting captured, staying still, or moving in loops.
- Multiple rewards for collecting treasures and smart movement.
- Special bonus rewards for surviving under certain conditions.

-This dynamic setup helps the agent learn through a balance of risk, strategy, and reward.

-To train the agent effectively, we used **Deep Q-Network (DQN)** and **Policy Gradient** methods. These reinforcement learning techniques help the model learn optimal strategies through interaction with the environment.

---

3. Objectives of the Project

The primary objectives of the project are:

- To create a grid-based environment simulating a treasure hunt game.
- To apply reinforcement learning algorithms (Policy Gradient and DQN) to train an agent.
- To compare the results of both algorithms in terms of performance, training speed, and efficiency.
- To provide a visual interface using PyQt5 to display the agent's progress and actions in real-time.

---

4. System Architecture

The system is divided into the following components:

1. **Custom Gym Environment:** The grid-based environment where the agent navigates to collect treasures while avoiding traps and enemies.
2. **Reinforcement Learning Algorithms:**
   - **Policy Gradient**: A method for learning policies through gradient ascent to maximize expected rewards.
   - **DQN (Deep Q-Learning Network)**: A model-free, off-policy algorithm that uses deep learning to approximate Q-values for action selection.
3. **PyQt5 Interface**: A graphical user interface to visualize the agent's learning process and interactions within the grid.
4. **Training Pipeline:** The training loop for both algorithms, where the agent interacts with the environment, learns, and updates its policy or Q-values.

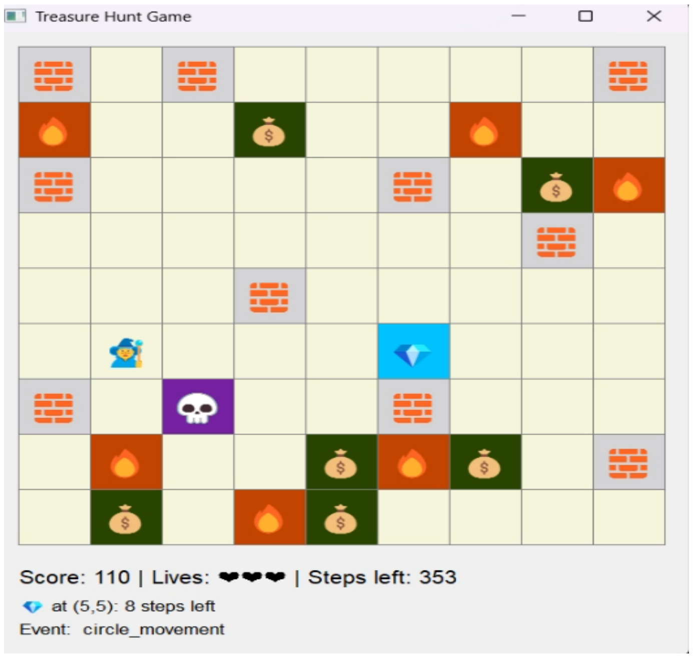
---

5. Custom Gym Environment

Environment Setup

The Treasure Hunt environment is designed as a grid with the following features:

- **Grid Dimensions:** The grid has a size of N×NN \times N cells, where each cell can contain a treasure, trap, or enemy, or be empty.
- **Agent Movement:** The agent can move up, down, left, or right, as well as pick up treasures when near them.
- **Treasure:** The agent earns positive rewards when it picks up treasures.
- **Traps:** If the agent moves into a trap, it receives a negative reward.
- **Enemies:** The agent loses points or may get captured if it encounters an enemy.
- **Empty**: Represents free space. The agent can move freely through this cell.
- **Obstacle**: Blocks the agent's movement. The agent cannot move through this cell.
- **Special Treasure**: A rare and high-value item. Grants significantly more reward than regular treasures. These are usually placed in hard-to-reach or risky areas, encouraging strategic decision-making.



Score: 110 | Lives: ❤❤❤ | Steps left: 353

💎 at (5,5): 8 steps left

Event: circle_movement

Action Space

The agent can take five actions:

- **Move Up**
- **Move Down**
- **Move Left**
- **Move Right**

State Space

The agent receives information about its current position and the surrounding environment. This could include:

- The positions of treasures, traps, and enemies.
- The agent's current location and its previous actions.

