Collaborative Artificial Intelligence with Reinforcement Learning

COMP 8037 – Major Project 1

Jingyang Dong – A00997028

11-7-2021

Table of Contents

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, OpenGL, and am experienced in programming in C, C++, C#, Java, JavaScript, HTML, and PHP.

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

1.2. Work Experience

- **Website Developer - Golbey's Law** **Apr. 2020 – Oct. 2020**
  - Created a trademark registration website with React.js, .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.

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. Reinforcement learning is an area of Machine Learning. It 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 behaviors, and eventually act with actions that help the players to win the game. For the goal of the project, the trained agents will be illustrated in a 2D top-down 4-

player racing game. The core mechanic of the game is to get the flag, evade other cars, and run. The game is similar to the game “capture the flag” in terms of the mechanics. There will be two teams of two and each has one flag representing their team. The player can shoot the opponent, and when this happens, any flag the opponent is carrying will drop. The car being shot at will be respawned in 3 seconds. The game will be built with Unity across platforms for Android and PC.

The agents will be attached with path-finding algorithms to constantly track where the other agents are during the gameplay. The learning process is about 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) constantly. The initial input would be the actions of the players. When actions taken do not provide any rewards (scoring), this episode of actions will be labeled as “do not do”, and the possibility of performing such actions would be weighted lower. In this project, the expected behavior of the agent-training process contains multiple stages: 1) learning the basics; 2) Increasing navigations; 3) Applying strategies.

In stage one, the expected outputs from the agents are: Moving in the base area, randomly shooting, not capturing the flag, and doing nothing. In stage two, the agent should learn to shoot at the opponent, navigate to the home/opponent base. In stage three, some strategies should be learned and applied. This includes home base Defense, opponent base camping, and teammates/opponents following.

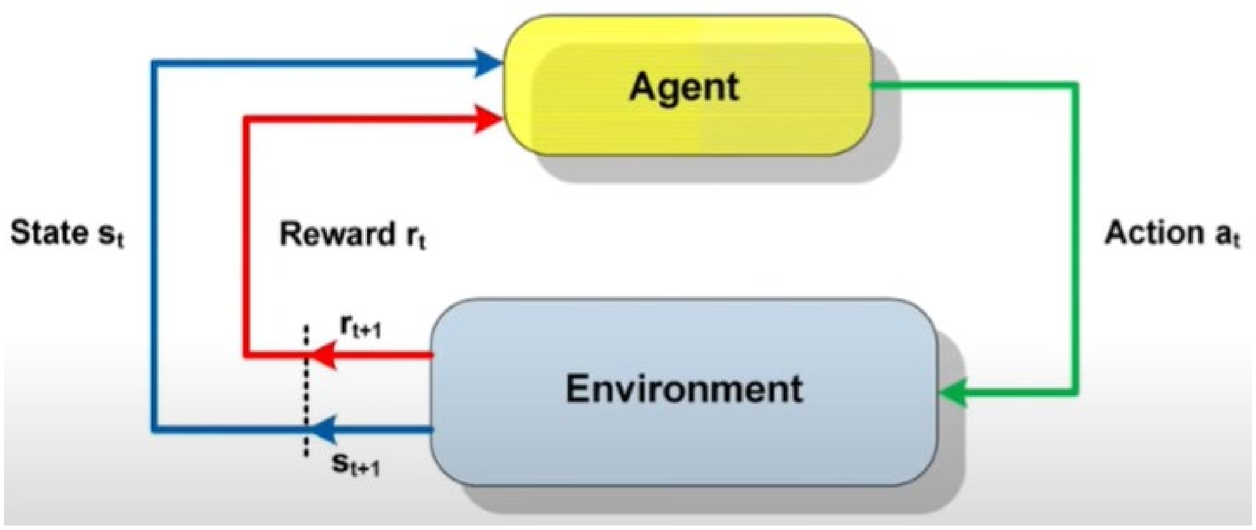

Figure 1. Reinforcement Learning States Machine An Introduction to Reinforcement Learning - Youtube. *https://www.youtube.com/watch?v=JgvyzIkgxF0.*

3. Problem Statement and Background

In the current game industry, most of the 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, to 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.

