Multi-Agent Reinforcement Learning System to Promote Teamwork and Emergent Behaviour in Drones

Digital Systems Project – UFCFZK-30-3

**Austin Harper**

Student ID: 17009191

Word Count: 10087

# Abstract

The use of drones for more complex purposes means that multiple intelligent drones may need to be used to complete a task that would not be possible for a single drone, which is called an emergent property. Using reinforcement learning, a multi-agent system has been created to allow for drone simulation to be explored with intelligent agents, which are able to perform teamwork.

Literature on the topic has been considered in order to find the most useful areas of study for this tool to be used within, and the tool has been designed to fill gaps between, and compound the work that has come before it. Users are able to use the core tool to train agent systems with customisable settings, evaluate those systems, and analyse results. The tool is designed to be built upon in the future, allowing additions of more settings and reinforcement learning techniques.

# Acknowledgements

Thanks to the attendees of the Project in Progress day, who gave valuable thoughts on the state of the project, in particular Jim Smith.

Thanks to the Github users *Kjell-K* and *hoangtranngoc,* whose repositories, *AirGym* and *AirSim-RL* respectively, provided a great reference for the API part of the project.

Finally, thanks to Phil Legg, for continued support and advice throughout the project. His feedback and enthusiasm for the project greatly aided in its development.

# Table of Contents

[Abstract 2](#_Toc70590616)

[Acknowledgements 3](#_Toc70590617)

[Table of Contents 4](#_Toc70590618)

[Table of Figures 6](#_Toc70590619)

[Table of Tables 7](#_Toc70590620)

[Introduction 8](#_Toc70590621)

[Aims and Objectives: 9](#_Toc70590622)

[Outline of Content: 9](#_Toc70590623)

[Literature Review 11](#_Toc70590624)

[Background Research: 11](#_Toc70590625)

[Findings: 13](#_Toc70590626)

[Requirements 15](#_Toc70590627)

[1. Must: 15](#_Toc70590628)

[2. Should: 16](#_Toc70590629)

[3. Could: 17](#_Toc70590630)

[4. Won’t: 18](#_Toc70590631)

[User Stories: 18](#_Toc70590632)

[Methodology 20](#_Toc70590633)

[Design 22](#_Toc70590634)

[Design Diagrams: 22](#_Toc70590635)

[Architecture and Design Principles: 24](#_Toc70590636)

[Tools and Libraries: 24](#_Toc70590637)

[Menu Designs: 24](#_Toc70590638)

[Implementation 26](#_Toc70590639)

[Stage 1 – Prototyping and Preparation: 26](#_Toc70590640)

[Stage 2 – Q-learning for 1 Agent: 27](#_Toc70590641)

[Stage 3 – Multiple Q-learning Agents and Teams: 29](#_Toc70590642)

[Stage 4 – Tool Development and Testing: 30](#_Toc70590643)

[Stage 5 – Reinforcement Learning Techniques: 31](#_Toc70590644)

[Testing: 33](#_Toc70590645)

[Requirements Testing: 33](#_Toc70590646)

[Execution Screenshots: 35](#_Toc70590647)

[Performance Testing: 38](#_Toc70590648)

[Encouraging Teamwork: 40](#_Toc70590649)

[Aims and Objectives Met: 43](#_Toc70590650)

[Project evaluation 44](#_Toc70590651)

[Requirements and Tool Utility: 44](#_Toc70590652)

[Limitations: 44](#_Toc70590653)

[Further Work: 45](#_Toc70590654)

[Reflections: 45](#_Toc70590655)

[Conclusion 48](#_Toc70590656)

[References/Bibliography 49](#_Toc70590657)

[Literature Reviewed: 49](#_Toc70590658)

[Other Sources: 50](#_Toc70590659)

[Tools and Software Libraries: 51](#_Toc70590660)

[A) Appendices 52](#_Toc70590661)

[Appendix 1 – Source Code via GitHub: 52](#_Toc70590662)

[Appendix 2 – README.md: 52](#_Toc70590663)

[Appendix 3 – Weekly Logs via GitHub: 54](#_Toc70590664)

# Table of Figures

[**Figure 1 - Gantt Chart** 21](#_Toc70585951)

[**Figure 2 - Activity Diagram showing Q-Learning for One Agent - Visual Paradigm Online** 23](file:///C:\Users\Austi\Documents\Uni\Year%203\DSP\AHarper%20-%20Multi-Agent%20Reinforcement%20Learning%20System%20to%20Promote%20Teamwork%20and%20Emergent%20Behaviour%20in%20Drones.docx#_Toc70585952)

[**Figure 3 - Sequence Diagram showing Data Flow of Q-Learning for One Agent - Visual Paradigm Online** 23](file:///C:\Users\Austi\Documents\Uni\Year%203\DSP\AHarper%20-%20Multi-Agent%20Reinforcement%20Learning%20System%20to%20Promote%20Teamwork%20and%20Emergent%20Behaviour%20in%20Drones.docx#_Toc70585953)

[**Figure 4 - Activity Diagram showing the Flow of Tasks in a Multi-Agent Q-Learning System** 24](file:///C:\Users\Austi\Documents\Uni\Year%203\DSP\AHarper%20-%20Multi-Agent%20Reinforcement%20Learning%20System%20to%20Promote%20Teamwork%20and%20Emergent%20Behaviour%20in%20Drones.docx#_Toc70585954)

[**Figure 5 - Class Diagram showing how Environment, Drone, and Agent Classes are Linked** 24](file:///C:\Users\Austi\Documents\Uni\Year%203\DSP\AHarper%20-%20Multi-Agent%20Reinforcement%20Learning%20System%20to%20Promote%20Teamwork%20and%20Emergent%20Behaviour%20in%20Drones.docx#_Toc70585955)

[**Figure 6 - Main Menu of the created Tool** 25](#_Toc70585956)

[**Figure 7 - A Drone Moving to a Set Location through the AirSim/Gym API** 27](#_Toc70585957)

[**Figure 8 - A Single Drone using Q-Learning to Navigate** 29](#_Toc70585958)

[**Figure 9 - Text UI Viewer** 30](#_Toc70585959)

[**Figure 10 - Drones (circled) moving to Random Starting Locations** 31](#_Toc70585960)

[**Figure 11 - A Selection of Custom Settings for Training** 32](#_Toc70585961)

[**Figure 12 - Preset Training and Evaluation Options in the Evaluation Menu** 32](#_Toc70585962)

[**Figure 13 - Variable Values on Breakpoint for calculate\_reward()** 36](#_Toc70585963)

[**Figure 14 - Variable Values after calculate\_reward() Breakpoint, showing Euclidean Reward** 36](#_Toc70585964)

[**Figure 15 - Breakpoint on get\_action(), Values of Observations are shown** 37](#_Toc70585965)

[**Figure 16 - Breakpoint showing the value of Action1, returned after get\_action()** 37](#_Toc70585966)

[**Figure 17 - AirSim Viewer, executing a previously saved run** 38](#_Toc70585967)

[**Figure 18 - Custom Training Setup** 38](#_Toc70585968)

[**Figure 19 - Text UI Viewer, showing Previous Training Run with a Moving Flag** 38](#_Toc70585969)

[**Figure 20 - Text UI Viewer, showing next step of run in Figure 19** 39](#_Toc70585970)

[**Figure 21 - config.ini Settings, showing changed velocity and destinations** 39](#_Toc70585971)

[**Figure 22 - Text UI Viewer, showing Previous Training Run of 100,000 Episodes** 41](#_Toc70585972)

[**Figure 23 - Text UI Viewer, showing Steps 0 and 1 from Test P.7 Evaluation** 42](#_Toc70585973)

[**Figure 24 - Text UI Viewer, showing Steps 2 and 3 from Test P.7 Evaluation** 43](#_Toc70585974)

[**Figure 25 - Text UI Viewer, showing Steps 4 and 5 from Test P.7 Evaluation** 43](#_Toc70585975)

[**Figure 26 - Text UI Viewer, showing Steps 17 and 18 from Test P.7** 44](#_Toc70585976)

# Table of Tables

**Table 1 - User Stories**

**Table 2 - Requirement based Test Cases**

**Table 3 - Performance Testing**

# Introduction

Drones are seeing an increase in use in our modern world, from delivery of packages by companies such as Amazon (Georgiadis, 2021), to military uses such as ‘Loyal Wingman’ systems (Chuter, 2021), and the availability of drones is allowing for anyone to buy and fly a drone. When working as a swarm, drones can display emergent properties similar to that of a collective intelligence, when an individual is incapable of performing an action but a swarm is able to, such as the drone display at the 2021 New Year’s Eve celebration in London. In addition the improvement in drone technology presents reasons to develop countermeasures, such as the disruption caused by drones at Gatwick airport in 2018 (Shackle, 2020), or the possibility for drone swarms to constitute a new weapon of mass destruction (Kallenborn, 2021).

Many future uses of drones require more intelligent and autonomous drone controllers, able to make quick decisions given limited information about an environment, and multiple drones to be acting as a team. These same techniques could also be applied to drones for use as countermeasures to malicious drones. Multi-agent reinforcement learning can be used to train agent teams to evade or capture each other. It has been seen that such systems can “induce agents to learn complex strategies and skills.” (Baker et. al., 2020, p.10). Teams of agents trained in this manner may display the use of different strategies to compete and improve performance.

This project was inspired by the OpenAI Gym: Hide and Seek paper, Baker et. al. (2020). In a sense this project is an extension of the authors’ paper, in a more open environment with the possibility of more realistic uses.

The tool created for this project aims to allow training of multi-agent reinforcement learning, and drone simulation. The system allows for the agent controlled simulated drones to exhibit emergent behaviour and teamwork in order to achieve a team goal, one team aiming to reach a point, another team aiming to counter them. It is built in a clear and modular manner in order to allow for further additions to the tool in future, such as different reinforcement learning techniques. It serves as a tool to answer the question, how can teamwork and emergent behaviour in multi-agent reinforcement learning drones be induced?

The creation of the tool includes an API for AirSim (Microsoft, 2021) and OpenAI Gym (OpenAI, 2021) to allow for these techniques to be applied to simulated drones in an Unreal Engine (Epic Games, 2021) environment. The tool was developed using Python (Python Software Foundation, 2021), on the Ubuntu operating system.

## Aims and Objectives:

The project objectives are:

* To create a system allowing for multi-agent reinforcement learning, where the agents control simulated drones in a 3 dimensional space.
* Experiment with learning techniques allowing for agents to learn to evade, capture, and use teamwork to achieve goals.
* To encourage teamwork and emergent behaviour in multi-agent systems.
* To use and implement reinforcement learning techniques in a simulated environment.

The main focus of the project was to create a tool allowing for many Reinforcement Learning settings to be explored, trained, and viewed. The system is aimed at researchers in the field, with experience in reinforcement learning implementation. Additions and modifications to the main tool will be made easier by the modular nature of the system, where new techniques can be implemented into the main system with few changes. The system could be used as a proof of concept to show the different uses that multi-agent reinforcement learning can provide.

Experimentation with settings and reinforcement learning techniques can be used to find the best way to encourage teamwork and emergent behaviour between agents.

## Outline of Content:

The Literature Review contains a critical review of studies that have taken place in multi-agent reinforcement learning, and more specifically for drone simulation. Studies are summarised, with the salient points, limitations, outcomes, and possible extensions noted. It was found that a useful tool would allow comparisons between different reinforcement learning techniques, the ability to easily include new features, and adjustable settings for the environment itself.

The Requirements section details the planned requirements for the tool, in a prioritised order. The arguments for using these requirements are laid out, as is the effect on the finished tool.

Methodology covers how the project was approached. In this case implementation was based around Agile development practises. A Gantt chart shows the timeframes of work.

In the Design section the design principles and architecture of the tool are discussed, using diagrams to visualise. Object Oriented design was used wherever possible to allow for more modular software and therefore more utility considering the possible extensions of the tool.

The stages of Implementation detail how Agile practises were used to develop the tool incrementally. Test Cases and results are also included here as the test cases were developed before each requirement was introduced to the system. Performance testing is used to show how the reinforcement techniques perform when applied using the tool. An exploration into how teamwork can be induced is conducted, and the aims and objectives are revisited.

Finally, the Evaluation covers the analysis of how well suited the tool actually is to the intended purpose, limitations of the tool and extensions that could be made, and personal reflections on the project.

The outcome of the project was the creation of a function system allowing for multi-agent reinforcement learning using drone simulation, where variables can be adjusted and results compared.

# Literature Review

This project was inspired by the OpenAI Gym: Hide and Seek paper, Baker et. al. (2020), however other current and previous literature have been reviewed in order to identify the significance of this project.

## Background Research:

Baker et. al. (2020) show how agents can create a self-supervised autocurriculum with distinct rounds of emergent strategy, with coordination and tool use. This is through two teams of agent playing hide and seek against each other, continually adapting and improving on their performance. Agents use standard reinforcement learning algorithms and are given intrinsic motivation to perform tasks by giving agents a team reward. The results of Baker et. al. showed 6 distinct rounds of competing strategies, proving that multi-agent autocurricula can lead to physically grounded human-relevant behaviour. The agents learned to coordinate with their team and divide labour by using team-based rewards. The authors found that the number of training episodes defined the intelligence and strategies of agent teams, in this case the agent teams only learned to move from the first stage of random movement after 25 million episodes of hide and seek. Another 75 million episodes and the seeking agents devise tool use strategies to counter the hiders, with all strategies exhausted only after 380 million episodes of training. The environment used was identified to be inherently bounded and will not surpass the 6 modes presented.

Another key piece of literature is Barton et. al (2018), where the authors present a method for assessing collaboration in multi-agent reinforcement learning. In order to asses this method Barton et. al simulated a predator-prey task where a team of predator agents were tasked with reaching a prey agent that was faster than any one predator. As with Baker et. al., the predators received a team reward on contact with the prey. The agent learning technique used neural networks representing policy and Q-Learners, where behaviour coordination is aided due to each agent’s state and actions being passed to each critic network, in order to become joint action learners instead of independent learners. Barton et. al. used convergent cross mapping (CCM) to assess collaboration, examining the causal influence one time-series has on another. It was found that this algorithm demonstrated better performance than any fixed methods, however this does not necessarily indicate coordinated behaviour between agents. This environment could be added to, including giving the prey agent a goal to fulfil in addition to evading the predators.

Cetin, Barrado, and Pastor (2020) aim to counter drones with other drones, with the countering drones controlled using deep reinforcement learning. The authors use AirSim, OpenAI Gym, and Keras-RL in order to train the agent and simulate drones. The agent uses a Convolutional Neural Network, to process inputs such as image from drone camera, velocity, Euclidean distance to the goal, and produces the action output as one of three options, straight, yaw left, yaw right. This paper is limited as the drones are always on the same plane, therefore becoming less complex with only 2 dimensions to act within. It could be extended by including more than 3 actions to take, and allowing the other drone to be agent controlled.

Muñoz et. al. (2019) implement a reinforcement learning system for delivery drones to avoid obstacles. The study compared the performances of Convolutional Neural Networks (CNN), and Joint Neural Networks (JNN) that include Deep Q-Network learning using AirSim in a realistic Unreal Engine environment. The algorithms were applied to an image from the drone camera, in order to calculate an action. The results found that the JNN outperformed the CNN with better rewards, consistent results, and faster convergence. The authors also showed that agents could be around 90% accurate in the simulation, which would drop to 50-60% in a realistic environment. This study could be extended to use moving obstacles, and compare how different methods can perform.

Similarly Akhloufi, Arola, and Bonnet (2019) use a deep Convolutional Neural Network (CNN) to learn, another CNN for object detection, and search area proposal based on particle filters to predict the actions of and track a target UAV. This was also applied to a real drone for an outdoor test. The deep reinforcement learning technique is effective but can be slowed due to processing required. It would be interesting to apply these techniques to a more complex environment, and attach such algorithms to drone in a multi-agent system.

Bou-Ammar, Voos, and Ertel (2010) present a reinforcement learning autopilot for a Quadrotor UAV, compared to a nonlinear controller. Fitted value iteration was used in training the agent. The goal in this environment was to use low level controls to hover the UAV. It was found that the nonlinear controller performed better, but the learning algorithm also achieved a satisfying result. The learning algorithm also did not require knowledge of the model to be designed. To extend this study the UAV could be trained to move to a location using these controls.

Pham et. al. (2018) propose an algorithm in order to allow UAV’s to learn to cooperate to most efficiently map an area. To enable joint cooperative action, Pham et. al. use game theoretic correlated equilibrium. In this scenario, the UAV’s are focussed on team mapping, without any other variables such as obstacles or opposition. This is similar to Lowe et. al. (2017) where the authors explore deep reinforcement learning in multi-agent domains. The authors show learned policies requiring agent coordination, and present this method in cooperative and competitive scenarios. Lowe et. al. use a method that outperforms traditional reinforcement learning algorithms in a variety of environments, however it could be further tested in more complex environments.

