

Collaborative Artificial Intelligence with Reinforcement Learning

COMP 8047 – Major Project 1

Jingyang Dong – A00997028

8-15-2022

Table of Contents

1. Introduction.....	3
1.1. Student Background	3
1.1.1. Education.....	3
1.1.2. Work Experience	3
1.2. Project Description	3
1.2.1. Essential Problems.....	4
1.2.2. Goals and Objectives	4
2. Body.....	5
2.1. Background.....	5
2.2. Possible Alternative Solutions	7
2.3. Chosen Method	7
2.4. Design Details	9
2.4.1. Deliverables	9
2.4.2. Algorithms and Approaches	10
2.4.3. Reinforcement Learning Components	15
2.4.4. Game Play Design, Player Profile, Storyboard, and Prototype.....	17
2.4.5. Use Case Diagram.....	18
2.4.6. Sequence Diagrams	19
2.4.7. Class Diagram	19
2.4.8. Flow Chart.....	22
2.4.9. Installation and Instruction Manual	24
2.5. Testing Details and Results.....	27
2.5.1. Unit Testings.....	28
2.5.2. Manual Testings.....	31
2.5.3. Testing Results.....	36
2.5.4. Tests for Game.....	36
2.5.5. Tests for Reinforcement Learning	40
2.5.6. Implications of Implementation	42
2.5.7. Innovation.....	45
2.6. Complexity.....	46
2.7. Research in New Technologies.....	47

2.7.1.	Research used in Project	48
2.8.	Technical Challenges	50
2.8.1.	The Learning Curve.....	50
2.8.1.1.	Training Multiple Agents in the Unity Game	50
2.8.1.2.	Changing the Cooperative Game to Fit Needs	51
2.8.1.3.	The Training Efficiency.....	52
2.9.	Future Enhancements	52
2.10.	Timeline and Milestones	53
3.	Conclusion	55
3.1.	Lessons Learned	55
3.2.	Closing Remarks	56
4.	Appendix.....	57
4.1.	Approved Proposal	57
4.2.	Project Supervisor Approvals	57
5.	References	58
6.	Change Log	59

1. Introduction

1.1. Student Background

I received the Diploma of Computer System Technology from BCIT in 2020, completing the Cloud Computing option. I have developed games using the Unity Engine, and OpenGL, and am experienced in programming in C, C++, C#, Java, JavaScript, HTML, and PHP.

1.1.1. Education

British Columbia Institute of Technology (BCIT) - Burnaby

- **Degree, Bachelor of Technology** **Sep. 2020 – (April. 2022)**
 - Games Development Option
- **Diploma, Computer Systems Technology** **Jan. 2018 – Apr. 2020**
 - Computer System Cloud Computing

1.1.2. Work Experience

- **Website Developer - Golbey's Law** **Apr. 2020 – Oct. 2020**
 - Created a trademark registration website with React.js, and .Net, and intergraded it with WordPress. (checkmarks.ca)
- **Website Developer (Group Project) - Wander Her** **Sep. 2019 – Dec.2019**
 - Implemented new features with prebuilt React.js, Redux, and Node.js Web applications.

1.2. Project Description

This project is integrating a reinforcement learning approach to create a game that allows multiple Artificial intelligence agents to react and collaborate. The game's core mechanic is to get the ball, hand the ball to the one who is faster to get a point and repeat. The game is similar to the game "capture the flag"(represented as a ball in this game) in terms of the mechanics, but not competing with each other. Two agents that were trained with the RL algorithm will be illustrated in a 2D top-down 2-player

cooperations game, showing the strategies that outperform the random AIs. The game is built with PyGame and is available to play with the executable on Windows, or running python files with other operating systems.

1.2.1. Essential Problems

In the current game industry, most Artificial Intelligence is aiming to beat the player with given instructions, but there is not much aiming for collaboration with the players. While there are some successful approaches, for example, single-agent reinforcement learning (Q-learning), the performance suffers when the number of agents grows. This is one of the main reasons reinforcement learning is yet widely used for games with multi-agent.

Agents suffer in Q-learning because all agent policies are naturally evolving, and when the number of actions overwhelms the agent, it will create problematic situations. A famous example is an approach in the game “Montezuma’s Revenge”, in which the goal of the agent is to navigate ladders, jump over obstacles, grab a key, and navigate to the door to complete the current level [1]. The problem is that by taking random actions, the agent is not seeing a reward for most of the tasks it accomplished. Because the number of actions needed to get a reward is huge, the agent would have no idea what to do to get the positive reward solely based on the random generation. With multiple agents, the situation gets worsens and it is expected to experience the agents being stuck, making no positive progress. This project aims to create a real-world experience when it comes to gaming, where every agent learns and acts independently to cooperate and compete with other agents. The environments reflecting this degree of complexity remain an open challenge.

1.2.2. Goals and Objectives

The goal of the project is to create a game like capture-the-flag that allows the agents, with a reinforcement learning approach, to cooperate with other agents.

The depth of this project includes the following:

- The AI will learn from scratch how to observe, act, cooperate and compete in customized environments. This includes
 - Training a population of agents.
 - Each agent should learn its internal reward signal (meaning it can generate its own internal goals, such as capturing the ball or passing it to others).
- The AI itself will use the input data and previously learned the outcome of previous data to come to an outcome on what decision to do next.
- The AI will work on discovering the behaviours of other agents and apply strategic decision-making.
- The trained agents perform better than the baseline – random AIs.

2. Body

2.1. Background

With supervised learning, one can easily implement the cost function and run gradient descent on it, but for Reinforcement Learning (RL), algorithms have many moving parts that are hard to debug, and they require effort in tuning to get good results. To better understand the project with some basic RL knowledge before the details, RL is about taking suitable action to maximize rewards in a particular situation. It employs a system of rewards and penalties to compel the computer to solve a problem by itself (see figure.1). Algorithms will be created for the agent to constantly learn, receive feedback, adjust behaviours, and eventually act with actions that help the agents to win the game.

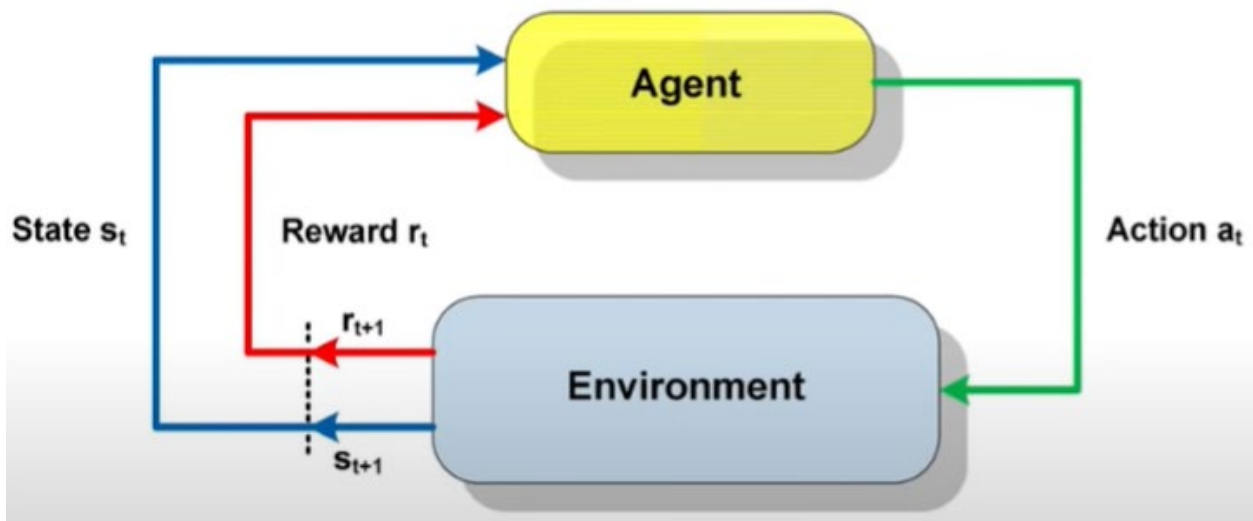


Figure 1. Reinforcement Learning States Machine An Introduction to Reinforcement Learning - Youtube.

<https://www.youtube.com/watch?v=JgvyzlkqxFO>.

The learning process is about constantly inputting the state of the agent (or the actions the agents take) and updating the value of the state (the weight of how likely the agent performs the same action that led to the reward). The initial input would be the actions of the players. When actions taken do not provide any rewards (scores), this episode of actions will be labelled as “do not do”, and the possibility of performing such actions would be weighted lower. In this project, the expected behaviour of the agent-training process contains multiple stages: 1) learning the basics of available moves; 2) Increasing navigation to try to reach the ball; 3) Applying strategies with another agent for getting the points in the game efficiently.

In stage one, the expected outputs from the agents are: recognize the moves in the game area; randomly moving like the random AI; not capturing the ball and doing nothing. In stage two, the agent should learn to navigate to the ball or the friendly character. In stage three, some strategies should be learned and applied. This includes waiting for the ball to respawn, passing the ball to the faster player, and finding the fastest routes.

Note that the project is designed as a 2D implementation with limited mechanics for reducing the complexity of training a population of agents.

2.2. Possible Alternative Solutions

There are alternative solutions to this project for both making the game and training the agents.

Initially, the concept game is planned to be made with Unity. Due to the higher computer specifications to run Unity, developing with Unity ended up with low performance due to high loading time and lags that happened occasionally. This happened with the “ML-development-kit-tool” as well in the development process. The tool is supported by Unity without modifications to the libraries. However, each iteration of a simple 2D game made in Unity takes about 2 minutes with the playable frame that is originally set. Nevertheless, Unity and other game-making software that supports 2D game development are capable of the game-making process. For example, the Unreal Engine, GameMaker Studio, Corona, and Godot Engine are part of the list.

In terms of the reinforcement learning process that includes multiple agents, there are fewer options despite the various algorithms there exist for RL algorithms. As this part of the RL is fairly new and has gradually become more popular in the past few years, more third-party libraries became useful. Since this project simply asks for an RL algorithm that solves the problem of helping the agents to decide the next moves in each episode, other algorithms remain effective and are possible alternative solutions to this project. Those algorithms include Advantage Actor Critic (A2C), Soft Actor-Critic (SAC), Deep Q-Learning (DQN was used in the Unity Engine initially), and Augmented Random Search (ARS), and possibly their variations or improved versions.

2.3. Chosen Method

The chosen solution was initially implemented in Unity with the ML agent Toolkit for combining the AI portion of the game. With further research and the aim of completing the project, it is proven that using Python with the supported libraries like Gym, PettingZoo, and PyTorch (later switched to Stable Baselines 3) was a better solution than the limited, time-consuming ML-agent kit overall. Especially when it comes to 2D

game simulations as Unity is widely used for 3D environments and does not provide full functionalities compared to other solutions. With the change of language and game engine, modified gameplay is chosen to ensure the success of the agents with RL algorithms. The original plan contains 2 friendly players and 2 enemy players with both cooperation and competing elements in the game. However, since training one agent was found to be time-consuming, 4 players could lead to the exponential growth of data needed as well as the training time. Since the designing process of the algorithm remains impractical, the 2 player cooperation game is made for proving the concept of the project. Two players are capable of showing the cooperation and strategies needed in a customized environment where each player is faster in a different area in the game with one goal – pass the ball to the bucket.

In terms of the algorithms picked, Q-learning was the initial attempt with the states and actions provided in Unity. However, the Proximal Policy Optimization (PPO) algorithm is later found to be a better fit as the simplicity of the new proof-of game supports image identification effectively. Thus, the observation spaces for inputs were provided as images. Even though the algorithm is different, the general goal and its concept remain the same – to help the agents to better decide what available moves could lead to more rewards.

The library used for training the multiple agents in the project is the “stable-baseline3” implementation of the RL learning algorithms PPO. This training process is possible due to those libraries. Otherwise, the complicated RL algorithms shall be made from scratch using Pytorch, which took the developer 20 hours to get familiar with the parameters and fundamental knowledge of such algorithms yet failed to succeed in completing the customized version of it. Not to mention the algorithms picked might not be as efficient after training under a specific environment, such as the conceptual game that is made. For example, with the same PPO algorithm trained, the agents were performing worse than the random AI in the initial implementation of the game that has various shapes and similar colours for the objects in the game, which made the PPO

algorithm harder to recognize the states for agents. Such vaguenesses should be eliminated in the project than trying with risk in the limited period of the project. Moreover, the Supersuit library is the chosen solution for reducing the headaches with the correctness of the data during the training process and it is proven to be effective. To better connect all dependencies used in this project, Anaconda package management is used to ensure the versions between language and packages are free of conflict. This would save tens of hours if one installation or update leads to reinstalling the packages and solving the conflicts then.

In terms of project management, the Agile Scrum approach was chosen to develop this project. Agile Scrum is great when changes occur frequently during the development. Since the Unity game needs to be developed in the starting phase of the project, the game should be adapted to changes if some of the approaches of the agent training fail. Also, the feedback on the game should be gathered as early as possible to ensure fewer changes to the core mechanics of the game and allow more focus on the agent training. This will be possibly very difficult to perform with the waterfall approach. Therefore, Agile Scrum will allow me to keep testing during development and keep adjusting to users' feedback.

A development cycle (sprint) is planned bi-weekly. I will add desired tasks to the Trello backlog before each sprint. New tasks and finished tasks were updated with priority labels during and after each sprint. This approach made the development process more organized and efficient.

