School of Computer Science

CT 413 – Final Year Project
Formal Report

Reinforcement Learning using queries

Author:

Abdul-Hameed Adagun

Supervisor:

Jamal Nasir

Acknowledgement

As I wrap up this final year project, I find myself overwhelmed with gratitude for the support and encouragement I've received along the way. I would like to thank Jamal, my supervisor, for all the help he has provided me with throughout this project. Without him I would have struggled greatly. His experience and expertise have guided me in the right direction regarding my project.

I also want to extend my thanks to the staff at the University of Galway, whose dedication to education and mentorship has laid the foundation for my academic growth.

I would also like to thank my colleagues and managers that I had while working at Ericsson during my placement. I will use the skills that I learned there for the rest of my life. These skills helped me complete this project to a high standard.

Lastly, I am grateful to my friends and family for their unwavering support, understanding, and encouragement throughout this challenging yet rewarding journey.

Glossary

FYP	Final Year Project
AFK	Asking for Knowledge
UI	User Interface
RL	Reinforcement Learning
IDE	Integrated development environment
AI	Artificial Intelligence
ML	Machine Learning
PPO	Proximal Policy Optimization
A2C	Advantage Actor-Critic

Table of figures

Table of Contents

Chapter 1. Introduction

1.1 Project Overview

In this project, I introduce a novel approach to reinforcement learning (RL) that leverages querying as a means of learning. Traditional RL algorithms rely on trial and error to discover optimal strategies, often requiring extensive exploration of the environment. My approach seeks to expedite the learning process by incorporating querying, allowing the agent to actively seek information from its environment.

1.2 Project Objectives

I developed this project using python. Using the already existing q-baby ai as a reference I hoped to be able to improve on it. Q-baby AI [7] can be used to solve four level one tasks, object in box, danger, go to favorite and open door. Both the object in box and the danger task can only be solved 100% of the time using queries. Opening the wrong objects or stepping on the danger tile terminates the game. The go to favorite and open door can both be solved using trial-and-error however it is more efficient using queries.

The project can be broken down into core and extra objectives. The core objectives being an integral part of the project and the extra objectives are not essential parts of the project.

1.2.1 Core Objectives

A core project goal would be improving the performance of the Q-baby AI. There are many ways I could do this. I will be experimenting with each of the following:

Memory Optimization - Continuously optimize the 'notebook' mechanism to handle larger volumes of information efficiently. This could involve exploring different memory structures or algorithms to improve the scalability and speed of information retrieval.

Enhanced Knowledge Querying - Make the AFK agent's query ability more natural. Explore techniques from natural language processing and information retrieval to improve the relevance and efficiency of the queries.

Complex Task Combinations: Introduce more intricate task combinations beyond level 4. Experiment with combining tasks in novel ways to create increasingly complex scenarios. This could involve creating new task types.

1.2.2 Extra Objectives

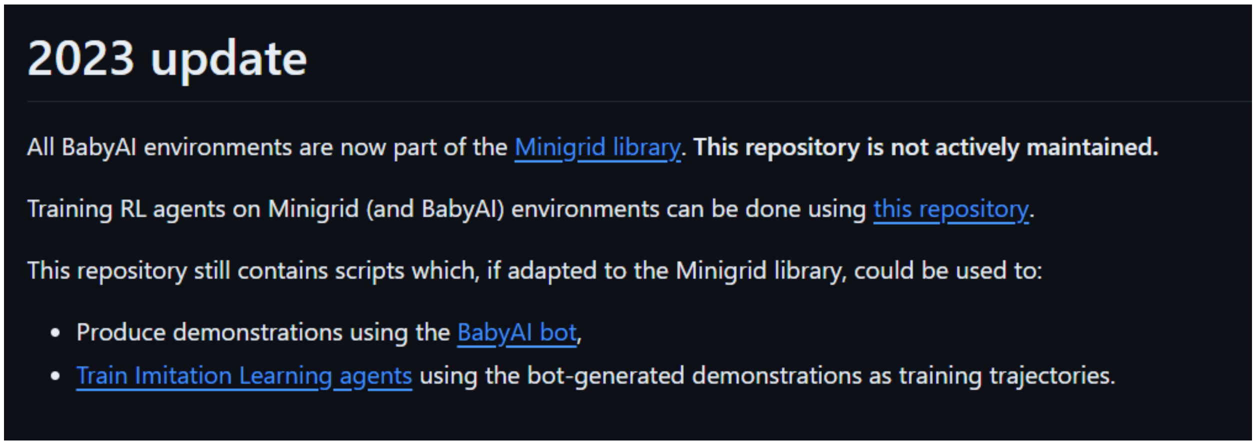
An extra project goal would be to design a user-friendly interface or interaction model that allows users to understand and interact with the AI. This could be beneficial for both researchers and potential end-users of the technology.

Another extra project goal would be establishing robust evaluation metrics and benchmarks to assess the AI's performance accurately. This could involve creating standardized tests or tasks to measure its progress.

1.2.3 Objectives not implemented

A lot of the objectives that I intended to implement could not be achieved. This was due to technological constraints. The baby ai repository was no longer being maintained meaning I could not use any of its environments. This can be seen in Figure 1 which can be found in the readme file of the BabyAI repository [6]. The q baby ai repository was coded using BabyAI.

The model that I had originally hoped to improve on (q-baby ai) was not accessible to me. The model kept throwing errors at me. After two weeks of trying hard to fix them, I decided to try a new repository and build my own model that queries the user. This new repository was called rl-starter-files. [5]



2023 update

All BabyAI environments are now part of the Minigrid library. **This repository is not actively maintained.**

Training RL agents on Minigrid (and BabyAI) environments can be done using this repository.

This repository still contains scripts which, if adapted to the Minigrid library, could be used to:

- Produce demonstrations using the BabyAI bot,
- Train Imitation Learning agents using the bot-generated demonstrations as training trajectories.

Figure 1 (BabyAI unavailable) [6]

A new core objective involved comparing two well-known algorithms with the new model that I built. These two algorithms will be PPO and a2c.

1.3 Project Deadlines

1.3.1 Project Tasks

Here are my projects tasks that I curated in November 2023. Due to me not being able to setup Q-baby ai my project tasks looked vastly different to what I had originally planned.

Task ID	Planned task	Number of weeks to complete
1	Project Setup	1
2	Framework Understanding and Analysis	2
3	Optimization and Testing Phases	2
4	Data Collection and Performance Enhancement	3
5	Reporting Preparation	1
6	Report Finalization	2

Week 1–2 (08/01/2024 - 21/01/2024): Project Setup

Week 1:

- Set up GitHub repository for version control.
- Finalize research on Q-baby AI framework and relevant literature.

Week 2:

- Setup Q-baby AI.

Week 3-4: Framework Understanding and Analysis

Week 3:

- Review the existing Q-baby AI implementation.
- Analyze the designed tasks and their interactions within the framework.
- Identify any initial performance issues or areas for potential improvement.

Week 4:

- Allocate specific areas of focus for performance enhancement based on the analysis done.

Week 5-6: Optimization and Testing Phases

Week 5:

- Initiate optimization efforts aimed at improving AI performance metrics.
- Implement targeted changes in the framework to enhance efficiency or accuracy.

Week 6:

- Conduct comprehensive testing to evaluate the impact of optimization strategies on the AI's performance.
- Collect data and metrics to measure the effectiveness of the implemented changes.

Week 7-9: Data Collection and Performance Enhancement

Week 7:

- Validate the optimized AI performance across different task combinations and levels.
- Gather comprehensive data on performance metrics, ensuring consistency and reliability.

Week 8:

- Perform thorough analysis on the collected data to evaluate the effectiveness of optimization efforts.
- Compare the results against previous performance benchmarks to gauge improvement levels.

Week 9:

- Conduct experiments to fine-tune the exploration strategy and memory mechanisms.

Week 10: Reporting Preparation

Week 10:

- Start drafting the project report, detailing the optimization strategies employed and their impact on AI performance.
- Begin structuring sections related to results and analysis.
- Continue writing and refining the project report, incorporating analysis of performance data and validation results.
- Develop visual aids or illustrations to present key findings effectively.

Week 11-12: Report Finalization

Week 11:

- Finalize the project report, ensuring coherence and completeness in all sections.
- Review the entire document for consistency, accuracy, and adherence to project objectives.

Week 12:

- Perform a final review of the report, addressing any remaining issues or adding concluding remarks.
- Prepare presentation materials.

1.3.2 Constraints

The main constraints that I encountered during the project's development were:

Time: I underestimated the amount of time other modules took. This led to me being under pressure to make sure that I had enough time each week for the project.

Scope: My initial scope changed early in the project's development due to unforeseen circumstances. This meant that I had to adapt and make quick decisions.

1.3.3 Deliverables

- Project Definition Document - 27th November 2023
- Final Project Report - 28th March 2024
- Project Demonstration - 2nd - April 2024
- Viva Voce - 9th April 2024

1.4 Report Structure

This report includes five chapters. The five chapters are the introduction, research, development, the application and results. This ensures that each section flows logically, providing a thorough understanding of the topic at hand. In this section I will show what each chapter entails.

Introduction

The introduction serves as the gateway to the report, setting the stage by presenting the problem statement, objectives, and scope of the study. Additionally, the introduction provides a brief overview of the subsequent chapters, allowing the reader to know what lies ahead.

Research

In the research chapter, we delve into the existing literature relevant to the study. This section offers a comprehensive review of prior research and areas warranting further exploration. By gathering diverse sources, we establish a foundation upon which our own investigation builds, demonstrating an understanding of the subject matter. Here, readers gain insight into the significance of the research topic, its relevance to the field, and the potential implications of the findings.

Development

The development chapter shifts focus to the methodology employed in conducting the research. Here, we outline the approach taken and the tools used. Within these pages, readers will find a comprehensive exploration of the development journey, from inception to implementation.

Application

In the application chapter, we apply insights to practical scenarios. This section showcases the relevance of our research in informing decision-making processes. Through simulations we demonstrate the potential applications of our work, bridging the gap between theory and practice.

Results

Finally, the results chapter presents the outcomes of our research endeavor, including findings, statistical analyses, and key insights derived from the data. Here, we interpret the results in relation to the research questions and objectives outlined in the introduction, offering interpretations, implications, and avenues for future inquiry. By presenting clear and concise summaries of our findings, we empower readers to draw their own conclusions and contribute to ongoing discourse in the field.

Chapter 2. Research

2.1 Research Origins

Before this project, I had little knowledge to no knowledge on RL. The most I knew about it was from Machine Learning and Artificial Intelligence lectures that I attended. To improve my knowledge, I read research papers and watched a Stanford RL course on YouTube.

2.2 Literature Review

Here I will review everything I have learned that has helped me to complete this project.

Definition of RL terms

- **Environment**: The environment represents the external system with which the agent interacts. It consists of a set of states, actions, transition dynamics, and rewards. At each time step, the agent perceives the current state of the environment and selects an action to execute.
- **Agent**: The agent is the decision-making entity that learns to interact with the environment. Its objective is to learn a policy—a mapping from states to actions—that maximizes its long-term cumulative reward. The agent learns from its interactions with the environment through trial and error.
- **State**: A state represents a particular configuration or situation of the environment at a given time step. It encapsulates all relevant information that the agent needs to make decisions, including observations, past actions, and any other contextual information.
- **Action**: An action is a decision taken by the agent to influence the state of the environment. The set of possible actions depends on the specific problem domain and can range from discrete choices to continuous control signals.
- **Reward**: A reward is a scalar feedback signal provided by the environment to the agent after each action. It indicates the immediate benefit or cost associated with the action taken in a particular state. The agent's goal is to maximize the cumulative reward it receives over time.

Reinforcement learning involves four main components which includes optimization, generalization, exploration and delayed consequences.

Optimization

Whenever creating an RL model, the goal is to find an optimal way to make the best decisions.

Generalization

Policies are used in RL models; they help make current decisions based on past actions. Pre-programming policies to include all actions depending on the environment may be computationally difficult. This is why generalization is important.

Exploration

RL model usually involves an agent and the environment. The agent learns about the environment through trial and error. We don’t know whether a decision was good or bad until we try it. Each decision has a reward associated with it. It is possible that we will never reach a certain outcome if the model never took the decisions the came with that outcome. The agent’s action affects the subsequent data that it receives.

Delayed Consequences

You can’t be sure of a decision that you are making now can’t have an impact later. This is why decisions should not only have immediate benefits but also long-term consequences.

RL vs ML

In an ML model there is no exploration or delayed consequence. In contrast to RL models there is no supervisor, only a reward signal. There is nothing in place to tell the RL model the right action to take. Instead, it uses trial and error. Also, when you get feedback, it is delayed. Compared to a ML model where the feedback is instantaneous.

RL vs AI

AI models differ from RL models in that AI models do not use exploration. Typically, in AI models a set of rules and how they affect the environment is known. The model may require search, but this is not exploration. You can think of exploration as a way of discovering the rules of the environment.

Sequential Decision Making

When making a decision the goal is to maximize the total expected rewards when balancing the immediate rewards and the long-term rewards.