The work of Selvakumar and Bakolas (2020) present the use of MinMax Q-Learning in order to address a pursuit-evasion game. The authors present two opposing agents, one aiming to reach a target destination, the other aiming to capture its opponent. MinMax works to maximise the chances of an agent achieving the best reward, while minimising the chances of an opposing agent achieving the best reward. In this case a payoff matrix is generated, where the values of time-to-target and time-to-capture are computed. This study would be an excellent starting point to explore how MinMax could be used in Multi-agent learning.

Panait and Luke (2005) survey the cooperative multi-agent learning literature over a broad variety of topics. The authors identify two key features of multi-agent systems that are unique, the ability to learn emergent behaviours, and the use of multiple learners which opens game-theoretic issues. This creates a very useful resources for categorising multi-agent systems and points to further research in each category. This method could be further studied and improved with a modular Q function.

Itoh (1991) presents theories on how to incentivise teamwork between agents. The paper refers to a centralised agent called the principal that organises other agents, where efficiency can be increased if agents work cooperatively on a task together, or focus on their own individual tasks. This paper may be limited by its age, as many theories may be outdated or irrelevant now that the field has progressed and computational power increased.

## Findings:

The literature reviewed leaves a space for a multi-agent drone simulation system, where agents are tasked with either reaching a goal or countering other drones trying to do so. This is similar in scope to Barton et. al. (2019) in that there are agents trying to evade, but also gives those evading agents an extra task. Muñoz et. al. (2019) showed that the piloting of a drone to a target, using Q-Learning, was possible in an AirSim simulation where obstacles were present, and Pham et. al. (2018) use game theory to enable agent cooperation.

It has already been seen that a reinforcement learning autopilot can hover a Quadrotor UAV using low level controls, in the study by Bou-Ammar, Voos, and Ertel (2010), and Akhloufi, Arola, and Bonnet (2019) used deep reinforcement learning to track a target UAV. These features will not be included in the scope of this project, as they have already been investigated, however they could be added as an extension to the project.

Two teams of reinforcement learning enabled agents, with team rewards and goals, would allow for many different scenarios to be experimented with. Many techniques should be possible to be used and their performance compared, therefore the tool should be resilient to changes and additions. Drone simulation would serve to visualise the tactics of agents and how it could be applied to real life, including obstacles to be avoided. In addition, the studies in literature used many episodes and steps in order to train agents, therefore the system should be able to train for many episodes, and be able to be analysed.

# Requirements

The requirements of the system were difficult to define due to the experimental nature of the project, where it is needed to predict what end users would find most useful. The best solution to this was found using the Must, Should, Could, Won’t (MoSCoW) method. This allowed for identification and categorisation of requirements into different priority groups, and also defined the scope of the system. These requirements detail the expected end product of the project, and the inclusion of each requirement is rationalised.

Intending to follow Sommerville’s requirements engineering process as closely as possible, discovery of requirements through stakeholders was augmented to the discovery of requirements based on reviewed literature, treating the researchers as possible stakeholders (Sommerville, 2016, p.111). This helped to identify the gap in research for multi-agent drone simulation, and more importantly what has already been covered in detail such as the low level control of drones through reinforcement learning as seen in Bou-Ammar, Voos, and Ertel (2010).

Key: (F) = Functional Requirement, (NF) = Non-Functional Requirement.

## Must:

*The system must:*

* 1. Include an API to allow AirSim to be used as part of an OpenAI: Gym environment. (NF)
  2. Be programmed in Python and run in Unreal Engine. (NF)
  3. Include multiple Agents. (F)
  4. Report the Reward function for the Agents, which is based on minimising the distance between the travelling Agent(s) and the target flag, or the opposite for the counter Agent(s). (F)
  5. Show the performance of Agents, visually. (NF)
  6. Form two ‘teams’ of Agents, in order to have adversaries to challenge each other. (F)

*Each Agent must:*

* 1. Have reinforcement learning techniques implemented. (F)
  2. Take the inputs: (F)
     1. The current kinematics of the drone.
     2. The location of the other drones in play.
     3. The location of the target flag.
  3. Output a command to move the drone to a new position. Simplified to cardinal directions. (x+, x-, y+, y-, z+, z-). (F)

These requirements define the core of the system. The ability to apply reinforcement learning to a multi-agent system, and allow for these agents to be simulated as drones, is the key use case of the tool. AirSim and OpenAI Gym are used in order to simulate the drones, and allow for reinforcement learning techniques to be applied with minimal complexity. Python allows for useful reinforcement learning libraries to be used, such as TensorFlow and Keras, and Unreal Engine is used to run AirSim. Multiple agents are required, with the reward of each agent being team based (calculated from whichever targeting agent is closer to the goal state), which is simply reversed for the counter team.

These agents will be controlled using reinforcement learning, where the location of all drones and the destination is observed, and a corresponding action is decided. Considering the reviewed literature, the main reinforcement learning technique implemented should be Q-Learning. As many other techniques seen use the base of Q-Learning, such as Deep Q-Networks (DQN) in the study from Muñoz et. al. (2019), or MinMax Q-Learning used by Selvakumar and Bakolas (2020).

Visual representations of agents will be produced using data from the system, to allow viewing and analysing of different techniques, such as the viewing of trained agents in post using AirSim.

## Should:

*The system should:*

* 1. Include a moving target flag that the travelling Agents are trying to reach. (F)
  2. Allow for the environment to be changed, including the addition of obstacles. (NF)
  3. Allow for the following variables to be easily tweaked: (F)
* Drone Velocity
* Number of Agents on each team
* Location of target flag
* Modes, allowing some observations or action to be activated and deactivated

*The Agents should:*

* 1. Have a number of different reinforcement learning techniques able to be applied to them, in order to assess what techniques are best. (F)
  2. Include the data from the front camera of the drone it controls as an observation. (NF)

The ‘Should’ requirements detail the features that should be implemented upon the core of the tool for additional utility.

A moving target will allow for more realistic environments, where agents must be able to predict the path of the target and act accordingly.

Obstacles in the environment will allow for assessment of object avoidance in the agents, in addition flag location changes will mean agents are trained for new situations. These variance option will extend upon the work of Muñoz et. al. (2019).

Altering the variables of requirement 2.3 can add complexity and variance to training, and can allow for more opportunities for teamwork through changes in difficulty. Adjusting drone velocity in AirSim will change the speed of agent movement, and number of agents on each team is variable between 1 and 2 on the target finding team, as one agent is required, and 0, 1, or 2 on the counter team.

Different reinforcement learning techniques should be included for application on agents to allow for the comparison of performances. The techniques used should take cues from the literature reviewed, Deep Q-Networks (DQNs) saw high usage throughout the studies, as did the Double DQN variant, such as the studies by Muñoz et. al. (2019) and Akhloufi, Arola, and Bonnet (2019). MinMax is another popular technique that saw results in the reviewed papers, as seen in Selvakumar and Bakolas (2020), and an Actor-Critic Network was used by Baker et. al. (2020).

Different modes such as an evaluation mode that can be used after training, can make training and evaluation faster and more straightforward.

Finally, the image from the drone camera can be included to further increase the realism of observations. This was used by many of the authors of papers in the field, applied with a Convolutional Neural Network in order to produce actions with minimal knowledge of the environment.

## Could:

*The system could:*

* 1. Include Car agents for a similar task. (NF)
  2. Include a realistic Unreal environment, to show how the agents adapt to a realistic environment. (NF)
  3. Compare how an Agent trained against adversaries will perform against an Agent trained without adversaries, when the adversaries are removed. (NF)

*The Agents could:*

* 1. Include the ability for drones to ‘stun’ each other when close enough. (F)
  2. Include more complexity for drone movement, allowing the drones to move more freely instead of just in directions. (F)

The ‘Could’ requirements are detailed as extensions to the core of the tool. A Car agent could be included in order to test algorithms the best algorithms for drones against the best algorithms for cars.

Realistic environments would be a true test of realism, especially if combined with use of drone cameras. Combined with drone cameras, this would allow for similar functionality as the studies by Muñoz et. al. (2019), allowing extensions to this paper in more lifelike circumstances.

Allowing saving and reuse of agent data would mean that agents trained in different scenarios can be compared against each other within the system. Drone stunning or similar abilities would make counter team agents more powerful, meaning that target team drones would be unable to take actions for a time.

Complex low level control drone movement has been achieved in studies such as Bou-Ammar, Voos, and Ertel (2010), therefore it has been proven to be possible with reinforcement learning, and this would add another layer of realism and complexity to the agents.

## Won’t:

*The system won’t:*

* 1. Allow for testing in real life. (NF)
  2. Simulate the technology that would be needed in real life, such as for communication between drones for teamwork. (F)
  3. Include any other supplementary tests in order to assess agent intelligence. (NF)

The system is not intended for real life use, which would require much more complex systems for observing and acting, and much more training time, however this could be a possible extension in future. Agent intelligence is not inherently able to be tested in this system, which merely allows for comparisons of the targeting and countering behaviour.

## User Stories:

|  |  |
| --- | --- |
| Story | Usage |
| Train Agents | As a user, I want to train a multi-agent system, and view the training live as a drone simulation. |
| View Training Run | As a user, I want to be able to revisit and view a previous training run that has taken place to identify the behaviour of agents. |
| Change settings | As a user, I want to adjust the settings for a training run, to allow for experimentation and improvement. Drone velocity, number of agents on each team, location and movement of target flag, use of drone cameras should all be adjustable. |
| Evaluate Agents | As a user, I want to evaluate the performance of agents in a more sterile environment, where agents do not learn. |
| Change environment | As a user, I want to change the environment that drone agents are simulated in, to add obstacles or change the play area. |
| Save and Load Agent | As a user, I want to save the data for an agent, to be able to reuse the trained agent another time. |
| Change Reinforcement Learning | As a user, I want to change the type of learning that agents use, in order to experiment and improve. |
| Additions | As a user, I want to add new reinforcement learning techniques to the system for agents to use, without requirement much change of code. |

**Table 1 - User Stories**

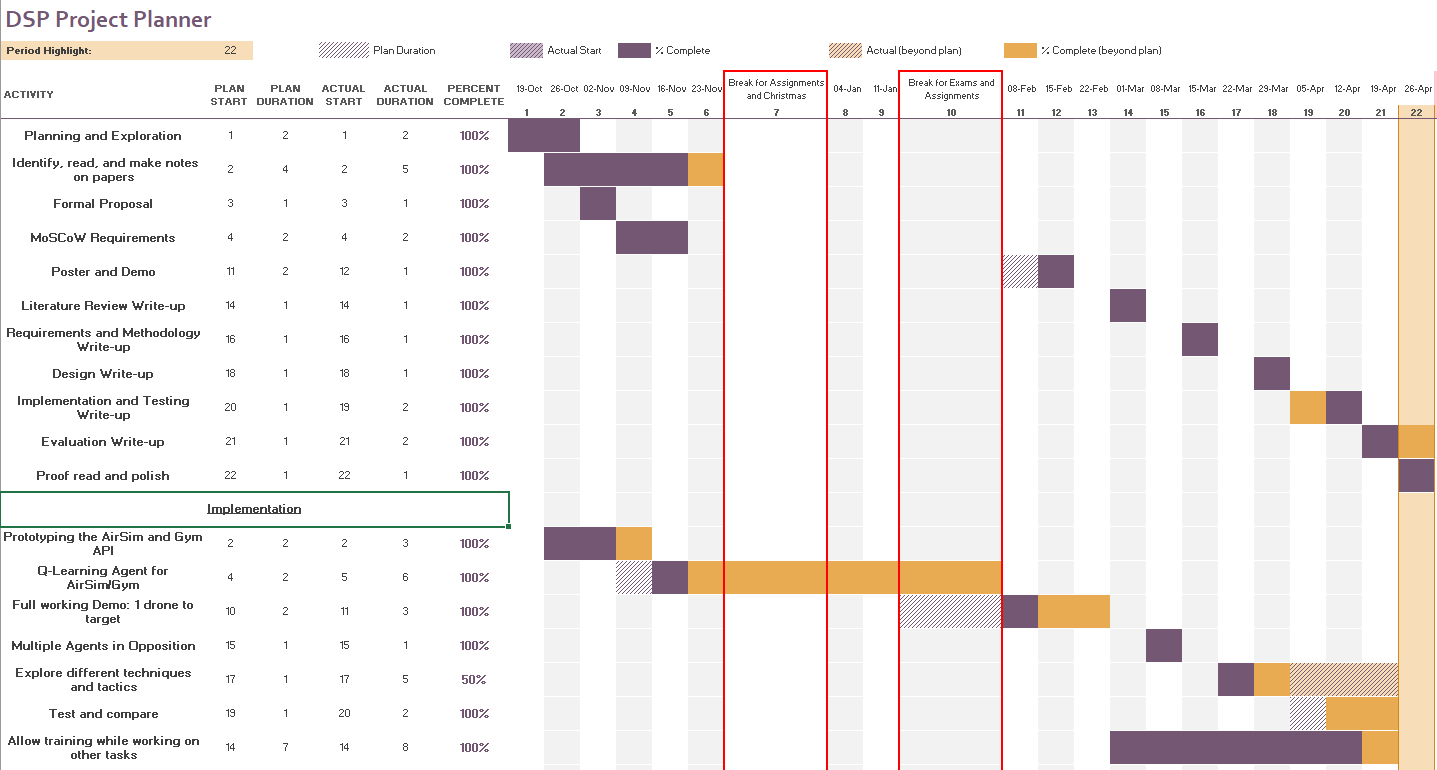
# Methodology

Due to previous experience with Agile methodology during placement, the project was developed with Agile in mind. This included following sprints dedicated to a specific section of work where possible, usually over one week, and recording what happened each sprint in weekly logs.

These sprints were more formal during term 2, as scheduling allowed for more time to be dedicated to the project consistently. These were planned before the Project in Progress day, and alternated between implementation and documentation in order to ensure all areas were progressed.

In addition, detailed system specifications and design documentation have been minimised in favour of faster, incremental development. Incremental design was used throughout, a new functionality would be designed and thought through, implemented, and tested before moving on to the next. This ensured a fully working product was always available.

Full Agile development was not feasible due to the lack of a customer and a team, however the key principles were followed wherever possible.



**Figure 1 - Gantt Chart**

**Figure 1** shows how different tasks were planned to be completed, and how they were actually completed. This plan was first formally created on the 4th January 2021, and last updated on the 28th April 2021.

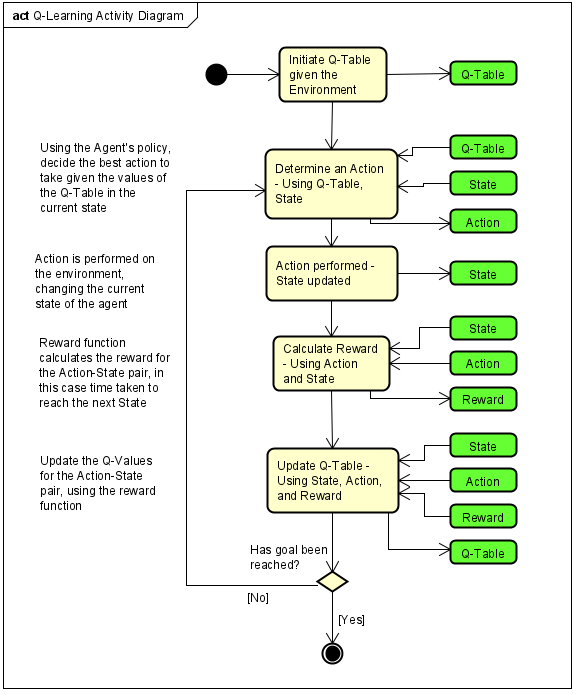
The implementation of the project can be split into stages. The first stage included prototyping and preparation of the environment using OpenAI Gym and AirSim. Stage 2 involved implementing Q-learning for 1 agent moving to target. Next, stage 3 of implementation handled multiple Q-learning agents on teams in opposition. In stage 4, the tool itself was developed into something that is more user friendly and that fulfils the set requirements. Testing of the tool also took place here. Finally, other reinforcement learning techniques were explored during stage 5.