2.4. Design Details

2.4.1. Deliverables

The deliverables in this project are:

- Proof-of-concept game:
 - Project executables and readme.txt
 - Demo of the RL algorithm trained agents
 - Demo executable for the random AI

- Report: Major Project report

2.4.2. Algorithms and Approaches

Training agents in multi-agent systems requires teammates and opponents in the environment to generate a learning experience. In this project, several different approaches to the algorithm were attempted for the agents to perform well and for learning purposes. The first approach is to apply an algorithm similar to the agent's behaviour with its previous actions. In other words, the agent is trained by playing against its policy. While self-play variants can be effective in some multi-agent games, this method turned out to be a failure. Unfortunately, as this approach requires a series of actions recorded by humans, the system of such structure is difficult to make and is being dropped for this project.

With the shift of focus moving to the RL algorithms, Machine Learning Development Toolkit is used for implementing the multi-agent game attempt in Unity (Figure 2). However, due to the limitation of the toolkit and the unsupported of training with a such black box in Unity, as well as the inefficient and time-consuming training, this attempt is dropped and the focus moved to

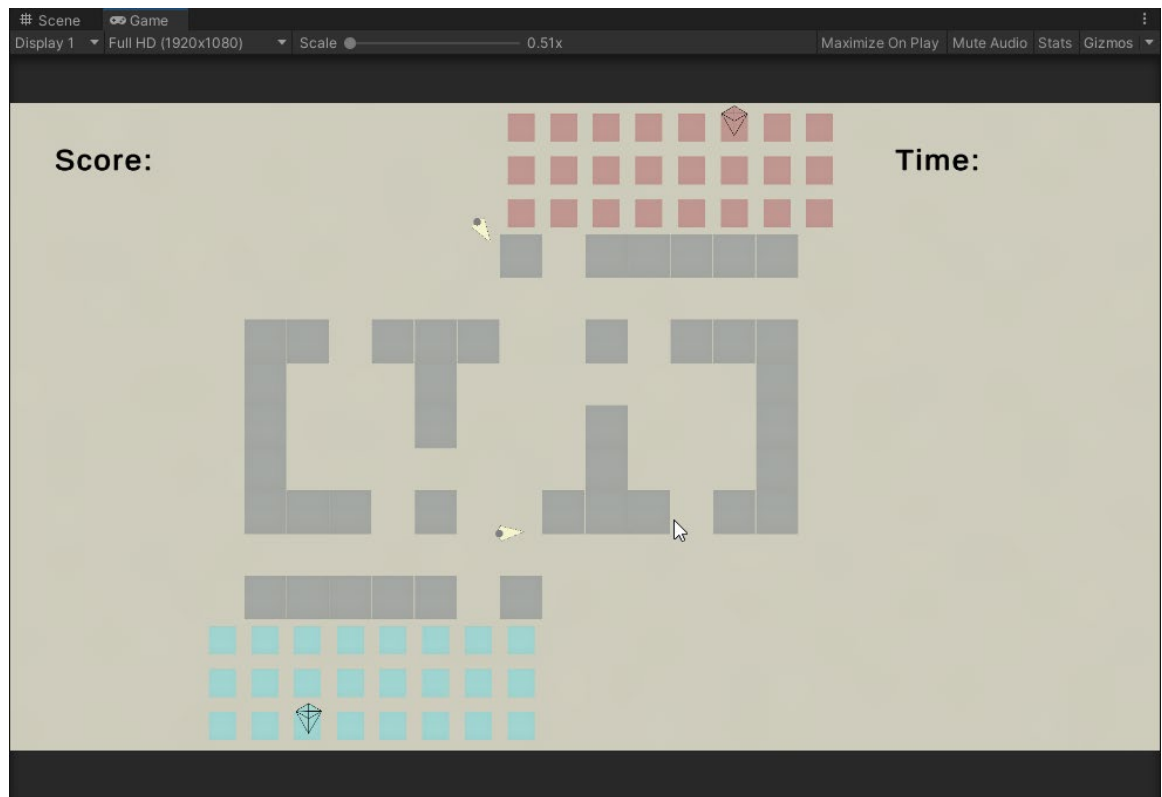


Figure 2 Initial Game Concept Implementation in Unity

The third approach is to apply the RL algorithm to train one agent instead of multiple ones. With the help of the Gym library, the agent is trained to get a flag in a command line 2D game (Figure 2).

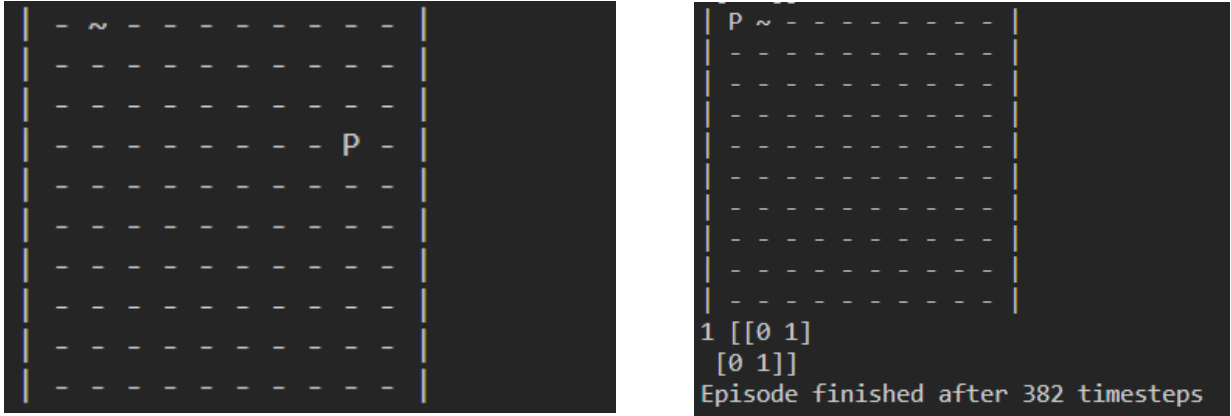


Figure 3 Initial Concept Game for Catching the flag. The States are: Start (left) and End (right).

This simple 2D game is built for testing whether RL is possible to reach the desired outcomes, which is to reach the goal in the game. The P symbol in Figure 2 is the agent (player), and the “~” symbol is the goal (flag). Q-Learning algorithm is used in this attempt. The equation is shown in Figure 3.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)}_{\text{new value (temporal difference target)}}$$

temporal difference

Figure 3: Q-learning algorithm

In this equation, S is a set of states; A is a set of actions per state; Executing an action in a specific state provides the agent with a reward. The goal of the agent is to maximize its total reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state and influencing current action by the potential future reward.

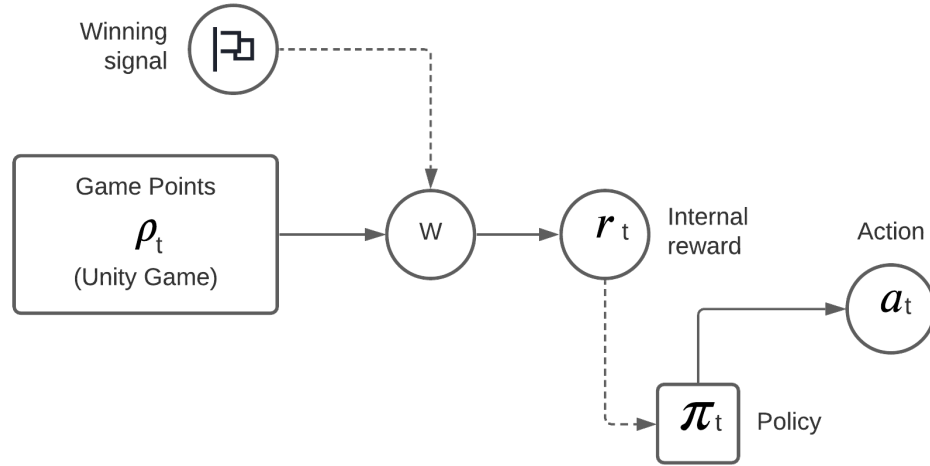


Figure 4: Agent architecture and benchmarking.

The action distribution Policy (π_t) is conditional at each time step t . The agent will be updated using reinforcement learning based on the agent's internal reward Signal (r_t), which will be obtained from w of game points (or scores from the winning screen) p_t [6]. The w is meant for setting win/loss signals as a reward of $r_t = 1, 0$, and -1 accordingly. With the different rewards the agents receive, different responses are expected for the agents under states where: a) agent is holding the ball; b) the ball is held by a teammate; c) the ball is put to the goal and respawned.

The final approach is aiming to provide a solution to train several P different agents in parallel that plays with each other, which introduces diversity amongst agents to stabilize training. Each agent learns from experience generated by playing with other agents sampled from the given population. It is also crucial to ensure a diverse set of agent behaviours are seen during training. With the PettingZoo library, it is possible to update the observations, rewards, and actions of each agent (Figure 5).

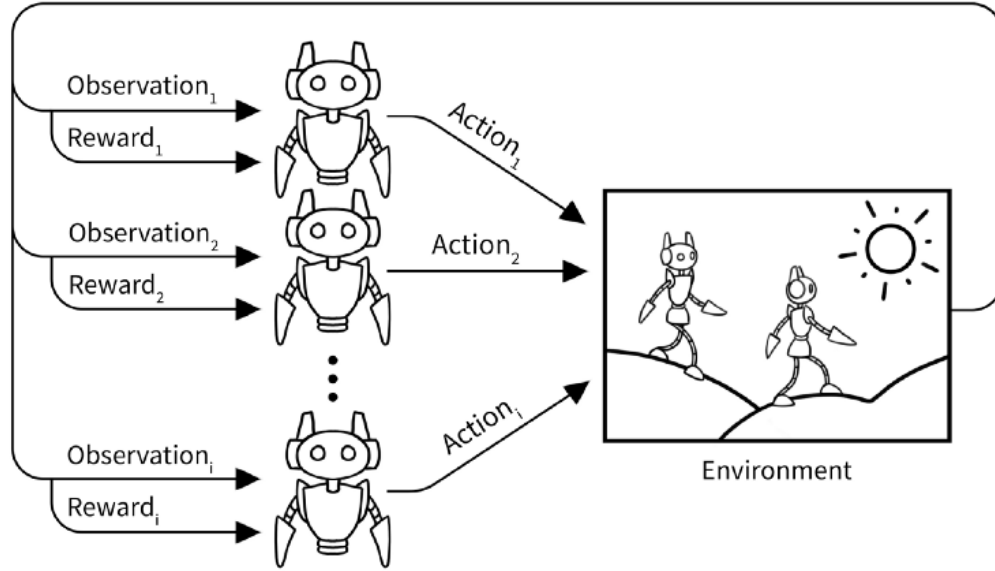


Figure 5 Multi-agent Reinforcement Learning Visualized. (Credit by Justin Terry, 2021)

To provide a faster training process, the agents were trained in 8 parallel environments that are running and calculating concurrently. The final approach uses the on-policy policy gradient RL algorithm. PPO is trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. In other words, PPO would use the observation space of the game to tell the agents what actions are better to make in the current step of the gameplay. The algorithm equation is shown in Figure 5.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

Figure 6 Proximal Policy Optimization Algorithm

- θ is the policy parameter
- \hat{E}_t denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively

- \hat{A}_t is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2.

This objective implements a way to do a Trust Region update which is compatible with Stochastic Gradient Descent and simplifies the algorithm by making adaptive updates. In practical tests, this algorithm should display one of the best performances on continuous control tasks, despite being simpler to implement.

Further, in the project, Stable Baselines 3 is used as it includes scalable, parallel implementations of the PPO algorithm, which uses Message Passing Interface (MPI) for data passing. Simply put, MPI is a standard designed to function in parallel computing. This project used Python3 and TensorFlow, which do support parallel processing.