States and rewards

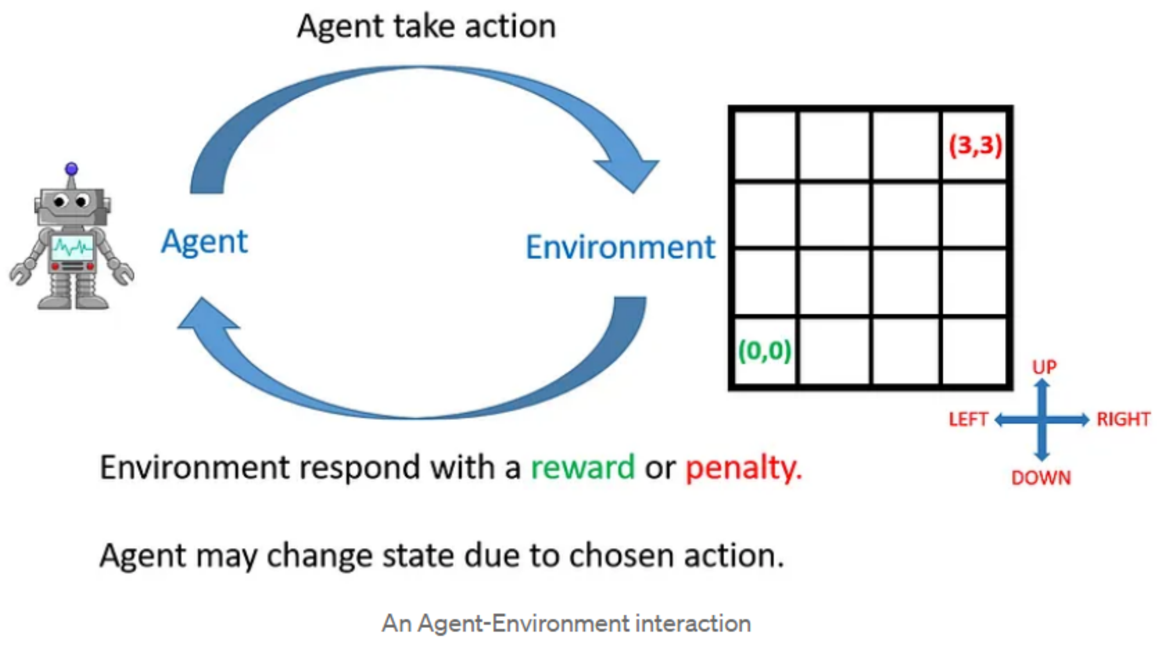

Agent take action
Agent
Environment
(3,3)
(0,0)
UP
LEFT
RIGHT
DOWN
Environment respond with a reward or penalty.
Agent may change state due to chosen action.
An Agent-Environment interaction

Figure 2 (Agent-Environment Interaction) [13]

Markov Assumption

Markov Assumption means that the current state is sufficient for the agent to make a decision. The previous states do not matter. "...given the present, the future does not depend on the past" [10]

Types of Sequential Decision Making

Deterministic

Given history and action there is a single observation and reward.

Stochastic

Given history and action there are many potential observations and rewards.

Policy

A policy is essentially a strategy or a set of rules that an agent uses to determine its actions in a given state. It’s a mapping from states to actions. The policy defines the behavior of the agent in its environment.

The goal in reinforcement learning is often to find the optimal policy, which is the one that maximizes the cumulative reward the agent receives over time. This can be achieved through various algorithms like Q-learning, SARSA, or policy gradient methods.

Choosing the right policy depends on the specific problem you're trying to solve and the nature of the environment the agent is interacting with. Sometimes, a deterministic policy might be more suitable, while in other cases, a stochastic policy might be better suited to capture uncertainty or explore different possibilities.

Value Function

A value function is a function that estimates the expected cumulative reward an agent can obtain from a given state (or state-action pair) onwards, following a particular policy. It essentially tells the agent how good it is to be in a particular state or to take a specific action in that state.

There are two main types of value functions:

1. **State-Value Function (V)**: This function, denoted as V(s), estimates the expected cumulative reward starting from a particular state under a given policy. In other words, it tells you how good it is to be in a certain state on average.
2. **Action-Value Function (Q)**: Also known as the Q-function, denoted as Q(s, a), this function estimates the expected cumulative reward starting from a particular state and taking a specific action, again under a given policy. It tells you the value of taking a certain action in a certain state.

These value functions are fundamental to many reinforcement learning algorithms. They help the agent evaluate different states and actions, guiding it towards making decisions that maximize its long-term rewards. Algorithms like Q-learning, SARSA, and policy gradient methods often use value functions to learn optimal policies.

By iteratively updating the value functions based on experiences gathered during interaction with the environment, the agent can gradually improve its decision-making capabilities and learn to achieve its goals more effectively.

Model

The model typically refers to the agent's understanding or approximation of the environment it's interacting with. This understanding helps the agent make decisions about how to act in that environment to maximize some notion of cumulative reward.

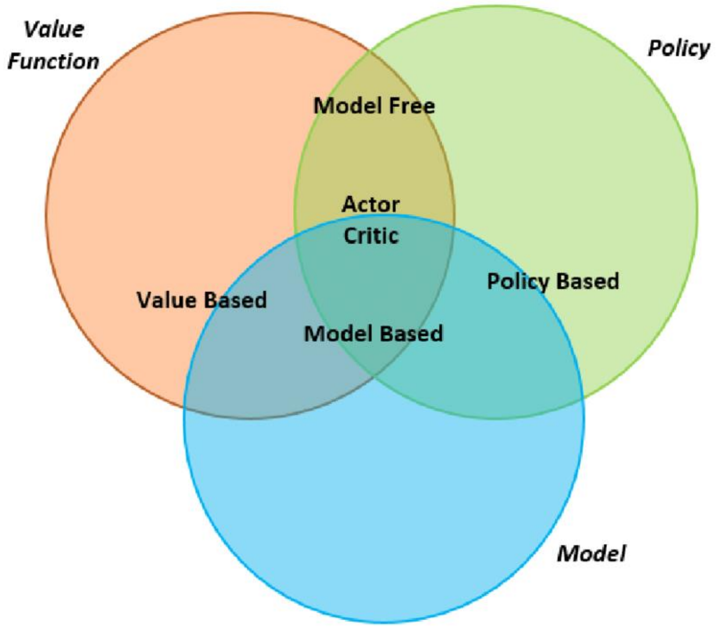


Figure 3 (Taxonomy of RL agents) ([14]

Exploration vs Exploitation

Exploration and exploitation are two key concepts in reinforcement learning that deal with how an agent interacts with its environment to maximize its rewards.

Exploration: Exploration involves trying out different actions to learn more about the environment. When an agent explores, it chooses actions that might not have been tried before or have been tried infrequently. Exploration is important because it allows the agent to discover new states, transitions, and rewards, which ultimately helps in refining its understanding of the environment and finding better policies.

Exploitation: Exploitation involves selecting actions that the agent believes will lead to the highest immediate reward based on its current knowledge. When an agent exploits, it leverages its existing knowledge to make decisions that maximize short-term rewards. Exploitation is crucial because it allows the agent to capitalize on what it has already learned and exploit the known good actions to accumulate rewards.

Balancing exploration and exploitation is a fundamental challenge in reinforcement learning. If the agent focuses too much on exploration, it may waste time trying out suboptimal actions and fail to exploit the best actions it has already discovered. On the other hand, if the agent focuses too much on exploitation, it may become stuck in a suboptimal policy and miss out on potentially better actions.

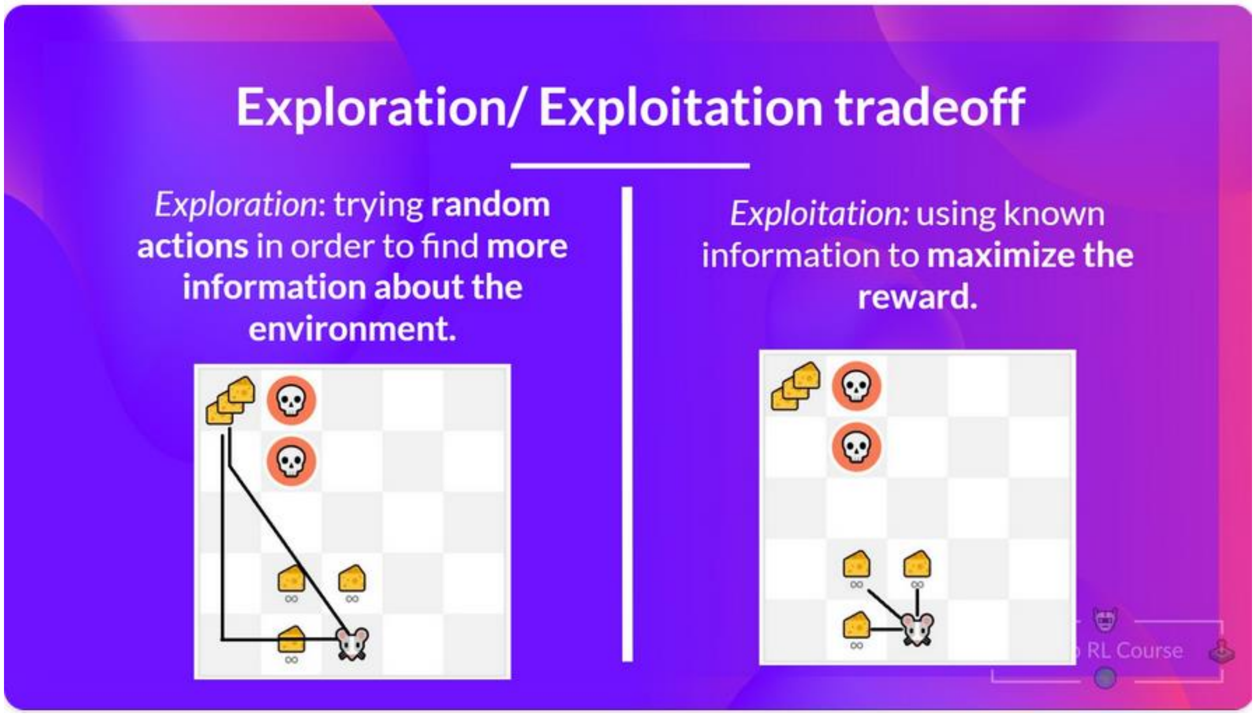

Exploration/ Exploitation tradeoff
Exploration: trying random actions in order to find more information about the environment.
Exploitation: using known information to maximize the reward.
RL Course

Figure 4(Exploration vs exploitation) [11]

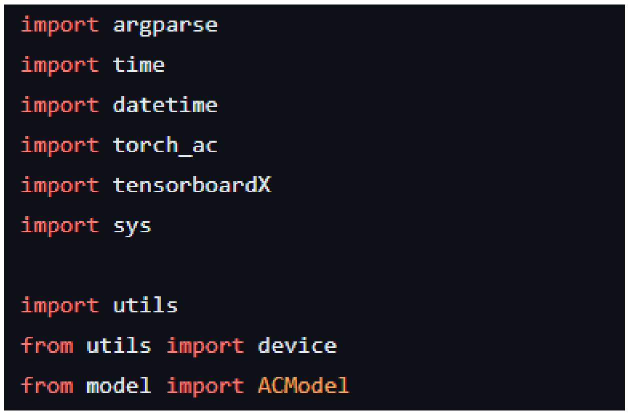
2.3 Innovative Approach

Originally, I had planned to improve on the existing q-baby-ai. As I was unable to set it up I had to change my approach. Using what I learned from my research I decided to use rl-starter-files and build from there.

They were three main scripts used in rl-starter-files. As well as a utility folder and a storage folder.