The reason agents suffer in Q-learning is that 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 worsen 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.

4. Scope and Depth

The minimum scope of the project is to create a capture-the-flag game that allows the agents, with a reinforcement learning approach, to cooperate and compete with the players and other agents.

The depth of this project includes the following:

In-Scope:

- The AI will learn from scratch how to see, act, cooperate and compete in unseen 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 a flag).
- 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 behaviors of other agents and apply strategic decision-making.
- The game will be a cross-platform game with a local network (PC and Android). Players can create and join a room to start the game.
- The game will allow up to 4 human players or with a maximum of 3 agents.
- An instruction manual with the rules of the game will be provided in the game.
- The game will have a frame rate greater than 30 frames per second.

Out of Scope:

- Mechanics that require multiple decisions to be made during gameplay. For example, adding different collectible items will change the behavior of the existing strategies, adding an extra layer of complexity, generates numerous branches of actions for AI to learn. This would lead to exponentially increased complexity for the agents to make decisions.
- Having dedicated servers for multiplayer through matching rooms online. Due to the lack of free Azure and AWS credits, the game will not be hosted on the cloud with extra costs.

Stretch Goals

- Publishing the proof-of-concept game in public stores. It is considered as a stretch goal mainly because of the high complexity required to successfully publish IOS apps in the App Store.
- Polishing the proof-of-concept game with a better gameplay experience. Since the game is mainly focused on demonstrating the cooperative agents, the polishing of the game itself is considered as a stretch goal. This includes a clean User Interface, various particle effects, and meaningful plays that lead to an immersive experience.

The feasibility of the project should be high. The 2D implementation with limited mechanics for this game reduces the complexity for training a population of agents, as well as the length of the game development to an obtainable goal for a timeframe of 360 hours, including research and testing.

5. Test Plan

Procedures and methodology

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 might be 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 2 [6].