Reward System

Positive Rewards:

- +10 for collecting a treasure.
- +5 special bonus if the agent survives 10 steps without falling into a trap or getting caught.
- +50 for collecting all treasures.

Negative Rewards:

- -10 for stepping into a trap.
- -20 for getting caught by an enemy.
- -1 for staying in the same position.
- -2 for moving in loops or repeating positions.
- -5 penalty if the agent reaches the maximum number of steps (timeout) without finishing the task.

-This reward structure pushes the agent to explore efficiently, avoid danger, and stay active throughout the episode.

6. Reinforcement Learning Algorithms

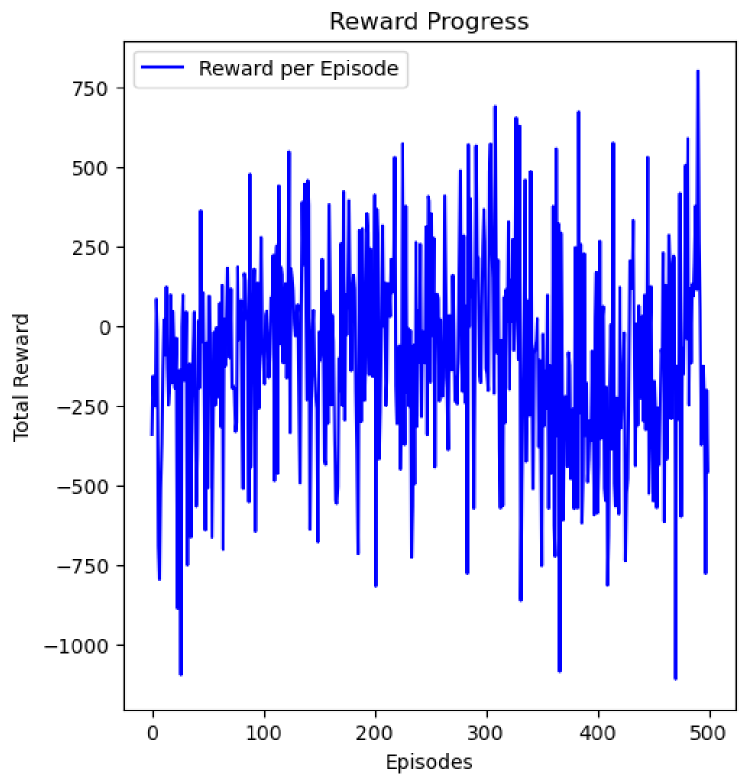
we implemented and compared several reinforcement learning (RL) algorithms to solve a custom environment. The algorithms tested were:

- Deep Q-Network (DQN)
- Vanilla Policy Gradient (VPG)
- Proximal Policy Optimization (PPO)

We started with built-in implementations and then developed our own versions of DQN from scratch . PPO was tuned extensively using its stable-baselines3 implementation.

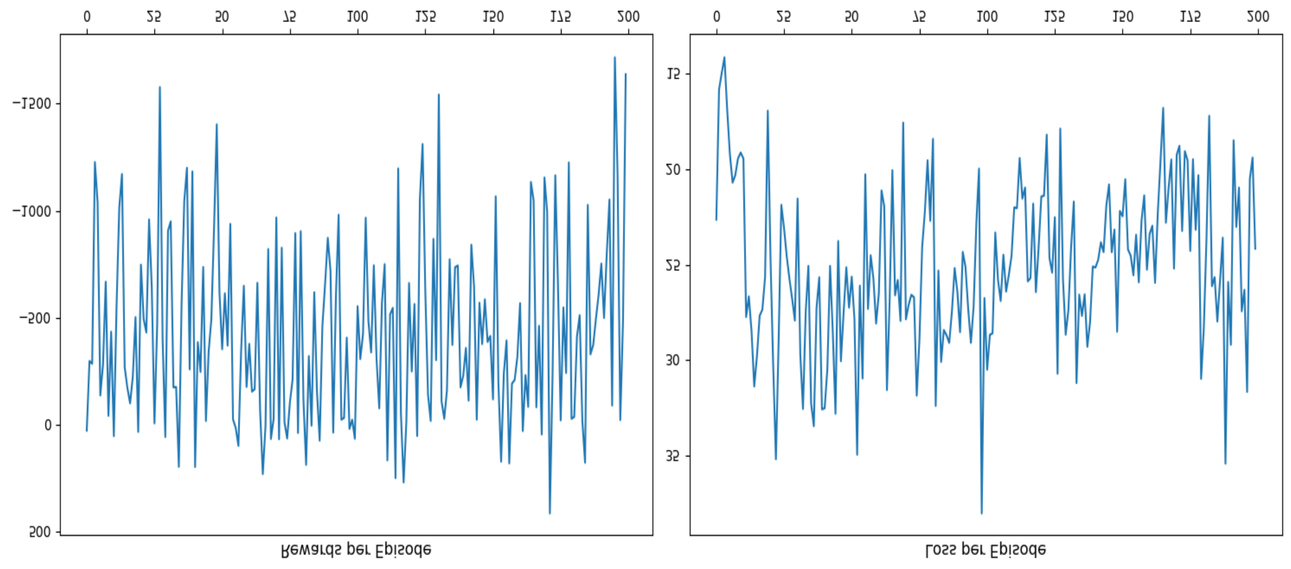
1. DQN (Deep Q-Network):