# Design

## Design Diagrams:

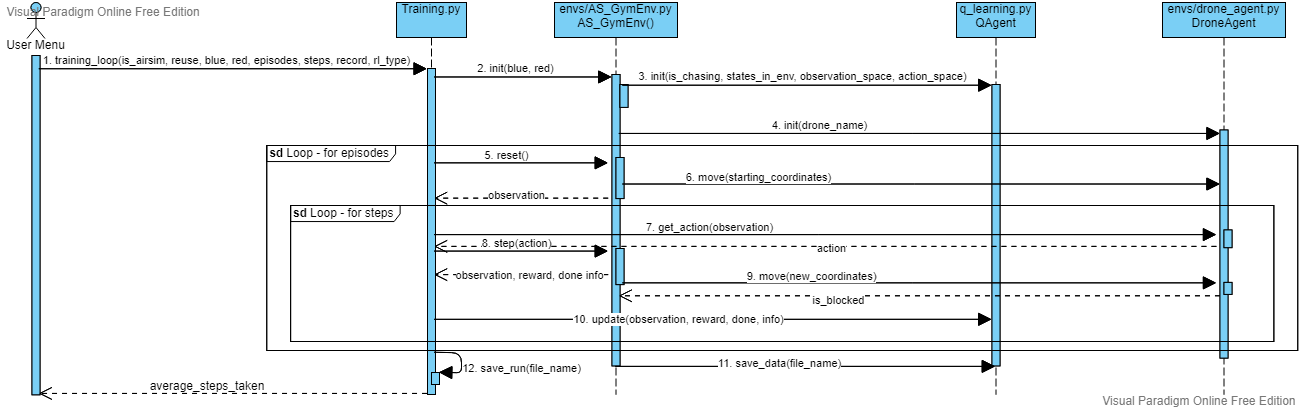
Visual Paradigm Online () was used to create the following diagrams. It is a free online modelling tool which allows for the creation of many types of diagrams.

Correct UML 2.0 notation has been used as a standard for modelling and Object Oriented design has been used. Activity diagrams are used to “capture the work (actions) that will be performed when an operation is executing” (Eriksson et. al., 2004, p.163).

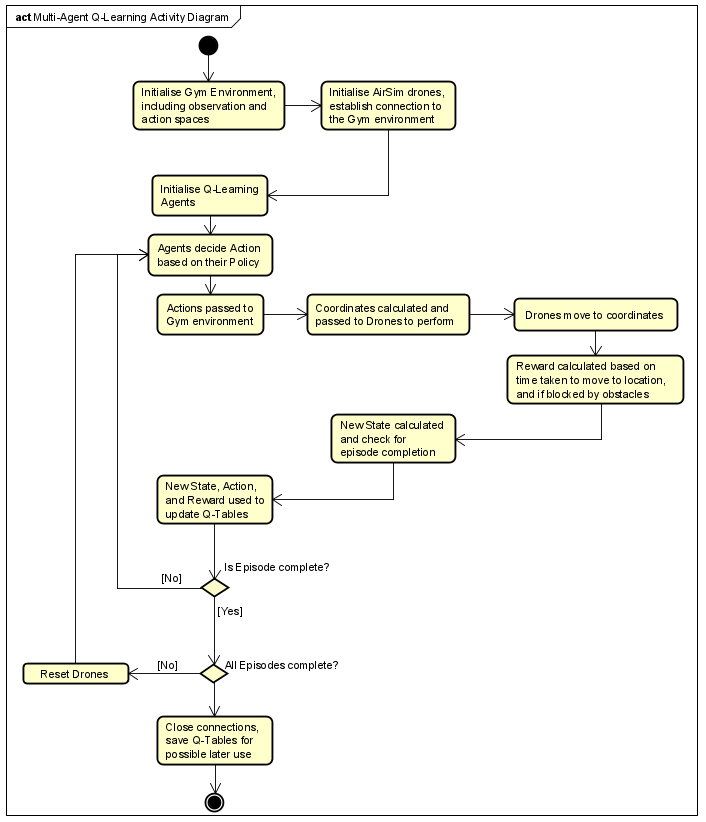
**Figure 2** shows the activity diagram for a single Q-Learning Agent in the tool. The elements on the right side show the variables that are affected by each stage, for example in order to determine an action, the Q-Table values for each action that could be taken within the state must be compared. These stages are described in note form on the left side of the diagram.

**Figure 2 - Activity Diagram showing Q-Learning for One Agent - Visual Paradigm Online**

**Figure 3** follows a similar core training loop through a sequence diagram. This diagram uses the structure of actual software functions and classes used in the tool, and shows how the data flows through each element.

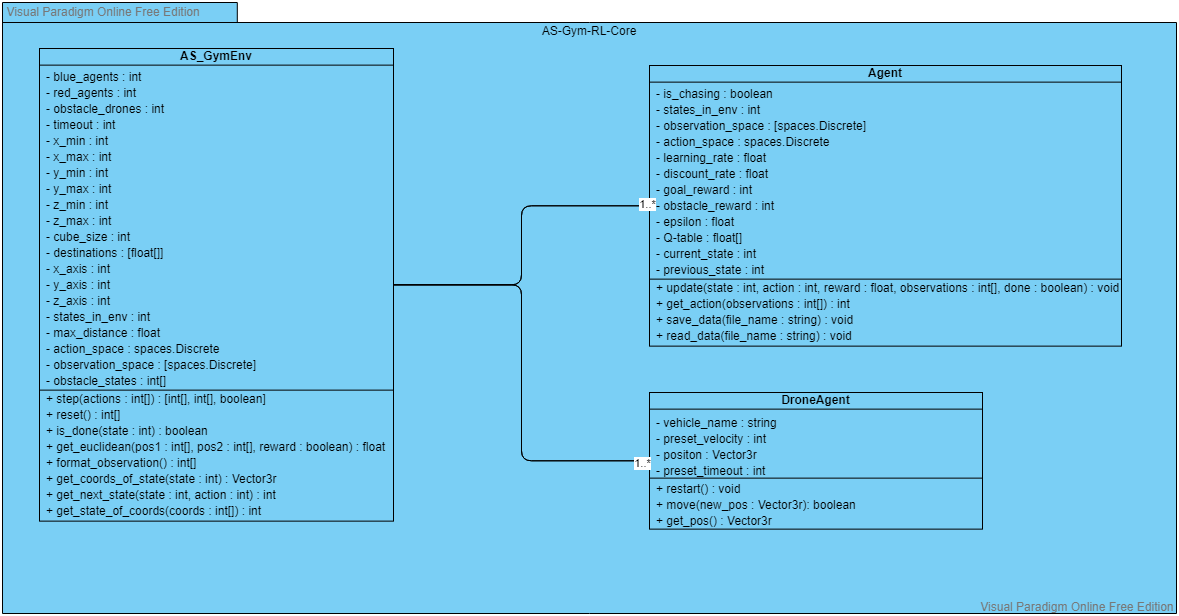


**Figure 3 - Sequence Diagram showing Data Flow of Q-Learning for One Agent - Visual Paradigm Online**

**Figure 4** visualises how Q-Learning is applied to multiple agents in the system, and how this links to the Gym environment and the AirSim drones.

**Figure 4 - Activity Diagram showing the Flow of Tasks in a Multi-Agent Q-Learning System**

The Class Diagram (**Figure 5**) shows how the API between AirSim drone (DroneAgent), the OpenAI Gym environment (AS\_GymEnv), and the reinforcement learning agent (Agent) was designed. A user defined number of Agents and Drones exist within the environment.



**Figure 5 - Class Diagram showing how Environment, Drone, and Agent Classes are Linked**

## Architecture and Design Principles:

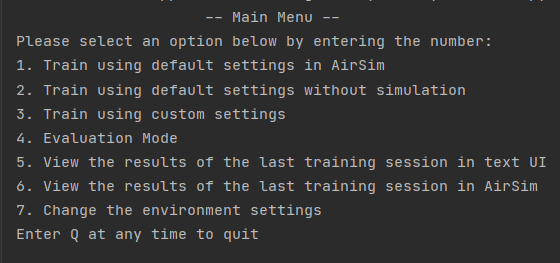
To allow for more opportunities for reuse and modularity, Object Oriented design principles were followed wherever possible and necessary. By using objects many portions of code can be swapped out for new addition or changes, for example the addition of a different reinforcement learning technique requires the get action, update, and save/read data functions to follow the same format as existing methods, then it can be used interchangeably. This change and more have been described in the README.md of Appendix 2, which aims to document the usage and useful design points of the tool.

## Tools and Libraries:

OpenAI Gym (OpenAI, 2021) is an interface to support reinforcement learning tasks. It acts as a library of environments that reinforcement learning techniques can be applied to, and allows for a standard environment to be used across many techniques. It was decided to be used because this matches the ideas behind the tool developed for this project, where different techniques and variables can be tested to assess how to achieve the best performance.

AirSim (Microsoft, 2021) and by extension Unreal Engine (Epic Games, 2021) are used to simulate the flight of multirotor drones. These technologies were used to visualise the movement of agents in a 3d space with the use of drones. In addition, AirSim also allows for realistic environments and extensions into more realistic control of drones, or use of cars instead of drones.

## Menu Designs:



**Figure 6 - Main Menu of the created Tool**

The basic design of the main menu, as seen in **Figure 6**, was designed to be a simple introduction to the tool allowing for users to access the areas to be used quickly. It was developed during stage 2 of implementation, where option of non-simulation training and viewing was deemed necessary for the sake of time of training. The menu serves as a basis for the requirements to be fulfilled within the options, custom settings offers most of the variable and option changes to be trained with. Environment settings such as the Unreal Engine environment itself cannot be changed from within the tool, therefore option 7 points to the instruction in README.md found in Appendix 2.

# Implementation

## Stage 1 – Prototyping and Preparation:

The system was implemented using Python, using the OpenAI Gym Python client and AirSim. Python was used as it can be used by both other technologies, and can be used with packages such as Keras to implement other reinforcement learning techniques.

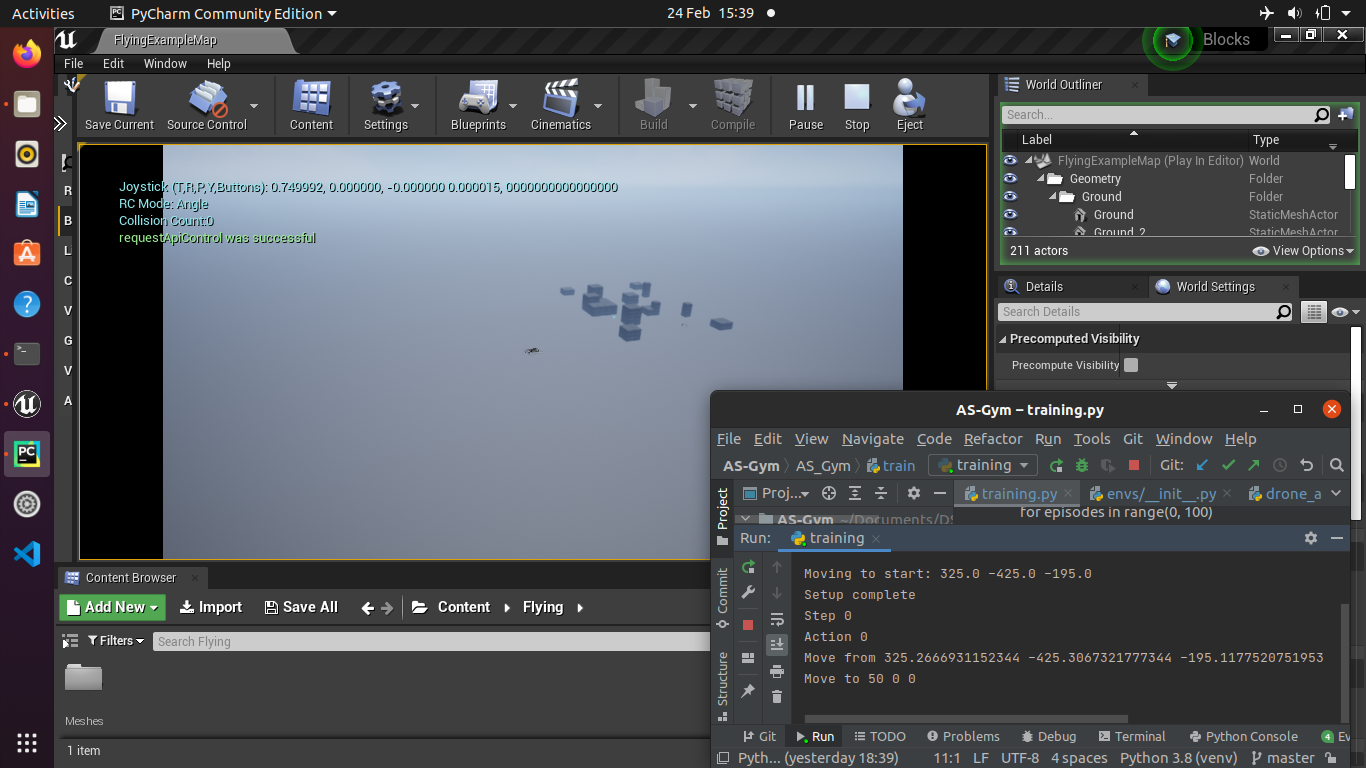
The initial task was to create an API to allow OpenAI Gym and AirSim to run together and communicate with each other. Github repositories from Kjell-K (2019) and Hoangtranngoc (2020) were very useful in this process, where these implementations could be used as a reference for this system.

This process included creating a custom Gym environment that linked into the AirSim client, and connected each Agent to a Drone in AirSim. Custom Gym environments must follow some conventions, where required methods have specific rules, such as step(), where an action must be input, and observation, reward, and done will be returned.

The environment is split into a 3d grid, of user defined size, in order to create states that agents will move between. Functions were created to get coordinates given a state, get the state given coordinates, and get the next state given a state and action.

A multi-rotor client was created to control the drones. It was decided that AirSim movement would be handled using the AirSim method ‘moveToPosition’, which moves a selected drone to a coordinate. Previous research such as Bou-Ammar, Voos, and Ertel (2010) showed that low-level control of drones using Reinforcement Learning agents was possible, therefore this system does not include this as an option as it is not a focus of the project and would have required additional training time.

Requirements implemented: 1.1, and 1.2.



**Figure 7 - A Drone Moving to a Set Location through the AirSim/Gym API**

## Stage 2 – Q-learning for 1 Agent:

The next task was implementing a reinforcement learning strategy for one agent. Q-Learning was decided as a good starting position. This is because it is relatively simple to implement compared to other options, and other techniques can be built upon the framework such as Deep Q-Networks.

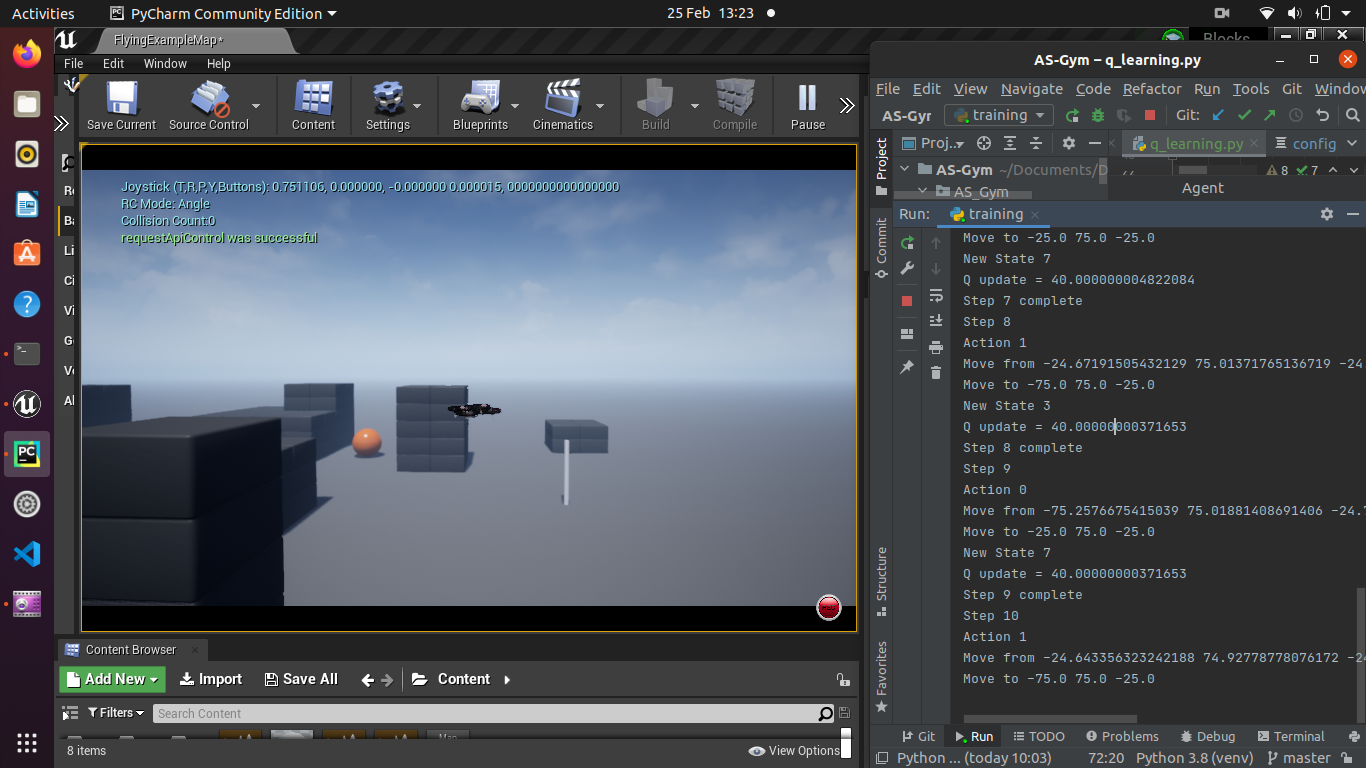
Q-Learning is based around a Q-Table framing a Markov Decision Process, which represents Action able to be taken in States, and the related Q value which is used to decide the best Action to take. To create the Q-Table, the Observation space and the Action space must be decided upon. The Action space was set to 6, the simplest form with one action for each direction, +x, -x, +y, -y, +z, -z. The Observation space was decided to be variable, splitting the play space into cubes of user defined size. The agents will move though the grid cubes in the environment, each representing an observation state. The user is able to set the maximum and minimum values for each axis, and the cube size. The other variable that makes up the Observation space is the amount of destinations, for each destination a separate set of states representing the environment must be trained.