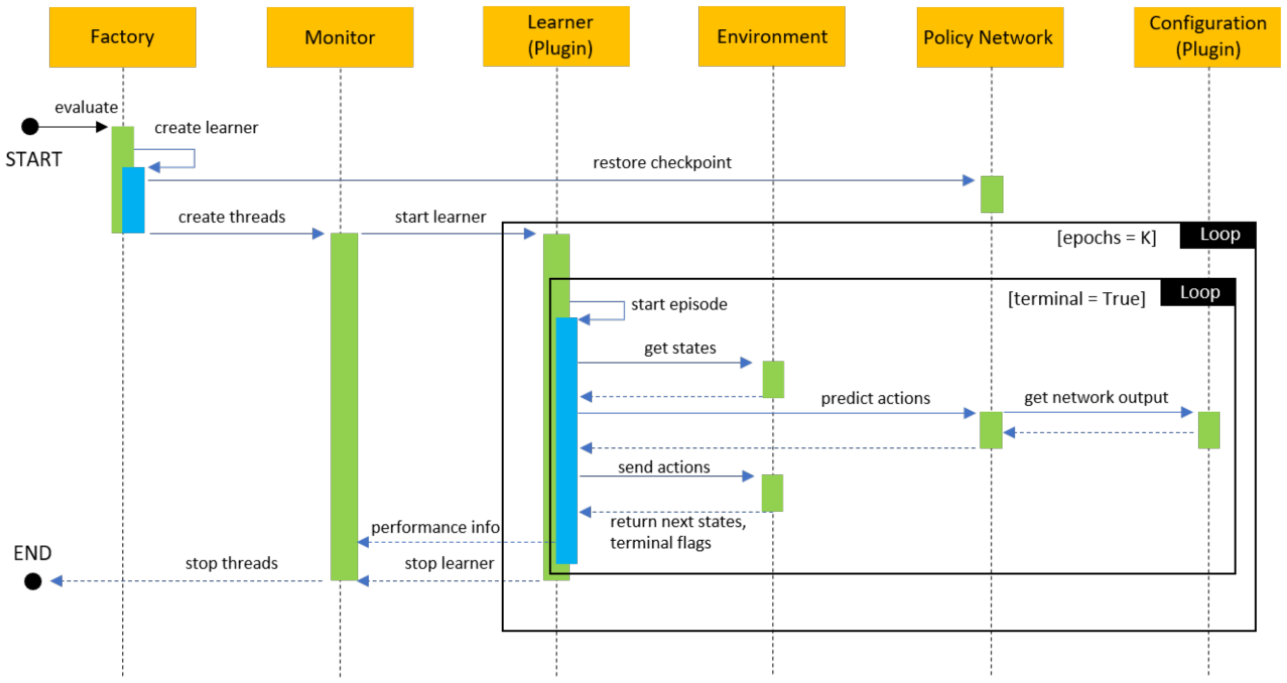

Figure 2: UML sequential diagram of the evaluation process.

For testing the algorithm, the difference is that it does not train the agents after the next states are returned. The results of the collected episodes will be verified possibly through exporting data as CSV files and printing on the Tensor board to evaluate results. The pass criteria would be determined by how well the performances of the collected data. 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.

Unit Testing

The NUnit test framework will be used to automate and test the Unity game. The PyTest framework will be used to test the Python project. These two frameworks were chosen due to their good reputations and their integrated IDE support. Each component of the game and the training models will have Unit Tests written for them. 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.

Test Cases

Test cases will be designed to test against these requirements and any edge cases that might exist. Below are some example test cases that will be used for testing.

Note: more test cases will be created during development.

- For the game

Test	Input	Expected Result
Transform of Game objects	Game controller	All functionality for position, scale, rotation, and offset are updated correctly upon collision.
Character controller	Game controller/Buttons	Character movement responding to the user input
UI Manager	Buttons	All UI elements staying in fixed positions
Sound Manager	Game controller/Buttons	Sounds are correctly set up, played, and removed.
Texture Manager	Resources	Textures loaded and handled correctly. All texture is verified with its ID.
Mechanics – shooting	Game controller/Buttons	Tests for shooting mechanics functioning when colliding with other objects/agents.

- For the agents-training

Test	Input	Expected Result
The rewards system	Data	All functionality for setting and updating the rewards algorithm is correct.
Shortest path algorithm	Given positions	The agents move towards the expected path calculated from the shortest path algorithm.
States	Data/action	The number of different states meets the expected number when an action is performed.
Reading scores from the game	Transformed data from the winning screen's scores.	The rewards are updated correctly with the transformed data from the winning page in the game.
Actions	Data/algorithm	The actions performed by the agents are getting better as the number of training increases.

6. Methodology

Methodologies and Approach

I am going to use the Agile Scrum approach 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 feedbacks of the game should be gathered as early as possible to ensure fewer changes to the core mechanics of the game and allow more focus for the agent-trainings. 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 will be updated with priority labels during and after each sprint. Developers tend to gradually deepen their understanding of the software during the development process.

Ideally, this approach will make my development process more organized and efficient.

Technologies & Tools

The capture-the-flag racing game will be developed on top of the Unity engine, rendered in 2D with 2D physics. The language used in this project will be mostly in C#, among with Python for developing the algorithm through TensorFlow (an open-source software library for machine learning and artificial intelligence). Machine Learning Agent Toolkit (ML-Agents) from Unity or TF-agent from google will be used for providing a training environment for the agents. Plugins like TensorForce, OpenAI Baselines, and RLLab might be needed for implementing the reinforcement learning algorithm.

Platform

The catching-the-flag game is targeting multi-platform, which includes Windows and Android.

Version Control

Git will be used as this project's version control system. All code for this project will be hosted on GitHub.

Integrated Development Environment (IDE)

The Unity game will be developed within Visual Studio 2019, and the reinforcement learning model will be implemented with Visual Studio Code. Visual Studio Code provides features including integrated debugging support, code analysis, and unit testing support.

Design Tool

Free web apps like Lucidchart.com, draw.io will be used to create the project's software architecture diagrams.

Planning Tool

Trello and Notion are used to maintain backlogs and task planning during each sprint.