- **Built-in version:** The built-in version was implemented using a standard library that provides a pre-built structure for DQN.



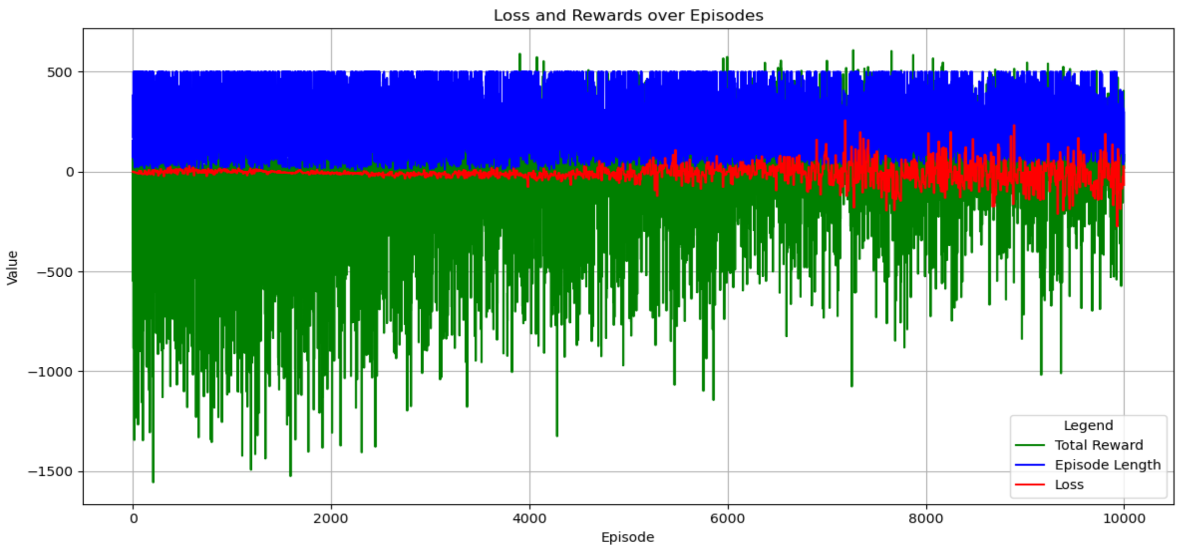
-

- **From scratch**: The core idea was to build a neural network that approximated Q-values from raw input data, and to store past experiences in a buffer to improve sample efficiency.