2.4.3. Reinforcement Learning Components

The five main components of the Reinforcement Learning concepts are as follows:

- Agents: Players (Red and Green squares)
- Environment: the playing grid for the game that consists of the agents, the ball, and the areas with speed variations in the game.
- State: the current state of the environment and the agent. Since the agents are moving or passing the ball most of the time, the carrying state is changing, and so is the agent who is responsible for it.
- Action: the action determines the possible movements for the agents (Up, Down, Left, Right, and passing the ball). Certain actions will be limited when the agent is against edges or players in the game.
- Reward: under the current state, the agent would receive feedback by taking certain actions. The reward can be positive for representing a better move

or negative for getting punished (Coding details shown in Figure 7). For example, in the game, the agents receive relatively greater rewards when putting the ball into the goal position compared to getting the ball or passing it to the other agent. (Coding details are shown in Figure 8).

```
133     def step(self, actions):
134         """
135         step(action) takes in an action for the current agent (specified by
136         agent_selection) and needs to update
137         - rewards
138         - _cumulative_rewards (accumulating the rewards)
139         - done
140         - infos
141         - agent_selection (to the next agent)
142         And any internal state used by observe() or render()
143         """
144         # If a user passes in actions with no agents, then just return empty observations, etc.
145         if not actions:
146             self.agents = []
147             self.__tick()
148             return {}, {}, {}, {}
149         self.steps += 1
150         self.rewards = {agent: 0 for agent in self.agents}
151         for agent in actions:
152             self.__handle_action(agent, actions[agent])
153
154         self.__tick()
155
156         env_done = self.is_terminal()
157         done = {agent: env_done for agent in self.agents}
158
159         observations = {
160             agent: self.observe(agent) for agent in self.agents
161         }
162
163         infos = {agent: {} for agent in self.agents}
164
165         if env_done:
166             self.agents = []
167
168         return observations, self.rewards, done, infos
169
170     def render(self, mode='human'):
```

Figure 7 Components for Each Iteration in Reinforcement Learning Environment

```

309     def __ball_update(self):
310         for ball in self.balls:
311             if ball.carrier:
312                 score_dest = pygame.sprite.spritecollideany(ball, self.dests)
313                 if score_dest:
314                     self.score.value += 1
315                     # showing score in cmd
316                     print(self.score.value)
317
318                     for agent in self.rewards:
319                         self.rewards[agent] += 10
320                     if ball.ball_find_carrier(ball, self.player1):
321                         self.rewards[self.player1.name] += 1
322                         self.rewards[self.player2.name] -= 1
323                     if ball.ball_find_carrier(ball, self.player2):
324                         self.rewards[self.player2.name] += 1
325                         self.rewards[self.player1.name] -= 1
326                     if self.random_pos:
327                         self.ball_spawn = self.__generate_random_pos()
328                         self.dest_pos = self.__generate_random_pos()
329                     self.ball.respawn(self.ball_spawn)
330                     self.dest.respawn(self.dest_pos)
331             else:
332                 player = pygame.sprite.spritecollideany(ball, self.players)
333                 if player is not None:
334                     ball.carry_by(player)
335

```

Figure 8 Setting Rewards for the Agents

2.4.4. Game Play Design, Player Profile, Storyboard, and Prototype

Since the game is mainly for demonstrating the results for the agents trained, the targeted group are the ones that are fans of algorithms and machine learning who may also love thinking about the optimal play. While the original design includes more strategies, the current game is simplified to basic strategies. Capture the Ball's prototype is a variation of the "Capture-the-flag" cooperation game involving 2 players (in this case, the agents) to reach a common goal. Since the map has areas that allow certain players to move faster, the best strategies would be the ones that utilize the speed advantages of the players. As the game is designed as a prove-of-concept action game, it does not have a storyline at the moment. The core mechanic of the game involves collecting the ball, passing the ball, and gaining points as a team. The more points gathered during the same period, the better the AI.

2.4.5. Use Case Diagram

The engine doesn't interact with the player too much as it is a test of the agent's performance with a basic game. The detailed controls are in the Instruction and User Manuals section of the report. Some of the use cases include the following (Figure 9). The players can make moves around the world. When the players are close to the ball, one player will get to hold it and interact with the goal object. Once the ball is reaching the goal area, both players will gain 1 point. If one player is touching the other when holding the ball, they can pass the ball to the other one. The players can also exit the game.

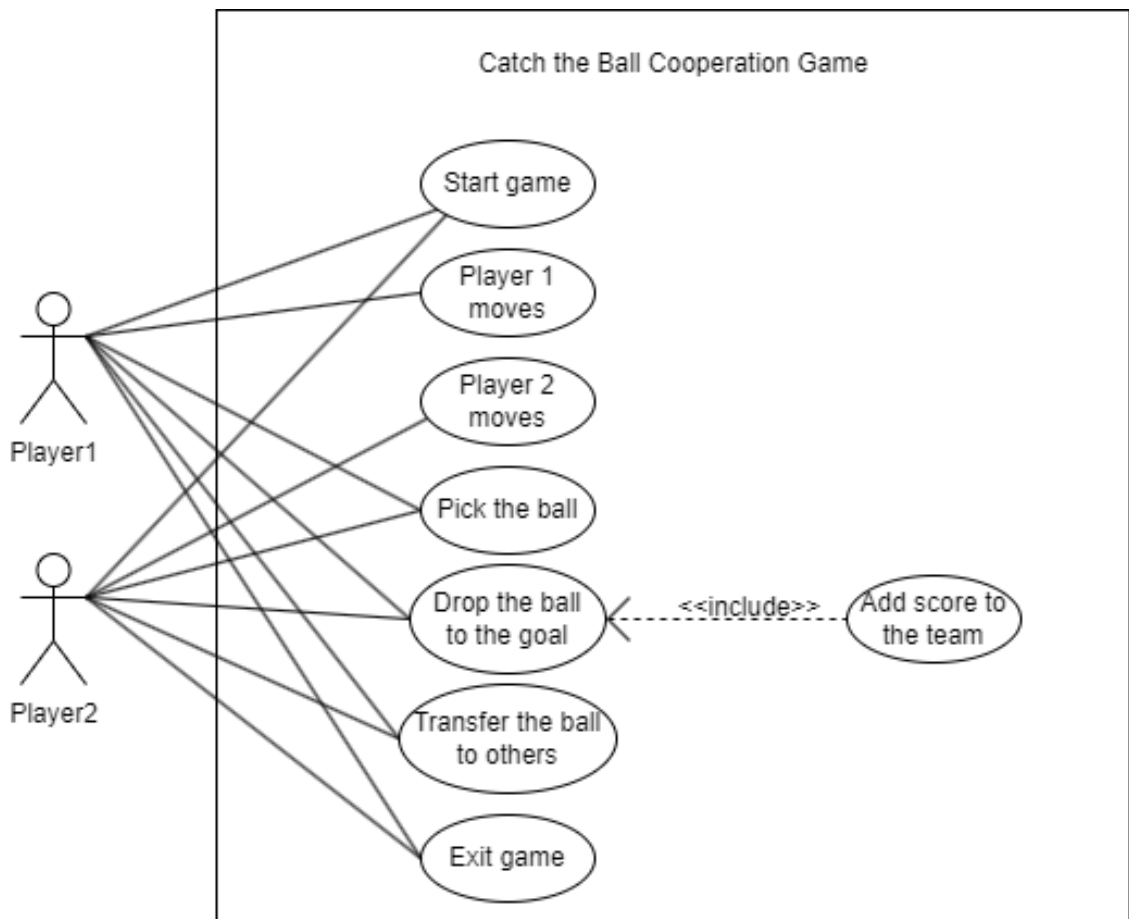


Figure 9 Use Case Diagram

2.4.6. Sequence Diagrams

Figure 10 shows the overall training process for the single and multiple agents in a frame of the game. The time it takes for a single frame to complete is dependent on the refresh rate of the computer it runs on. For the developer's computer, the game runs at 60 frames per second (FPS) normally. However, when training the agents with RL, the FPS could reach 291 yet still takes about 4 hours for finishing 1 million steps. Within each iteration, the agent will get the current state, predict the actions, and perform the actions. Note that the Learner is the Stable Baselines 3 library and the game is the customized environment.

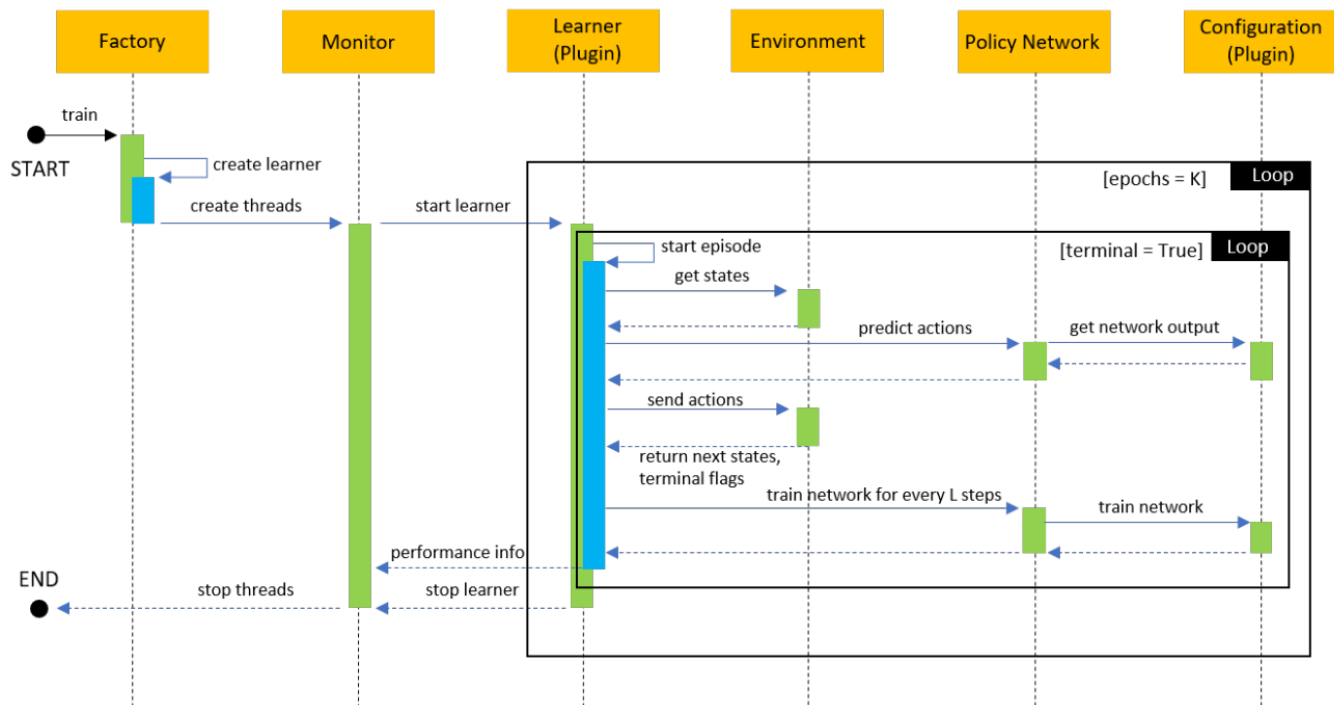


Figure 10: UML sequential diagram of the agent training process.

2.4.7. Class Diagram

Figure 11 is the class diagram for the PyGame 2D capture-the-ball cooperation game. The game is going to be implemented using a singleton. The game can be played by 2 players or 2 AIs, either random AIs or the 2 agents

trained with the RL algorithm. The environment of the game is customized for agent training and bridged to the base game. The sprites in the game, including the score (not rendered in the demo as it affects the performance), are rendered through `pygame.Rect` on the `pygame.Surface` shown on the game panel. The game is run with `ParallelEnv` which allowed 8 concurrent games during the training process.

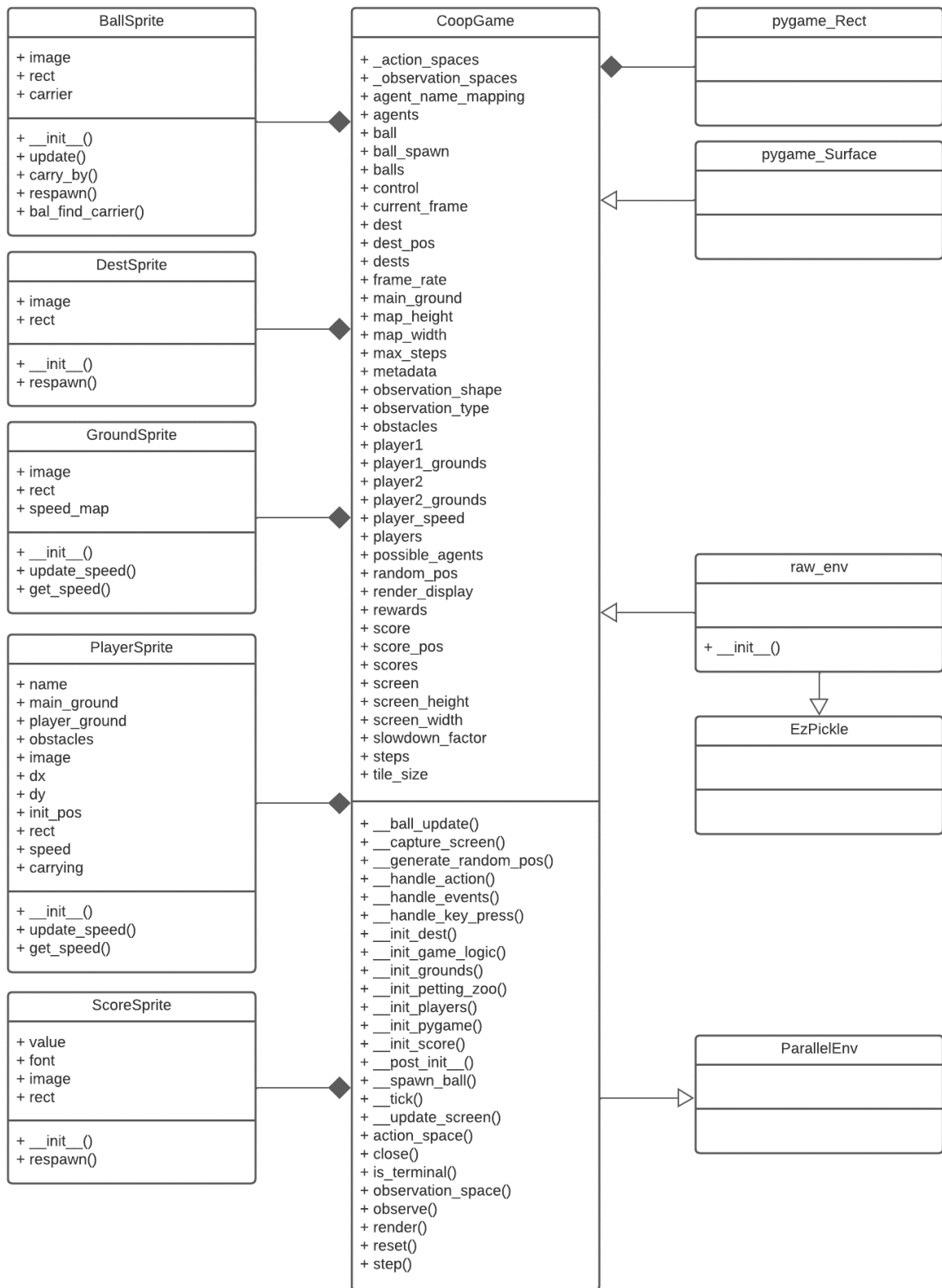


Figure 11: Class Diagram of the game with PyGame

2.4.8. Flow Chart

Figure 12 is the flow chart showing how training is done with PettingZoo in this project. Firstly, the environment of the game is initialized with default values. Then, after observing the observation_space, rewards, done, and agent_selection for x times (x is the number of agents) as previously explained, updating the values requires y times (y is the number of actions or steps required based on the changes of rewards). Finally, training each iteration for z times (z is the number of episodes) to complete the training process for multiple agents. Since the game is run in a parallel environment, multiple flow charts are followed in the training with different states.