An Epsilon Greedy policy was used to select the action given a state, where the Epsilon value is the probability of selecting an explorative action (random action), instead of an exploitative option where the action with the highest Q-Value is taken.

Once an action is taken and a reward is assigned, the Q-Table must be updated. The Bellman equation is used to update the previous value to represent the new information from the action just taken, and the influence of past actions and rewards.

Q(st, at) ← Q(st, at) + α[rt+1 + γ *max* Q(st+1 ,at+1) – Q(st, at)]

Some unexpected behaviour occurred in AirSim due to the simulation being more realistic than anticipated. For example when using ‘moveToPosition’ the drones still had inertia from their movement velocity, meaning they would often overshoot the target or be unstable. This problem was solved by calculating the coordinates of the next state for each movement, instead of moving a set distance.

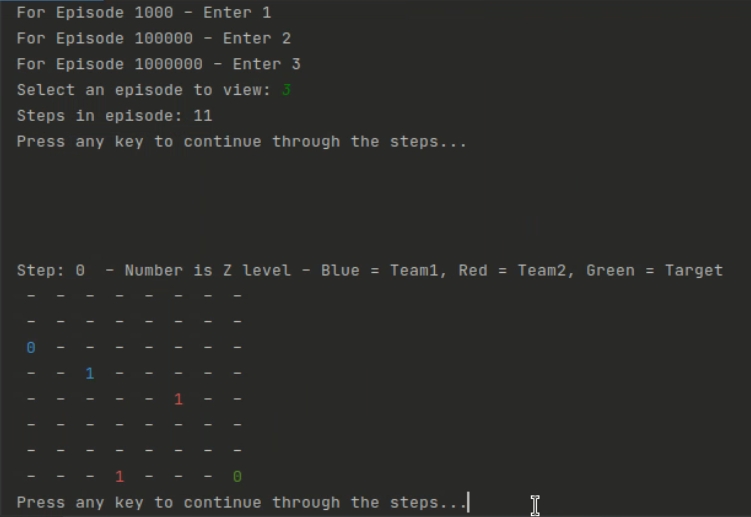


**Figure 8 - A Single Drone using Q-Learning to Navigate**

During implementation, it was clear that training using AirSim was infeasible due to the computational resources needed to run the simulation. In order to run many episodes in a short period of time, another environment was implemented in parallel that did not include simulation. This environment allowed for very fast training of agents in order to view immediate results, however the downside was the non-graphical nature that is only provided by drone simulation.

This led to another development, allowing the saving of training session to be viewed in post. These saved runs can be viewed using AirSim, or through a text based grid, allowing for analysis of training sessions. The text based grid helps the user to view the training sessions in detail, due to the small size of drone in the Unreal Engine, as seen in **Figure 10**.

Requirements implemented: 1.7, 1.8, and 1.9.



**Figure 9 - Text UI Viewer**

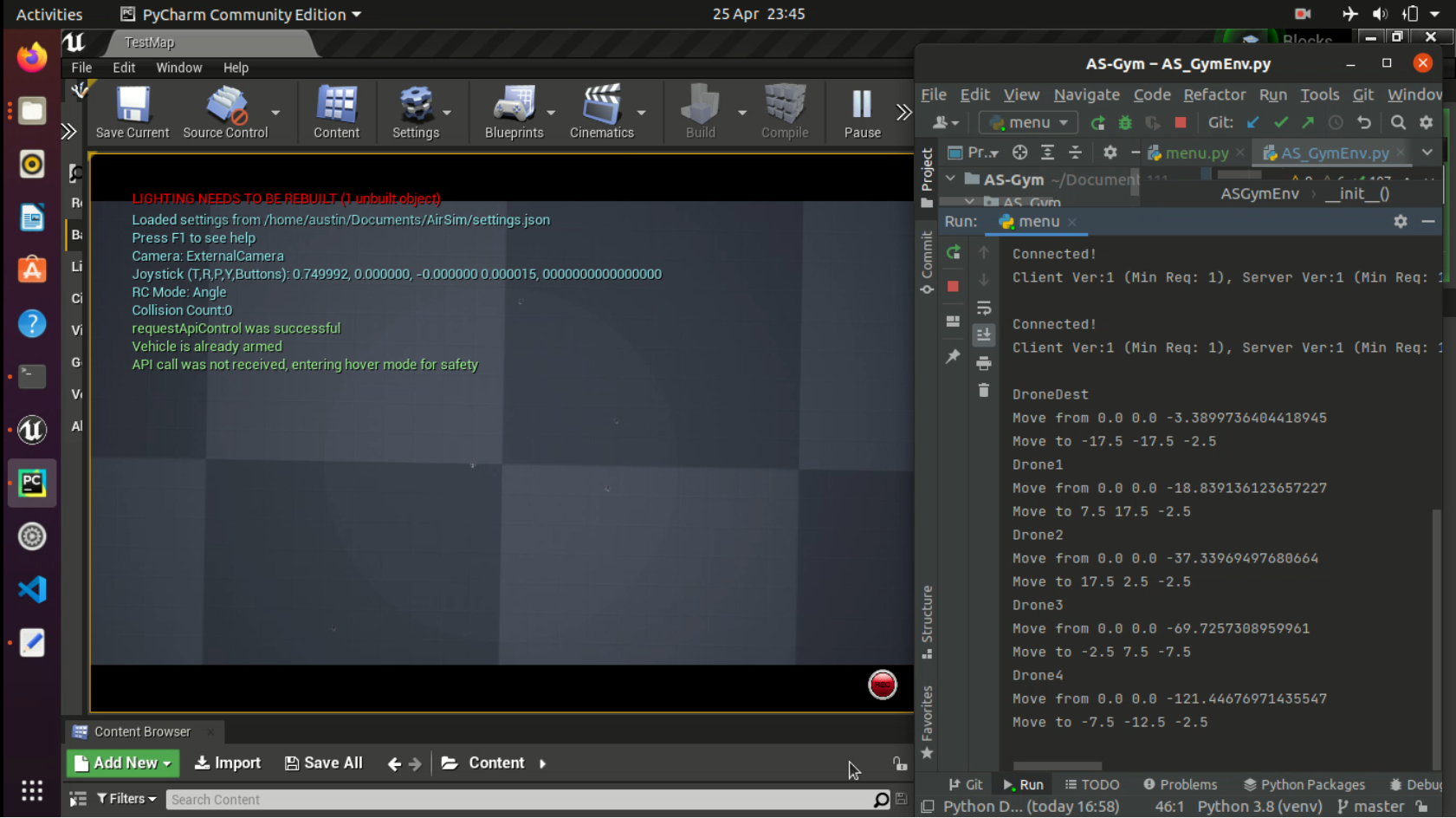
## Stage 3 – Multiple Q-learning Agents and Teams:

The team of counter agents act as obstacles for the evasion team, meaning if they attempt to move into a counter agent then their reward is the same as if they tried to move into an obstacle, and the counter agent receives a positive reward. This also happens if the counter agent move into the same state as an evasion agent, the evasion agent’s action will be treated as if it was a move into an obstacle.

The base reward is calculated by finding the Euclidean distance between each targeting drone and the goal state, using the closest distance, and normalising and inversing the value. This is then used as the team reward for the targeting team, unless another reward is received for reaching goal state or moving into an obstacle. The negative of this reward is given to the counter team, meaning that the reward of the counter team is dependent on the performance of the targeting team. The goal reward for the counter drones is changed from the targeting team, as this team receives a team success reward if the end of the episode is reached without the targeting team reaching the goal state. In addition, each counter agent will receive a goal reward if they successfully block the path of a targeting agent. This system allows for tactics using teamwork and emergent behaviour to be explored.

To allow for better and simpler adjustment of the multi-agent system, the agent classes and drone controller classes have been implemented as part of the Gym environment. This means that the Gym environment is no longer standard, however this is not an issue for this project as it was not designed to be used as a third party environment for Gym. It would be possible to make this standard if it was to be submitted to Gym.

Requirements implemented: 1.3, 1.4, 1.5, and 1.6.



**Figure 10 - Drones (circled) moving to Random Starting Locations**

## Stage 4 – Tool Development and Testing:

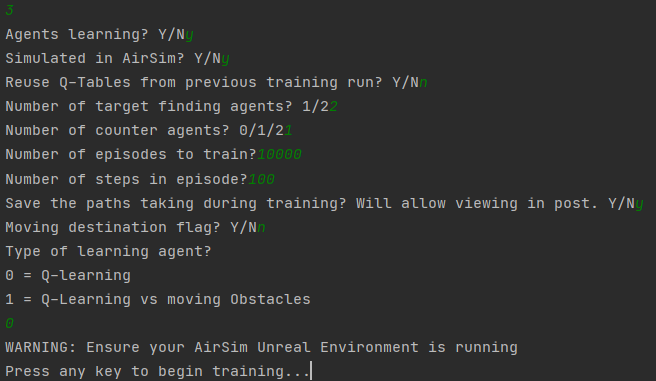
A non-simulated environment was developed to ensure ease of training and testing. Many options were created to allow the user to run custom training sessions quickly and easily. The results of the sessions were saved to a file so that they could be later viewed and analysed. The Q-tables of agents are also saved so that training can be continued, or agents evaluated.

Training with custom settings allows for the most variance. Simulation type, reuse of Q-Tables, number of agents on each team, the types of agent (the reinforcement learning technique, and if both teams use reinforcement learning), episodes and steps to train, session saving, destination, and velocity are all available to be set.

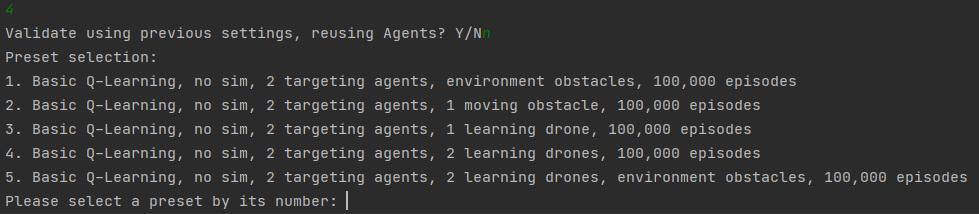
The most difficult part of this stage was implementing all the custom settings and variables into the code, as many checks had to be performed at many points within the code. For example in order to implement the option to have different numbers of agents, checks must be performed at all points where any optional agent is called as otherwise a call will go to an undefined agent. This has effected the modularity and readability of the code in some areas.

Over training, a message is output every 1000 episodes containing the episode number, and how many steps the goal state was reached in. This is to ensure the user know that the tool is running as expected, as there can be very long training sessions.

Requirements implemented: 2.1, 2.2, and 2.3.



**Figure 11 - A Selection of Custom Settings for Training**



**Figure 12 - Preset Training and Evaluation Options in the Evaluation Menu**

## Stage 5 – Reinforcement Learning Techniques:

Finally, more reinforcement learning techniques were experimented with using the TensorFlow and Keras packages. Deep Q-Networks (DQN) and Actor Critic Networks (ACN) were identified to be used as these techniques saw use in reviewed literature.

DQN reinforcement learning combines Q-Learning with neural networks. The value of states are not stored, but are estimated by a state value function, which allows for operation within continuous state spaces. Compared to Q-Learning, in which a State and Action are input, and a Q value is received, DQN uses a single State as an input, and outputs the estimated values of the Actions in the action space. ACN learning splits Actor and a Critic neural networks, where the Critic estimates the Q value, and the Actor determines the Action based on the output of the Critic.

These techniques were experimented with, using resources such as Patel (2017), however some issues arose regarding the shape of different inputs when running multiple agents. In addition, much of the advice regarding these libraries and techniques was outdated, as TensorFlow and Keras are frequently updated, meaning that what has worked previously may not work in the present build.

As the last stage of implementation, this stage was cut short due to time constraints therefore it was unable to be completed. However, the implementation that was completed enabled future addition of more techniques, through the addition of more modular Agent classes that can be defined in only one section of code.

Requirement 2.4 was partially implemented here, however as neither of these techniques were complete it cannot be fulfilled.

## Testing:

Test cases were developed on an implementation basis, where the test case was described before the stage where the requirement was planned to be implemented. Therefore only the implemented requirements appear in the test cases, with a full discussion of requirements in the evaluation.

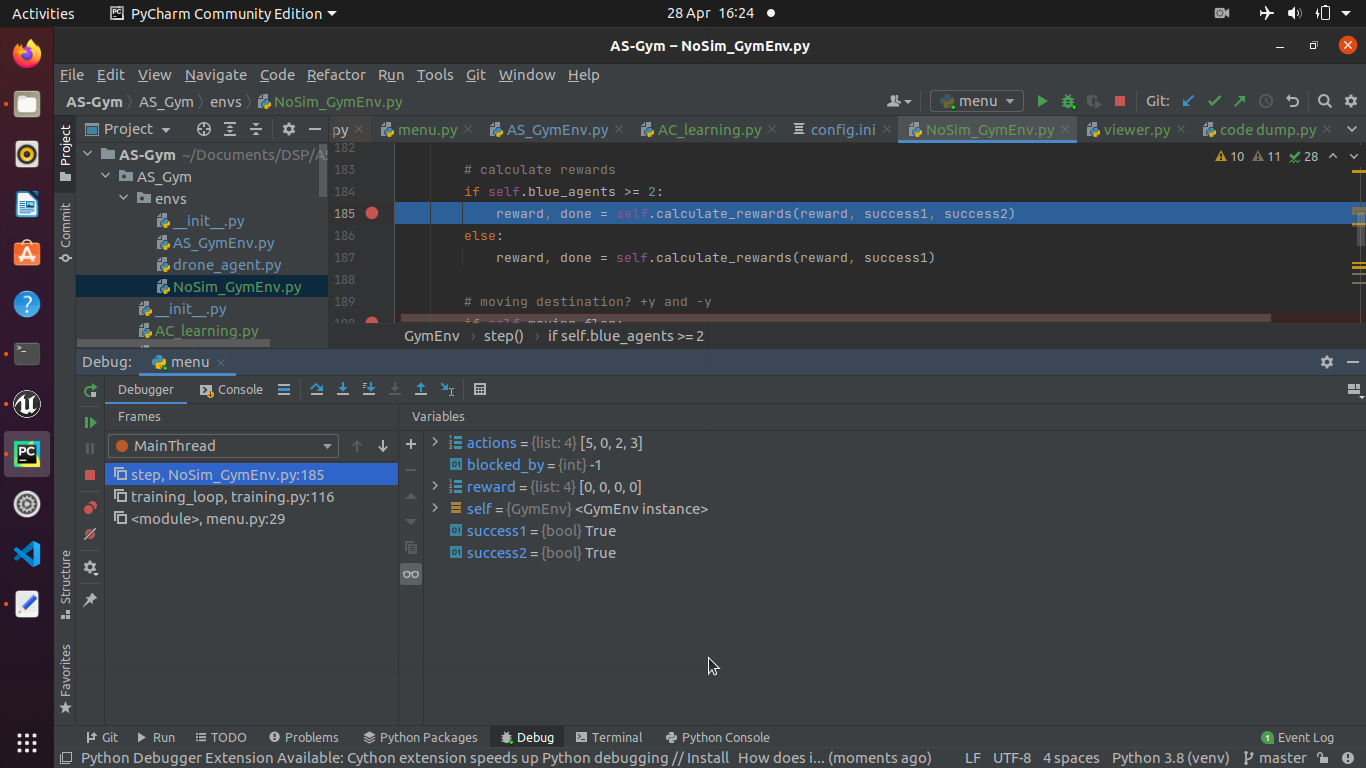
The requirement testing aims to cover all implemented requirements, with some using black box test cases to test the system as a whole, and some white box test cases in order to test the internal structure of the system.