Train.py

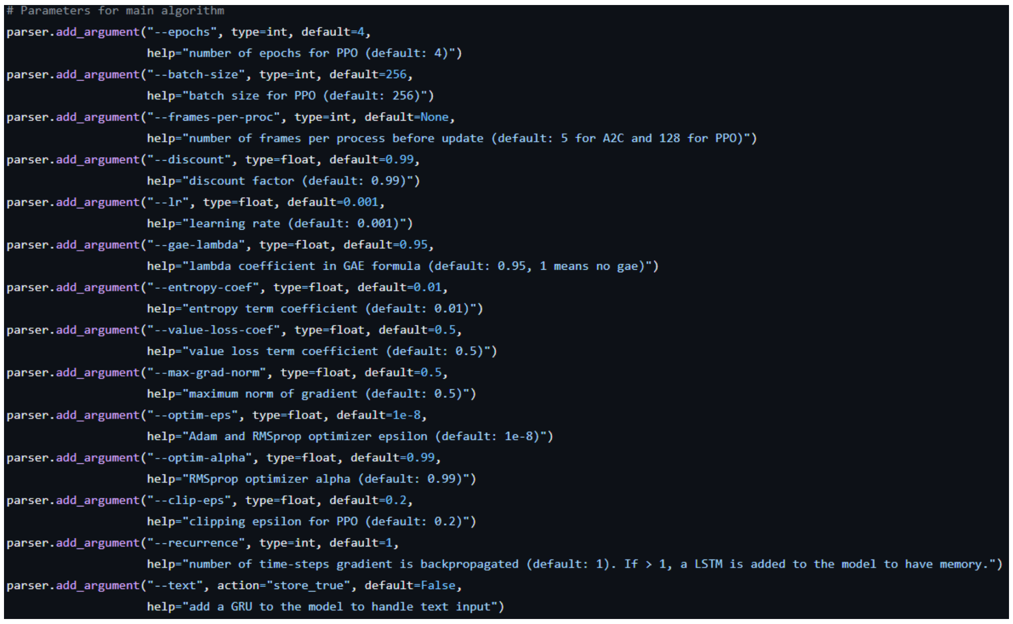
Imports: This section includes all the necessary imports for the script to function properly. These imports often include libraries like TensorFlow, PyTorch, or other reinforcement learning frameworks, as well as any custom modules or utility functions needed for training.



```python
import argparse
import time
import datetime
import torch_ac
import tensorboardX
import sys

import utils
from utils import device
from model import ACModel
```

Figure 5(Train.py - Imports)

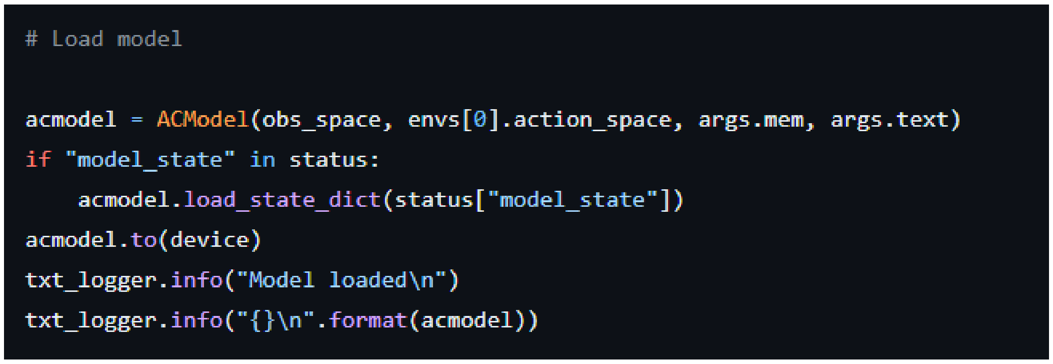
Argument Parsing: The script may parse command-line arguments to allow for customization of training parameters such as the learning rate, discount factor, number of episodes, etc. This makes the script more flexible and reusable for different experiments.



```python
# Parameters for main algorithm
parser.add_argument("--epochs", type=int, default=4,
                    help="number of epochs for PPO (default: 4)")
parser.add_argument("--batch-size", type=int, default=256,
                    help="batch size for PPO (default: 256)")
parser.add_argument("--frames-per-proc", type=int, default=None,
                    help="number of frames per process before update (default: 5 for A2C and 128 for PPO)")
parser.add_argument("--discount", type=float, default=0.99,
                    help="discount factor (default: 0.99)")
parser.add_argument("--lr", type=float, default=0.001,
                    help="learning rate (default: 0.001)")
parser.add_argument("--gae-lambda", type=float, default=0.95,
                    help="lambda coefficient in GAE formula (default: 0.95, 1 means no gae)")
parser.add_argument("--entropy-coef", type=float, default=0.01,
                    help="entropy term coefficient (default: 0.01)")
parser.add_argument("--value-loss-coef", type=float, default=0.5,
                    help="value loss term coefficient (default: 0.5)")
parser.add_argument("--max-grad-norm", type=float, default=0.5,
                    help="maximum norm of gradient (default: 0.5)")
parser.add_argument("--optim-eps", type=float, default=1e-8,
                    help="Adam and RMSprop optimizer epsilon (default: 1e-8)")
parser.add_argument("--optim-alpha", type=float, default=0.99,
                    help="RMSprop optimizer alpha (default: 0.99)")
parser.add_argument("--clip-eps", type=float, default=0.2,
                    help="clipping epsilon for PPO (default: 0.2)")
parser.add_argument("--recurrence", type=int, default=1,
                    help="number of time-steps gradient is backpropagated (default: 1). If > 1, a LSTM is added to the model to have memory.")
parser.add_argument("--text", action="store_true", default=False,
                    help="add a GRU to the model to handle text input")
```

Figure 6(Train.py – Argument Parsing)

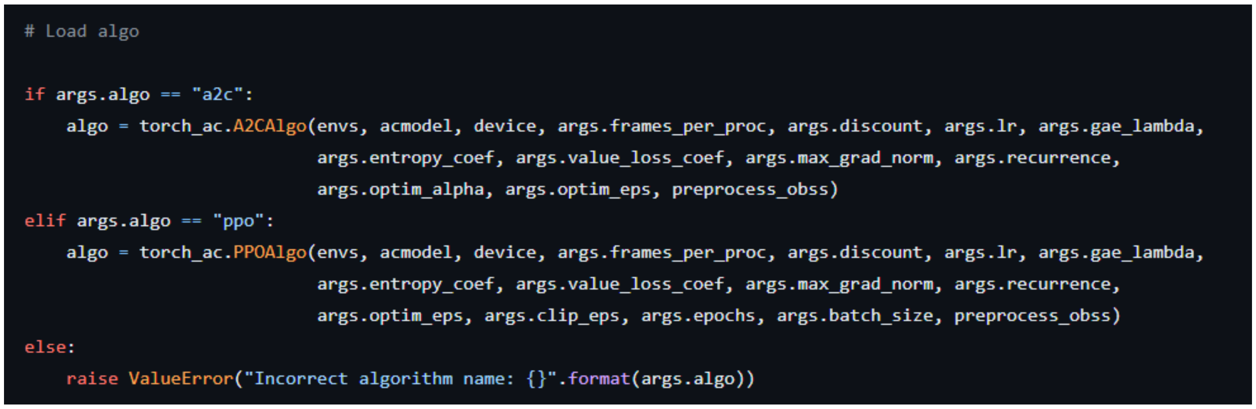
Model Initialization: Here, the script initializes the model where the agent will learn and interact with the environment.



```
# Load model

acmodel = ACModel(obs_space, envs[0].action_space, args.mem, args.text)
if "model_state" in status:
    acmodel.load_state_dict(status["model_state"])
acmodel.to(device)
txt_logger.info("Model loaded\n")
txt_logger.info("{}\n".format(acmodel))
```

Figure 7 (Train.py – Model Initialization)

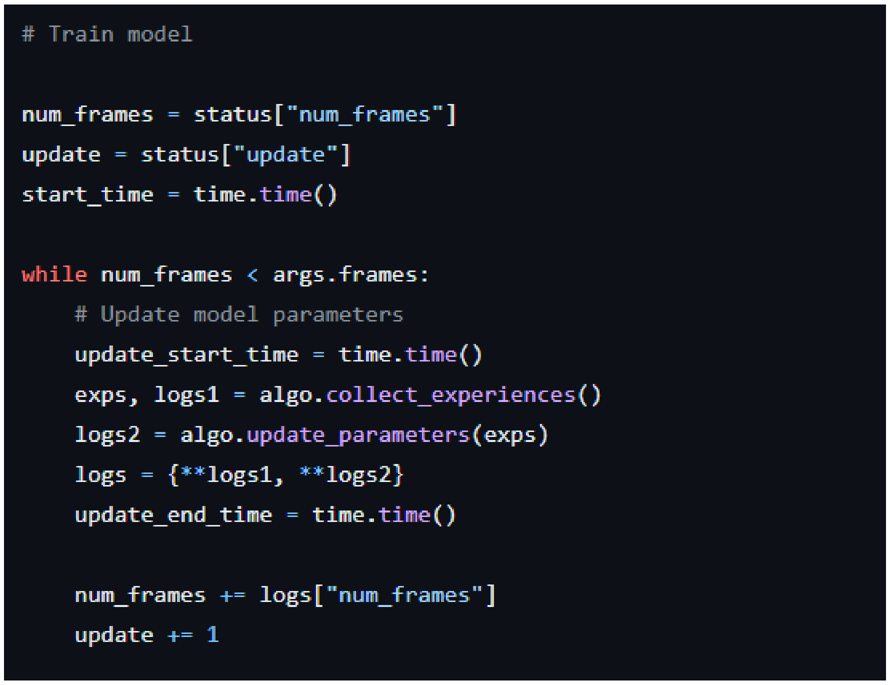
Agent Initialization: The reinforcement learning agent itself is initialized in this section. Depending on the specific algorithm being used, the agent may have different architectures and hyperparameters. The algorithm used are found in the torch_ac repository.



```
# Load algo

if args.algo == "a2c":
    algo = torch_ac.A2CAlgo(envs, acmodel, device, args.frames_per_proc, args.discount, args.lr, args.gae_lambda,
                            args.entropy_coef, args.value_loss_coef, args.max_grad_norm, args.recurrence,
                            args.optim_alpha, args.optim_eps, preprocess_obss)
elif args.algo == "ppo":
    algo = torch_ac.PPOAlgo(envs, acmodel, device, args.frames_per_proc, args.discount, args.lr, args.gae_lambda,
                            args.entropy_coef, args.value_loss_coef, args.max_grad_norm, args.recurrence,
                            args.optim_eps, args.clip_eps, args.epochs, args.batch_size, preprocess_obss)
else:
    raise ValueError("Incorrect algorithm name: {}".format(args.algo))
```

Figure 8 (Train.py – Agent initialization)

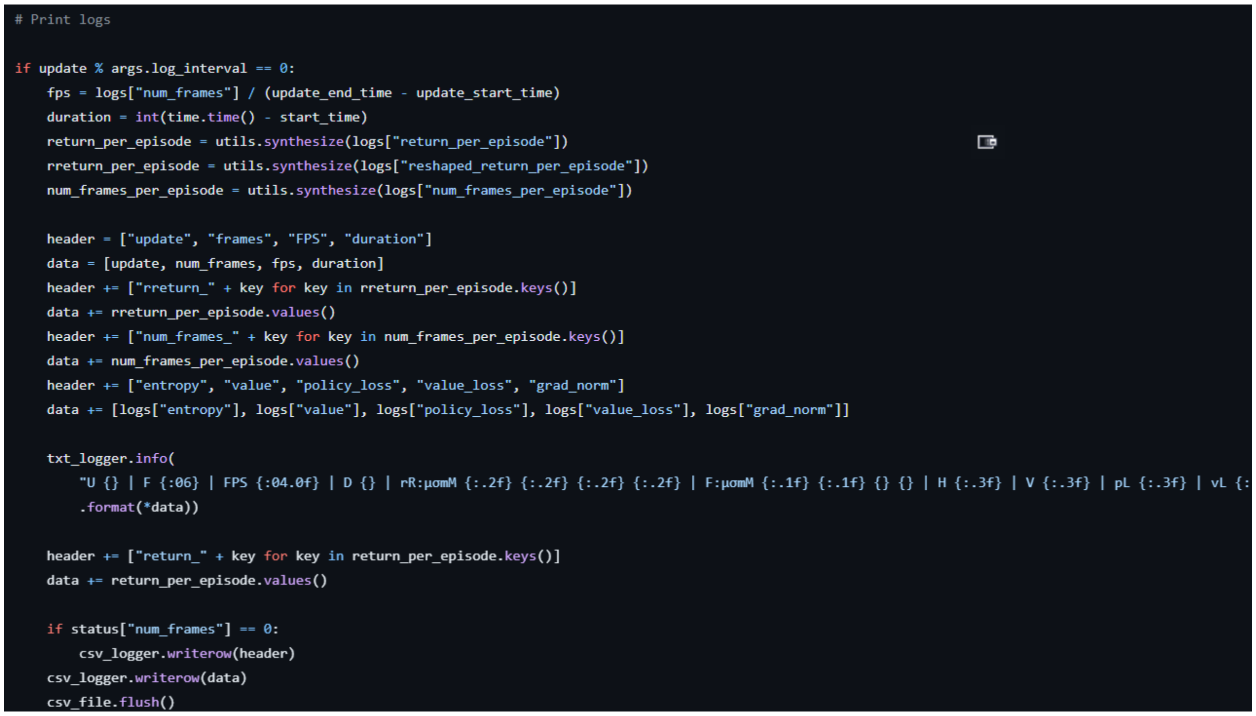
Training Loop: This is the heart of the script, where the agent learns to perform the task specified by the environment. The loop typically consists of iterating over a specified number of episodes, with each episode comprising multiple steps of interaction between the agent and the environment.



```
# Train model

num_frames = status["num_frames"]
update = status["update"]
start_time = time.time()

while num_frames < args.frames:
    # Update model parameters
    update_start_time = time.time()
    exps, logs1 = algo.collect_experiences()
    logs2 = algo.update_parameters(exps)
    logs = {**logs1, **logs2}
    update_end_time = time.time()

    num_frames += logs["num_frames"]
    update += 1
```

Figure 9 (Train.py - Training loop)