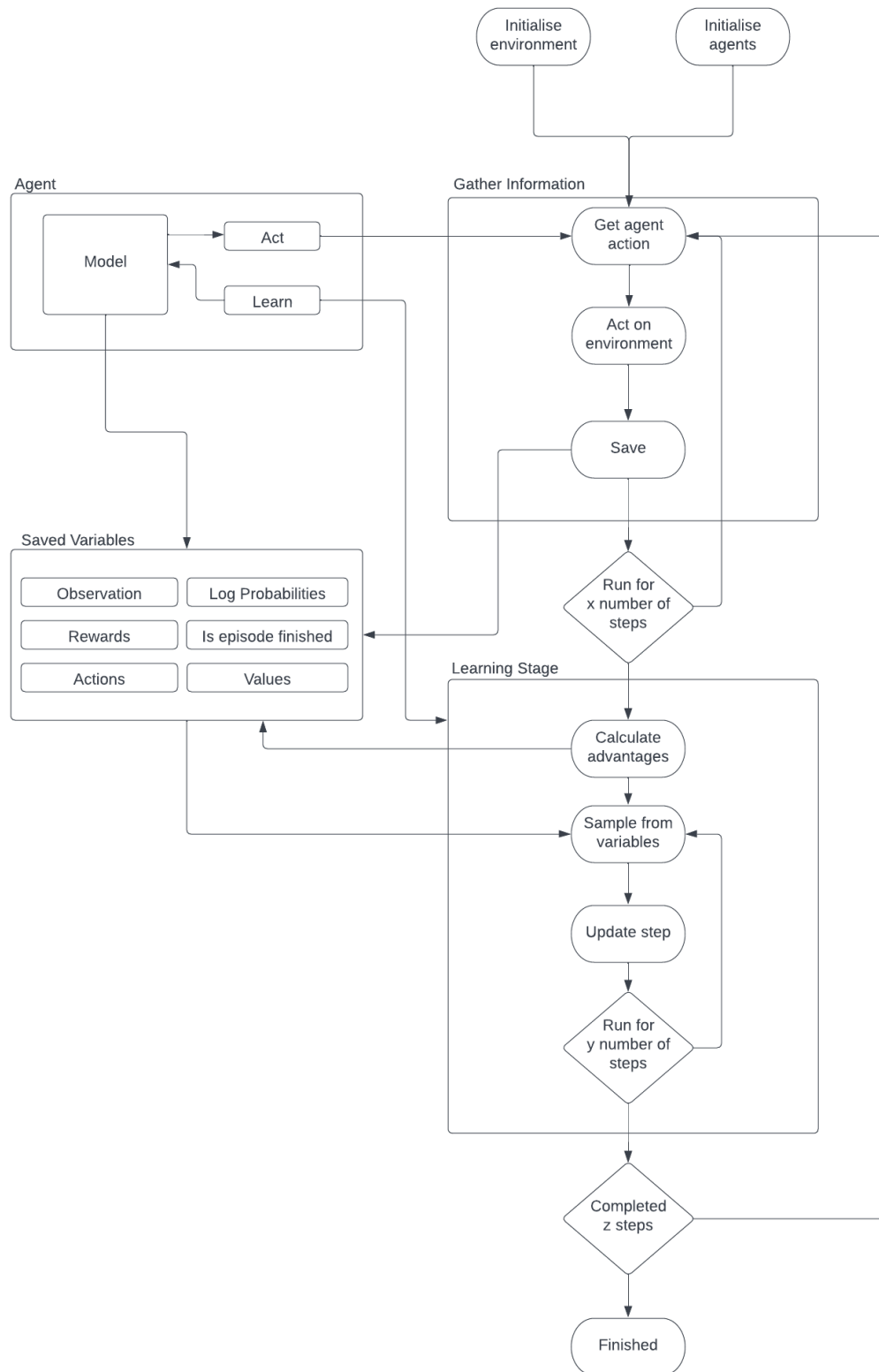


Figure 12 The Flow Chart

2.4.9. Installation and Instruction Manual

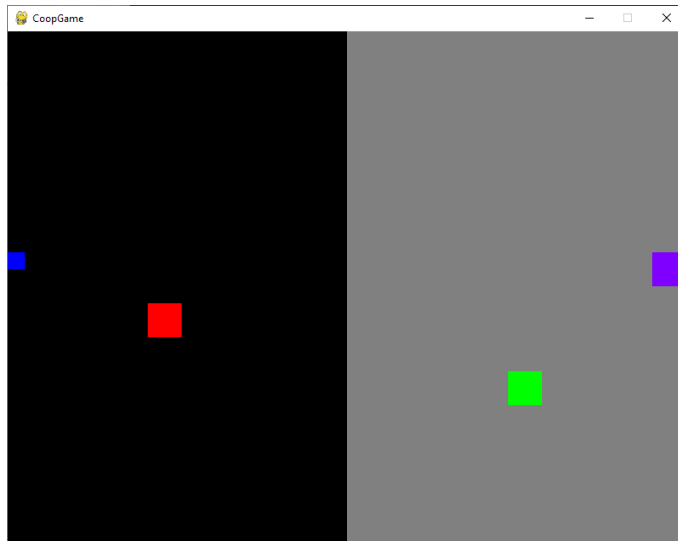
The game should run all by the executable itself with the manual mode and the demonstration version of the random AI. However, running the trained agents requires PyTorch, which does not currently support being packaged as executables. Therefore, before running the rf.py file, a couple of dependencies shall be installed. First, the project is required to be run in Python 3.7.13. Secondly, PyTorch, as well as Cuda, is needed if not already had on the system. Even though the project is built using the package management system Anaconda, the project can run without it. Simply go to the [link](#) and check the “Pip” option in the Package area would also work. The image below is the guideline for the installation.

PyTorch Build	Stable (1.12.1)		Preview (Nightly)		LTS (1.8.2)
Your OS	Linux		Mac		Windows
Package	Conda	Pip	LibTorch		Source
Language	Python			C++ / Java	
Compute Platform	CUDA 10.2	CUDA 11.3	CUDA 11.6	ROCm 5.1.1	CPU
Run this Command:	pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu113				

- pip install pygame
- pip install NumPy
- pip install gym==0.21.0
- pip install stable-baselines3[extra]
- pip install supersuit==3.3.3
- pip install pettingzoo==1.17.0

After installing the said processes, run the python file with the command “python rf.py enjoy”.

This should allow the program to run the game showing 2 agents playing the game with their scores printed on the command line.



The elements for the game are:

- Blue square: the ball that can be collected.
- Purple square: the goal.
- Red and green squares: player one and player two.

The black and gray areas of the game provide different speeds to different players. For example, the red player has a normal speed in the black area but its speed becomes one-third of its original speed when the red player is moving in the gray area. The vice versa applies to the green player while its speed becomes one-third of its normal speed in the black area.

Moving the players to the ball area will automatically pick up the ball. The ball can be automatically passed to another player by moving in the direction of

other players (if applicable). When the agent puts the ball to the goal (purple area), the team score 1 point.

The actions available are the following.

- A – Move left for the red player.
- Left arrow key – Move left for the green player.
- D – Move right for the red player.
- Right arrow key – Move right for the green player.
- W – Move up for the red player.
- Up arrow key – Move up for the green player.
- S – Move down for the red player.
- Down arrow key – Move down for the green player.
- Exiting the game: Click the close window button on the top right of the application with the mouse.

The command-line prompt after running the game should look something like this:

```
(Testing) C:\Users\mahou\Desktop\Multi-agentCollaborativeGame>python test_env.py
pygame 2.1.2 (SDL 2.0.18, Python 3.7.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
1
2
3
```

Optional: For applying the training process, enter `python rf.py train`

- For running and testing the model before it is fully trained, save points can be applied as the following.
 - `python rf.py train -i .\models\v0\ppo\rl_model_100000_steps`
 - Note that “rl_model_100000_steps” is the name of the .rar file.
- For retraining the model with the already trained model, the feature can be applied after it is fully completed, as the following.
 - `python rf.py train -i .\models\v0\ppo\rl_model_final -l`

2.5. Testing Details and Results

For this project, I will have unit tests for both the game and the agent-training functions. They will test each component of the code in isolation, which means it can be limited when trying to catch bugs in some edge cases where they only occur when the game runs as a unit. However, these unit tests will allow me to have tests running constantly everything when the code is committed. Since testing the reinforcement learning algorithm is difficult, other methods are implemented for the evaluation process. This includes manually testing with different values of parameters while training the agents. The overall test process is shown in Figure 13 [6].

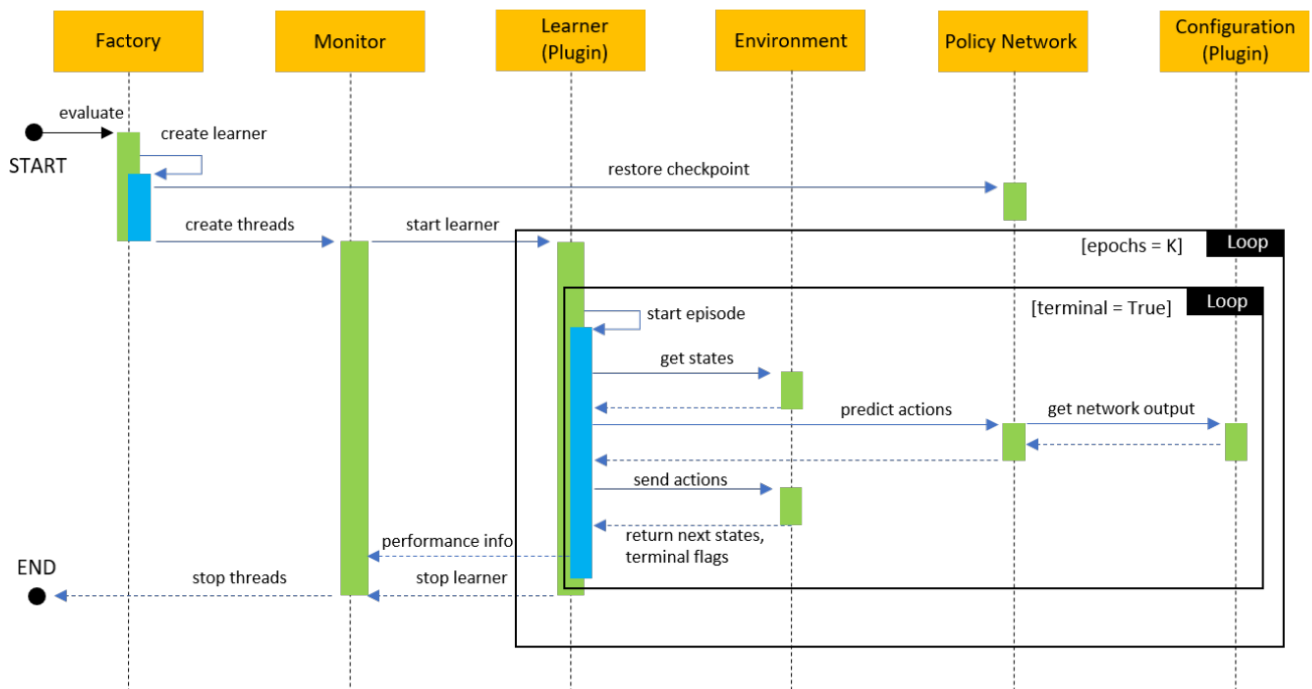


Figure 13: UML sequential diagram of the evaluation process.

For testing the PPO algorithm, the results of the collected episodes will be verified and saved as a model zip file for evaluating the results. The pass criteria would be

determined by how well the performances of the collected data. In the case of the PPO algorithm, it is the observation saved as an image. If the overall performance is with very badly performed data where the agents are not performing actions that maximize the rewards, it is considered a failure.