### Requirements Testing:

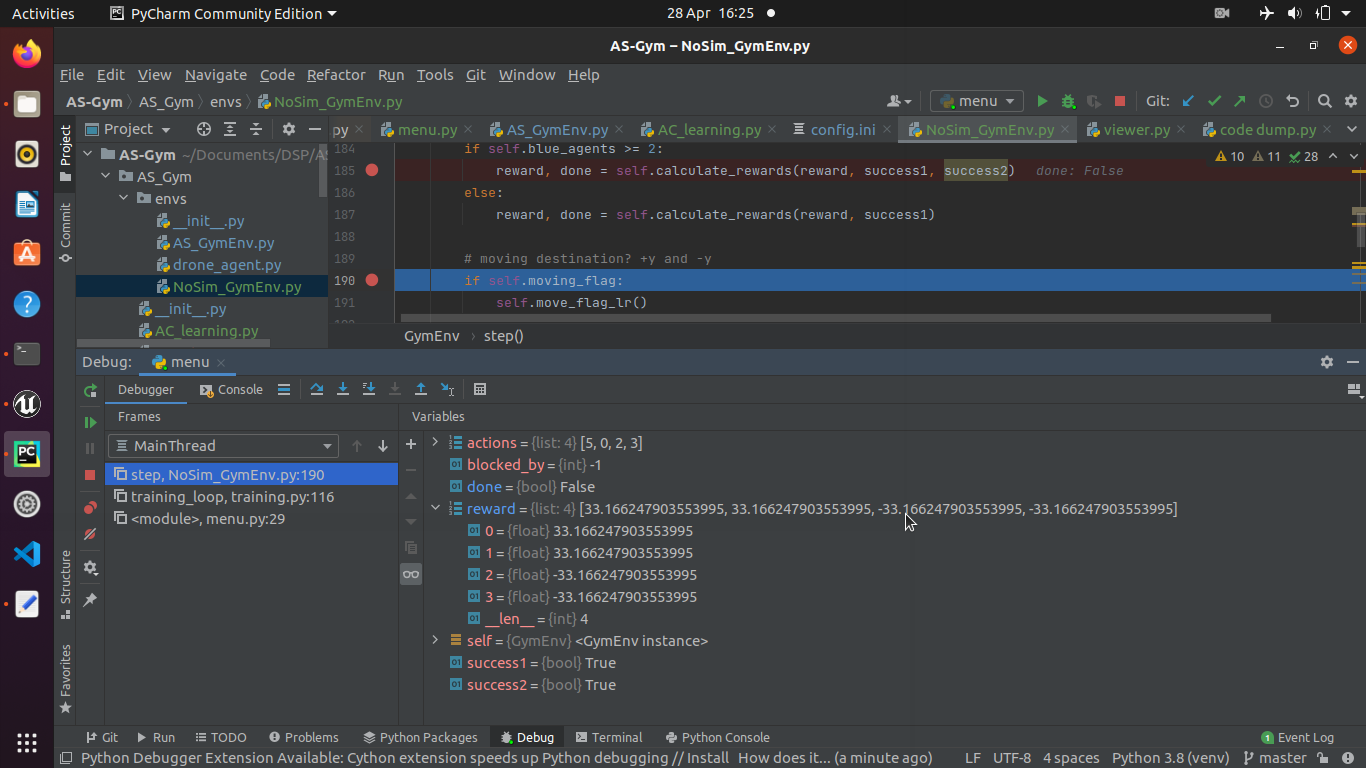
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Description | Inputs | Expected Outcome | Actual Outcome | Requirements |
| R.1. | Initialise an AirSim environment with 2 drones on each team | Run AirSim project in Unreal Engine,  Select option 1 from the main menu | Drones will move to random starting positions | As expected. See **Figure 10**. | 1.1, 1.2, 1.3 |
| R.2. | View a training run in AirSim, and in the Text UI | Run AirSim project in Unreal Engine,  Select option 2 from the main menu, wait for completion,  Select option 6 from the main menu,  View episode 100000,  Select option 5 from the main menu,  View episode  100000, | The same steps will be viewed in AirSim viewer and Text UI viewer | As expected.  See **Figure 17** and **Figure 22**. | 1.5 |
| R.3. | Using a breakpoint, review calculate\_rewards() of GymEnv, reviewing the parameters and the rewards returned | In NoSim\_GymEnv.py, place a breakpoint on calculate\_rewards(),  Note the parameters when breakpoint occurs,  Step over the breakpoint,  Note the reward returned,  Verify the result | Reward returned from calculate\_rewards() is Euclidean based where agent is not blocked or has reached goal state | As expected. See **Figure 13** and **Figure 14**. | 1.4 |
| R.4. | Using a breakpoint, review get\_actions() of QAgent, reviewing the parameters and the actions returned | In training.py, place a breakpoint on get\_actions(),  Note the parameters when breakpoint occurs,  Step over the breakpoint,  Note the actions returned,  Verify the result | Parameters of get\_actions() include states of all drones and the goal state, returns an integer between 0-5 | As expected. See **Figure 15** and **Figure 16**. | 1.8, 1.9 |
| R.5. | Change test settings for number of agents, and run evaluation mode | Select option 3 from main menu,  Use 1 agent on each team, run for 1000 episodes, moving target flag,  Run evaluation mode,  Select option 5 from main menu | There will only be 1 agent on each team, and a moving flag | As expected. See **Figure 19** and **Figure 20**. | 2.1, 2.2, 2.3 |
| R.6. | Change configuration options for drone velocity and location of goal states. Compare to test R.1. | Edit config.ini to decrease drone velocity to 5 from 10, and change the z coordinates of destinations to -7.5 from -2.5,  Run test R.1. | Drones will move to random starting locations, slower than test R.1. and moving to different destinations | As expected. See **Figure 21**. | 2.3 |
| P | Assess the performance of Agents given different settings | Evaluate performance using the Performance Testing in **Table 3**. | Average steps will decrease as episodes increase, | See Performance Testing in **Table 3**. | 1.3, 1.4, 1.5, 1.6, 1.7, 2.3, 2.4 |

**Table 2 - Requirement based Test Cases**

### Execution Screenshots:

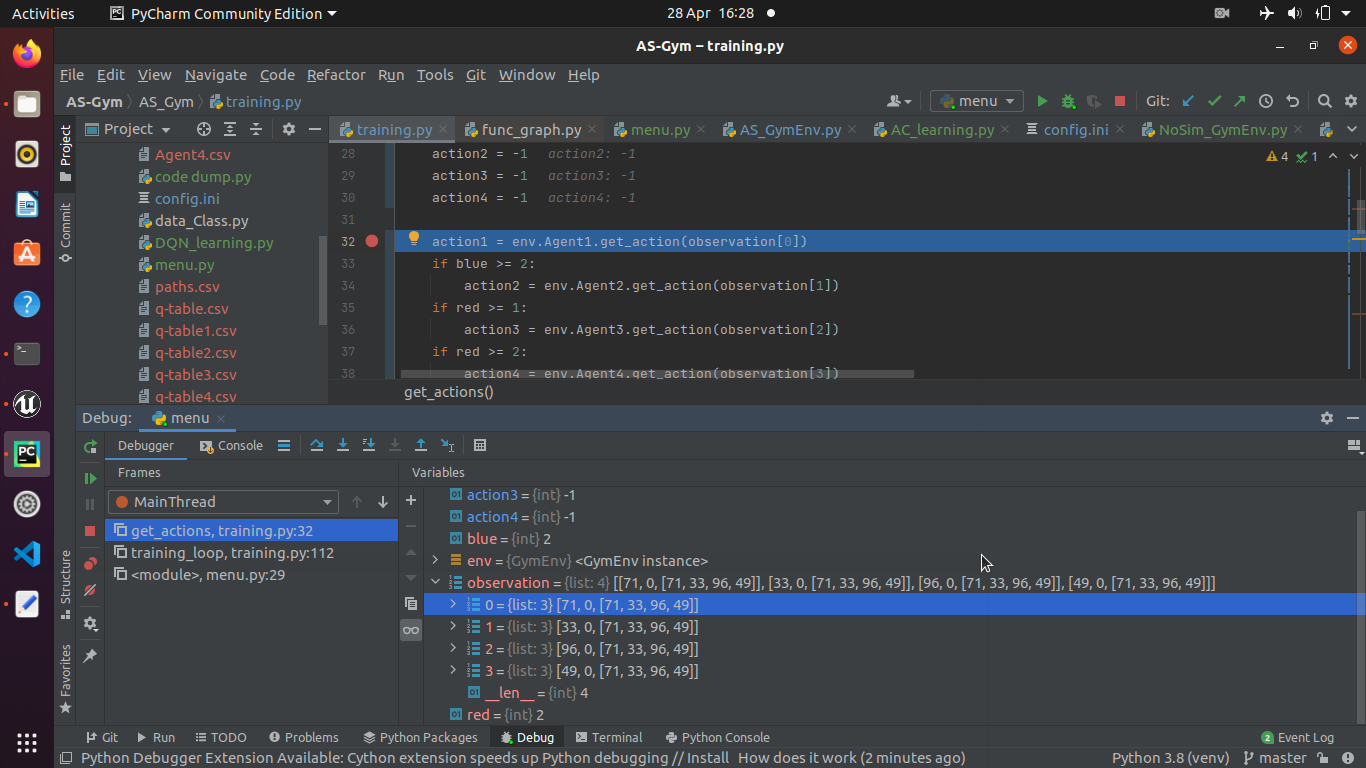


**Figure 13 - Variable Values on Breakpoint for calculate\_reward()**

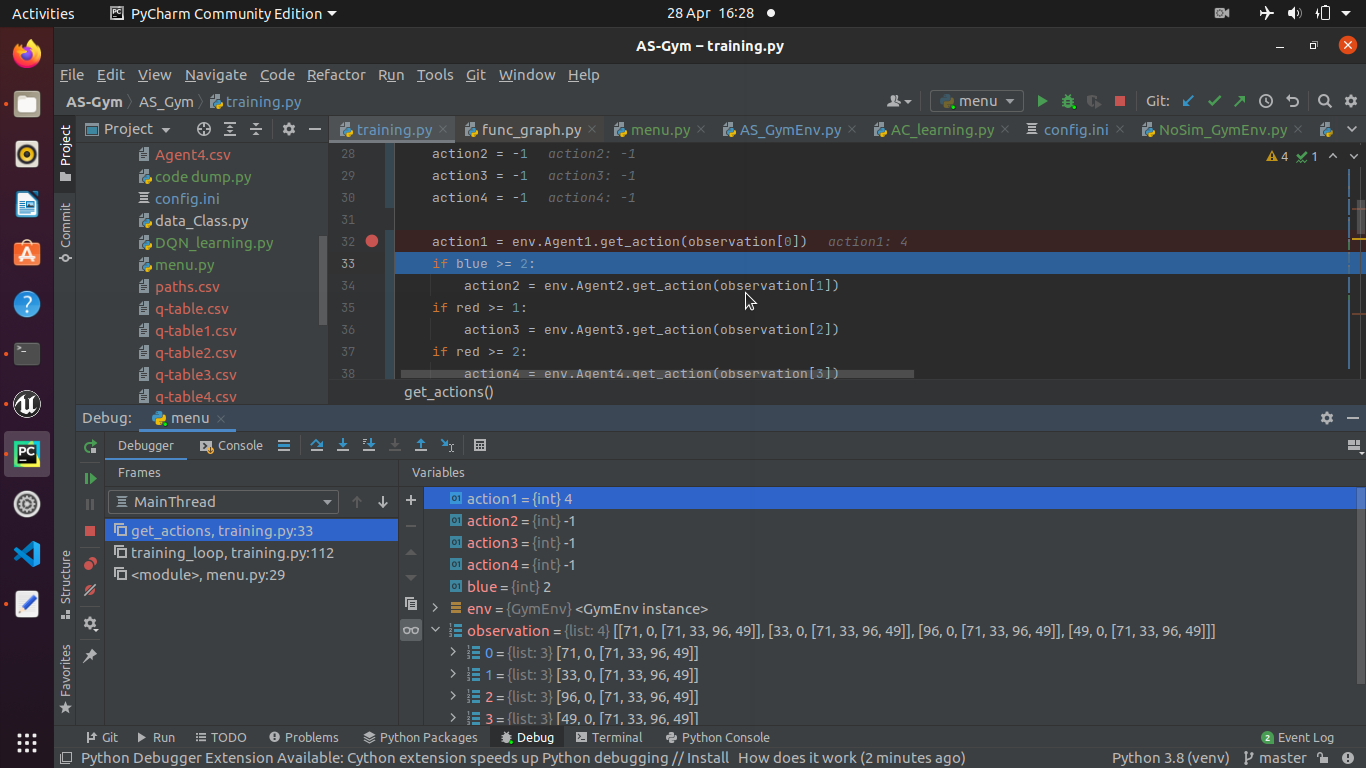


**Figure 14 - Variable Values after calculate\_reward() Breakpoint, showing Euclidean Reward**

**Figure 13** and **Figure 14** show the variable values before and after calculate\_rewards() is executed. Before it is executed, the value of reward is [0, 0, 0, 0], meaning no other reward overrules the Euclidean reward. Afterwards the reward values are based on the distance of the closest targeting agent to the goal state, with the targeting team being rewarded positively, and the counter team rewarding negatively.