7. System/Software Architecture Diagram

Algorithms

Training agents in multi-agent systems requires teammates and opponents in the environment to generate a learning experience. In this project, several different approaches for the algorithm might be needed 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 own policy. While self-play variants can be effective in some multi-agent games, this method can be unstable and does not support concurrent training. The second approach is aiming to provide a solution to train a number of 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 behaviors are seen during training. Several maps should be made in the game to provide meaningful learning behavior and experiences.

Algorithm Details: Q-learning will be used in this project. The equation is shown in Figure 3.

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

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.

In the project, the architecture for the agent is designed as Figure 4.

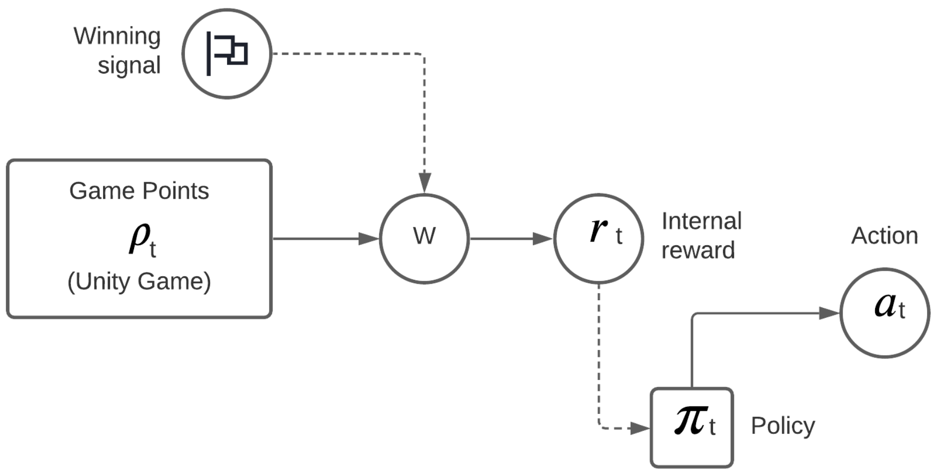

Figure 4: Agent architecture and benchmarking.

The action distribution Policy (πt) is conditional at each time step t. The agent will be updated using reinforcement learning based on the agent's internal reward Signal (rt), which will be obtained from w of game points (or scores from winning screen) ρt. The w is meant for setting win/draw/loss signals as a reward of rt = 1, 0, and -1 accordingly. With the different rewards the agents receive, different responses are expected for the agents under states where: a) agent flag is taken; b) Opponent flag is held by a teammate; c) Opponent flag is held by the agent; d) Agent is respawning.

Diagrams

Figure 5 is the class diagram for the Unity 2D capture-the-flag racing game with shooting mechanics. The game is going to be implemented using the Entity Component System (ECS). It is also designed with local networks that a player can join rooms that others created. The ML-agent or TF-agent will be providing the environment for agent training and bridged to the base game.