2.5.1. Unit Testings

The unit test framework will be used to test the Pygame part. It is important to ensure that the code written passes all the Unit Tests. This will particularly reduce the number of bugs introduced during game development. The Unit Tests will be run again after each sprint to ensure new changes are working as expected. Figure 14 is the test results for the unit tests (all passed). Details are shown below.

```
(majorproject) C:\Users\mahou\Desktop\Multi-agentCollaborativ
eGame>python -m unittest -v test_test_env.py
pygame 2.1.0 (SDL 2.0.16, Python 3.7.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
test_BallCarriedBy (test_test_env.TestMain) ... ok
test_BallSprite (test_test_env.TestMain) ... ok
test_DestSprite (test_test_env.TestMain) ... ok
test_GroundSprite (test_test_env.TestMain) ... ok
test_PlayerAction (test_test_env.TestMain) ... ok
test_PlayerCollision (test_test_env.TestMain) ... ok
test_PlayerSprite (test_test_env.TestMain) ... ok
test_ScoreSprite (test_test_env.TestMain) ... ok
test_ScoreUpdate (test_test_env.TestMain) ... ok
test_UpdateSpeed (test_test_env.TestMain) ... ok

-----
-----
Ran 10 tests in 0.005s

OK
```

Figure 14 Unit Test Results

The unit tests for testing the game include as follows. The ground sprites in the game are rendered with the expected position and initialization settings are assumed to be correct.

```
class TestMain(unittest.TestCase):
    # Test the function env.py
    def test_GroundSprite(self):
        ground = GroundSprite(pygame.Color(0, 0, 0),
                               (0, 0, 20, 20))
        self.assertEqual(ground.rect, (0, 0, 20, 20))
```

The ground sprites in the game can apply the speed changes to the players. Both the initialization and the updates are assured to be correct.

```
def test_UpdateSpeed(self):
    ground = GroundSprite(pygame.Color(0, 0, 0),
                          (0, 0, 20, 20))

    obs = pygame.sprite.Group()
    player = PlayerSprite("1",pygame.Color(0, 0, 0), (0, 0), 10, 10, ground, obs, 6)
    players = pygame.sprite.Group()

    ground.update(player.speed)
    players.add(player)
    self.assertEqual(ground.speed_map.__contains__(10), False)
```

The Player sprites in the game are rendered with the expected position and multiple initializations are assumed to be correct.

```
def test_PlayerSprite(self):
    obs = pygame.sprite.Group()
    ground = GroundSprite(pygame.Color(0, 0, 0),
                          (0, 0, 20, 20))
    player = PlayerSprite("1",pygame.Color(0, 0, 0), (0, 0), 0, 0, ground, obs, 5)
    player2 = PlayerSprite("2",pygame.Color(255, 255, 255), (200, 200), 0, 0, ground, obs, 5)
    players = pygame.sprite.Group()
    players.add(player)
    players.add(player2)
    self.assertEqual(len(players), 2)
    self.assertEqual(player.rect, (0, 0, 0, 0))
```

The available actions of the Player inputs such as moving left, right, up, and down in the game are changed with the positions expected. The initializations are also assured to be correct.

```
# expected move to certain positions...
def test_PlayerAction(self):
    obs = pygame.sprite.Group()
    ground = GroundSprite(pygame.Color(0, 0, 0),
                           (0, 0, 20, 20))
    player = PlayerSprite("1", pygame.Color(0, 0, 0), (0, 0), 1, 0, ground, obs, 5)
    rect = player.image.get_rect()
    player.step(0)
    players = pygame.sprite.Group()
    players.add(player)
    self.assertEqual(rect, (0, 0, 1, 0))
```

The players can collide with other players correctly with the test of the function PlayerCollision.

```
def test_PlayerCollision(self):
    obs = pygame.sprite.Group()
    ground = GroundSprite(pygame.Color(0, 0, 0),
                           (0, 0, 20, 20))
    player = PlayerSprite("1", pygame.Color(0, 0, 0), (0, 0), 3, 1, ground, obs, 10)
    player2 = PlayerSprite("2", pygame.Color(0, 0, 0), (2, 2), 3, 1, ground, obs, 10)
    player3 = PlayerSprite("3", pygame.Color(0, 0, 0), (0, 0), 3, 1, ground, obs, 10)

    collision_with_player2 = player.obstacle_collide(player, player2)
    collision_with_player3 = player.obstacle_collide(player, player3)
    # Only player 3 is at the same location. Collision should occur
    self.assertEqual(collision_with_player2, False)
    self.assertEqual(collision_with_player3, True)
```

The destination and the ball sprites in the game are rendered with the positions expected. The initializations are also assured to be correct.

```
def test_DestSprite(self):
    # color, pos, width, height
    dest = DestSprite(pygame.Color(0, 0, 0),
                      (0, 0), 20, 20)
    self.assertEqual(dest.rect, (0, 0, 20, 20))
```

```
def test_BallSprite(self):
    obs = pygame.sprite.Group()
    # color, pos, width, height
    ball = BallSprite(pygame.Color(0, 0, 0),
                      (0, 0), 20, 20)
    self.assertEqual(ball.rect, (0, 0, 20, 20))
```

The balls can be carried by players with the test of the BallCarriedBy function. The function also updates the state of who the carrier is successful.

```
def test_BallCarriedBy(self):
    obs = pygame.sprite.Group()
    ground = GroundSprite(pygame.Color(0, 0, 0),
                          (0, 0, 20, 20))
    ball = BallSprite(pygame.Color(0, 0, 0),
                     (0, 0), 20, 20)
    player = PlayerSprite("1",pygame.Color(0, 0, 0), (0, 0), 0, 0, ground, obs, 5)
    player2 = PlayerSprite("2",pygame.Color(0, 0, 0), (0, 0), 0, 0, ground, obs, 10)
    # First carried by player 2, then 1
    ball.carry_by(player2)
    self.assertEqual(ball.carrier, player2)
    ball.carry_by(player)
    self.assertEqual(ball.carrier, player)
```

The score sprites and the values after initializations are assumed to be correct. The score counter is updated and working as expected when the player or agent passes the ball to the goal.

```
def test_ScoreSprite(self):
    pygame.font.init()
    obs = pygame.sprite.Group()
    # pos
    score = ScoreSprite((0, 0))
    obs.add(score)
    self.assertEqual(score.value, 0)
```

```
def test_ScoreUpdate(self):
    pygame.font.init()
    # update score to 2
    score = ScoreSprite((0, 0))
    score.value = 2
    score.update()
    self.assertEqual(score.value, 2)
```

2.5.2. Manual Testings

The parallel environment and the performance are assured to work as expected with the help of the parallel_api_test(env) function. This function also

made sure the requirements needed for Stable Baselines 3 and the iterations in the game (each step) are provided with the correct inputs.

```
env = parallel_env(**env_kwargs)
parallel_api_test(env)
env = aec_env(**env_kwargs)
```

The performance of the game is tested for checking whether the performance of the “Stable baselines 3” library is taking longer time than expected in each step. A for loop that takes a longer time, affecting the overall performance was noticed and modified with the help of the `performance_benchmark(env)` function.

```
env = aec_env(**env_kwargs)
performance_benchmark(env)
env.reset()
```

The observations of the game are assured to be saved as a screenshot with the correct format and settings. The function `test_save_obs(env)` is used for checking whether the observations are correctly saved with the format that can be accepted while feeding the inputs to the game steps.

The `random_demo(env)` function is responsible for testing the visual of the random AI’s actions in the game. The inputs are assured to be expected as the inputs for the AI are random with the correct type (integer).

```
env.reset()
test_save_obs(env)
random_demo(env)
```

The training process is also assured with the expected outcomes listed as the following. The variables listed are assured to be expected as the inputs for the agents. Other variables like `fps`, `iterations`, `time_elapsed`, and `total_timesteps` are also assured to be recorded correctly in the process. The details are as follows (Figure 15).

time/	
fps	261
iterations	4455
time_elapsed	34868
total_timesteps	9123840
train/	
approx_kl	4.5441266e-05
clip_fraction	0.135
clip_range	0.00878
entropy_loss	-1.57
explained_variance	-1.95
learning_rate	2.2e-05
loss	-0.0178
n_updates	17816
policy_gradient_loss	-0.00146
value_loss	1.19e-09

Figure 15 RL Variables Recorded During the Training Process with PPO Algorithm

```

parser = argparse.ArgumentParser()
parser.add_argument("action", choices=["train", "enjoy"])
parser.add_argument("-i", dest="model_name", type=str, default="models/v0/ppo/rl_model_final")
parser.add_argument("-l", dest="inherit", action="store_true", default=False)

args = parser.parse_args()

if args.action == "train":
    train(args.model_name, args.inherit)
else:
    enjoy(args.model_name)

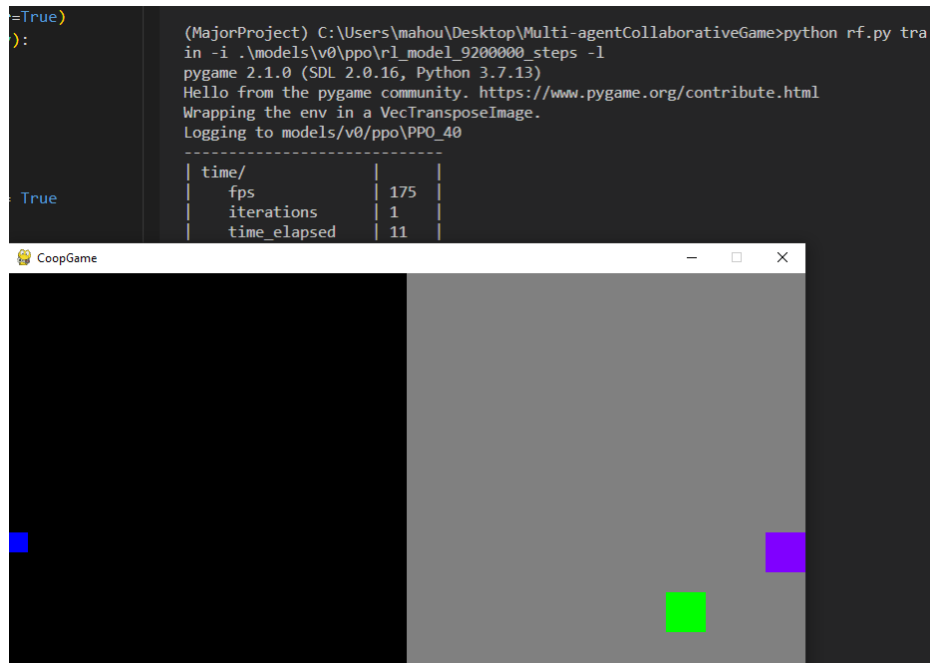
```

This is tested with the following commands:

- Training the model with the default parameters set in file rf.py.



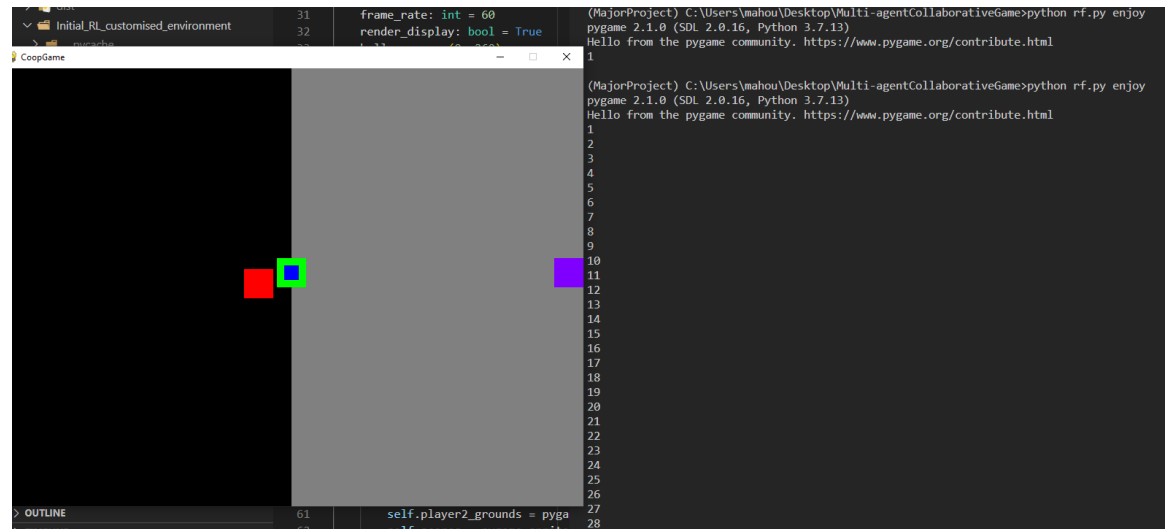
- Training and saving to the zip file named “rl_model_9200000_steps”.



- Loading with the model with 9.2 million steps as the save point.