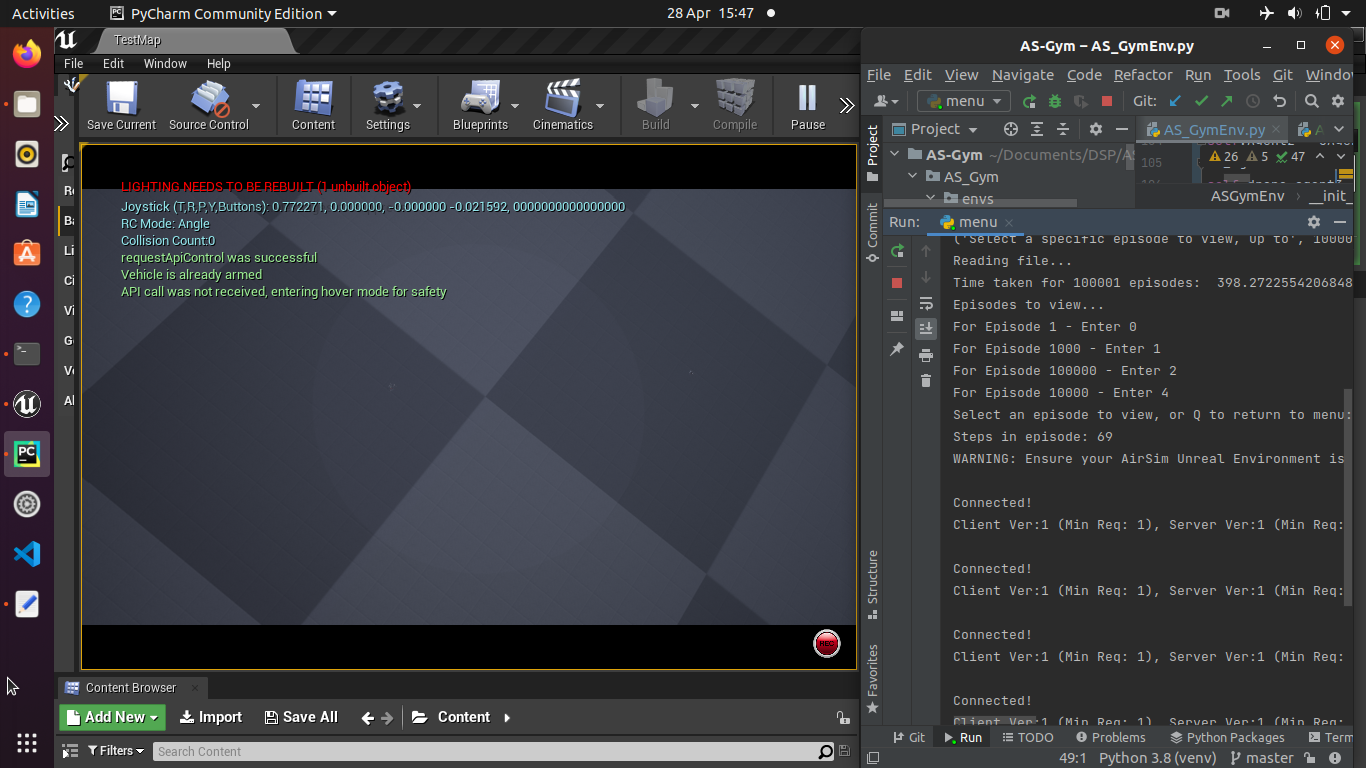


**Figure 15 - Breakpoint on get\_action(), Values of Observations are shown**

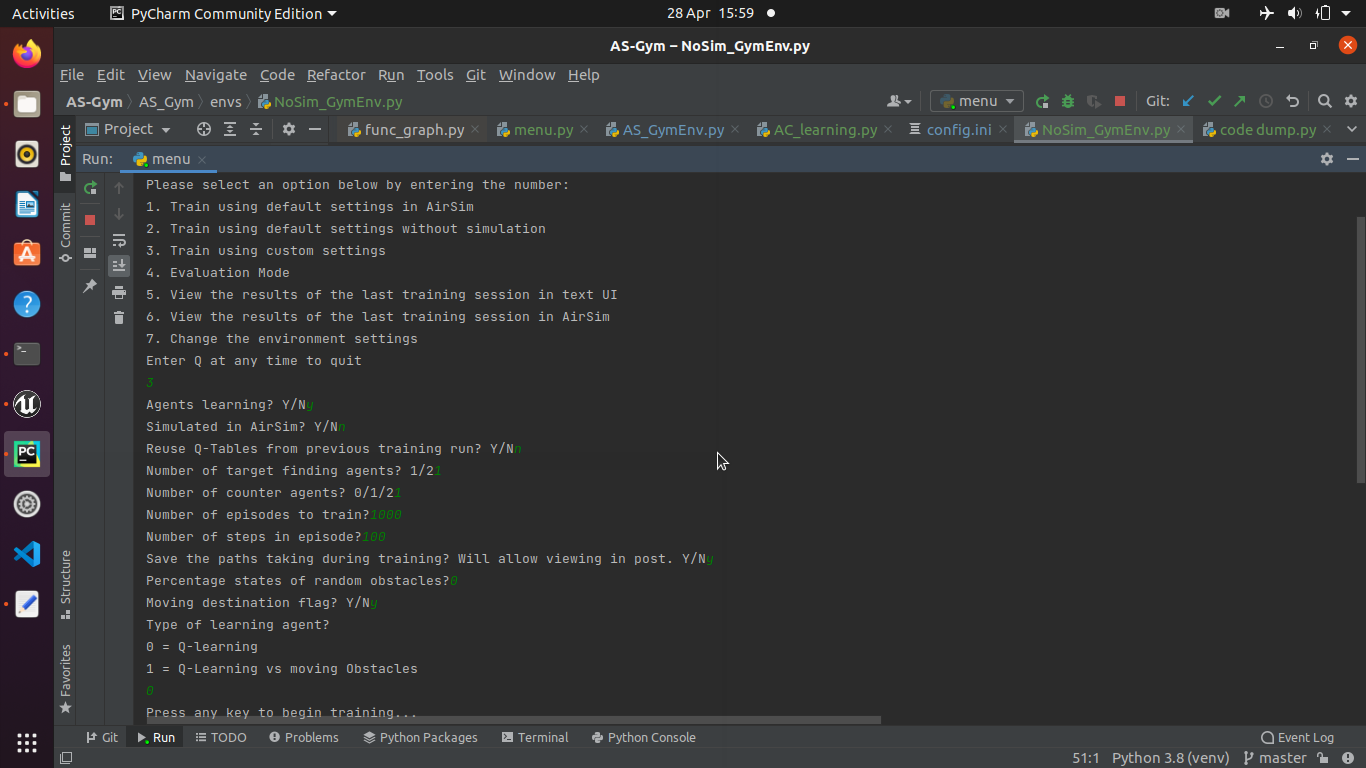


**Figure 16 - Breakpoint showing the value of Action1, returned after get\_action()**

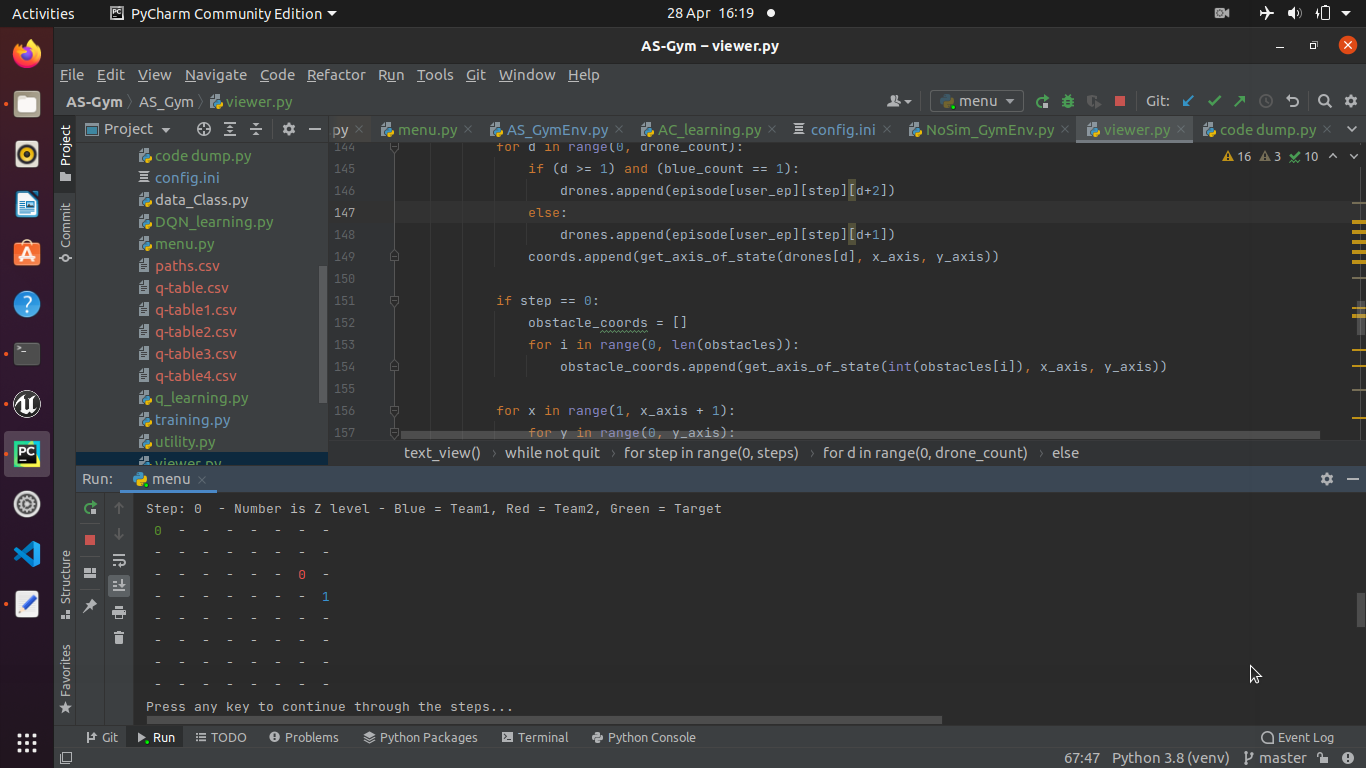
**Figure 15** and **Figure 16** show the variable values before and after get\_action() is executed. The observation is entered as a parameter, and an Action, in this case 4, is produced as a result based on the Q-Learning algorithm.

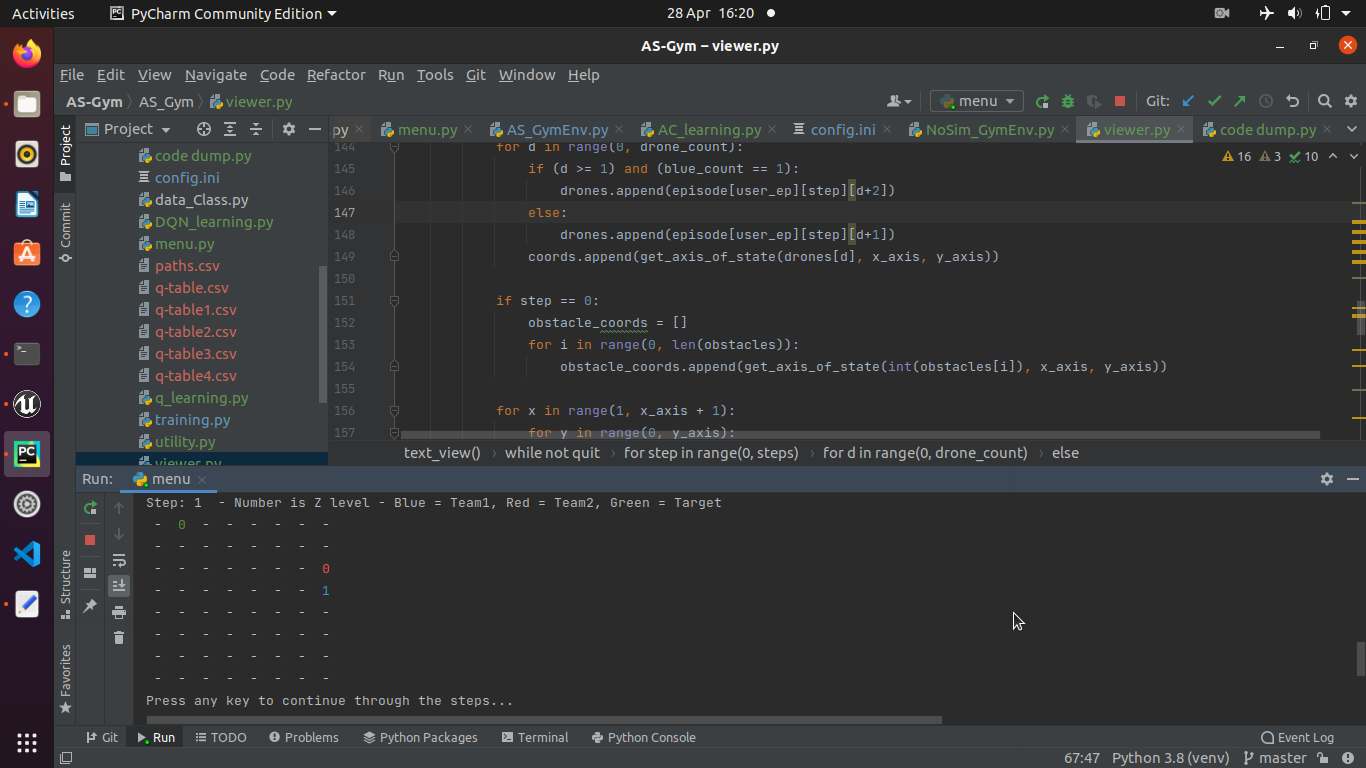


**Figure 17 - AirSim Viewer, executing a previously saved run**

**Figure 17** shows a previous training run being reproduced in AirSim after learning has been completed.

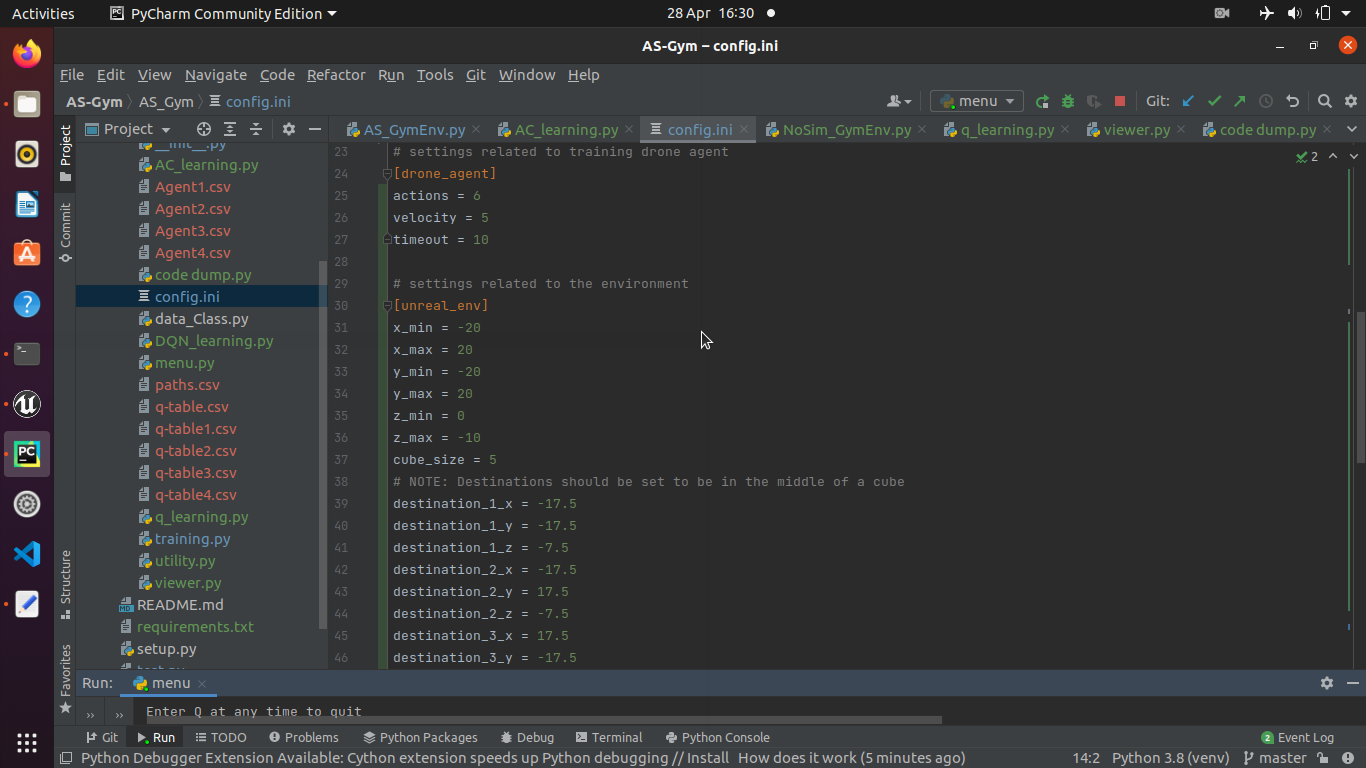
**Figure 18** shows how the different custom settings can be trained with.

**Figure 18 - Custom Training Setup**



**Figure 19 - Text UI Viewer, showing Previous Training Run with a Moving Flag**

**Figure 20 - Text UI Viewer, showing next step of run in Figure 19**



**Figure 21 - config.ini Settings, showing changed velocity and destinations**

### Performance Testing:

Evaluate performance of 2 target finding Q-Learning Agents in a non-simulated environment. All test cases train agents for the episodes defined, for 100 steps each episode. Then these agents are evaluated with the same settings, for 100,000 episodes, to get an average number of steps to reach the target.

These tests were run with the configuration settings set to a learning rate of 0.2, discount rate of 0.2, epsilon of 0.1, goal reward of 999, and obstacle reward of -100.

It is important to note that although agent learning does not continue during evaluation, agent starting locations, goal state locations, and obstacle locations are still randomised.

|  |  |  |
| --- | --- | --- |
| Test | Settings | Average Steps Taken |
| P.1.100 | No obstacles – 100 episodes | 91.77 |
| P.1.1000 | No obstacles – 1,000 episodes | 50.46 |
| P.1.100000 | No obstacles – 100,000 episodes | 32.84 |
| P.1.1000000 | No obstacles – 1,000,000 episodes | 31.62 |
| P.2.100 | Environment obstacles (5%) – 100 episodes | 84.82 |
| P.2.1000 | Environment obstacles (5%) – 1,000 episodes | 67.81 |
| P.2.100000 | Environment obstacles (5%) – 100,000 episodes | 41.85 |
| P.2.1000000 | Environment obstacles (5%) – 1,000,000 episodes | 35.53 |
| P.3.100 | 1 Moving Obstacle – 100 episodes | 91.27 |
| P.3.1000 | 1 Moving Obstacle – 1,000 episodes | 61.85 |
| P.3.100000 | 1 Moving Obstacle – 100,000 episodes | 30.21 |
| P.3.1000000 | 1 Moving Obstacle – 1,000,000 episodes | 29.91 |
| P.4.100 | 1 Counter Agent – 100 episodes | 85.20 |
| P.4.1000 | 1 Counter Agent – 1,000 episodes | 68.10 |
| P.4.100000 | 1 Counter Agent – 100,000 episodes | 35.28 |
| P.4.1000000 | 1 Counter Agent – 1,000,000 episodes | 40.68 |
| P.5.100 | 2 Counter Agents – 100 episodes | 82.18 |
| P.5.1000 | 2 Counter Agents – 1,000 episodes | 59.07 |
| P.5.100000 | 2 Counter Agents – 100,000 episodes | 40.65 |
| P.5.1000000 | 2 Counter Agents – 1,000,000 episodes | 39.65 |
| P.6.100 | 2 Counter Agents + Environment Obstacles (5%) – 100 episodes | 90.04 |
| P.6.1000 | 2 Counter Agents + Environment Obstacles (5%) – 1,000 episodes | 65.31 |
| P.6.100000 | 2 Counter Agents + Environment Obstacles (5%) – 100,000 episodes | 57.78 |
| P.6.1000000 | 2 Counter Agents + Environment Obstacles (5%) – 1,000,000 episodes | 51.56 |
| P.7.10000000 | 2 Counter Agents + Moving Destination – 10,000,000 episodes | 61.68 |

**Table 3 - Performance Testing**

The results of the performance tests show how different variable settings can affect the performance of the system. Over all settings the results over 100 episodes are varied, however this is likely due to the agent policy selecting a random action if there is not ‘best’ action, i.e. an action with a higher Q value as these all start as 0. However some interesting results are found above 1000 episodes trained.