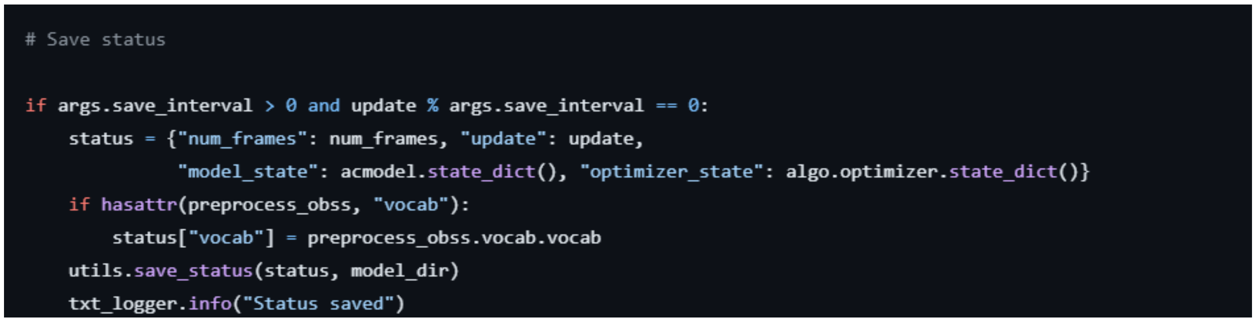
Logging and Visualization: During training, it's essential to log relevant metrics such as episode rewards, episode lengths, and any other statistics of interest. These metrics can help monitor the agent's progress and diagnose potential issues. Visualization tools may also be used to plot learning curves or visualize the agent's behavior.



```
# Print logs

if update % args.log_interval == 0:
    fps = logs["num_frames"] / (update_end_time - update_start_time)
    duration = int(time.time() - start_time)
    return_per_episode = utils.synthesize(logs["return_per_episode"])
    rreturn_per_episode = utils.synthesize(logs["reshaped_return_per_episode"])
    num_frames_per_episode = utils.synthesize(logs["num_frames_per_episode"])

    header = ["update", "frames", "FPS", "duration"]
    data = [update, num_frames, fps, duration]
    header += ["rreturn_" + key for key in rreturn_per_episode.keys()]
    data += rreturn_per_episode.values()
    header += ["num_frames_" + key for key in num_frames_per_episode.keys()]
    data += num_frames_per_episode.values()
    header += ["entropy", "value", "policy_loss", "value_loss", "grad_norm"]
    data += [logs["entropy"], logs["value"], logs["policy_loss"], logs["value_loss"], logs["grad_norm"]]

    txt_logger.info(
        "U {} | F {:06} | FPS {:04.0f} | D {} | rR:μσmM {:.2f} {:.2f} {:.2f} {:.2f} | F:μσmM {:.1f} {:.1f} {} {} | H {:.3f} | V {:.3f} | pL {:.3f} | vL {:
        .format(*data))

    header += ["return_" + key for key in return_per_episode.keys()]
    data += return_per_episode.values()

    if status["num_frames"] == 0:
        csv_logger.writerow(header)
    csv_logger.writerow(data)
    csv_file.flush()
```

Figure 10 (Train.py - Logging and visualization)

Model Saving: Once training is complete, the trained model parameters are saved for later use or evaluation. This allows you to deploy the trained agent in a real-world environment or continue training from where you left off.



```
# Save status

if args.save_interval > 0 and update % args.save_interval == 0:
    status = {"num_frames": num_frames, "update": update,
              "model_state": acmodel.state_dict(), "optimizer_state": algo.optimizer.state_dict()}
    if hasattr(preprocess_obss, "vocab"):
        status["vocab"] = preprocess_obss.vocab.vocab
    utils.save_status(status, model_dir)
    txt_logger.info("Status saved")
```

Figure 11 (Train.py - Model Saving)

Visualize.py

Argument Parsing: The script starts by parsing command-line arguments using argparse. These arguments include the environment name, the trained model name, the random seed, the number of environment resets at the beginning, whether to select the action with the highest probability, the pause duration between two consecutive actions, whether to store the output as a GIF, the number of episodes to visualize, and whether to add LSTM or GRU to the model.

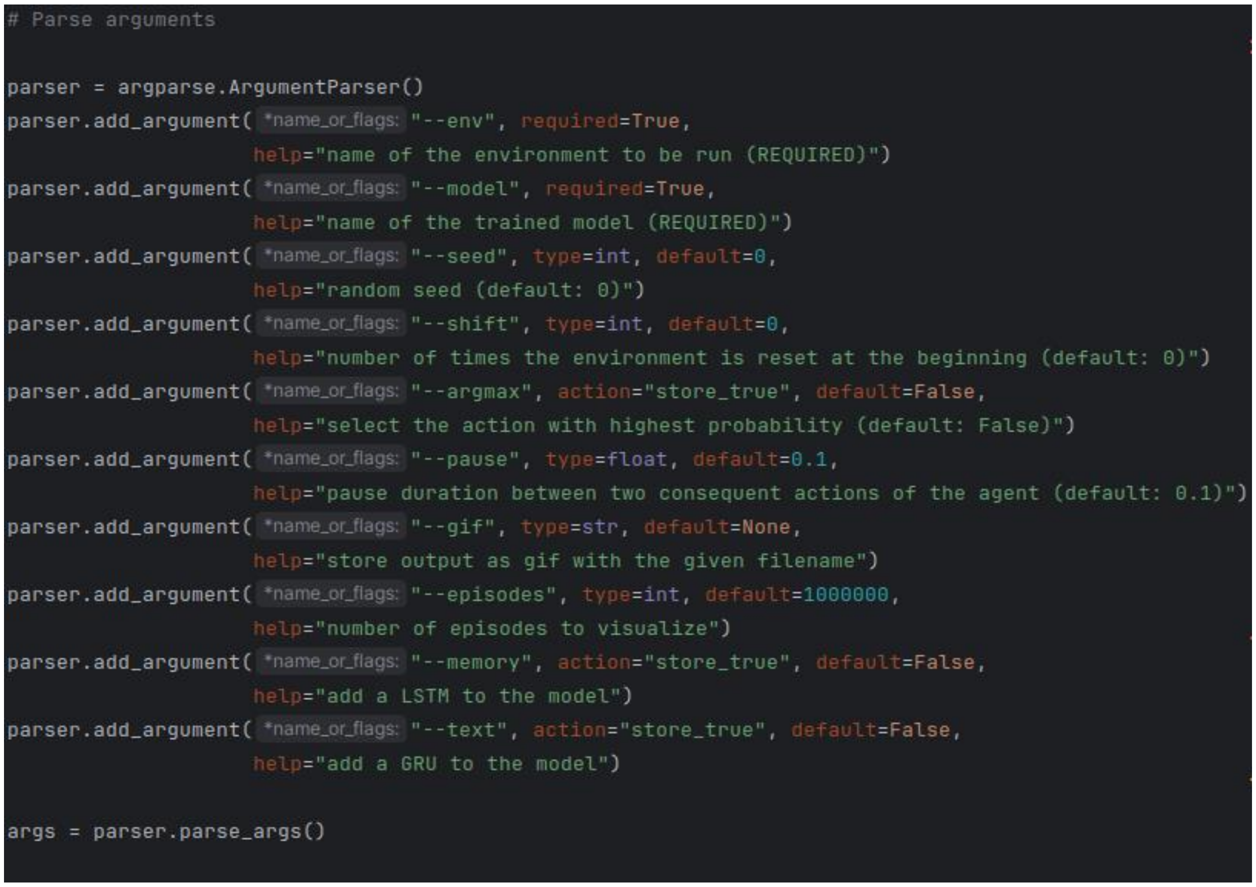

```
# Parse arguments

parser = argparse.ArgumentParser()
parser.add_argument( *name_or_flags: "--env", required=True,
                    help="name of the environment to be run (REQUIRED)")
parser.add_argument( *name_or_flags: "--model", required=True,
                    help="name of the trained model (REQUIRED)")
parser.add_argument( *name_or_flags: "--seed", type=int, default=0,
                    help="random seed (default: 0)")
parser.add_argument( *name_or_flags: "--shift", type=int, default=0,
                    help="number of times the environment is reset at the beginning (default: 0)")
parser.add_argument( *name_or_flags: "--argmax", action="store_true", default=False,
                    help="select the action with highest probability (default: False)")
parser.add_argument( *name_or_flags: "--pause", type=float, default=0.1,
                    help="pause duration between two consequent actions of the agent (default: 0.1)")
parser.add_argument( *name_or_flags: "--gif", type=str, default=None,
                    help="store output as gif with the given filename")
parser.add_argument( *name_or_flags: "--episodes", type=int, default=1000000,
                    help="number of episodes to visualize")
parser.add_argument( *name_or_flags: "--memory", action="store_true", default=False,
                    help="add a LSTM to the model")
parser.add_argument( *name_or_flags: "--text", action="store_true", default=False,
                    help="add a GRU to the model")

args = parser.parse_args()
```

Figure 12 (Visualize.py – Argument Parsing)

Setting Seed and Device: The script then sets the seed for all randomness sources to ensure reproducibility. It also sets the device (CPU or GPU) for computations.

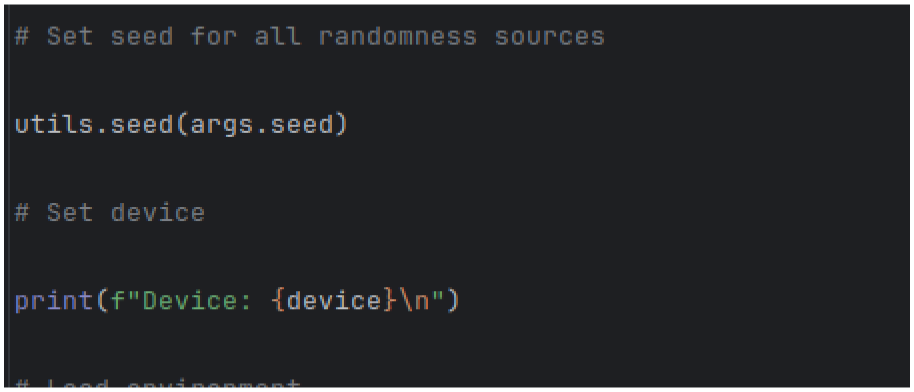
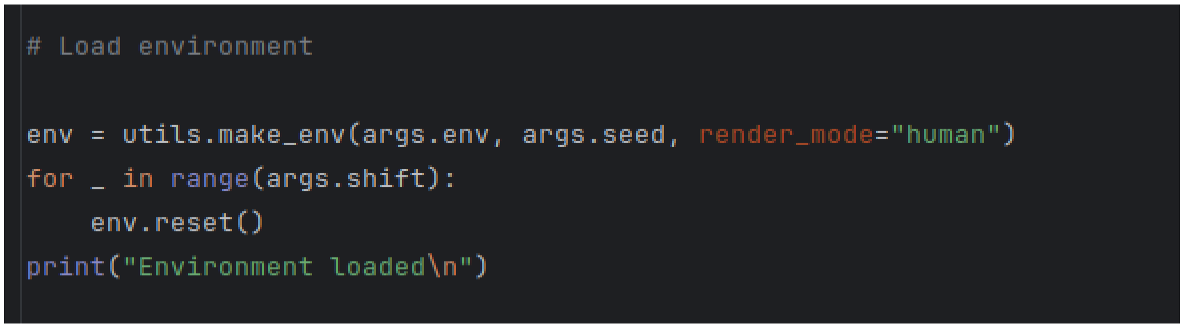

```
# Set seed for all randomness sources

utils.seed(args.seed)

# Set device

print(f"Device: {device}\n")
```

Figure 13 (Visualize.py – Setting seed and device)

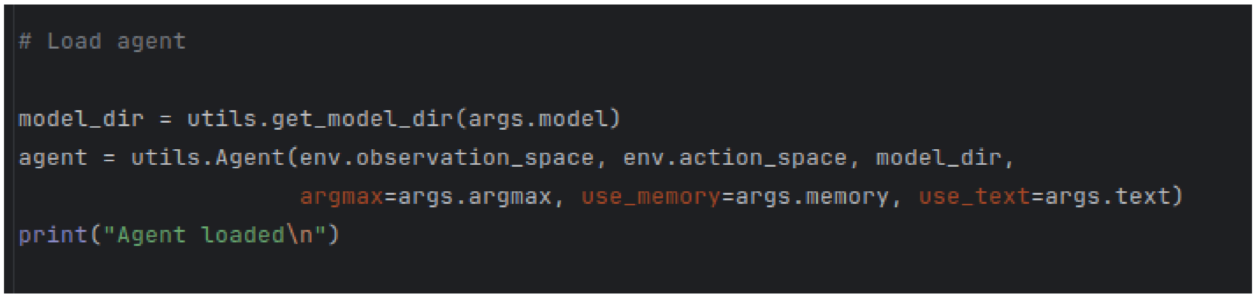
Loading Environment: The script loads the environment using the make_env function from the utils module. The environment is reset a number of times as specified by the shift argument.



```python
# Load environment

env = utils.make_env(args.env, args.seed, render_mode="human")
for _ in range(args.shift):
    env.reset()
print("Environment loaded\n")
```

Figure 14 (Visualize.py - Loading environment)