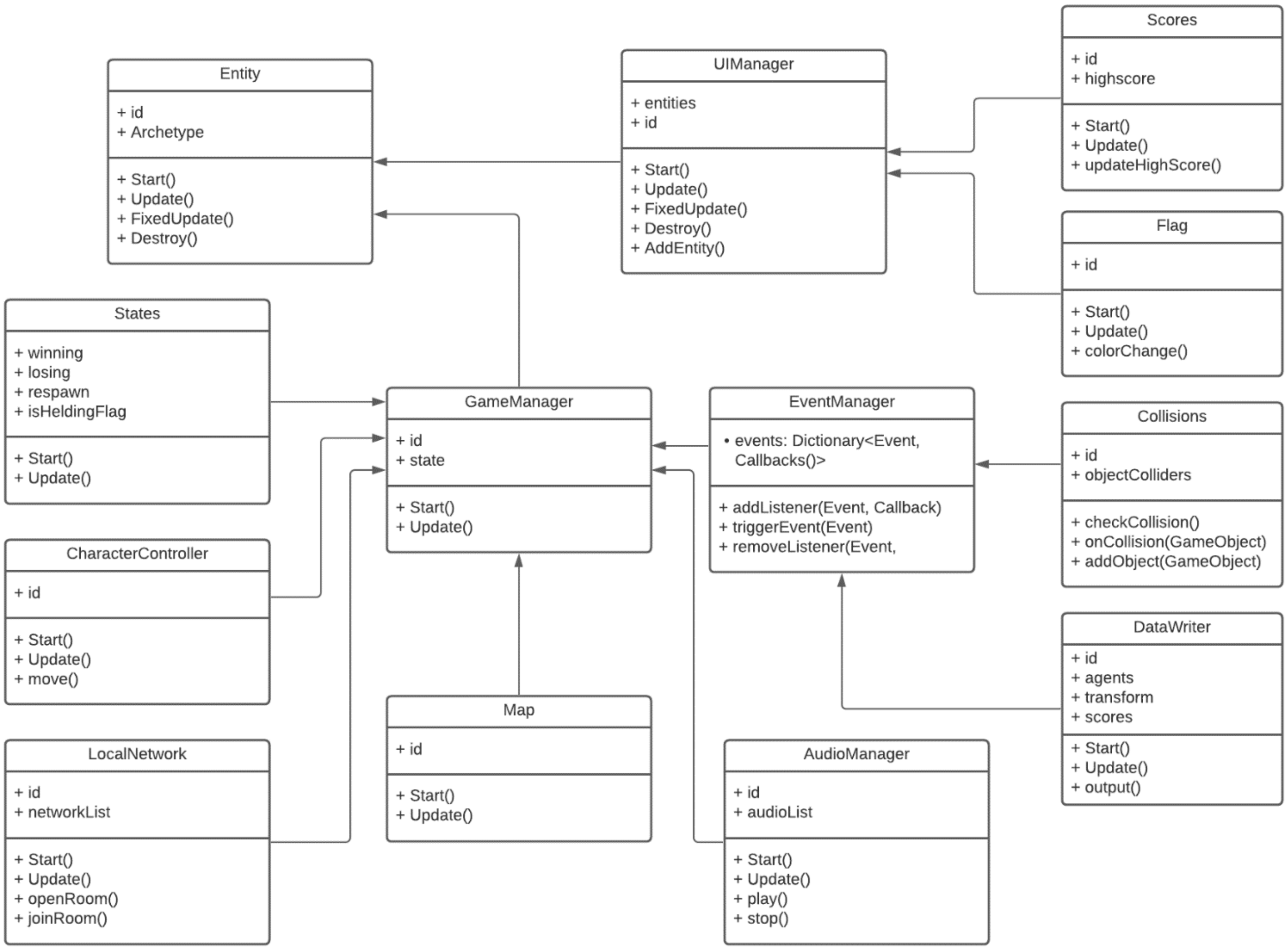

Figure 5: Class Diagram of the Unity game

Figure 6 shows the overall training process for the single and multiple agents. Within each iteration, the agent will get the current state, predict the actions, and perform the actions [6].