- Loading the completely trained model

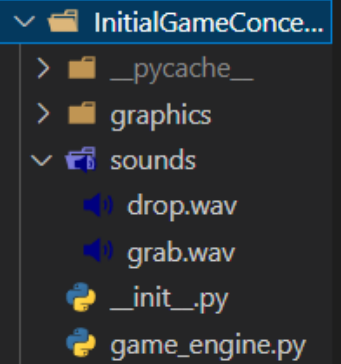
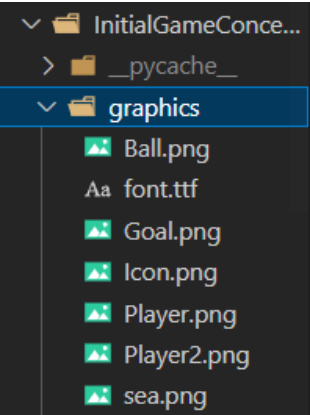


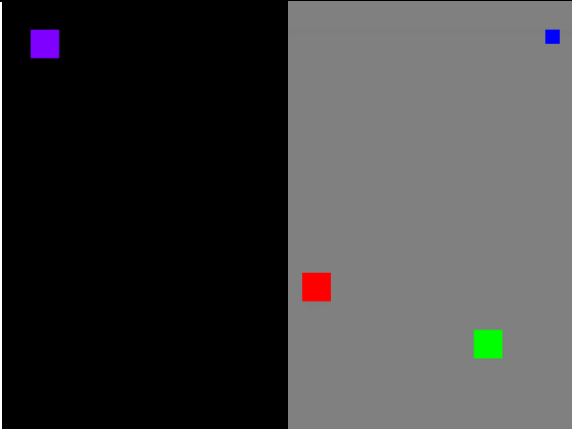
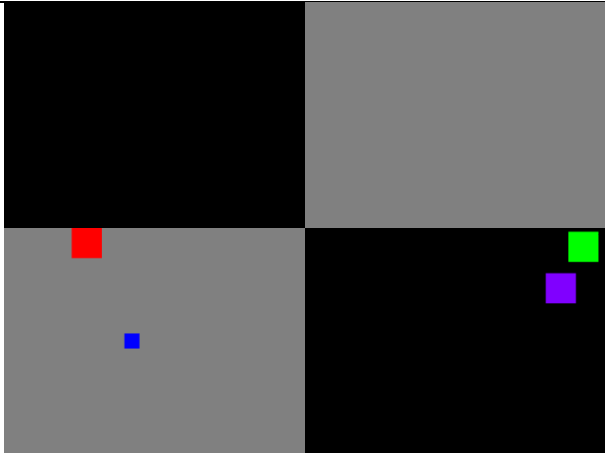
2.5.3. Testing Results

This section contains a collection of tests I for expected results, their priorities, and the overall test results. This section intends to cover general interactions available within the game to ensure it has been tested, with both unit testings and manual testings. Please note that because of the repetitive sounds playing in the background while training the AI as well as its non-essential for the completion of the project, the audio in the game is taken off. This includes the audio for moving, picking the ball, dropping the ball, passing the ball to the other agent, getting the score, and the background music. The textures and images attached to the game objects are also dropped because they affect the results of the PPO algorithm, making the agents perform slightly less effective after the same amount of training (14 hours for 10-million-step training explained in the Time e Efficiency section). The functional requirements and the test results are shown below.

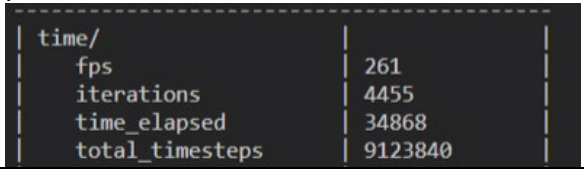
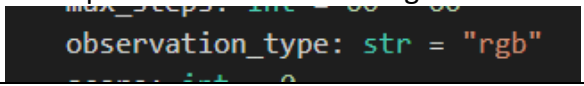

2.5.4. Tests for Game

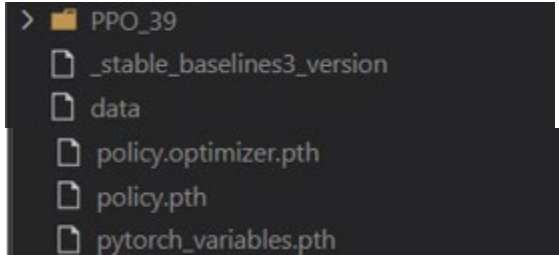

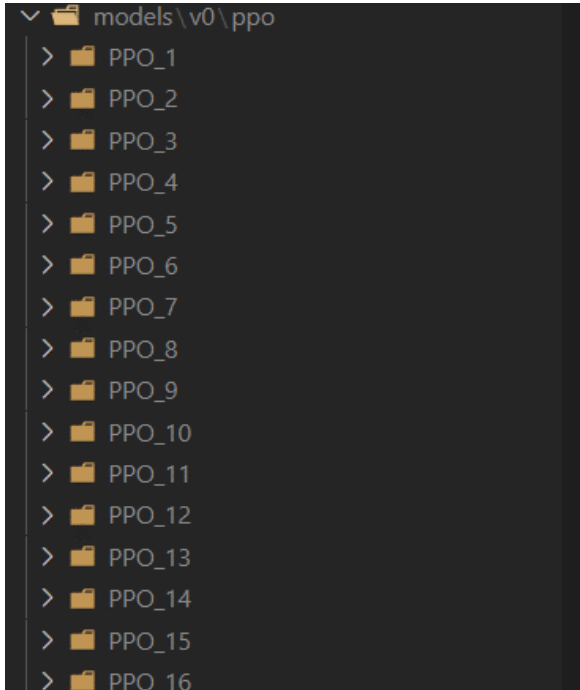
Description	Components Tested	Priority	Expected Result	Status
Transform of game objects	Pygame Sprites	Essential	All position, scale, and offset functionality are updated correctly upon collision.	Passed
Character controller	Player key inputs	Essential	Character movement responding to the user input. No out-of-bounds actions should happen. <pre>def render(self, mode='human'): if self.render_display: pygame.display.update() if self.frame_rate != 0: pygame.time.Clock().tick(self.frame_rate)</pre>	Passed
UI manager	Types of the Sprites	Revised, Essential	All UI elements stay in fixed positions. (The score is removed because it affects the agent's behaviours as a sprite)	Passed

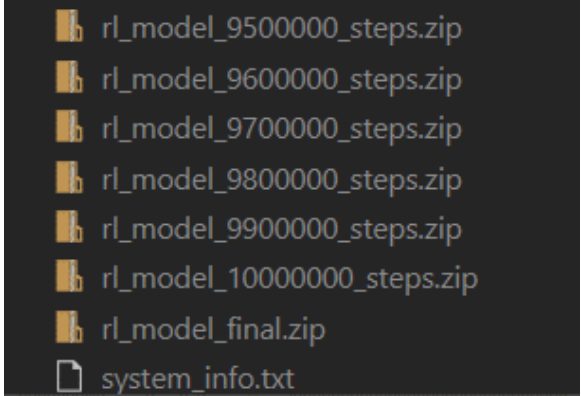
	test_env.py test file		<pre>performance_benchmark(env) env.reset() test_save_obs(env) random_demo(env)</pre>	
Audios and SFX	Pygame sounds	Non-essential	<p>The sound plays as expected. Dropped due to unnecessarities while training.</p> 	Dropped
Texture manager	Resources	Revised, Non-essential	<p>Game sprites for the objects in the game are loaded and handled correctly. However, this is dropped due to its negative effects (making the agents perform worse than expected) while training the agents with the PPO algorithm.</p> 	Dropped

Random spawning positions	Training Process with PPO	Non-essential	 <p>Although implemented, the random position of the ball and the goal position leads to poor performance (worse than random AIs) after several 14-hour-trainings. Since the algorithm, as well as the data fed, are expected further development requires different approaches to Algorithms. Since this requires changes to many aspects of the project (including the algorithms and data fed) and is not required in the initial draft of the project, the feature is considered out of scope.</p>	Dropped
Advanced maps	Training Process with PPO	Non-essential	 <p>Although implemented, the advanced map leads to poor performance (worse than random AIs) after several 14-hour-trainings. Same as the previous test: since this requires changes to many aspects of the project and is not required in the initial draft of the project, the feature is considered out of scope.</p>	Dropped

2.5.5. Tests for Reinforcement Learning

Description	Components Tested	Priority	Expected Result	Status
The rewards system	Inputs and outputs for rewards	Essential	All rewards should be calculated as the datatype provided and updated correctly upon change of state.	Passed
States of the current observation	Types of data in the observation space	Essential	All position, speed, and offset functionality are updated correctly upon collision.	Passed
Actions	Available moves from the agents (as 0, 1, 2, or 3)	Revised, Essential	All agent's movements should be updated correctly. The position of the objects should be at the desired places.	Passed
Is the agent finished (Done)	Types of the key inputs	Essential	The information is needed for passing to PettingZoo. When the agent is done, it should be labelled.	Passed
Performance	Frames rendered	Essential	The training process of the game should perform with at least 200 FPS.  <pre> time/ fps 261 iterations 4455 time_elapsed 34868 total_timesteps 9123840 </pre>	Passed
PPO Input data sanitize	Key inputs from p	Essential	The Inputs are saved as an image.  <pre> observation_type: str = "rgb" </pre>	Passed
Training the agents with PPO	The input image taken	Essential	The image is saved as expected.  <p>\\Multi-agentCollaborativeGame\\saved_observations\\coop_pa</p> <p>green_player.png red_player.png</p>	Passed

Saving the trained model	Resources (zip files in the folder)	Essential	<p>Game states in the game are saved with the correct training steps and the formats of the data are handled correctly (this is formatted as data and policies).</p> 	Passed
Reading the model	Resources (zip files in the folder)	Essential	<p>Game states in the game are loaded and the formats of the data are handled correctly.</p>  	Passed

Inherently training the model from saved points	Resources (zip files in the folder)	Essential	<p>Game states in the game that are previously trained can be loaded and performs as expected. The formats of the data are handled correctly.</p> <pre>def train(model_name, inherit, save_freq=100000): env = load_env(train=True) if inherit is True: model = PPO.load(model_name) model.set_env(env)</pre> <pre>parser.add_argument("-l", dest="inherit", action="store_true", default=False)</pre> 	Passed
Loading parallel environments concurrently	ParallelEnv's inputs and outputs	Essential	<p>The game is trained with 8 parallel environments without major performance flaws. The inputs and outputs were as expected in the concurrent environments.</p> <pre>def linear_schedule(initial_value: float): def func(progress_remaining: float) -> float: return progress_remaining * initial_value return func</pre>	Passed

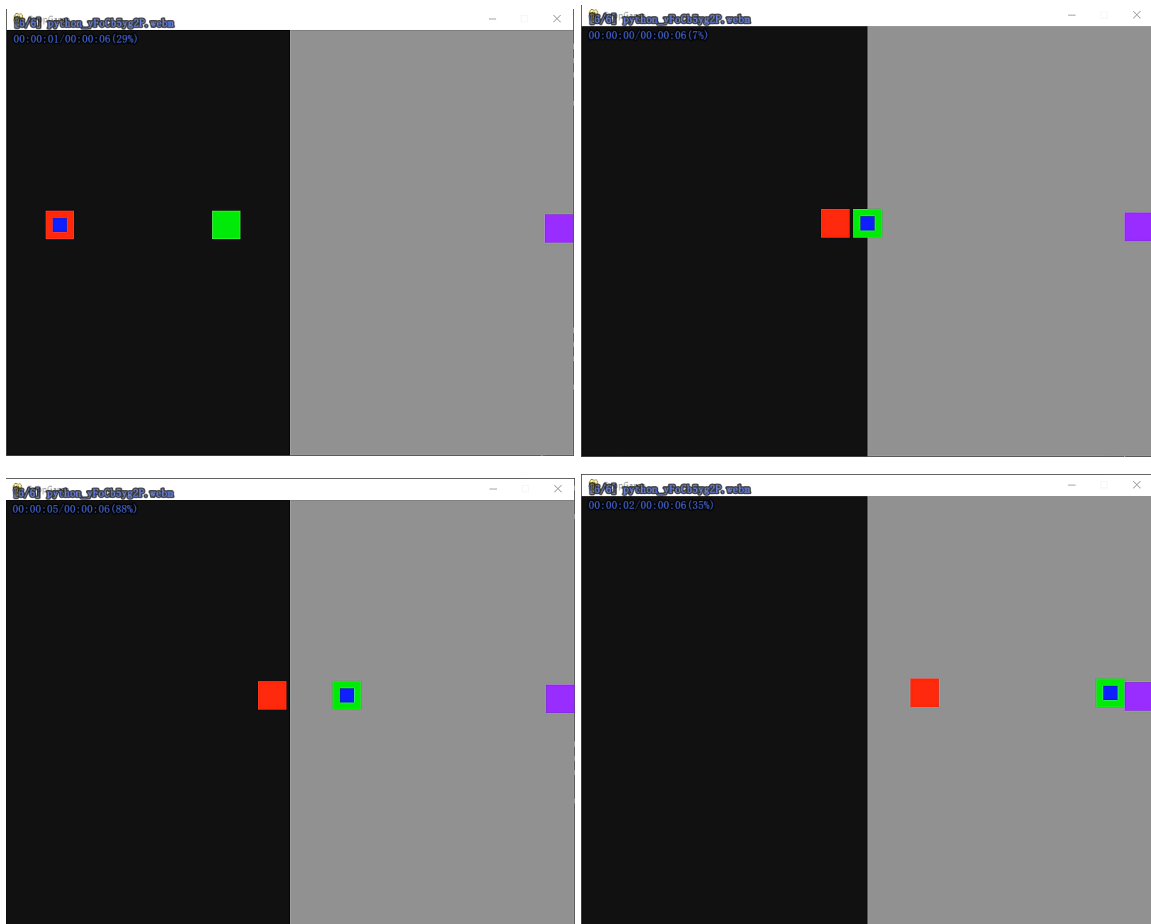
2.5.6. Implications of Implementation

Two prove-of-concept demos and a game were developed to help evaluate how effectively the agents cooperate in the game. Since the game does not require a player and an agent, the actual game with full control for both players is suitable for the deliverable. The first demo includes an executable showing the 2 random AIs performing

in the game. It was designed with the sole intent of comparison as the randomly activated AIs are the baseline of the project.