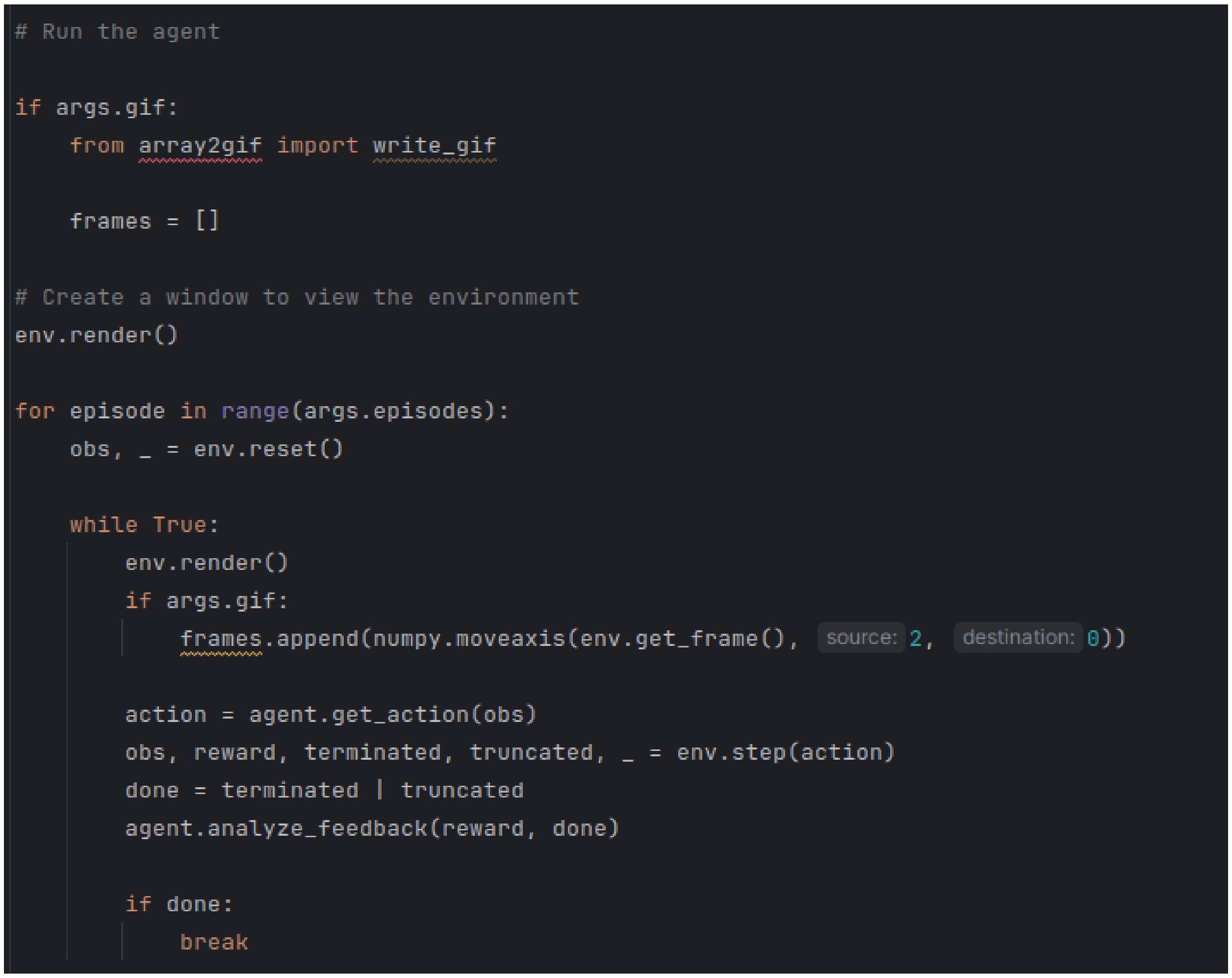
Loading Agent: The script loads the trained agent. The agent is an instance of the Agent class from the utils module. The agent takes an observation from the environment and decides on an action to take.



```python
# Load agent

model_dir = utils.get_model_dir(args.model)
agent = utils.Agent(env.observation_space, env.action_space, model_dir,
                    argmax=args.argmax, use_memory=args.memory, use_text=args.text)
print("Agent loaded\n")
```

Figure 15 (Visualize.py - Loading Agent)

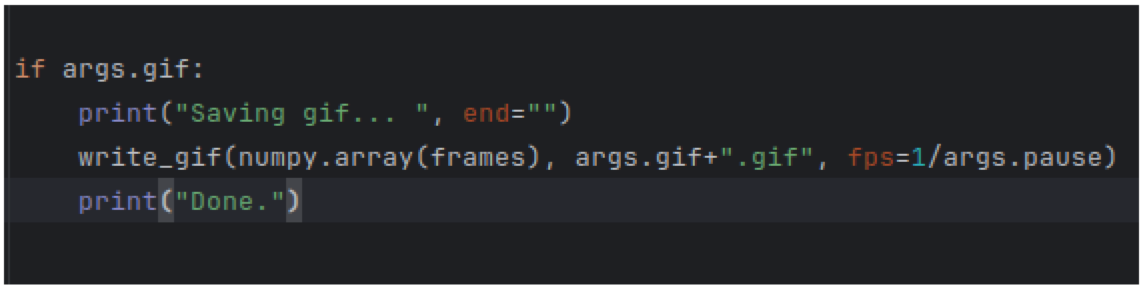
Running the Agent: The script enters a loop where the agent interacts with the environment. For each episode, the environment is reset and the agent starts taking actions until the episode ends (either by reaching a terminal state or by truncation). If the gif argument is provided, the script captures frames from the environment at each step and stores them in a list.



```
# Run the agent

if args.gif:
    from array2gif import write_gif

    frames = []

# Create a window to view the environment
env.render()

for episode in range(args.episodes):
    obs, _ = env.reset()

    while True:
        env.render()
        if args.gif:
            frames.append(numpy.moveaxis(env.get_frame(), source: 2, destination: 0))

        action = agent.get_action(obs)
        obs, reward, terminated, truncated, _ = env.step(action)
        done = terminated | truncated
        agent.analyze_feedback(reward, done)

        if done:
            break
```

Figure 16 (Visualize.py - Running the agent)

Saving as GIF: If the gif argument is provided, after all episodes are completed, the script saves the collected frames as a GIF using the write_gif function from the array2gif module.



```
if args.gif:
    print("Saving gif... ", end="")
    write_gif(numpy.array(frames), args.gif+".gif", fps=1/args.pause)
    print("Done.")
```

Figure 17 (Visualize.py - Saving as GIF)

PPO

PPO, which stands for Proximal Policy Optimization, is a popular reinforcement learning algorithm known for its stability and efficiency. It belongs to the class of policy gradient methods and is designed to optimize both sample efficiency and computational efficiency. [2]

A2C

A2C stands for Advantage Actor-Critic. It's a type of reinforcement learning algorithm that combines elements of both policy-based methods (Actor) and value-based methods (Critic). [3]

PPO vs A2C

	PPO	A2C
Policy Update Mechanism	Utilizes a clipped surrogate objective function to ensure stable and gradual policy updates. The objective is optimized to balance between policy improvement and stability, with a clipping mechanism to prevent overly large updates.	Updates the policy using the advantage function, which measures the relative advantage of each action compared to the average action taken in a given state. The advantage function guides the policy updates by indicating which actions lead to better-than-average outcomes.
Sample Efficiency	Generally more sample efficient compared to A2C, as it optimizes the policy using more efficient updates and employs a clipped surrogate objective to prevent large policy changes.	May require more samples to converge compared to PPO, as it relies on simple policy gradient updates without additional mechanisms for stability or efficiency.
Stability	Known for its stability during training, thanks to the clipped surrogate objective and careful balancing of policy updates. The clipping mechanism prevents drastic changes to the policy that could lead to instability.	While generally stable, A2C may be more sensitive to hyperparameter settings and require careful tuning to avoid issues like policy oscillation or divergence.
Parallelization	Can be parallelized effectively, with multiple instances of the environment running concurrently to collect experiences. This parallelization can significantly speed up training, especially in	Also supports parallelization, but may not be as inherently parallelizable as PPO due to differences in the policy update mechanism.

	distributed computing environments.	
Adaptability	Offers various hyperparameters that can be tuned to adapt to different environments and tasks. The clipped surrogate objective and other parameters allow for fine-tuning to achieve optimal performance and stability.	Similarly adaptable, with hyperparameters like learning rate, entropy coefficient, and network architecture that can be adjusted to suit different scenarios.
Implementation Complexity	May have a slightly higher implementation complexity compared to A2C due to the clipped surrogate objective and additional tuning parameters. However, there are well-documented implementations available, making it relatively straightforward to use.	Generally simpler to implement compared to PPO, as it involves straightforward policy gradient updates and does not require additional mechanisms like clipping.

AC Model

The AC (Actor-Critic) model is a type of reinforcement learning architecture that combines elements of both policy-based and value-based methods. It is designed to process observations from the environment, learn meaningful representations of the state space, and produce actions that maximize expected rewards while also estimating the value of being in a particular state. It's a versatile architecture that can be adapted to a wide range of reinforcement learning tasks, including those involving image and text inputs.

2.4 Conclusion

In conclusion, the research section has provided a comprehensive exploration of the existing literature relevant to my study. Moving forward, the insights gained from the research section will guide further sections. It is important to know the basics which will allow me to move forward towards the more advanced techniques.

Chapter 3. Development

3.1 Tools

PyCharm

PyCharm is an IDE used primarily for Python programming. It's developed by JetBrains, the same company behind popular IDEs like IntelliJ IDEA and PhpStorm. PyCharm offers a wide range of features to facilitate Python development, including code analysis, debugging and version control integration. PyCharm version control integration, allowing me to manage my RL project's codebase efficiently. Also, its project management features make it easy to organize your RL project's files, dependencies, and resources. You can create virtual environments, manage packages, and structure your project in a way that enhances productivity and maintainability.

GitHub

GitHub is a web-based platform that provides hosting for software development projects using the Git version control system. Allowing me to track changes, revert to previous versions if needed. This is crucial for managing the iterative development process typical in RL projects.

3.2 Libraries

Torch-ac

Torch-ac is a library that includes two algorithms. These two algorithms are what I'm going to use in this final year project. They have been explained earlier in this report. The two algorithms were PPO and A2C.

Minigrid

"minigrid" is a minimalist grid world environment designed for use in reinforcement learning research and experimentation. It provides a simple yet flexible framework for creating grid world environments where agents can learn and interact. I will be using this library to code each of the tasks as it offers a variety of environments with different complexities, ranging from simple rooms to more complex layouts with obstacles and multiple rooms.

TensorboardX

TensorboardX is a Python library that provides a way to visualize data. It is an extension of TensorFlow's native TensorBoardX tool, allowing users to log various types of data during training and then visualize it in the TensorBoardX interface. This will be useful for visualizing the results for the fifth chapter.

NumPy

NumPy is a fundamental Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. This will be used throughout my project.

gymnasium

Gymnasium is a toolkit for developing and comparing reinforcement learning algorithms. It provides a wide range of environments, from simple grid worlds to complex simulated physics environments, where agents can learn and improve their behavior through trial and error.

Gymnasium is designed to be easy to use and flexible, making it a popular choice for researchers, students, and practitioners working in the field of reinforcement learning. It provides a simple and consistent API for interacting with environments, allowing users to focus on developing and testing their algorithms without worrying about the specifics of each environment's implementation.

tkinter

Tkinter is a standard Python library used for creating graphical user interfaces (GUIs). It provides a set of tools and widgets for building desktop applications with interactive elements such as buttons, menus, text entry fields, and more.

3.3 Conclusion

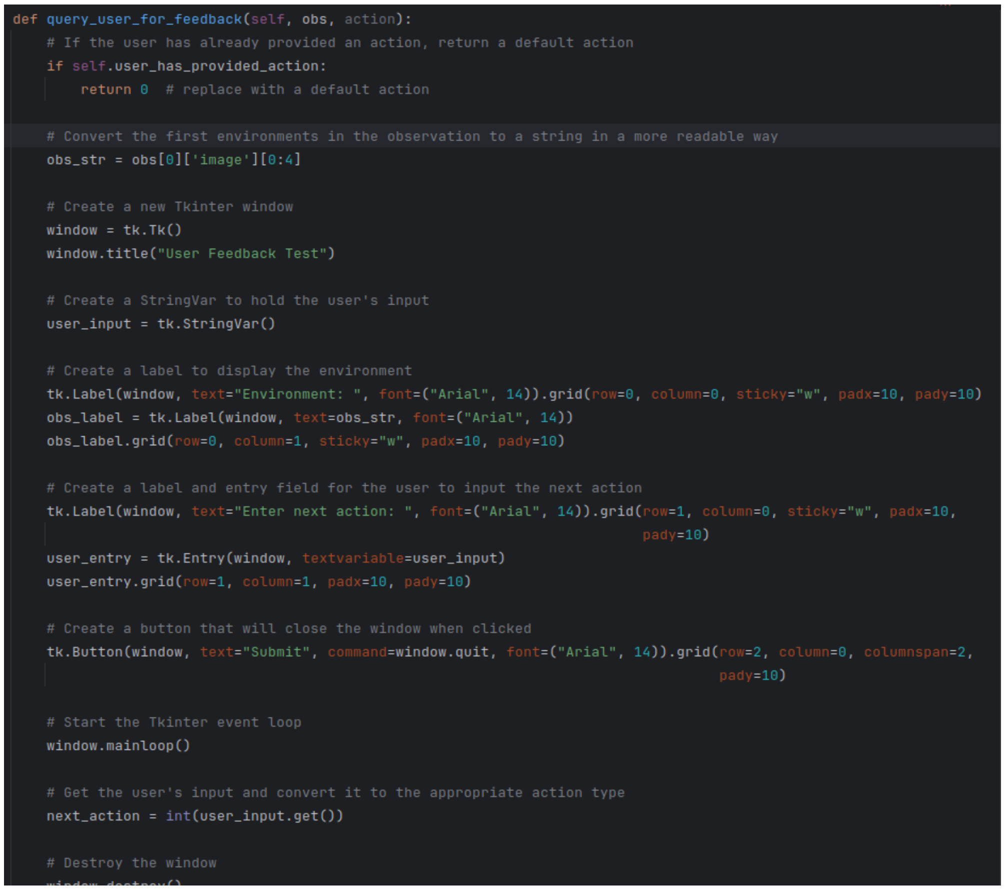
Within this chapter, we have delved into the details of the development phase of our project, showing the tools, libraries, programming languages, and other technological components that were instrumental in its realization. As shown by the array of technologies showcased, our project demanded a multifaceted approach. The forthcoming chapter showcases the functionality and features of the application.