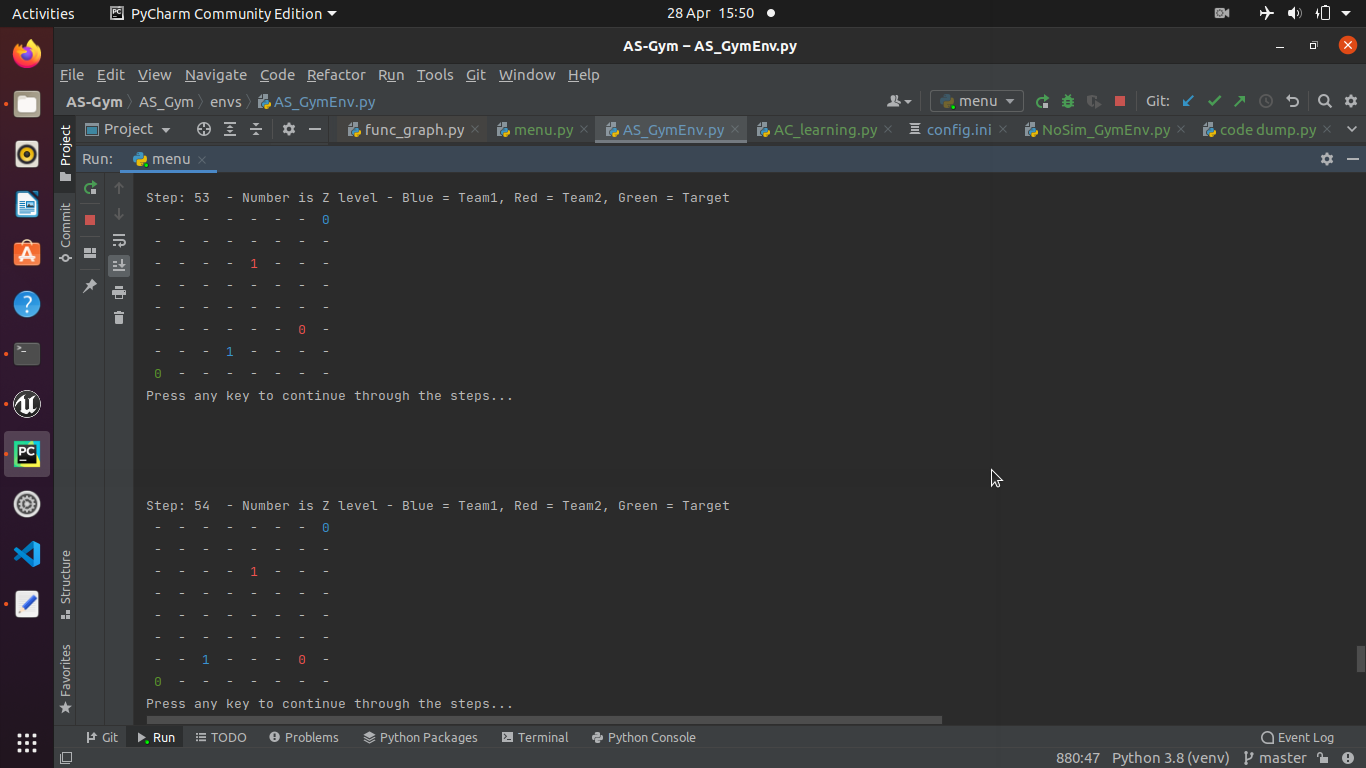
Interestingly P.1 and P.3 tests show similar results, showing that a simple moving obstacle does not pose an issue to a team of two drones, once they have been sufficiently trained over 100,000 episodes. This proves that the agents can learn to avoid an obstacle in minimal extra steps.

In addition, when trained against one counter agent the targeting team performs better over 100,000 steps than over 1,000,000 steps, likely showing that the counter agent learns to better capture and block the opposing team as it trains. At this point in training the counter agent is a more disruptive opponent to a highly trained targeting team than an environment that is 5% filled with obstacles, or far more effective than a simple moving obstacle.

Also of interest is the fact that a single counter agent is as effective as two counter agents over 1,000,000 episodes. This is likely due to the fact that it may take many episodes for a team to fully develop, for example the study from Baker et. al. (2019, p.7) required “132.2 million episodes… over 34 hours of training” to reach a high level of team work, the 4th strategy of 6.

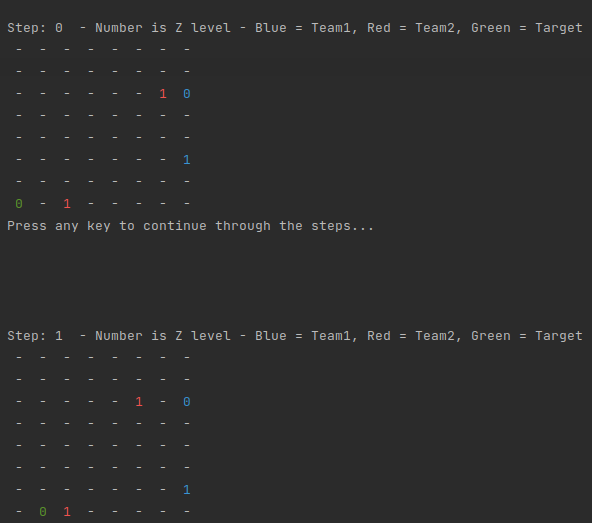
## Encouraging Teamwork:

The beginnings of teamwork can be seen over just 100,000 episodes, for example **Figure 22** show a screenshot of a very interesting run, where both ‘blue’ agents started in the top right corner, and the ‘red’ agents started in the bottom left corner near the goal state. As stated in the key of the viewer, ‘blue’ team represents target finding agents, ‘red’ team represent countering agents. During this run one ‘blue’ agent moved into the top right corner and stop moving while the teammate agent started slowly moving towards the bottom left. The counter drones both continuously move towards the top right corner, clearly they had previously received rewards from blocking the ‘blue’ agent that is there, however this opened a space for the other ‘blue’ agent to reach the goal state.



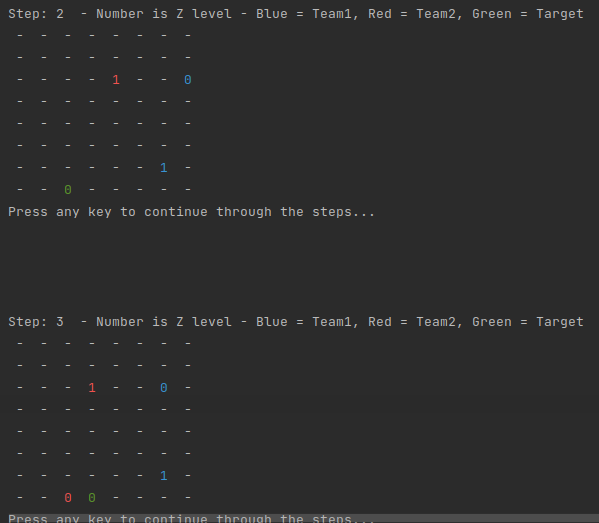
**Figure 22 - Text UI Viewer, showing Previous Training Run of 100,000 Episodes**

Of course this type of behaviour is less likely when agents have not been extensively trained, however it serves as a good example of how teamwork can be used by the agents. The ‘blue’ agent in the top right had received rewards for staying in place previously, when its teammate had reached the goal, and the counter agents had received rewards for blocking either drone, with the immobile drone being more easy to block.



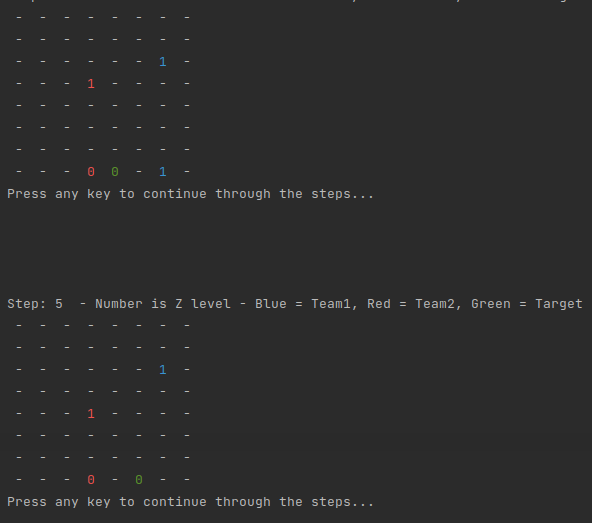
**Figure 23 - Text UI Viewer, showing Steps 0 and 1 from Test P.7 Evaluation**

Counter agent teamwork can be seen in test P.7, where agents trained for 10,000,000 episodes (over 16 hours) where counter agents teams learned more effective blocking techniques, such as following the destination in order to block more directly. **Figure 23** and **Figure 24** show this more direct approach being taken by one counter agent.



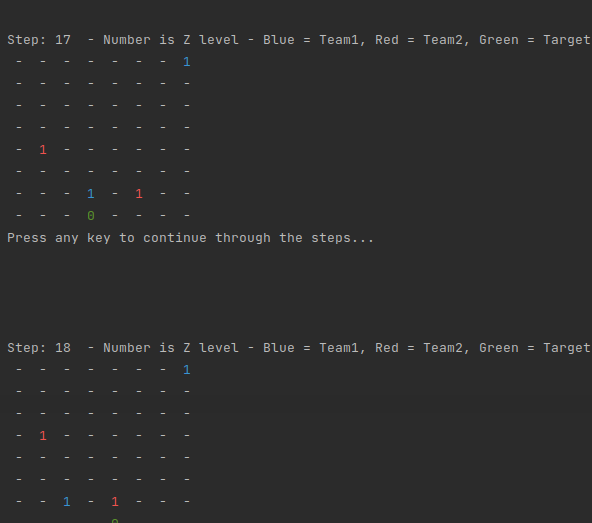
**Figure 24 - Text UI Viewer, showing Steps 2 and 3 from Test P.7 Evaluation**

This strategy seems to have worked, as ‘blue’ agents are apprehensive to directly move to a goal state, meaning they may have been blocked and received a bad reward previously. It also appears that drones took to using the higher Z states to perform most of their actions, we can assume this is part of strategy development, where the ‘blue’ team first moved to this plane and were not blocked. **Figure 25** and **Figure 26** show this behaviour from one ‘blue’ agent, which has many chances to move onto the goal state but does not. Step 5 shows that the agent is directly above the goal state, overlapping on the viewer. This could likely be changed by increasing the discount rate of previous rewards, and increasing learning rate of agents.



**Figure 25 - Text UI Viewer, showing Steps 4 and 5 from Test P.7 Evaluation**

These strategies used are reminiscent of the work of Baker et. al. (2020), where agents created distinct rounds of strategy and emergent behaviour. This appears to have been achieved combination of setting team based rewards calculated using the closest ‘blue’ agent to the goal, punishment rewards for moving into an obstacle, and rewards if a ‘blue’ agent reaches the goal, or if a ‘red’ agent moves into the same state as a ‘blue’ agent. The end of the episode also results in a reward for one team, depending on which team succeeded. The agents will always work towards maximising their own reward.



**Figure 26 - Text UI Viewer, showing Steps 17 and 18 from Test P.7**

## Aims and Objectives Met:

Although some areas of the tool are basic by design, the aims and objectives of the project have been met.

The system allows for multi-agent reinforcement learning, and the Agents control simulated drones in a 3d space. They can also be trained outside of the simulation, and replayed within.

The tool gives users the ability to experiment with learning techniques allowing for Agents to learn to evade, capture, and use teamwork to achieve goals. This was seen in the performance testing, where different settings made the agents perform in different ways.

The objective, ‘To encourage teamwork and emergent behaviour in multi-agent systems.’ is more difficult to define, however the tool does allow for teamwork and emergent behaviour to be explored. This point is discussed further as part of the evaluation.

Finally the use and implementation of reinforcement learning techniques in a simulated environment has been completed, however only one technique has been implemented, where more techniques would have been preferable to allow for comparisons. This would have aided the experimentation of teamwork and emergent behaviour if included.

# Project evaluation

## Requirements and Tool Utility:

The fully implemented requirements fulfil all of the Must requirements, and most of the Should requirements. Partially implemented requirements include 2.4, the implementation of multiple reinforcement learning techniques to assess the performance of each technique. In addition, due to the modular nature of the core requirements, all of the Could requirements would be straightforward to implement on top of the existing project. This is an important feature of the tool, providing opportunities for more techniques and settings to be applied in order to assess their capabilities for teamwork.

Considering that the highest priority requirements have been fulfilled by the tool, it is fit for the intended purpose. It provides a core function that allows for the training and evaluation of multi-agent systems using drone simulation, with many easily adjustable variables and extra features for faster training and clearer viewing. Saving of training and evaluation runs means that comparisons between different techniques and settings are possible, and saving of agents means that agents trained in different conditions can be introduced to new conditions or systems. These systems mean that agents can be trained over long periods of time, and those agents can be analysed in post to view the emergent behaviour displayed.

## Limitations:

The tool developed is limited by a lack of real usage testing. Due to this it is difficult to discern whether it would be a useful tool for the intended purpose due to the impossibility of customer feedback. In addition the only working reinforcement learning technique is Q-Learning, which is fairly inefficient in such an environment and does not scale as well as other techniques.

Another limitation of the tool is its appearance, which is quite raw. This will likely dissuade or confuse possible users, who may believe the product is unfinished or of poor quality. This is similar to the text viewer, which provides a limited function of understanding and analysing multi-agent systems in post. There are some issues such as overlapping objects, which reduce the clarity of the tool. This lack of analysis use also holds the system back from being useful to perform long training runs over many episodes, as the results cannot be easily viewed in a user friendly manner.

## Further Work:

This project could be extended in many ways, and was designed to be a core for larger projects to take build upon. The extension of the tool has been considered, with focus on allowing more reinforcement learning techniques to be implemented in a straightforward manner. In addition, it would be possible to include more drone based functions to the agents, such as using the camera, allowing movement with low-level controls, or new Unreal Engine environments.

The tool itself could be greatly improved with a new UI, which would make it more user friendly and provide a better user experience. A key point of this would be to revisit the viewer function, to include more tools for analysis. In addition visual viewer, perhaps creating short videos of runs, would greatly increase the user experience and allow for more interesting analysis. For example the videos and visual aids produced by Baker et. al. (2020) show an excellent method for increased readability. This improvement could also include automating more of the process such as opening the AirSim requirement when required.

In addition, as previously stated all requirements that were not completed in this instance of the tool would be beneficial and simple to add due to the core of the system being designed with these in mind.

## Reflections:

In retrospect, there are many points of the project that could be improved upon. From the outset the Aims and Objectives should have been more specific in order to have a clear vision of the end of the project. This would have allowed for a more focussed approach from the start. More succinct aims could have been achieved by reading more studies, completing the literature review before setting the objectives, or performing more prototype testing to reign in the ambition of the project as opposed to what will actually be feasible.

The reviewed literature allowed a greater knowledge of the field, and provided insight into the possible functions of the tool. To reach further into the topic at large, the studies reviewed could be over the broader field of multi-agent systems. This would help in identification of specific adjustable variables and how the tool could be applied over more areas.

More specific requirements in places, such as 2.4 where reinforcement learning techniques could have been explicitly stated, would have resulted in a clearer ‘definition of done’ and less ‘creep’ in the scope. This was an issue throughout the project, where not having a clear point to fulfil a requirement led to the continued implementation of other features, including a non-simulated environment, and multiple ways to view a past run. However, these additions certainly add to the function of the completed tool, and provide more varied usage, even though this came at the sacrifice of some defined requirements. In addition, requirements and test cases should have been developed in tandem in order to ensure all requirements were testable and had clear completion states.

The Agile methods used were very useful, however would have been much more efficient if all sprints were able to be fully dedicated to the project, however on many occasions other responsibilities were forced to take priority. This was not taken into account during planning, meaning that development time for one sprint may only constitute a day or less. In future it would aid development if sprints were fully dedicated to one project at a time, with all responsibilities accounted for in planning, splitting large chunks of time between priorities.