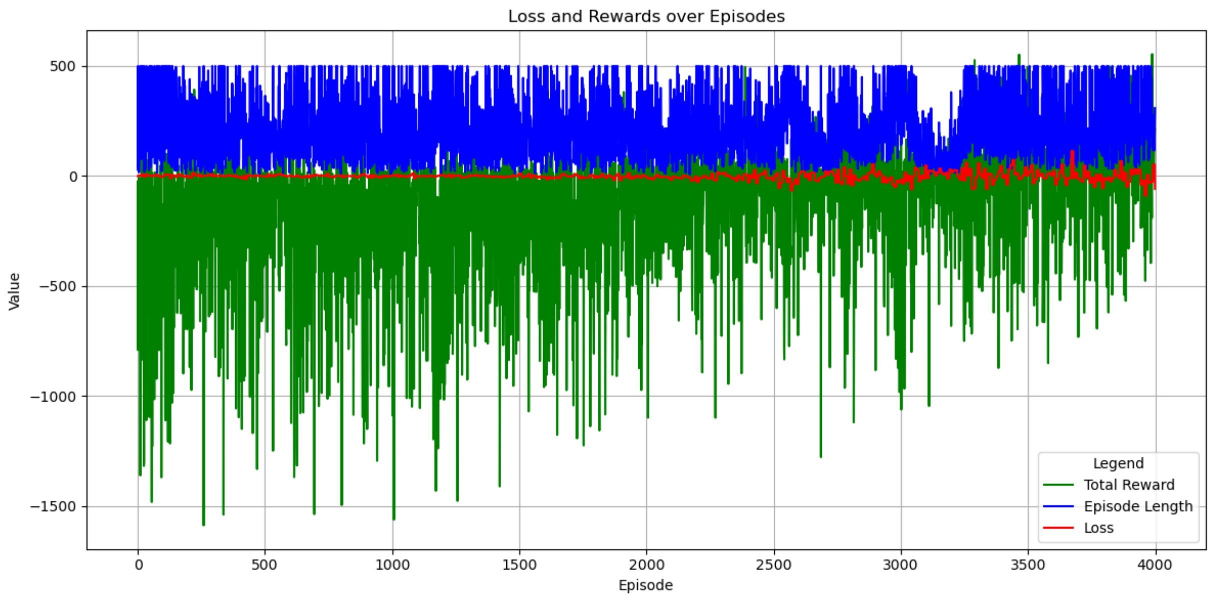
2. VPG (Vanilla Policy Gradient):

- **From scratch:** The custom implementation allowed full control over the policy network's architecture and gradient updates. This helped to optimize the agent’s learning process and stability.



3. VPG with Grid Search:

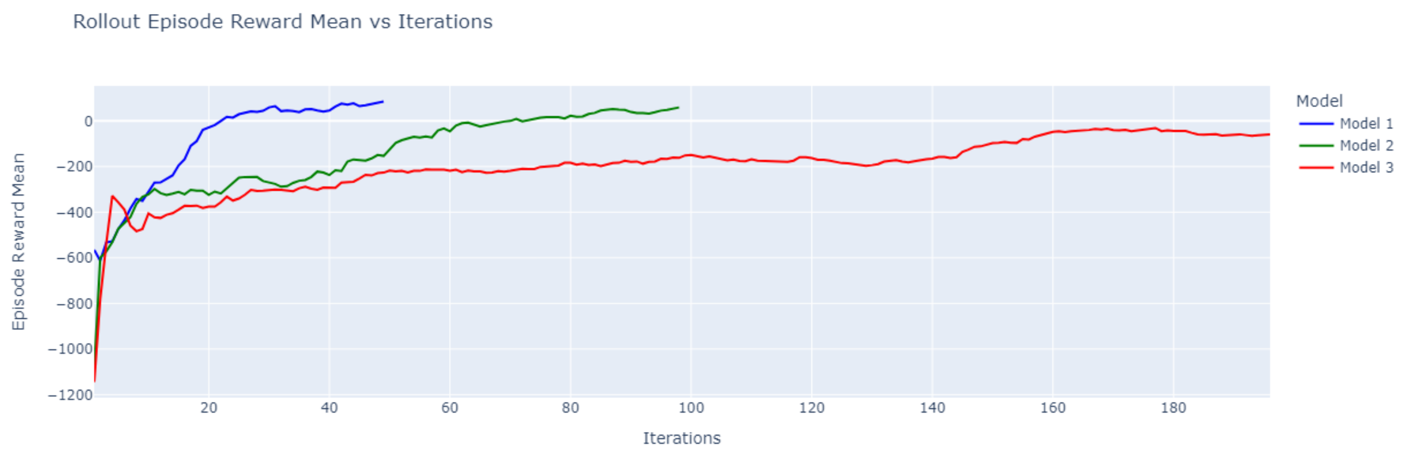
- The grid search was used to systematically evaluate different hyperparameter combinations and find the best settings for the policy gradient method. This method improves VPG’s performance by finding the most suitable hyperparameters for the task.