Chapter 4. The application

4.1 Application Architecture

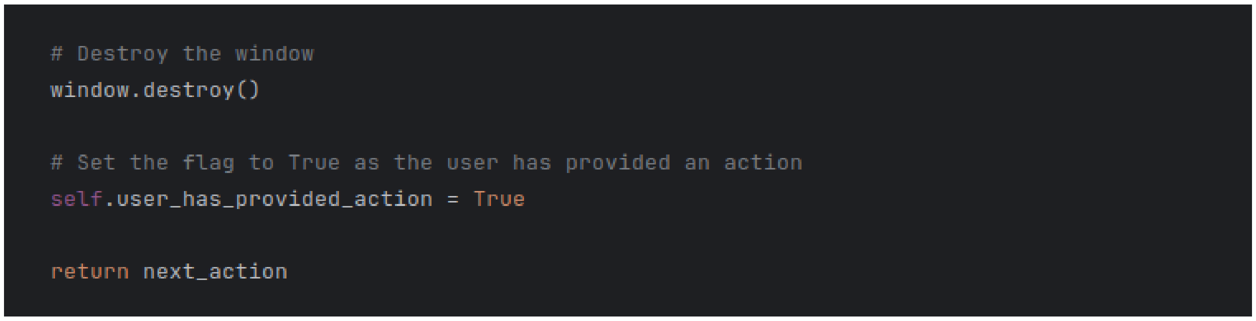
Function added

I added a function called query_user_for_feedback to both algorithms. This function shows the environment in array form and allows the user to decide the agent's next action. This changes the way the agent learns. It is using the tkinter library to create a Gui interface to enter the action space. This can all be seen in Figure 18.



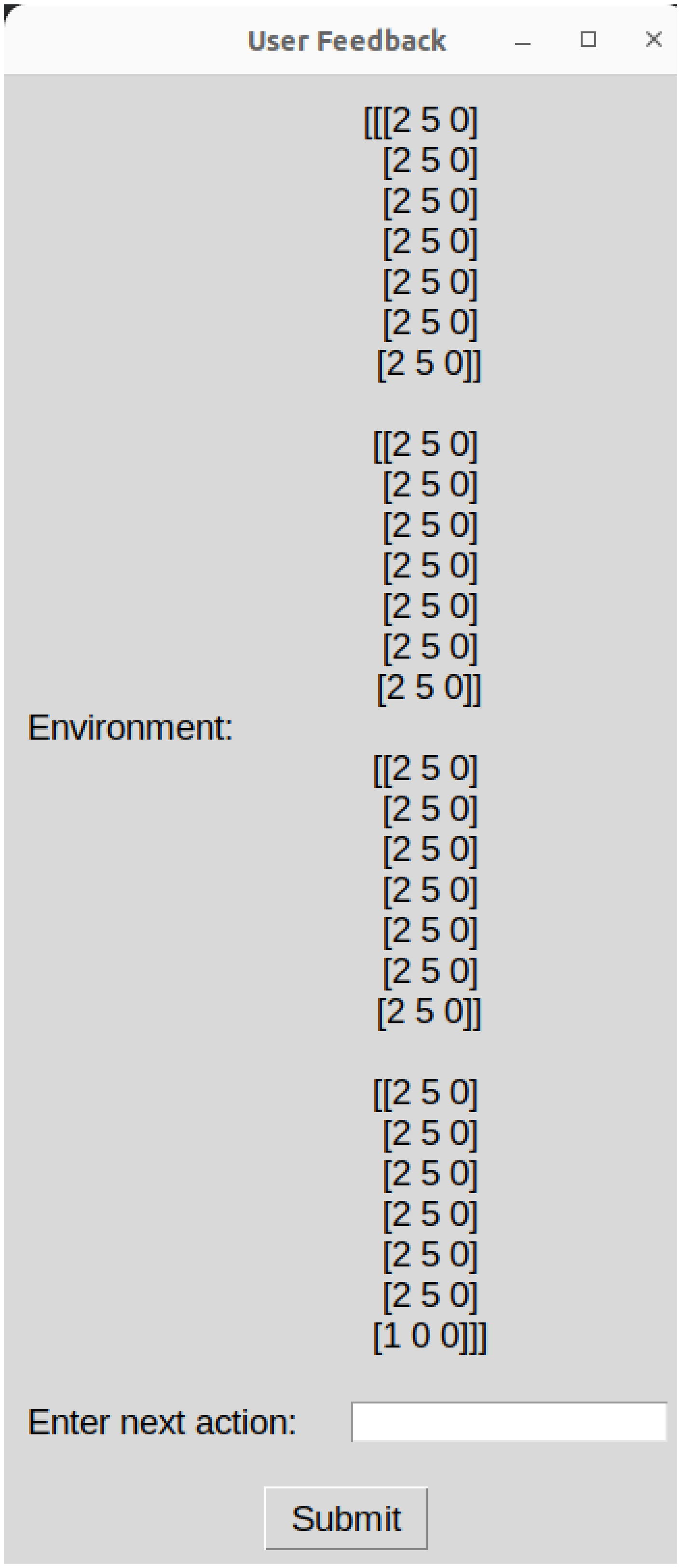
```python
def query_user_for_feedback(self, obs, action):
    # If the user has already provided an action, return a default action
    if self.user_has_provided_action:
        return 0  # replace with a default action

    # Convert the first environments in the observation to a string in a more readable way
    obs_str = obs[0]['image'][0:4]

    # Create a new Tkinter window
    window = tk.Tk()
    window.title("User Feedback Test")

    # Create a StringVar to hold the user's input
    user_input = tk.StringVar()

    # Create a label to display the environment
    tk.Label(window, text="Environment: ", font=("Arial", 14)).grid(row=0, column=0, sticky="w", padx=10, pady=10)
    obs_label = tk.Label(window, text=obs_str, font=("Arial", 14))
    obs_label.grid(row=0, column=1, sticky="w", padx=10, pady=10)

    # Create a label and entry field for the user to input the next action
    tk.Label(window, text="Enter next action: ", font=("Arial", 14)).grid(row=1, column=0, sticky="w", padx=10,
                                                                          pady=10)
    user_entry = tk.Entry(window, textvariable=user_input)
    user_entry.grid(row=1, column=1, padx=10, pady=10)

    # Create a button that will close the window when clicked
    tk.Button(window, text="Submit", command=window.quit, font=("Arial", 14)).grid(row=2, column=0, columnspan=2,
                                                                                   pady=10)

    # Start the Tkinter event loop
    window.mainloop()

    # Get the user's input and convert it to the appropriate action type
    next_action = int(user_input.get())

    # Destroy the window
    window.destroy()
```

Figure 18 (Function added)



```
# Destroy the window
window.destroy()

# Set the flag to True as the user has provided an action
self.user_has_provided_action = True

return next_action
```

Figure 19 (Function added contd.)



```
Environment:
[[[2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]]

 [[2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]]

 [[2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]]

 [[2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [2 5 0]
  [1 0 0]]]

Enter next action:
```

Figure 20 (User feedback)

Above in Figure 19 is the environment in array form. The next action the user enters would depend on the action space for that environment as well as the environment at that current state. This is done only during one episode. For the rest of the environments the agent learns using trial and error.

Environments used

Throughout this process, I used 5 environments you can think of them like the tasks that I defined earlier in the report. The visual representation of these environments can be seen in Figure 21-25. In each of the environments below I am assuming that the user is using the ppo algorithm. To use the a2c algorithm you would change the ppo to a2c in the algo argument.

Door Key

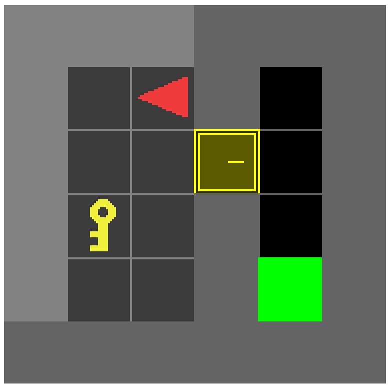


Figure 21 (Door Key Environment)

Command to train: python3 -m scripts.train --algo ppo --env MiniGrid-DoorKey-5x5-v0 --model DoorKey --save-interval 10 --frames 80000
Command to visualize: python3 -m scripts.visualize --env MiniGrid-DoorKey-5x5-v0 --model DoorKey

Go to door

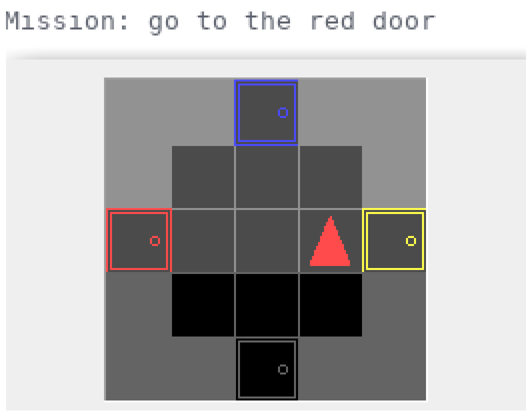

Mission: go to the red door

Figure 22 (Go to door Environment)

Command to train: python3 -m scripts.train --algo ppo --env MiniGrid-GoToDoor-5x5-v0 --model GoToDoor --text --save-interval 10 --frames 1000000

Command to visualize: python3 -m scripts.visualize --env MiniGrid-GoToDoor-5x5-v0 --model GoToDoor --text

Red blue doors

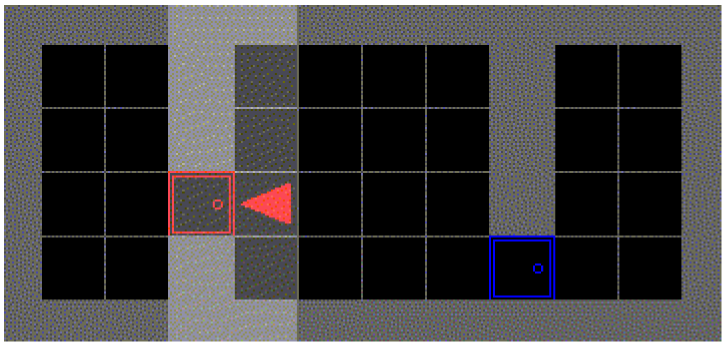


Figure 23(RedBlue doors Environment)

Command to train: python3 -m scripts.train --algo ppo --env MiniGrid-RedBlueDoors-6x6-v0 --model RedBlue --save-interval 10 --frames 80000
Command to visualize: python3 -m scripts.visualize --env MiniGrid-RedBlueDoors-6x6-v0 --model RedBlue

Red Ball

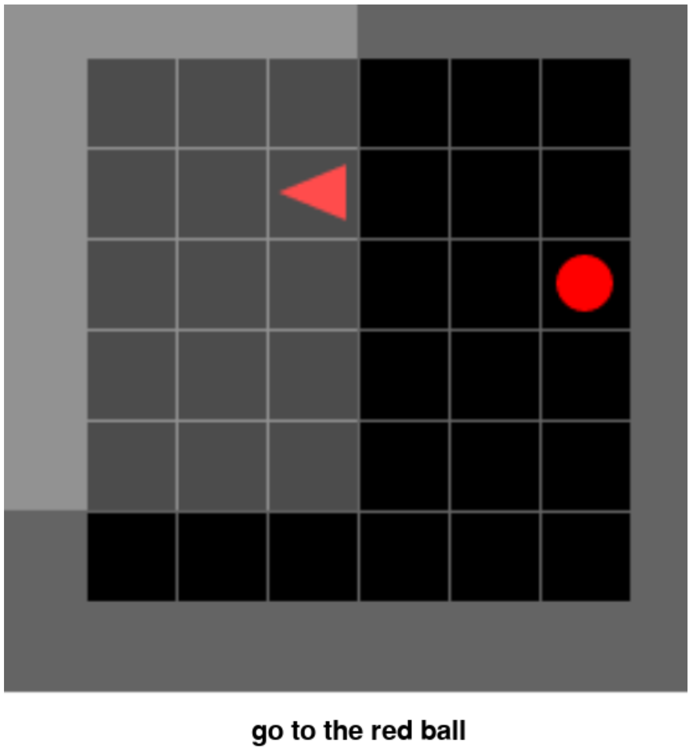


Figure 24 (RedBall Environment)

Command to train: python3 -m scripts.train --algo ppo --env BabyAI-GoToRedBallNoDists-v0 --model RedBall --save-interval 10 --frames 80000
Command to visualize: python3 -m scripts.visualize --env BabyAI-GoToRedBallNoDists-v0 --model RedBall

Open Red Door

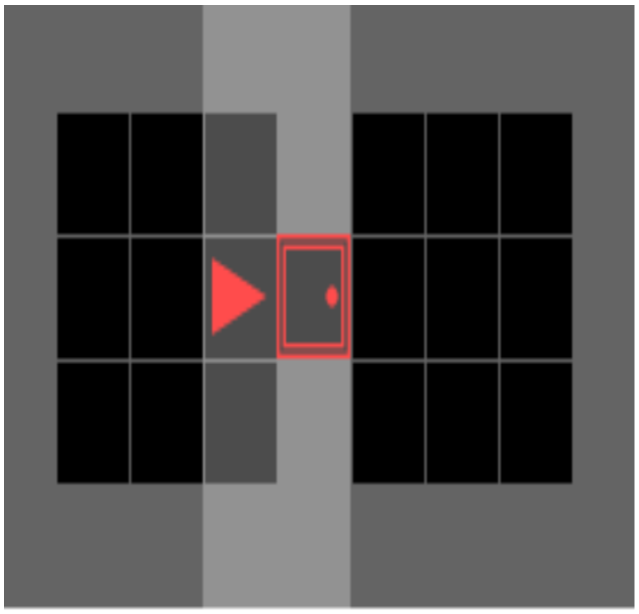


Figure 25 (Open Red door Environment)

Command to train: python3 -m scripts.train --algo ppo --env BabyAI-OpenRedDoor-v0 --model RedDoor --save-interval 10 --frames 80000
Command to visualize: python3 -m scripts.visualize --env BabyAI-OpenRedDoor-v0 --model RedDoor

4.3 Conclusion

Within this chapter I showcase the steps needed to set up my application. From what needed to be train certain environments to what is needed to be visualized. As well as the vital function added that allows the agent to query the user to help the algorithm make its next decisions.