Regarding Design and Implementation, the key issue was being too ambitious by using many unfamiliar tools. Unreal Engine, AirSim, OpenAI Gym, TensorFlow and Keras were all used with minimal previous knowledge. In addition, reinforcement learning itself had not been implemented before. This led to many simple mistakes that otherwise could have been avoided, which compounded on top of each other. The lack of understanding of these tools also affected the design process, where it was difficult to design using these technologies without fully understanding how they work, leading to a long period of prototyping. On the other hand, this project has been an excellent introduction into reinforcement learning and the many tools that can be used alongside it.

Throughout the project, feedback has been very useful, both from Phil Legg as supervisor, and from Jim Smith and others during the project in progress day.

Phil provided advice for requirements engineering and suggested using MoSCoW prioritisation, gave a guide for beginning a literature review by collating sources and writing a short summary of key points and limitations. Feedback on the first draft of requirements pointed towards more testable, provable, and specific requirement, which was taken into account for the second draft. However, these points should have been further implemented into the requirements for the reasons previously discussed. In addition, Phil provided many excellent ideas for additions and extensions that helped form the tool into a more modular system that could be used for many functions.

More feedback was received around the project in progress day, including ideas that eventually found their way into the core of the tool. These included the saving of paths of agents in order to later run in AirSim, moving obstacles as the first multi-agent implementation, and scaffolding, which is the training of agents for more simple tasks and adding more complex elements such as obstacles to the trained agents. This advice put the saving of paths and agent data to the forefront, as it would make the tool far more versatile for large training projects. In addition, Microsoft Azure was discussed and researched in order to train agents faster, however this was discarded as the reinforcement learning tools for Azure are currently in preview only.

# Conclusion

From the beginning of the project, the key aim was to create a tool to experiment with teamwork and emergent behaviour. This basic premise has been completed, at the current stage of the system it can be considered a core tool to be built upon, a proof of concept for a larger and more ambitious system. A multi-agent reinforcement learning system has been created, that allows for variables and settings to be adjusted within the system, and can perform training runs on drone simulation software. Runs can be analyses, albeit basically, meaning the results of different techniques can be compared.

In addition, much has been gained from the development of the project itself. In future this will be a project to provide insight into personal assignments, and use the lessons learned here to improve the quality of future work. In addition this project has served as an introduction to the tools and techniques of reinforcement learning, an incredibly interesting field with exciting new possibilities.

# References/Bibliography

## Literature Reviewed:

* Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B. and Mordatch, I. (2019) Emergent Tool Use From Multi-agent Autocurricula. In: Rush, A., ed. International Conference on Learning Representations. Addis Ababa, Ethiopia, 26 April 2020. (no place): OpenReview.net, pp. 1-28.
* Barton, S.L., Waytowich, N.R., Zaroukian, E. and Asher, D.E. (2018) Measuring collaborative emergent behavior in multi-agent reinforcement learning. In *International Conference on Human Systems Engineering and Design: Future Trends and Applications.* Reims, France, 25 October 2018. New York City, USA: Springer, Cham, pp. 422-427.
* Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. and Mordatch, I. (2017) Multi-agent actor-critic for Mixed Cooperative-competitive Environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* *(NIPS'17).* Long Beach, USA, 4 December 2017. New York, USA: Curran Associates Inc., pp. 6382–6393.
* Panait, L. and Luke, S. (2005) Cooperative Multi-agent Learning: The State of the Art. *Autonomous Agents and Multi-agent Systems* [online]. 11 (3), pp. 387-434. [Accessed 15 March 2021].
* Bou-Ammar, H., Voos, H. and Ertel, W. (2010) Controller design for quadrotor UAVs using reinforcement learning. In *2010 IEEE International Conference on Control Applications.* Yokohama, Japan, 8 September 2010. New York City, USA: IEEE, pp. 2130-2135.
* Muñoz, G., Barrado, C., ÇEtin, E. and Salami, E. (2019) Deep Reinforcement Learning For Drone Delivery. *Drones* [online]. 3 (3), p. 72. [Accessed 16 March 2021].
* Akhloufi, M.A., Arola, S. and Bonnet, A. (2019) Drones chasing drones: Reinforcement learning and deep search area proposal. *Drones [online].* *3*(3), pp. 58-72. [Accessed 16 March 2021].
* Itoh, H. (1991) Incentives to Help in Multi-agent Situations*. Econometrica: Journal of the Econometric Society* [online]. 59 (3), pp. 611-636. [Accessed 30 November 2020].
* Çetin, E., Barrado, C. and Pastor, E. (2020) Counter a Drone in a Complex Neighborhood Area by Deep Reinforcement Learning. *Sensors*, *20*(8), pp. 2320-2325. [Accessed 5 April 2021].
* Pham, H.X., La, H.M., Feil-seifer, D. and Nefian, A. (2018) Cooperative and Distributed Reinforcement Learning of Drones For Field Coverage. *Computing Research Repository (Corr)* [online]. abs/1803.07250 [Accessed 11 November 2020].
* Selvakumar, J. and Bakolas, E. (2020) Min-Max Q-Learning for Multi-Player Pursuit-Evasion Games. *Electrical Engineering and Systems Science (eess)* [online]. *arXiv preprint arXiv:2003.03727*. [Accessed 5 April 2021].

## Other Sources:

* Sommerville, I. (2016) *Software Engineering* [online]. Tenth Edition. London, England: Pearson Education. [Accessed 5 April 2021].
* Eriksson, H.E., Penker, M., Lyons, B. and Fado, D. (2004) *UML 2 Toolkit* [online]. Second Edition. Indianapolis, USA: Wiley. [Accessed 5 April 2021].
* Dawson, C.W. (2015) *Projects in computing and information systems: a student's guide* [online]. Third Edition. London, England: Pearson Education. [Accessed 5 April 2021].
* Tran, H. / Medium (2020) *Deep Reinforcement Learning for autonomous vehicles with OpenAI Gym, Keras-RL in AirSim simulator*. Available from: <https://medium.com/analytics-vidhya/deep-reinforcement-learning-for-autonomous-vehicles-with-openai-gym-keras-rl-in-airsim-simulator-196b51f148e4> [Accessed 05 November 2020].
* Zohdi, T.I. (2020) The Game of Drones: Rapid Agent-based Machine-learning Models For Multi-uav Path Planning. *Computational Mechanics* [online]. 65 (1), p. 217–228. [Accessed 30 November 2020].
* Ohta, M. (2003) Direct Reward and Indirect Reward in Multi-agent Reinforcement Learning. In *Robot Soccer World Cup*. National Institution of Advanced Industrial Science and Technology, Tokyo, Japan, 19 June 2002. Heidelberg, Berlin: Springer, pp. 359-366.
* Chuter, A. (2021) British Shell Out Seed Funding for ‘Loyal Wingman’ Combat Drone. *DefenseNews* [online]. 25 January. Available from <https://www.defensenews.com/global/europe/2021/01/25/british-shell-out-seed-funding-for-loyal-wingman-combat-drone/> [Accessed 23 April 2021].
* Georgiadis, P. (2021) UK Regulator Gives Green Light to Delivery Drone Trials. *Financial Times* [online]. 20 April. Available from <https://www.ft.com/content/66487d88-a6b3-4e46-9b8a-00e38e93d3af> [Accessed 23 April 2021].
* Shackle, S. (2020) The Mystery of the Gatwick Drone. *The Guardian* [online]. 1 December. Available from <https://www.theguardian.com/uk-news/2020/dec/01/the-mystery-of-the-gatwick-drone> [Accessed 23 April 2021].
* Kallenborn, Z. (2021) Meet the Future Weapon of Mass Destruction, the Drone Swarm. *Bulletin of the Atomic Scientists* [online]. 5 April. Available from <https://thebulletin.org/2021/04/meet-the-future-weapon-of-mass-destruction-the-drone-swarm/> [Accessed 23 April 2021].
* Kjell-K (2019) *AirGym* – GitHub Repository. Available from: <https://github.com/Kjell-K/AirGym> [Accessed 23 October 2020].
* Hoangtranngoc (2020) *AirSim-RL* – GitHub Repository. Available from: <https://github.com/hoangtranngoc/AirSim-RL> [Accessed 23 October 2020].
* Patel, Y. (2017) *Reinforcement Learning w/ Keras + OpenAI: DQNs*. Available from: <https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c> [Accessed 22 March 2021].
* Patel, Y. (2017) *Reinforcement Learning w/ Keras + OpenAI: Actor-Critic Models*. Available from: <https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69> [Accessed 22 March 2021].

## Tools and Software Libraries:

* Python Software Foundation (2021) Python3 (2021) [computer program]. Available from: <https://www.python.org/> [Accessed 19 October 2020].
* OpenAI (2021) Gym (2021) [computer program]. Available from: <https://gym.openai.com/> [Accessed 19 October 2020].
* Microsoft (2021) AirSim (2021) [computer program]. Available from: <https://microsoft.github.io/AirSim/> [Accessed 19 October 2020].
* Epic Games (2021) Unreal Engine (4.24) [computer program]. Available from: <https://www.unrealengine.com/en-US/> [Accessed 19 October 2020].
* Google Brain (2021) TensorFlow (2021) [computer program]. Available from: <https://online.visual-paradigm.com/> [Accessed 19 October 2020].
* Visual Paradigm (2021) Visual Paradigm Online (2021) [computer program]. Available from: <https://online.visual-paradigm.com/> [Accessed 10 April 2021].

# Appendices

## Appendix 1 – Source Code via GitHub:

The source code of the project and other design files.

Available from: <https://github.com/Austish101/MultiAgent-AS-Gym-RL>

## Appendix 2 – README.md:

Description and detail of the running and working of the developed tool.

Available from: <https://github.com/Austish101/MultiAgent-AS-Gym-RL/blob/master/README.md>

This tool is designed for users with prior programming knowledge, and knowledge of reinforcement learning.

Pre-requisites:

- AirSim and Unreal Engine, for running the drone simulations. https://microsoft.github.io/AirSim/#how-to-get-it

- OpenAI Gym, needed for reinforcement learning environments. https://gym.openai.com/docs/

- Python with packages: gym, airsim, tensorflow.

Menu:

To access the core of the tool, run Menu.py and use the text UI.

- AirSim options refer to the simulation of drone agents, through Unreal Engine and the blocks environment. Ensure this environment is ready before running AirSim options.

- Without simulation options do not simulate agents movements with drones, allowing for faster training. There is no visual to this during training, but the training session can be saved to view later.

- Custom settings allow for many options to be altered, recommended for use once the user is familiar with the tool. Detailed in the custom settings section. Reruns will be offered on the same instance.

- Evalution mode allows for presets to be run. If a training run has been completed it will allow for evalutaion mode to be run. Presets detailed in Evaluation section.

- In evaluation mode learning is not continued over the same training course so the average steps taken over the otherwise same training run can be analysed.

- View in text UI will allow for training runs that have been saved to be viewed using a text based UI. It allows for various episodes to be viewed, and displays a 2d representation of the 3d space for each step.

- The colour represents what is in the space, the number represents the z value. Overlapping objects are possible.

- View in AirSim will rerun the saved training run in AirSim.

- Changing environment settings is handled outside of the menu, refer to the section below.

- Input validation is handled, and the user can enter Q to quit the program in most inputs.

Custom settings:

- Learning agents - defines whether agents will be updated in this run. Differenciates training runs vs evaluation runs.

- AirSim or Non-simulated environment

- Reuse Q-Tables - should the data for the agent be reused from the last saved training run.

- Allows for continuing agent training, or evaluation, or training with different settings. Q-tables are automatically saved after each run, to files Agent1/2/3/4. Previous data will be overwritten.

- Number of agents - defines how many agents will be on each team. There must be one target finding agent.

- Number of episodes - how many episodes should the run consist of.

- Number of steps in episode - how many time steps should be allowed in each episode.

- Episode will end if a target finding agent reaches the goal, or if the number of steps in reached then the counter agents recieve a success reward.

- Saving paths - Will the states of each step and episode be saved to paths.csv. This can then be used to view the run in the viewer.

- Obstacle rate - Only for non-simulated environments. User enters a percentage of randomly placed obstacles in the environment. They remaing for this training run only.

- For AirSim obstacles, the Unreal environment will need to be changed.

- Obstacles in the Unreal environment will only be viewed in post if the same environment is used when viewing using the view in AirSim option.

- Obstacles in the non-simulated environment will only be viewable in the text UI viewer.

- Moving destination - allows for the goal state to be moving during training, +y until the edge of the environment, then back using -y.

- Agent types - Base allows all 'drones' to be agents and learn from experiences.

- Moving obstacles changes the counter team to be moving predictably duing training, +y until the edge of the environment, then back using -y.

Change environment and config settings:

- To edit obstacles, or redesign the simulation, use the Unreal engine. Refer to the AirSim and Unreal engine documentation.

- config.ini

- To change the play space of the simulation, goto the unreal\_env section and change the settings of the x,y,z max and mins, and the cube size.

- NOTE: cube size must be divisible by all max and mins, e.g. 5 given -20, 20, -10, 10 etc.

- NOTE: coordinates do not directly equate to Unreal coordinates, AirSim uses its own system based on the starting drone location. It also uses -z as upwards, and +z as downwards.

- To change the destinations used, goto unreal\_env and change the coordinates for destinations. Must be within the play space defined above.

- Recommended to set these coordinates to be in the middle of cube\_size defined.

- If needed to edit drone settings, goto drone\_agent where changes can be made to:

- number of actions - set to 6 for +x, -x, +y, -y, +z, -z. Rework needed to add more actions.

- velocity of drones - NOTE: too high a speed can cause drones to crash due to the simulation of AirSim

- timeout - how often will a moving drone check to see if it has collided with an obstacle.

- airsim\_settings section is unused, but would be used to define the input of a camera on a drone.

- learning\_settings can be used to change:

- learning rate of agents.

- discount rate of agents.

- the reward given to agents if their goal is complete.

- the reward given to agents if they collide with an obstacle, or are otherwise blocked.

- epsilon, defines the likelehood of exploring/exploting, i.e. choosing a random action, in epsilon greedy policy.

Evaluation:

- At the end of each run, the time taken and the avergage steps over all episodes is displayed.

- In text viewer, the steps for the selected episode is viewed.

- Presets in evaluation mode:

- Basic Q-Learning, no sim, 2 targeting agents, environment obstacles, 100,000 episodes.

- Basic Q-Learning, no sim, 2 targeting agents, 1 moving obstacle, 100,000 episodes.

- Basic Q-Learning, no sim, 2 targeting agents, 1 learning drone, 100,000 episodes.

- Basic Q-Learning, no sim, 2 targeting agents, 2 learning drones, 100,000 episodes.

- Basic Q-Learning, no sim, 2 targeting agents, 2 learning drones, environment obstacles, 100,000 episodes.

- If no runs have been completed on the current instance of the tool when entering evaluation mode, these can be run with learning agents.

AirSim:

- AirSim settings can be changed in AirSim/settings.json, such as:

- Number of drones initialsed, and the starting location.

- Camera type: following, manual, etc.

- NOTE: this will need to be edited for the correct amount of drones on startup.

- Defaults for this tool:

- Manual camera settings: "ViewMode": "Manual",

- In "Vehicles":"Drone1", "Drone2", "Drone3", "Drone4", "DroneDest", all with "VehicleType": "SimpleFlight" and different starting coordinates

Files:

menu.py - Starting menu for training and viewing.

training.py - Runs using defined settings, training\_loop() is the main function here calling to the rest where needed.

viewer.py - Allows viewing using text UI or AirSim of previously saved runs.

utility.py - Used to hold validation() function for input validation, used across multiple files.

config.ini - Holds config settings, see the change environment and config settings section above.

q\_learning.py - QAgent class for q learning agents.

AC\_learning.py/DQN\_learning.py - Partial implementation of Actor-Critic network and DQN reinforcement learning.

- NOTE: adding more learning techniques should be simple, users can use q\_learning.py as a template, as long as new techniques hold:

- update function - input observations, actions, rewards, done, etc.

- get\_action function - input observation, and output action.

- save\_data and read\_data functions .

Agent1/2/3/4.csv - Holds agent data to be reused.

paths.csv - Holds the saved paths of agents through runs.

env/AS\_GymEnv.py - The Gym environment for AirSim simulation.

env/NoSim\_GymEnv.py - Non simulated Gym environment.

env/drone\_agent.py - The drone controller to connect to AirSim through the Gym environment.

## Appendix 3 – Weekly Logs via GitHub:

These logs show the process and timeline of development, including personal thoughts, feedback received, and meeting notes.

Available from: <https://github.com/Austish101/MultiAgent-AS-Gym-RL/blob/master/design/DSP%20Weekly%20Logs.docx>