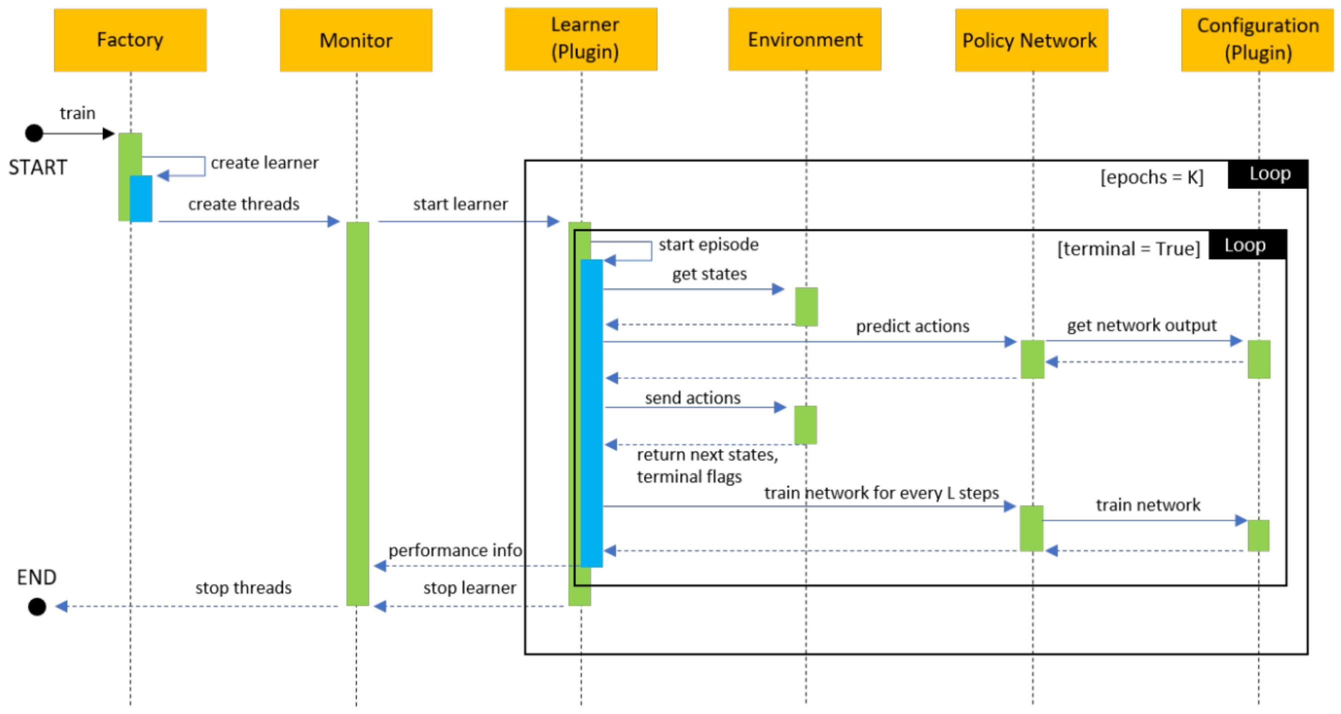

Figure 6: UML sequential diagram of the agent training process.

8. 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 – population-based deep reinforcement learning approach (In other words, 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 been 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.

9. 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. 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 includes:

- 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 most likely out of the scope based on my current knowledge.

10. Technical Challenges

There are two major categories of challenges associated with this project: Learning Curve and Algorithm Generation.

Some of these issues can be foreseen while others will be discovered as development progresses. Some of the foreseeable issues include:

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, Caffe, and Python, which are new to me. Also, since many frames of the game are needed for training the agents as well as the algorithm, the following order is decided: 1: Study Machine learning with "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 game cross-platform, game creation is challenging as well.

Algorithm Generation

The main focus of the project is to generate the algorithms and the corresponding policies to maximize the rewards to ensure the agents' actions are reasonable under the world that contains multiple agents. The algorithm is 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.

Low Specification for Training

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. Due to the current rising price for Graphics cards, purchasing a desktop with the newer models will not be an option. It is expected to use one of the following options for training the agent: a) Google Cloud or equivalent b) Desktops/Mac from the BCIT lab room. It is also expected to encounter difficulty for environment setting because their TensorFlow/Python/GPU driver versions may vary, which can fail in the environment setting.

11. Development Schedule and Milestones

The following table outlines this project's sprints and a breakdown of how time will be spent on each task. The milestones are structured around the project's sprints and are each two weeks apart as a result.

Task#	Description	Hours
1	Concept and requirement gathering Project design: proposal is written	30
2	**Sprint 1** A simple Unity Project practice with a tutorial with Entity-Component System Framework set up with GitHub	10
3	ECS system implemented for base game project and architecture blueprint fully understood. Shortest-path algorithm ready to implement	15
4	Prototype for the 2D game with simple replacement sprites and fundamental mechanics implemented	10
5	Testing with the fundamental mechanics with play test (possibly 3 people)	4
6	Implement the system to update scores and display on the UI element of the game	4
7	Writing Rule manual for the game and its control manual.	3
8	**Sprint 2** Full mechanics and rules implementation for the player with scores updates correctly	10
9	Implement a path-finding algorithm for the game logic (AI opponent). Research and create agents to perform the same actions as the player	10
10	Feature for creating room for multiplayer locally	5
11	Debug and testing with multiplayer feature	5
12	Polish the UI element to have the basic features of the game	7
13	Finding the free audios and sprites needed (itch.io)	4
14	Adds the audio elements to the game with a control system	2
15	Replace the current sprites and background with the new ones. Polish the UI elements	5
16	Feature for outputting the player/AI movement to an external file that can be training in TensorFlow/ML agent as input	10
17	Final debug and testing	5