Chapter 5. Results

5.1 Evaluation

In order to evaluate both the PPO and a2c algorithms that I am using I used the rreturn mean metric. The return, also known as the cumulative reward, is the sum of rewards obtained by the agent from the start of an episode until its termination. It reflects how good or successful a particular episode was for the agent. The mean return is the average cumulative reward. It represents the typical performance of the agent on average. Monitoring the rreturn mean is important for assessing the overall effectiveness and efficiency of the agent in achieving its objectives in the environment. By analyzing these metrics, I can gain insights into how well the agent is learning.

The prefix "r" in "rreturn mean" stands for "running" or "rolling," indicating that it represents a rolling average or a continuously updated mean over time. In this context, "rreturn mean" refers to a metric that computes the average return obtained by the agent over a moving window of episodes during training. This can be useful for tracking the agent's performance trends and smoothing out fluctuations in the performance metric. This provides a more dynamic and real-time view of the agent's performance. As the agent continues to learn and gather more experience, the rolling average is continuously updated to reflect its evolving capabilities.

It is expected that in the upcoming Figures the a2c algorithm will have more episodes than the PPO algorithm. A2c requires more samples to converge compared to PPO, as it relies on simple policy gradient updates without additional mechanisms for stability or efficiency. While generally PPO is more sample efficient compared to A2C, as it optimizes the policy using more efficient updates and employs a clipped surrogate objective to prevent large policy changes. This was covered earlier in the report.

Here the graphs for each of the environments and the models used:

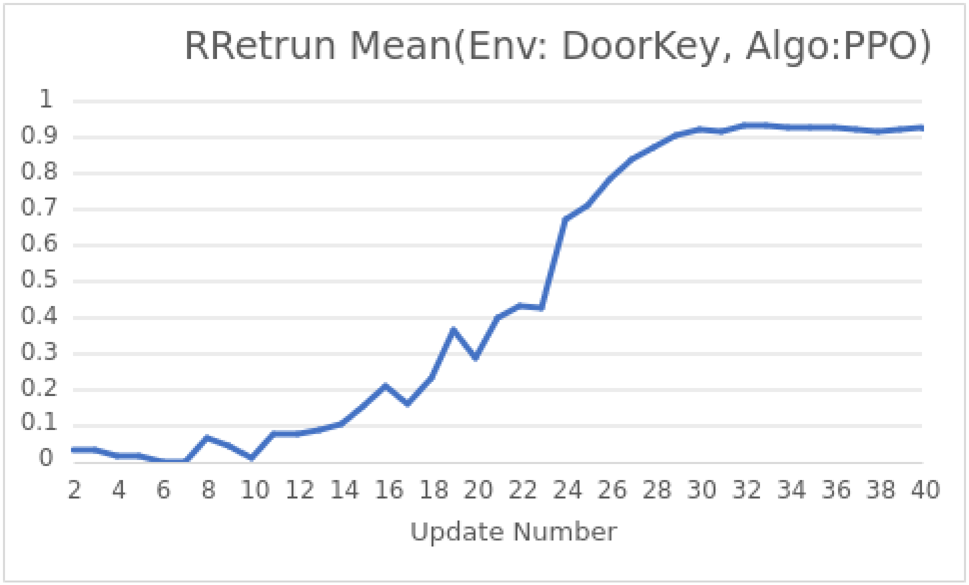


Figure 26 (Environment: DoorKey, Algorithm: PPO)

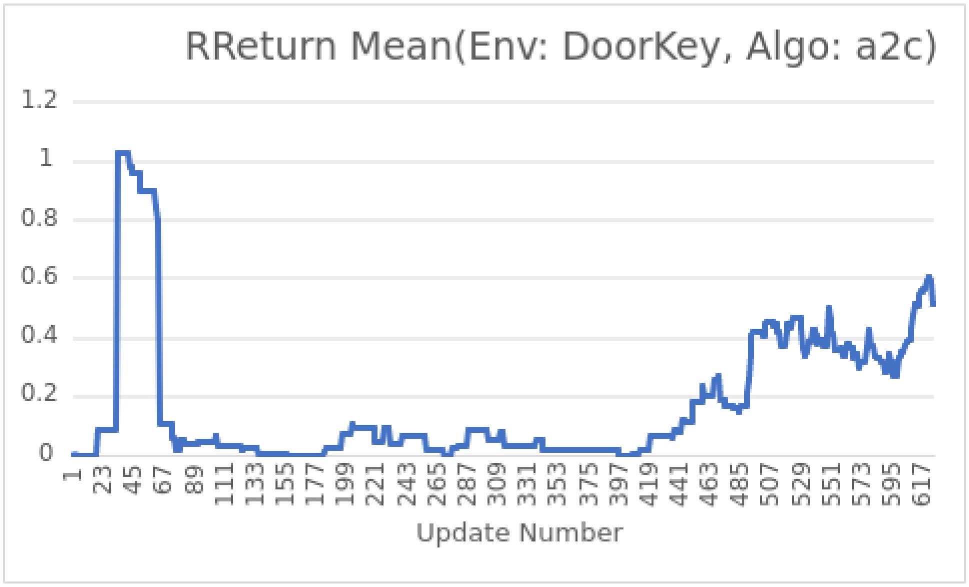


Figure 27(Environment: DoorKey, Algorithm: a2c)

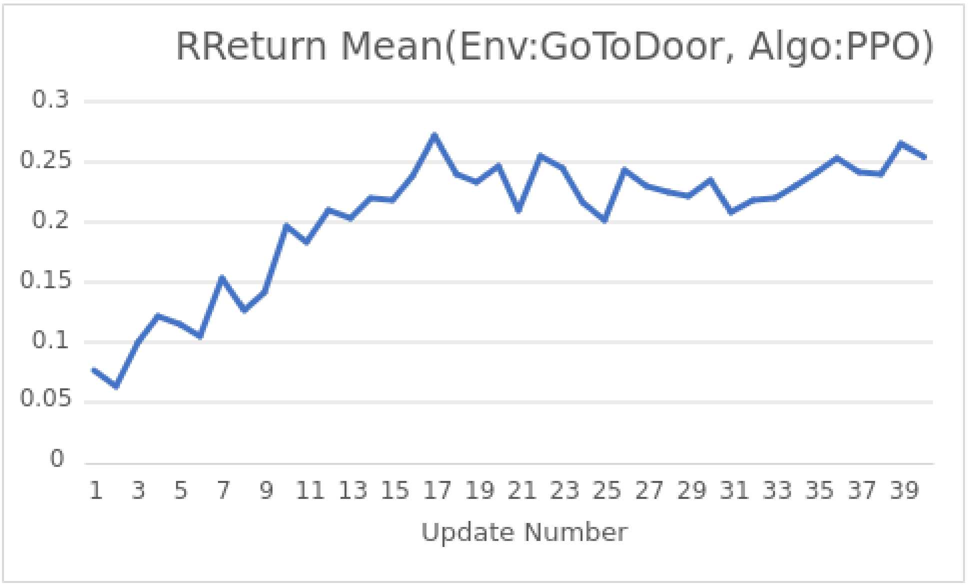


Figure 28 (Environment: GoToDoor, Algorithm: PPO)

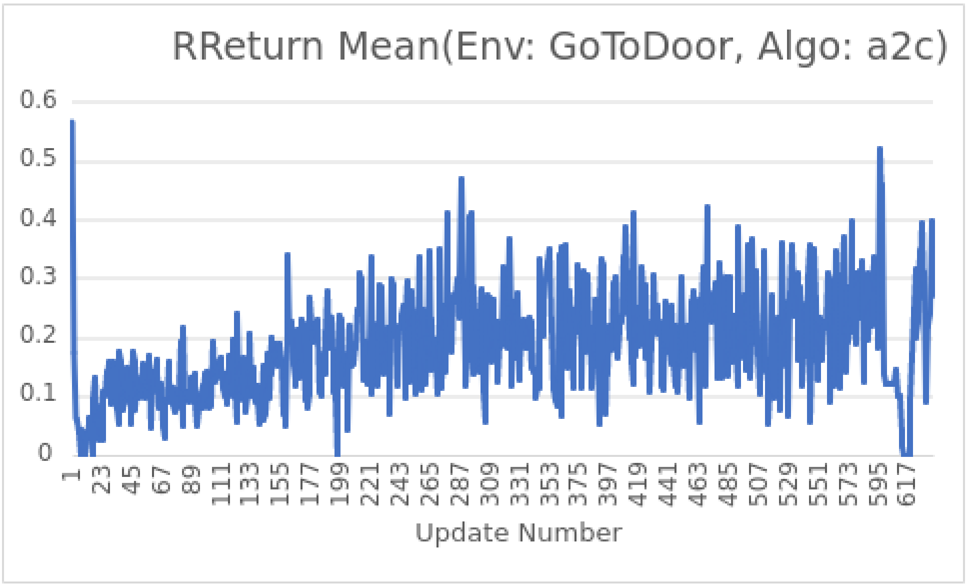


Figure 29 (Environment: GoToDoor, Algorithm a2c)

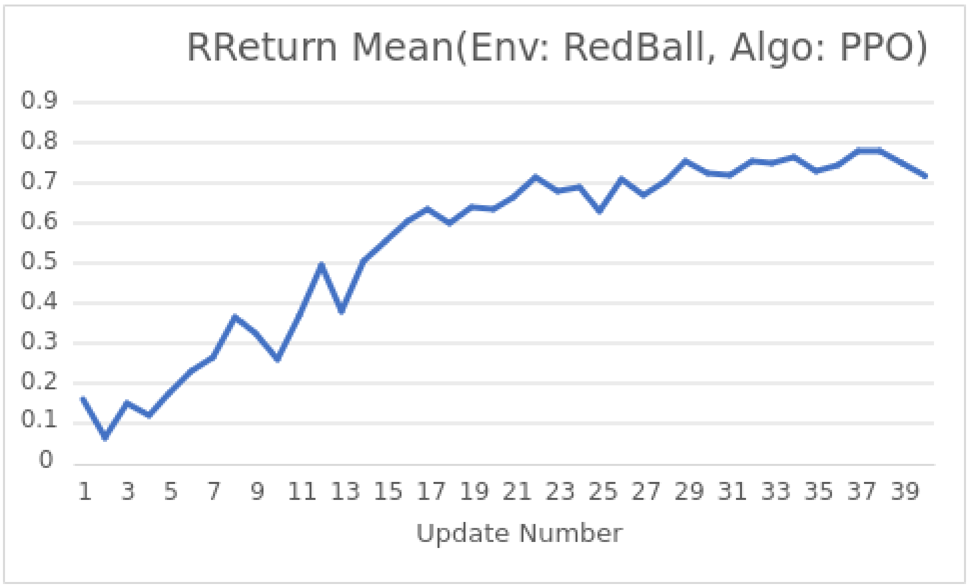


Figure 30 (Environment: RedBall, Algorithm: PPO)