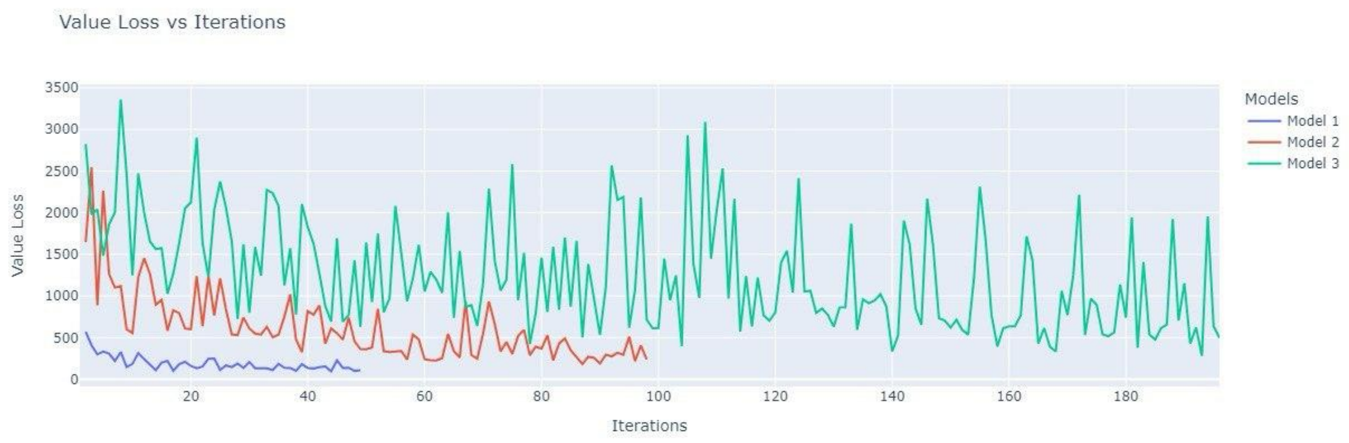
4. PPO (Proximal Policy Optimization):

- **Built-in version with tuning:** The built-in PPO version was tuned with hyperparameters, including learning rate, clip range, entropy coefficient, number of steps, and batch size. The tuning process aimed to enhance the algorithm's stability and convergence speed.

“Tunning reward”



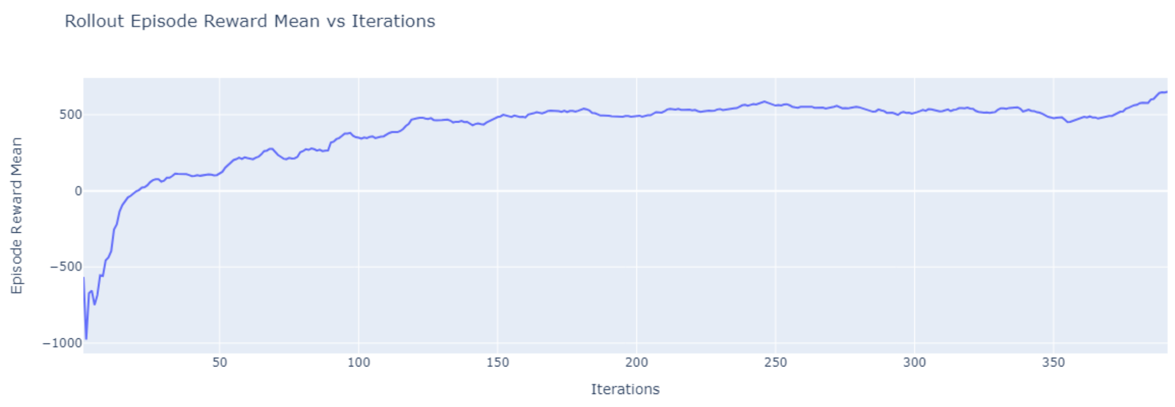
“Tunning loss”



