OOP Group Project Reflection Report

B123040044 侯廷翰
B123040049 劉育希
B123245016 柯伯諺

I. Abstract

This project explores the intersection of **object-oriented programming (OOP)** and reinforced learning (RL) through the implementation of three distinct environments: Mountain Car, Frozen Lake, and a custom multi-agent Car Racing environment. The primary objective was to demonstrate how OOP principles, Encapsulation, Inheritance, Polymorphism, and Abstraction, can be leveraged to build scalable, maintainable, and complex software systems.

Project Repo: https://github.com/ImMasterSam/oop-finalproject-team11

II. Mountain Car (Part 1) and Frozen Lake (Part 2)

A. Problems Description

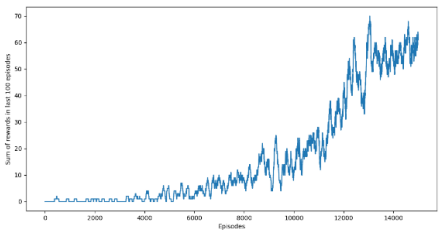
The Mountain Car environment is for us to test if the package is installed properly. For the second part, a Q-learning method is applied to the Frozen Lake environment. We will train a player to walk toward the target position on the frozen lake without falling into the ice hole.

B. Methods

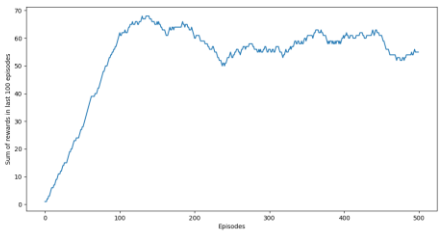
The training process has already been implemented in the given template. The way we can improve the result is to tune the hyperparameters of the training process, including *epsilon_decay_rate*, *min_epsilon*, and *learning_rate*. We applied **Optuna** [1] to quickly search for the optimal hyperparameters

C. Result

After 15000 episodes of training with the best tuned hyperparameters, the results can reach a success rate at **55% ~ 60%** (In 500 episodes).



◆ Training Process



◆ Run/Test Process

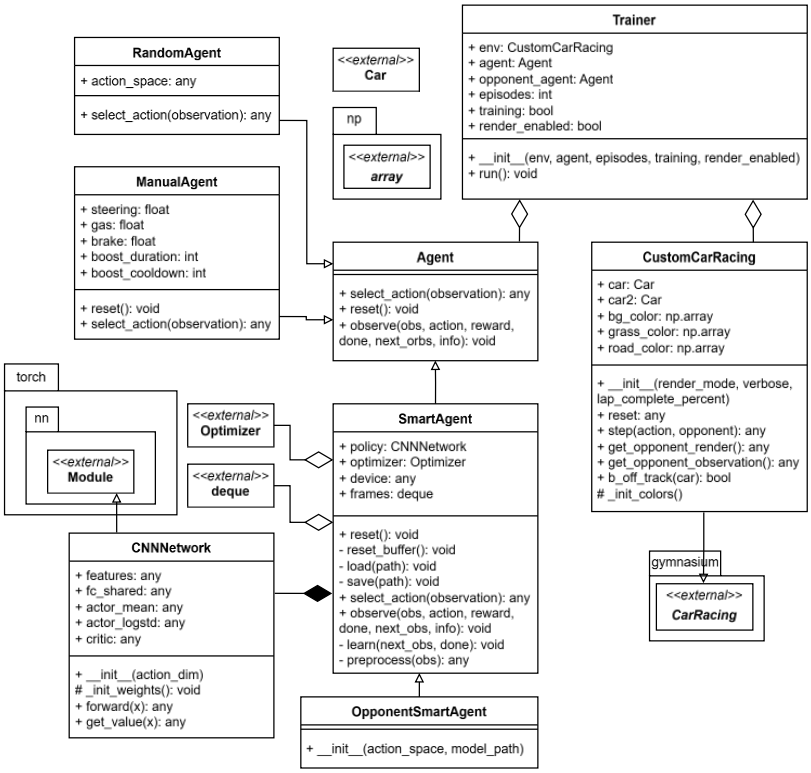
III. Car Racing Environment (Part 3)