The second demo shows how cooperative the 2 agents are with RL. The game state changes as each agent interact with the game and, in response, the PPO algorithm reevaluates how it intends to achieve its goals and perform any actions to help achieve them. As a result, the agents move with the optimal route in the center of the screen and passed the ball effectively (Figure 13). There is a huge difference when compared to random AIs which would take a long period to even get the ball (Figure 14).

For a better training experience with the time-consuming process, an inheritance system is made for saving the models that have been trained for hours. A save point feature is also made for testing the half-trained agents while the computer is still training the agents on the go. Those features have been time savers despite the time for training in general being over 120 hours. In the demo, the agents are done training for more than 10 million steps with the PPO RL algorithm. Even though they wander around in rare cases, both agents apply better strategies in the game compared to the baseline. Note that they could gain scores faster than humans as a human may hesitate at times while playing the game. Also, the game demo is provided for playing and testing the strategies available in a controlled manner.



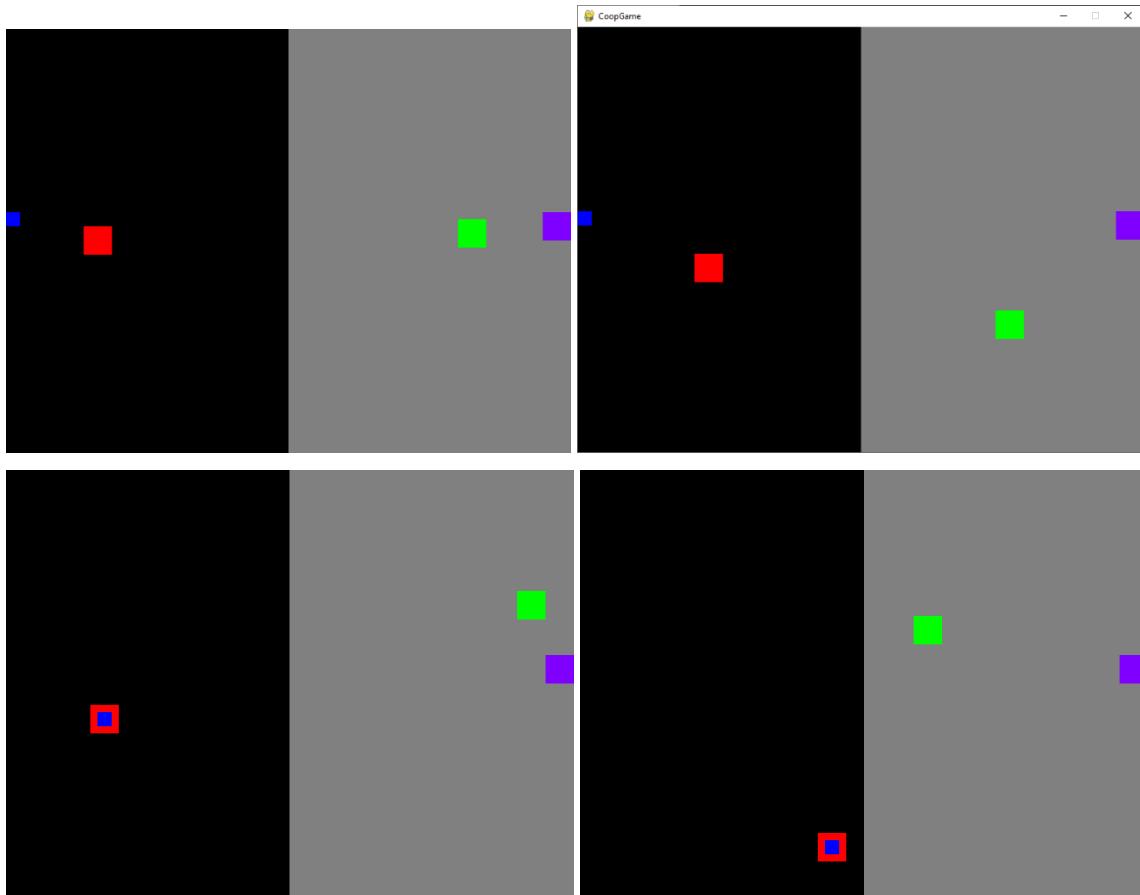


Figure 17 The Series of Actions Performed by Random AI. (Ordered as Top Left, Top Right, Bottom Left, and Bottom Right).

2.5.7. Innovation

The major innovation component of this project is that the gaming industry is lacking the collaborative experience with other agents that are constantly learning to help the players instead of fighting against the players. Even though the idea of using reinforcement learning is not new, it is yet explored as deeply as other machine learning techniques due to the high complexity and underwhelming sample inefficacy when it comes to integrating Q-learning Algorithms in a multi-agent environment.

A similar project has been completed by Google's research lab DeepMind in 2019. DeepMind used another advanced approach – the population-based deep reinforcement learning approach (In other words, the Deep Q-learning algorithm, which

is similar to Q-learning but replaced experience storage in Q-table with the neural network), and demonstrated with the first-person 3D game “Quake 3 Arena” [3]. However, this leading-edge technology has yet to be implemented by other gaming companies to further explore the possibilities of what collaborative multi-agent AI can do. It is innovative by exploiting the natural curriculum provided by multi-agent training and forcing the development of robust agents that can team up with humans.

2.6. Complexity

The major component that makes this project complex is the difficult nature for Q-learning to act and evaluate not in a single agent environment, where the variants are limited for demo purposes (under real-world scenarios where the agents themselves and other players perform unknown actions). This is a setting we call multi-agent learning: many individual agents must act independently and learn to interact and cooperate with other agents. This is an immensely difficult problem. In every frame in the game where agents react and adapt to each other, the agents are updating the weight and the value of their next possible move, but the number of the calculations will increase exponentially as the number of agents increases.

Some of the problems with the Q-learning algorithm include:

- Credit Assignment Problem: the problem occurs because the agent only gets a reward after a series of actions. If the agent performed well in the first few steps but performed a move that lost the game, the good performance taken by the agent is not trained as excellent moves. This could be fixed with a “sparse reward setting”, which gives the agent rewards with smaller sets of actions instead of the entire sequence.
- Sample-inefficient with multi-agent reinforcement learning: Since the agents are responsible for the evaluation of what actions were causing the rewards, numerous training times would be required for the agents to learn. This problem worsens when collaborations between the agents are needed.

While other more advanced and complex algorithms (Deep Q-learning, Actor-Critic, etc. [7]) can be used to solve the above issues, they are deemed to be exploratory and out of the scope based on my current knowledge.

2.7. Research in New Technologies

The capture-the-ball cooperation game was originally developed through the Unity engine, rendered in 2D with 2D physics. However, through the change of structure, the project is remade using Python's PyGame library in terms of the game engine. The language used in this project involves originally C#, then python. This project uses the open-source Python library Gym and PettingZoo for making agent training possible as they are the most advanced library for developing and comparing RL algorithms. For developing the algorithm, TensorFlow (an open-source software library for machine learning and artificial intelligence) was used. Even though the algorithm did not work as expected, it helped the developer to go in the right direction, choosing Stable Baselines3 (SB3) as the set of reliable implementations of RL algorithms in PyTorch (An open source machine learning framework for making complicated algorithms possible). Machine Learning Agent Toolkit (ML-Agents) from Unity and the TF-agent from google was initially used for providing a training environment for the agents. With the structure shift as well as the limit ML-Agents has, SB3 made the project possible for training with multiple agents running in the same customized environment.

Since the project requires very limited and precise environment settings for making the training process possible, a package management system "Anaconda" is used as it better serves the purpose of switching python environments and other needed libraries free of conflict. Also, to make sure the input for agent-training processes remains correctly without crashing, the "SuperSuit" wrapper for Gym and PettingZoo helped wrap RL environments to do preprocessing ('microwrappers'), thus making the PPO algorithm applied properly. For version control, Git is used as this project's version control system. All code for this project is hosted on GitHub.

For the Integrated Development Environment (IDE), the project was originally planned and tested on Unity, together with visual Studio 2019. This is later changed to Visual Studio Code as it provides features including integrated debugging support, code analysis, and unit testing support.

As for Designing and planning Tools, Free web apps like Lucidchart.com and draw.io are used to create the project's software architecture diagrams. Trello, Coda, and Notion are used to maintain backlogs and task planning during each sprint.

2.7.1. Research used in Project

The following is the list of technologies that were considered or used in the development of the Collaborative AI with RL project.

Game engine and implementation language: Unity was considered for the development of the project in the beginning months. However, with the high difficulty of setting the reward algorithm to multiple agents in each "step" iteration with Unity's build-in reinforcement learning kit, the project was then modified to a conceptual game made in PyGame due to Python's better support with the RL algorithms existed. With powerful libraries like PettingZoo and Gym that only supports Python, the developer decided to shift to one of the best languages for modifying machine learning customizations with Python. This also provided more freedom for changing the type of information needed to be fed to the agents.

Figuring out what algorithms make the agents perform as expected was one of the biggest technical challenges in this project due to the complex and advanced mathematics knowledge involved. This leads to more time spent on researching what algorithms are a better fit for the project to show the cooperation clearly while also being simple to be trained. With the research of corresponding policies maximizing the rewards for ensuring the personalities of agents' actions, it became more challenging in a world that contains multiple agents. The algorithms are not only difficult to be implemented in a multi-agent

environment but also prone to sample inefficiency and may cause optimization issues in the future. While the developer has some background on the fundamental mathematical concepts of reinforcement learning from various courses through BTech's AI course, gaining a full understanding of the conceptual working of various algorithms in the training process proved to be extremely difficult. To effectively make adjustments to the parameters in PPO (Proximal Policy Optimization) algorithm when training and testing multiple agents, the algorithm itself should be comprehensively understood. Even though an unexpectedly large amount of time was spent on learning both the underlying concepts behind PPO, SARSA (State-action-reward-state-action), and A3C (Asynchronous Advantage Actor Critic), the developer still resulted in a high-level knowledge of the subject. However, with more time and effort, the said algorithms are capable of handling the cooperative actions for the agents with correct data as the states. This requires some datatype changes for feeding the inputs and outputs to a recursive neural network or to fit other algorithms' corresponding needs. Unfortunately, due to allocated time constraints, the idea was cancelled.

To ensure the success of the project, the developer ultimately put the research on the subject on hold once enough structure of the game is made. The attempt of making a customized PPO algorithm with Pytorch was a failure after 20 hours of researching and designing. The developer then dropped the plan and decided to use the algorithm library that is made and supported in Python. Stable Baselines 3 is one of the helping hands that contains the implementation of the PPO algorithm. Even though the attempt was difficult and a full understanding of the concepts was not time efficient, understanding the inputs and outputs as well as applying the concepts was helpful and needed for the success of the training process.

2.8. Technical Challenges

There are two major categories of challenges associated with this project: Learning Curve and Training the Agents. The issues include:

2.8.1. The Learning Curve

A learning curve is expected due to the lack of experience in machine learning and related tools. Some of the popular tools that should be learned for the success of this project include Tensor Flow, Stable Baselines3, Gym, PettingZoo, and Python, which are new to the developer. Also, since many frames of the game are needed for training the agents as well as the algorithm, the following order is done: 1: Study Machine learning with “The UC Berkeley Deep Reinforcement Learning course”; 2: Create and test sample reinforcement learning examples with Python and Tensor Flow; 3: Create the game in unity; 4: Implement learning algorithm for one agent; 5: Implement a new algorithm for multiple agents; 6: Test and debug. Since I have no experience with creating multi-player games cross-platform, game creation is challenging as well. This is especially true since the game with 4 players is often complicated for the RL algorithm to properly form a positive chain of action to gain rewards as more states needed to be processed and learned.

2.8.1.1. Training Multiple Agents in the Unity Game

Training multiple agents successively in the Unity concept game initially created was extremely difficult due to 3 reasons. Firstly, with the graphic UI component of the game created in Unity running 60 frames per second, the training process takes a tremendously long time than training agents with no UI. Secondly, the environment should not be extremely difficult for agents to get rewards. Otherwise, the agents would have no clue what exact actions would gain rewards if it requires a sequence of actions. Thirdly, the tool kit built in Unity is less optimal (fewer options) for a customized environment like the

original game made, which made it difficult to connect the RL training part with the actual game inputs and outputs. In the Unity game concept, the proper reward was designed as a) getting the flag, b) avoiding the obstacle, c) returning to base, d) shooting the enemies, and e) protecting the ally. However, this is replaced with Python (refer to section 2.9. Research in New Technologies). Therefore, a simple conceptual environment is made first to test if multiple agents can be trained in a very basic game environment. With the help of a third-party library called “PettingZoo”, it was possible to set up a customized environment to train multiple agents. Originally with the game rendered with image sprites that contain similar shapes and colours, the PPO algorithm could not properly recognize the image process to 84*84 bit as input, which made the agents perform the same or worse than the random AI after training with 1 million steps. To successfully train multiple agents to reach the goal cooperatively in the new game environment, the game is simplified to 2 agents instead of 4. The mechanics of picking/dropping the ball were simplified as moving towards the direction would automatically send the ball as more actions lead to more calculations, making the training even slower. The shapes were simplified to squares and colours were well defined for recognition as well. The training process turned out to be faster than the original Unity game even though time PPO algorithm takes around 4 hours for 2 million-step training with RTX 3060. With other algorithms that do not use visual (image) recognition, the time was about 50 times faster.