18	Play-testing with the finished game	4
19	**Sprint 3** Documentation for the game created	10
20	Setting up the packs and environments needed for TensorFlow agent-training (Google Cloud/lab machines included)	10
21	Setting an example of AI training from the Udemy Courses or tutorials.	5
22	Training AI for the tested example to perform as expected	5
23	Creating bridge with the unity game and agent-training models (ML-agent/ TF-agent)	10
24	Setting up the algorithm for a single agent in the game created (setting up the policy and rewards from the output file generated in the game)	15
25	Training and testing a single agent and ensuring the agents perform the basic moves (moving to 4 directions and shooting)	10
26	**Sprint 4** Training and testing a single agent and ensure the agents perform some strategies while the actions of the player changes	10
27	Test multiple players in the game to ensure it works with different devices	10
28	Setting up the algorithm for multiple agents in the game created	30
29	**Sprint 5** Training and testing multiple agents and ensuring the agents perform the basic moves	30
30	Training and testing multiple agents and ensure the agents perform some strategies while the actions of the player changes	30
31	Expected waiting time for training the agents (Period of the time that training itself will take the place than coding/testing)	20
32	**Sprint 6** Apply the trained agents to the game and debugging/testing	20
33	Documentation for the AI agent training	10
34	Bug fixes with the trained agents running in the game	10
35	Creating .apk file for Android system and debugging	20
36	**Sprint 7** Final testing/optimizations	20
37	Final Report	30

- Estimated Total Hours: 448 hours
  - Implementing the proof-of-concept game: 123 hours

    - Creating, training, and testing the multiple agents: 265 hours
    - Documentation: 60 hours

12. Deliverables

The deliverables in this project are:

- Printed final report including the game's user manual
- Proof-of-concept game:
    - User manual PDF
    - Project solution and readme.txt
    - Game prototype executable for Windows PC
    - Game prototype installation package for mobile Android devices
- Report: Major Project report
- An Android smartphone with charger and USB cable (for demonstration)

13. Conclusion and Expertise Development

This project will further 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. It also takes less time to set up an environment for AI training and testing when compared to C++, which will allow more time to tackle down the core features in this project. Learning a new language means a learning curve is expected but it will allow me to face new challenges, adjust to new programming trials and errors, and better prepare me in the gaming industry. It 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. I always thought that machine learning is for the people who have the most powerful machines possible. Even though the training of multiple agents may require such specifications, it is possible to perform it through the cloud with less expense. I believe that with the project's scope, the knowledge and experience in machine learning will be expanded, making this project challenging yet achievable.

14. References

[1] An introduction to reinforcement learning - youtube. (n.d.). Retrieved September 30, 2021, from https://www.youtube.com/watch?v=JgvyzIkgxF0.

[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 behavior. 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

15. Change Log

Date	Description
Sep 15, 2021	Submitted Assignment 1: Project Idea Topic and Description
Sep 24, 2021	Received feedback for Assignment 1.
Sep 29, 2021	Submitted Assignment 2: Student Background, Project Description, Problem Statement and Background, Innovation, Complexity, and Technical Challenges.
Oct 13, 2021	Submitted Assignment 3: Project Scope and Depth, Conclusion and Expertise Development, and References.
Oct 15, 2021	Received feedback for Assignment 2.
Oct 27, 2021	Submitted Assignment 4: Test Plan, Methodology, System/Software Architecture Diagram.
Nov 07, 2021	Submitted Assignment 5: Development Schedule and Milestones, Deliverables, and conclusion section revised
Nov 22, 2021	Revised section 2: Project Description (the concept of Reinforcement learning) on page 2. Grammar revision on pages 2, 3, 4, and 11, 15.