A. Contents

In part3, we implemented a feature-rich racing environment that goes beyond the standard Gymnasium capabilities:

1) **Custom "Neon" Theme**: We overhauled the visual style of the track, implementing a dark-mode "Neon" theme with high-contrast colors to improve aesthetic appeal and visibility.
2) **Multi-Agent Racing**: Unlike the single-player functionality of the base library, our system supports two cars simultaneously. The environment handles physics for both the primary agent and an NPC opponent, enabling competitive racing scenarios.
3) **Advanced Agent Types**:
   - Smart Agent: A Reinforcement Learning agent governed by a PPO algorithm. It uses convolutional neural network (CNN) to perceive the track from raw pixel inputs, just like a human driver.
   - Random Agent: A baseline agent used for environmental testing and randomness benchmarks.
   - Manual Agent: Provides keyboard control for human players, featuring a custom Nitro Boost mechanic for strategic overtaking.

B. UML Graph



1) **Inheritance and Extension:** The solid arrow pointing to *CarRacing* represents an **IS-A** relationship. *CustomCarRacing* inherits all the core physics capabilities and rendering systems from *Gymnasium* library.
   - **Override:** We override reset() to initialize the second car and render() to apply the custom "Neon" color palette(_init_colors), demonstrating how inheritance allows for extending functionality without modifying the original source code.
2) **Abstraction Interface:** The *Agent* class is defined as <<Abstract>>, serving as strict blueprint.
   - **Contract:** It enforces that every subclass must implement the select_action(obs) method. This corresponds to the **Dependency Inversion Principle** --- high-level modules depending on key abstractions (*Agent*), not concrete implementations.

3) **Polymorphism in Action:** *ManualAgent*, *RandomAgent*, and *SmartAgent* are all distinct implementations of *Agent*.
4) **Composition and Association:** The line ending with a diamond indicates a **HAS-A** relationship.
   - **Decoupling:** *CustomCarRacing* maintains a reference to a second *Agent* instance (car2). This separates the environment from the intelligence (driving logic). The car objects exist within the race, but its decision-making logic is injected from the outside, following the **Strategy Pattern**.
5) **Encapsulation:** In the *SmartAgent* class, the minus signs (-) before policy and memory denote private attributes.
   - **Information Hiding:** The complex PPO neural network and replay buffer are hidden from outside world. The environment simply feeds observations to the public select_action method and receives controls back, knowing nothing about the underlying deep learning machinery.

IV. Conclusion

A. Used OOP Concepts

Throughout the project, we've used almost all the OOP concepts that we've mentioned in this semester, including abstraction, inheritance, encapsulation, polymorphism, and composition or aggregation in our project.

B. What have we learned?

To implement this project, we must learn how the OpenAI Gymnasium environment works. This is the first time we use the OOP concepts to complete a fine project. Besides the implementations of the project, we also need to use GitHub to manage the source code. GitHub allows us to work on the same project simultaneously on different branches. In total, we have 26 commits and 2 pull requests on our repository. It was a great help in finishing this project.

C. AI Tools

We've asked some AI tools to design our class structure. Like whether some part of the code should be encapsulated or the code should be shared between these objects by using an abstract base class or not. AI did a really good job on designing but the code implementation is quite poor.

V. References

[1] Optuna
https://optuna.org/

[2] Gym Documentation
https://gymnasium.farama.org/

[3] RL algorithm PPO
https://hackmd.io/@YungHuiHsu/SkUb3aBX6

[4] CNN convolution
https://brohrer.mcknote.com/zh-Hant/how_machine_learning_works/how_convolutional_neural_networks_work.html