2.8.1.2. Changing the Cooperative Game to Fit Needs

An unexpected obstacle also occurred during the training and testing of the algorithm. While the training of the co-op play with 2 agents in the game did work with the version that has no UI component, once the game UI is added, the image recognition algorithm was not working properly with the image sprites the developer had in the game. To cut down more time spent on figuring out and

implementing the whole PPO algorithm to fit the needs, the developer implemented an even simpler version of the game, changing the game structures for an easier environment set up and rendering the objects with squares in different colours instead of image sprites. The simpler version worked very well in the PPO algorithm so that all objects in the game can be recognized. Even though this portion of the project took far more time than expected due to the stated difficulties, the chosen development method can maximize the success rates in implementing the core components, proving their effectiveness and usefulness.

2.8.1.3. The Training Efficiency

Training multiple agents is a time-consuming task that can take hours with powerful newest generation graphics cards. Since the only device I have is a laptop with GTX 1050 -2G, the time it takes for training can be tripled when compared to larger models. Through the span of the project, a new desktop PC with GeForce RTX 3060 was bought to successfully train the agents despite the rising price of Graphics cards. However, the processing time is on average 13 to 14 hours for two agents to train for 10 million steps. It is tested that the agents could not perform better than random AI with several training less than 4 million steps. There were difficulties creating the proper environment, inputs, outputs, and version conflicts for the libraries used. TensorFlow version installed, Python used, and stable baselines were applicable with very specific versions, which costs the developer tens of hours setting up correctly at the end.

2.9.Future Enhancements

Some of the future enhancements that I would like to make are:

- In the future, the developer can train the agents with different RL algorithms to analyze not just the PPO that is needed with the image of

the current environment, but feeding in parameters as states of the agents to train with A2C and more algorithms alike. Due to the limited time and experiences the developer had with abandoning the already trained models and changing the framework for more uncertainties, only PPO is applied in the current settings.

- The game can also be altered with different layouts to further improve the strategies. For example, the map can be randomly generated with random pieces having the slowing down mechanics. However, this is only possible with an improved algorithm and more intense training that are over 10 million steps since the attempt of randomized starting and ending position after training 10 million steps ended up performing similar to the random AI.

Also, the number of agents could be expanded to 4 or more.

- Better Rendering: Currently, the rendering is fairly basic where it is just squared units with simple mechanics. Adding more complexity to the rendering like bitmapping, shadows and lighting would be nice to have.
- Game Logic: Further development on in-game logic would include shooting mechanics like originally planned, adding different conditions for winning and losing for creating more strategies, adding scoring systems, lives, leaderboard, etc. I would also develop this into an actual game instead of just a concept demonstration.

2.10. Timeline and Milestones

Milestones are made with Coda shown in the images below (or check the [link](#)).

Milestones 0

			Filter	Sort	Columns	Options	
	Tasks	Component	Start date	End date	Hours expected	Hours spent	
1	Project design and proposal	Concept and requirement gathering Project design: proposal is written	9/7/2021	11/7/2021	30	30	
2	Project proposal revision	Project proposal modifications and revisions after communicating with instructor	11/7/2021	11/22/2021	10	11	
3	Sprint 1 Unity project initialization	Unity github setup Library imported	1/3/2022	1/9/2022	5	8	
4	Researching Reinforcement Learning fundamentals	Getting familiar with RL algorithms, ideas	1/10/2022	2/1/2022	10	16	
5	Sprint 2 Understand Deep Q-Learning algorithm	Analysis on DQNs parameters	2/1/2022	2/12/2022	10	13	
6	Unity game engine structure set up	Created general singleton structure. Initially planned to use ECS system but dropped after 5 hours of research (Non-essential).	2/12/2022	2/14/2022	5	5	
7	Unity game engine: Scene set up with a map	Game engine: Creating the map using Tile Map 2D in Unity	2/14/2022	2/14/2022	5	6	
8	Game UI component	Finding and creating the object's sprites and background using Photoshop. Added Score text.	2/13/2022	2/13/2022	2	3	
9	Moving mechanics	PlayerController implementation and make sure no bugs happen during moving	2/14/2022	2/14/2022	2	3	
10	Character movement change	Added gun pivot for changing where the player want to shoot bullets and added the flag changes	2/14/2022	2/14/2022	2	3	
11	Make different layers for collision	Creating the object's layers	2/15/2022	2/15/2022	1	2	
12	Gameplay testing	Testing with the fundamental mechanics with play test (3 people)	2/16/2022	2/16/2022	3	3	
13	Rule manual	Originally written but changed after the game is changed	2/17/2022	2/17/2022	1	1	
14	Sprint 3 Researching the multi-agent training process for Unity game	Researching the "ML-tool kit" in Unity for applying multi-agent training. (After midterms)	2/18/2022	3/7/2022	10	14	
15	Sprint 4 Attempts for designing the customized pytorch PPO algorithm	Dropped the Pytorch algorithm design after 10 hours of tutorial and research as the parameters did not fit in the ML tool kit Unity provided, making it not a solution for the game created. (After finals) (Non-essential)	3/7/2022	4/14/2022	15	20	
16	Researching multi-agent training simply tutorial, training, and testing fundamentals	Shifting the focus of the Unity game to implementation of multi-agent training with Gym library.	4/14/2022	4/28/2022	4	6	
17	Sprint 5 One-agent Gym implementation before applying to multi-agent environment	Created a simple one-agent 1D Gym environment for testing if it works in Python	5/1/2022	5/6/2022	5	6	
18	One-agent Gym implementation Version 2 in 2D	Created a simple one-agent 2D Gym environment for testing if it works in Python	5/6/2022	5/10/2022	5	9	
19	Training and testing the Gym customized environment	Training and testing the one-agent Gym customized environment in both 1D and 2D command line environment. Flatten the data to the desired inputs and outputs. Each training takes about 45 mins.	5/7/2022	5/10/2022	15	16	
20	Sprint 6 A 2-palyer cooperation Pygame implementation	Create the simple conceptual game idea of a 2-palyer cooperation game and implemented with Pygame. (This process is delayed due to job applying and interviews)	5/10/2022	7/4/2022	5	8	

21	Added dropping and grabbing mechanics	Added mechanics for dropping and grabbing the ball to get score. Bug fixed with the original movements being strange at times.	7/4/2022	7/5/2022	2	3
22	Sprint 7 Research about combining with the PettingZoo library	Research for changing the game's input and outputs that supports the PettingZoo library	7/5/2022	7/22/2022	15	20
23	Created a simpler game with class structure	Restructure the current game to a simpler game with class structure that is capable to train using the PPO	7/5/2022	7/22/2022	10	16
24	Combined the game with PettingZoo library for multi-agent training	Added petting zoo environment with customized settings that is integrated in the game	7/5/2022	7/22/2022	20	21
25	Added screen capture feature	Added screen capture feature for PPO model to train	7/22/2022	7/22/2022	5	5
26	Training and testing multiple agents with PPO algorithm	Training and testing multiple agents and ensuring the agents perform the basic moves. Each training of 2 millions steps takes around 4 hours.	7/22/2022	7/26/2022	60	65
27	Bugs fix and updating the reward algorithm	Bugs fix and updating the reward algorithm so that the agents moves as expected	7/22/2022	7/22/2022	5	8
28	Retraining and retesting multiple agents with the updated reward system	The multiple agents perform the expected moves after training with updated parameters	7/26/2022	7/29/2022	60	60
29	Added models inherit feature	The inherit models feature makes the models can be trained with previously trained models	7/25/2022	7/27/2022	5	4
30	Final training and testing	Final training and testing the agents with different parameters	7/27/2022	7/29/2022	10	10
31	Build executable for windows	Added executables for demo purposes	8/8/2022	8/8/2022	2	2
32	Documentation and final report				40	43
32					379	440

3. Conclusion

3.1. Lessons Learned

Despite the great amount of time spent on the research process of the early development of the project, more technical documents and implementations should be explored before choosing the technology used in a project. During the development of the agent-training process, I found that the various dependencies were not supporting the specifications used for my applications, therefore tens of hours were spent setting and debugging environment settings. According to my previous experience, once the framework is set, it would become more difficult to change the fundamental structure to fit the incoming needs. When I decided to use another technology to set up the training environment for multiple agents, I had to drop the original libraries and related settings after research. A brand new game was created instead of

modifying the corresponding C# code and the unsupported settings with the algorithm picked. I realized that it is risky to adopt new technology in a project if the new one requires a different supporting environment and the format of settings in terms of inputs and outputs.

3.2.Closing Remarks

This project furthered my experience in my specialization in two main ways: experience using Python and experience training and optimizing machine learning in games.

The experience gained from being able to develop and test agents with Python will be of huge value because Python is the most popular language used for machine learning development and training. The implementation of the PyGame, setting RL algorithm inputs/outputs, and the training of the RL algorithm took less time. This helped as it ensured the success of the project. The agents that are trained with PPO performed smarter than the random AI and sometimes faster than a human being when playing the game. The project also allows me to experience the most experimental aspect of game creation – learning models for machine learning. I have been interested in machine learning for a while, but never had an opportunity or project that let me work with it. The project helped me to improve my software development skills in Python, machine learning, game making, and reinforcement learning algorithms. This project will also include all the software planning, design and management skills I have learned from BCIT and the software development team in Golbey Law.

4. Appendix

4.1. Approved Proposal



Major Project Proposal Approved - Dong, Jingyang A00997028



BCIT BTECH CST <BCIT_BTECH_CST@bcit.ca>
to jdong37@my.bcit.ca, me, BCIT, BCIT

Wed, Dec 8, 2021, 10:35 AM



Hi Jingyang,

The Major Project Review Committee has approved your COMP 8037 proposal. You will be registered into COMP8047 Major Project course in the upcoming Winter 2022 term. Your supervisor is Borna Nouredin.

Borna will provide technical assistance to you and ensure that your project is completed according to the proposal. Please send him the latest proposal. When you are done with the project, please submit the report to your supervisor so he can review the report and provide you with any comments.

Once your supervisor has approved your report and provided you with a written approval to be attached to the report, you may then submit the final report to the committee. Please make sure you allocate enough time for this approval process.

When you are ready to submit your report, please send a pdf copy of the final report to me here at cstbtech@bcit.ca. Any supporting documentation for your project, including codes, manuals, design documents, electronic copy of client letter, etc should also be sent over either in a zip/rar file or via link to a cloud/hosting service where we can retrieve it. If you have any questions about submitting your proposal for review please let me know.

If you do not wish for anyone other than the committee members and your supervisor to view your project report, you will need to submit a formal letter/documentation from your sponsor.

Please refer to the policy and requirement for major project report as described in the Major Projects Guidelines during the time you will be working on your project, as the guideline may go through updates several times a year. The Major Projects Guideline can be downloaded at <https://commons.bcit.ca/computing/files/2020/09/COMP80378047-Major-Project-Guidelines.pdf>.

Please note:

- Your Graduation Deadline is December 31, 2026.

- Your Project Due Date is April 15, 2022.

- If you wish to attempt to attend next June's Convocation, you will need to submit by the Project Due Date.

If you have any questions, please feel free to contact me.

Thanks,

4.2. Project Supervisor Approvals

My project supervisor is Borna Nouredin. He had approved my report a week before Nov 30th, 2021.



Borna Nouredin <borna_nouredin@bcit.ca>
to me

Nov 30, 2021, 5:26 PM



Hi John,

According to my records, you received feedback last week, so am assuming if you are asking if you can resubmit on Dec 6? If so, then yes, that is the idea: make the revisions required and note them in your change log, and make sure you have sent it to cstbtech@bcit.ca before 8am on Mon, Dec 6. If you have addressed all the comments, the proposal will get accepted, but if not, then unfortunately that means you will have to wait until next year and try again.

Let me know if that clarifies things.

Cheers,

Borna

5. References

- [1] An introduction to reinforcement learning - youtube. (n.d.). Retrieved September 30, 2021, from <https://www.youtube.com/watch?v=JgvyzlkxFO>.
- [2] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., & Graepel, T. (2019). Human-level performance in 3D multiplayer games with POPULATION-BASED reinforcement learning. *Science*, 364(6443), 859–865.
<https://doi.org/10.1126/science.aau6249>
- [3] Max Jaderberg, Wojciech Marian Czarnecki, Iain Dunning, Thore Graepel, Luke Marris. (2019, May 30). Capture the flag: The emergence of complex cooperative agents. Deepmind. Retrieved September 29, 2021, from <https://deepmind.com/blog/article/capture-the-flag-science>.
- [4] McIlroy-Young, R., Sen, S., Kleinberg, J., & Anderson, A. (2020). Aligning superhuman ai with human behaviour. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.
<https://doi.org/10.1145/3394486.3403219>
- [5] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. (n.d.). Retrieved October 1, 2021, from <https://proceedings.neurips.cc/paper/2017/file/68a9750337a418a86fe06c1991a1d64c-Paper.pdf>.
- [6] Nguyen et al., ‘Review, Analysis and Design of a Comprehensive Deep Reinforcement Learning Framework’.

- [7] Paczolay, G., & Harmati, I. (2020). A new advantage actor-critic algorithm for multi-agent environments. 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR). <https://doi.org/10.1109/ismcr51255.2020.9263738>.
- [8] Zhang, Y., Zhou, Y., Lu, H., & Fujita, H. (2021). Cooperative multi-agent actor-critic control of Traffic Network flow based on Edge Computing. Future Generation Computer Systems, 123, 128–141. <https://doi.org/10.1016/j.future.2021.04.018>.

6. Change Log

Date	Description
Aug 15, 2022	Submitted Project Report