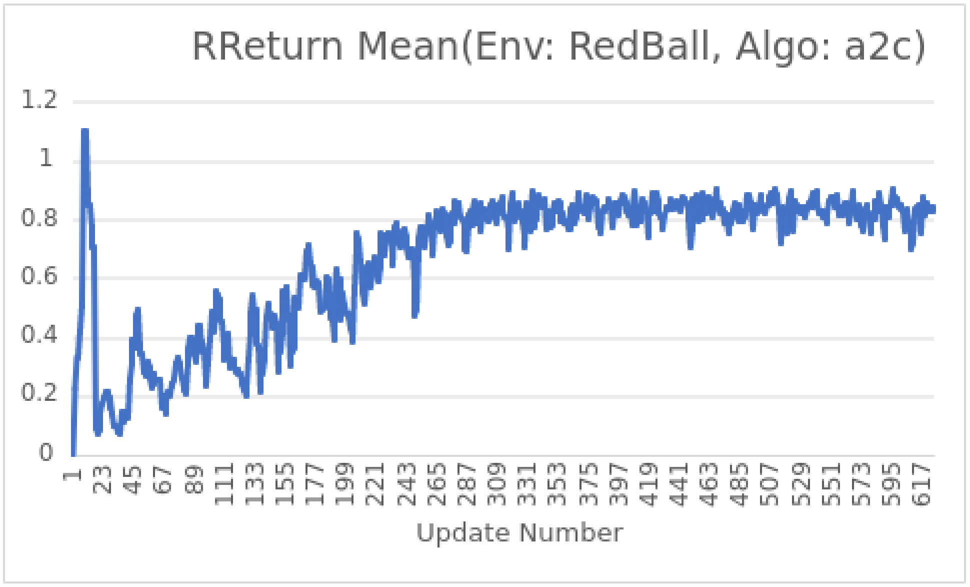


Figure 31 (Environment: RedBall, Algorithm: a2c)

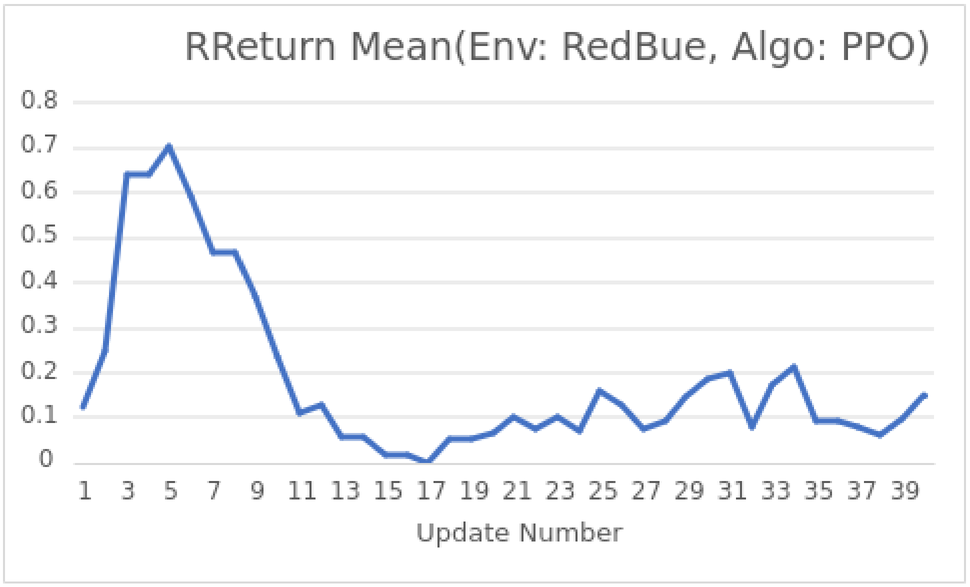


Figure 32 (Environment: RedBlue, Algorithm: PPO)

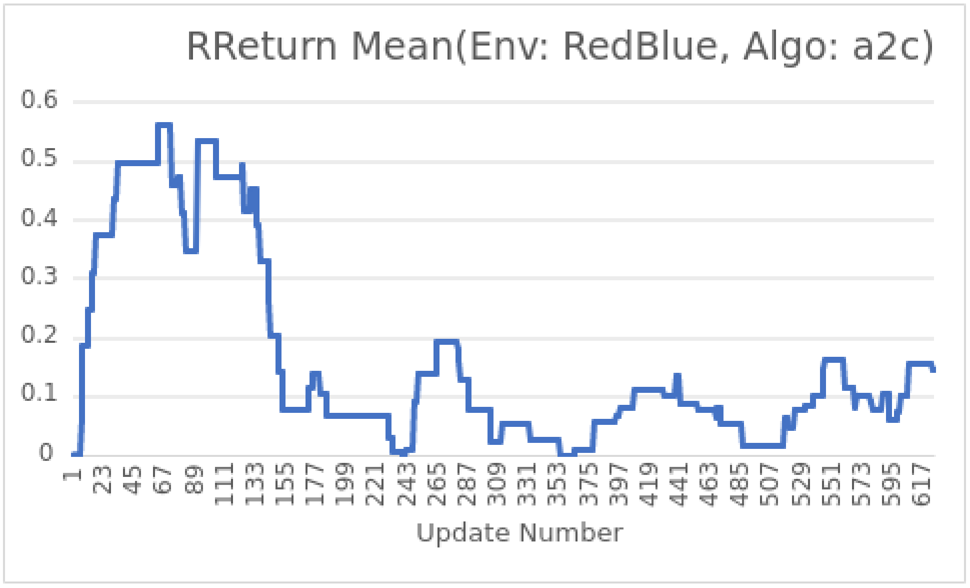


Figure 33 (Environment: RedBlue, Algorithm: a2c)

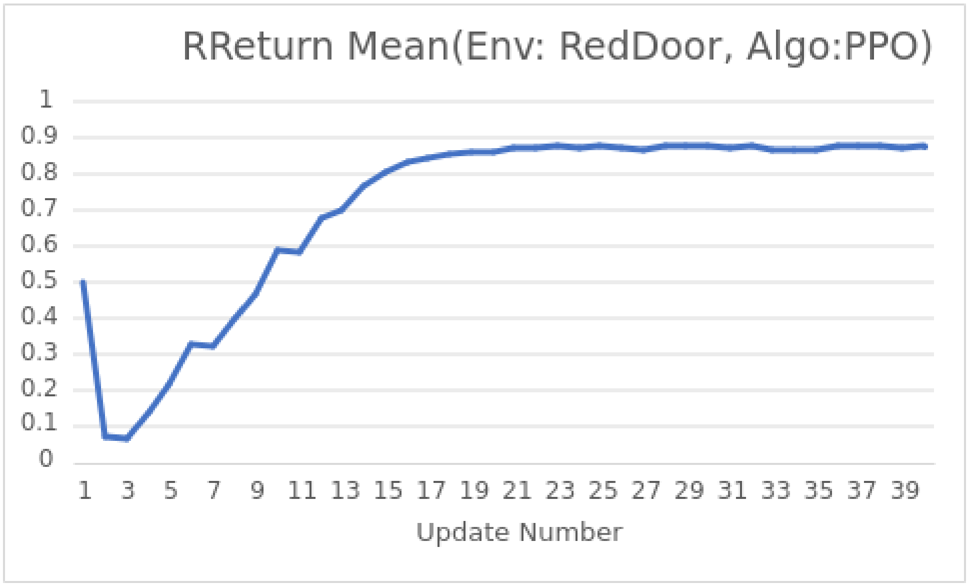


Figure 34 (Environment: RedDoor, Algorithm: PPO)

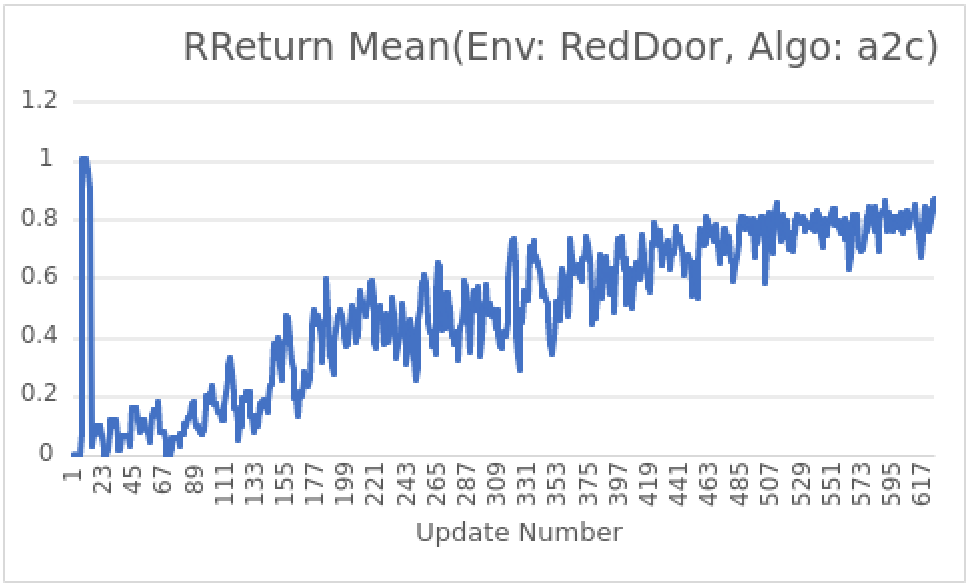


Figure 35 (Environment: RedDoor, Algorithm: a2c)

At first glance the PPO algorithm looks much more efficient than the a2c algorithm with these set of environments. With the PPO algorithm the rreturn mean seems to steadily increase with the exception of the RedBlue environment. In Figure 32 the rreturn means starts to decrease. This may indicate that it's struggling to explore and learn an effective policy in certain regions of the state-action space in this environment. The agent might be getting stuck in local optima or encountering challenging states that it's not able to handle effectively.

While with the a2c algorithm, the rreturn mean fluctuates a lot. This can be seen in Figures 27, 29, 31, 33 and 35, but especially in Figures 29, 31, and 35. This may be happening due to sensitivity to hyperparameters. Adjusting hyperparameters such as learning rates, exploration rates, or network architectures may help mitigate these fluctuations and stabilize the agent's learning process. Consistency in hyperparameters ensures a fair comparison between algorithms.

Also, in all the environments with the a2c algorithm the rreturn mean peaks early. This could be happening for a variety of reasons. The agent has learned to perform well in specific situations or states encountered early in training but struggles to generalize its performance to new or unseen states. This limited generalization capability can hinder the agent's ability to adapt to novel scenarios or environments, leading to suboptimal performance in the long run.

It could also indicate that the agent is exploiting a particular set of actions or states without adequately exploring other parts of the state-action space. Reinforcement learning agents need to balance exploration and exploitation to discover and learn optimal policies effectively. An early peak in the rreturn mean may suggest that the agent is favoring exploitation too soon without sufficiently exploring alternative strategies.

5.2 Deliverables

During this project I created an application that queries users during the training process of a RL model. After training the model is stored and can be used for evaluation. Any environment in the minigrid library or the baby-ai library can be used with this application. I decided to use five environments. The application can be seen in this GitHub repository: https://github.com/HameedAdagun/Final-Year-Project

5.3 Future Work

There are some things I would change to my application in the future. I would make the querying of users more user friendly. Right now, it would be difficult to understand if you are not familiar with the environment. Having a video demonstration of a user going through would also help with any difficulties encountered.

I would also add hyperparameter tuning for both algorithms. Both the parameters are constant, which helps with comparisons. Instead of having one set of hyperparameters, I would have multiple sets of hyperparameters. This eliminates the possibility of one algorithm performing better simply because it was given more favorable hyperparameter settings.

5.4 Conclusion

In conclusion, the evaluation of the Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) algorithms provides valuable insights into their performance in various environments. The utilization of the "rreturn mean" metric, representing the rolling average of cumulative rewards over time, allows for a dynamic assessment of the agents' learning capabilities. Using the PPO algorithm for the environments picked looks to be much more effective than using the a2c algorithm.

The development of an application for real-time user interaction during training, along with the ability to evaluate models across diverse environments, enhances the project's utility and flexibility. Future iterations could focus on improving user-friendliness through enhanced querying interfaces and video demonstrations, facilitating comprehension for users unfamiliar with the environments.

Overall, the evaluation and development efforts provide valuable contributions to reinforcement learning research and application, laying a foundation for further exploration and refinement in future endeavors.

References

[1] Torch-ac https://github.com/lcswillems/torch-ac/tree/master

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[3] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937). PMLR.

[4] https://www.youtube.com/playlist?list=PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u

[5] https://github.com/lcswillems/rl-starter-files

[6] https://github.com/mila-iqia/babyai/tree/master

[7] Liu, I. J., Yuan, X., Côté, M. A., Oudeyer, P. Y., & Schwing, A. (2022, June). Asking for knowledge (AFK): Training RL agents to query external knowledge using language. In *International Conference on Machine Learning* (pp. 14073-14093). PMLR.

[8] https://www.youtube.com/playlist?list=PLzuuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-

[9] https://www.davidsilver.uk/teaching/

[10]https://en.wikipedia.org/wiki/Markov_property#:~:text=A%20stochastic%20process%20has%20the,not%20depend%20on%20the%20past.

[11] https://huggingface.co/learn/deep-rl-course/unit1/exp-exp-tradeoff

[12] Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

[13] https://medium.com/analytics-vidhya/a-simple-reinforcement-learning-environment-from-scratch-72c37bb44843

[14] Oshingbesan, Adebayo & Ajiboye, Eniola & Kamashazi, Peruth & Mbaka, Timothy. (2022). Model-Free Reinforcement Learning for Asset Allocation.