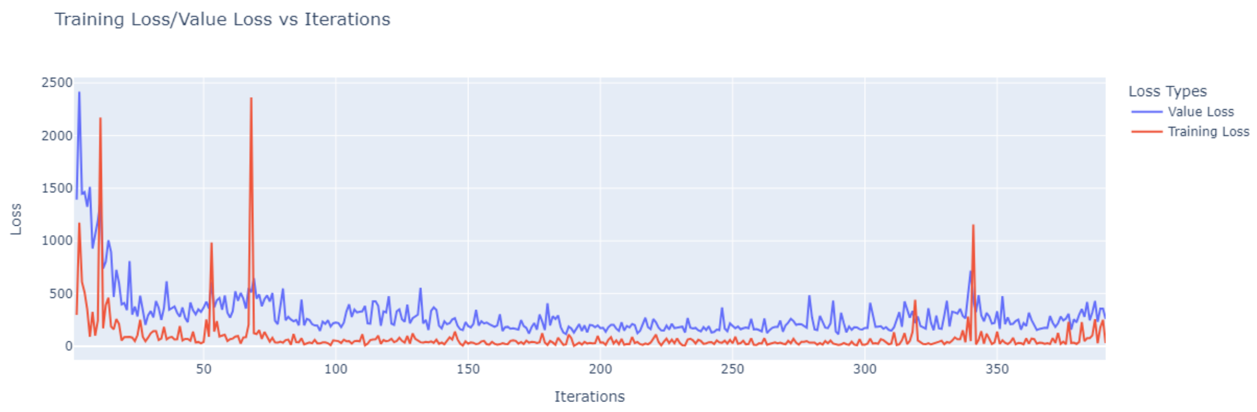
5. PPO (Standard version):

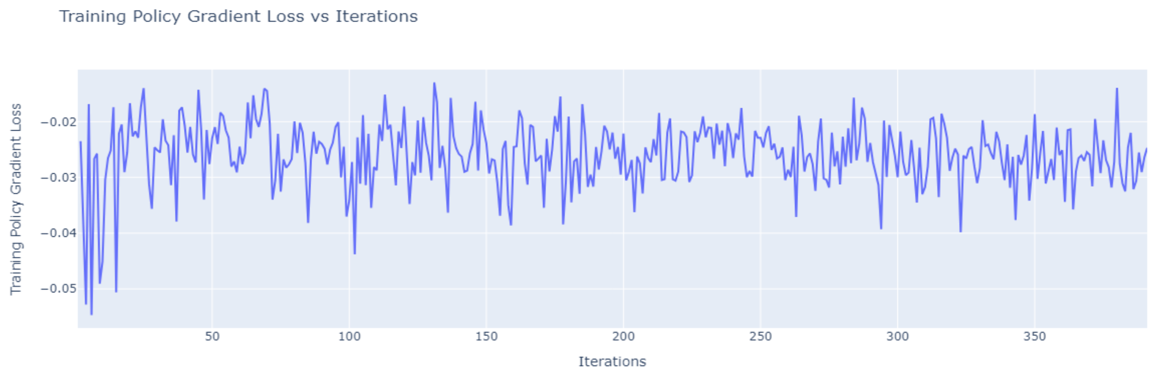
- The standard PPO implementation was used without extensive customization to the hyperparameters.

“Training reward”



“Training loss / Value loss”





6.5 Algorithm Comparison

We tested several RL algorithms: VPG, DQN (scratch and built-in), and PPO.

- VPG and DQN from scratch helped us understand the learning process.
- PPO was stable but needed more tuning.
- The built-in DQN gave the best results — fast learning, stable performance, and smarter agent behavior.

It was the most effective in training the agent to complete the task successfully.

7. Graphical User Interface (GUI)

The PyQt5 GUI is used to visualize the Treasure Hunt environment and track the agent’s progress. The GUI displays:

- The grid layout with treasures, traps, and enemies.
- The agent's current position and movements.
- Real-time updates of the agent’s actions and rewards.

---

8. Conclusion

This project successfully implements a custom gym environment for a treasure hunt game. The agent is trained using three reinforcement learning algorithms: Policy Gradient, DQN and PPO. After training the algorithms, a performance comparison is made based on efficiency, learning speed, and convergence. The project also includes a PyQt5 GUI for visualizing the environment and agent’s actions.
