

CCT College Dublin

ARC (Academic Research Collection)

ICT

Summer 2022

Problem Solving for Industry

Jozimar Basilio Ferreira
CCT College Dublin

Nicholas Chibuike-Eruba
CCT College Dublin

José Fernando González Anavia
CCT College Dublin

Jolomi (Oritsejolomi) Sillo
CCT College Dublin

Follow this and additional works at: <https://arc.cct.ie/ict>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Basilio Ferreira, Jozimar; Chibuike-Eruba, Nicholas; González Anavia, José Fernando; and Sillo, Jolomi (Oritsejolomi), "Problem Solving for Industry" (2022). *ICT*. 29.
<https://arc.cct.ie/ict/29>

This Undergraduate Project is brought to you for free and open access by ARC (Academic Research Collection). It has been accepted for inclusion in ICT by an authorized administrator of ARC (Academic Research Collection). For more information, please contact debora@cct.ie.

CCT College Dublin

Assessment Cover Page

To be provided separately as a word doc for students to include with every submission

Module Title:	Problem Solving for Industry (Capstone)
Assessment Title:	Project Report
Lecturer Name:	Muhammad Iqbal
Student Full Name:	Jozimar Basilio Ferreira, Nicholas Chibuike-Eruba, José Fernando González Anavia, Jolomi (Oritsejolomi) Sillo.
Student Number:	2018389, 2021302, 2021411, 2021276
Assessment Due Date:	16th May 2022
Date of Submission:	16th May 2022

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.



Problem Solving for Industry Capstone

GitHub Link: https://github.com/Jolomi2k9/RL_Game_AI

Table of Contents

Introduction	6
Business case	8
SWOT analysis.....	9
Strengths.....	9
Weakness.....	9
Opportunities.....	9
Threats	10
Conclusion.....	10
Technologies	11
Conclusion.....	11
Software Development Methodology	13
.....	13
Product Backlog	13
Sprints	14
Sprint progress.....	15
Reinforcement Learning	16
Policy.....	17
Reward	17
Dynamics.....	17
Fully connected network	18
Reinforcement learning Algorithms.....	20
Deep Q Network(DQN)	20
Bellman Equation.....	20
Markov Decision Process(MDP).....	22
Q-Learning.....	24
Temporal Difference	25
Deep Q- Learning(DQN)	27
Convolution Neural Network(CNN)	27
ReLU(Rectifier Linear Units).....	28
Action Selection Policy.....	30
Proximal Policy Optimization(PPO).....	31
Central optimization objective behind PPO.....	37

PPO vs DQN.....	39
DQN and PPO learning Example Using VizDoom and Unity	40
VizDoom and Unity learning environments.....	40
VizDoom.....	41
VizDoom environment information.....	41
Unity.....	41
Unity environment information.....	41
Observations on Example results	42
Unity Basic.....	42
VizDoom Basic.....	43
Conclusion.....	46
Misc.	47
Environment training by Nicholas Eruba	47
Appendix	62
Reflective Journal.....	62
Jozimar	62
Fernando Gonzalez Anavia	63
Jolomi.....	65
Nicholas.....	65
References	67

Introduction

Wallace Witkowski of *MarketWatch* estimates that 180 billion U.S dollars was generated by the whole video game market in 2020 (Witkowski, 2021), he also stated that it was bigger than movies and north American sports combined during the pandemic with expert expecting growth to continue after it.

Hence the video game industry is a lucrative sector, and with artificial intelligence already revolutionising the robotic and automotive sector (BOSCH, 2022), this group set out to see the applicability of artificial intelligence using reinforcement learning in the video game industry. To achieve this, we use algorithms that train agents, or Non-player character (NPC) in various game environments without being explicitly coded to do so, this enables developers who don't want to spend time writing coding and debugging various game character to get them trained using the algorithms we write and then used them in their environment.



Figure 1.0 FIFA 22

In video games the artificial intelligence (AI) is the framework that dictates the behaviour of nonplayer characters (Statt, 2019). These framework and techniques have stayed static for years with game developers using well established techniques to achieve the illusion of

intelligence according to Mike Cook of the Queen Mary University London, this method can be tedious and prone to errors that can take the player out of the immersion of the game (Statt, 2019).



Figure 1.1 AI controlled NPCs further immerse the player

This project seeks to significantly reduce or eliminate the amount of time developers spend programming and maintaining non-player-controlled characters while also creating NPC's that behave and interact with their environment in a more convincing way than traditionally programmed AI, increasing customer immersion and enjoyment while reducing development cost.

Business case

This project seeks to use reinforcement learning to develop AI agents used to controlled NPCs in video game worlds that are capable of mastering decision tasks in their video game environments.

Our job will be to develop algorithms and methods that can effectively train the AI agents using Reinforcement learning, which can be used in various gaming environments and scenarios such as racing games and first-person shooters. We then market these agents to video game developers for use in their game worlds. The developer can use our agents as-is in their game without modifications or they can train them further, using our algorithms, to tune the AI agents with various behaviours and capability with minimal or no need to write the code themselves.

With the use of reinforcement learning, our AI agents will learn using trial and error with rewards used to provide feedback to the AI. Over time the AI will master its environment and other AI and even possibly interaction with the human-gamer. This will produce AI controlled NPCs that behave and interact convincingly with their environments and the player, promoting player immersions while reducing developer workload.

Researchers at video game giant Electronics Art(EA), stated in an AI research paper that agents in developed with RL can help tests video games, making sure that they are balanced and engaging to the player (Dickson, 2021).

As gaming worlds become more extensive and complicated, it becomes more difficult for developers to ensure that their game worlds are engaging, enjoyable and bug-free.

Gaming businesses and developers have historically sought new ways and technologies to create new and engaging game worlds and experiences to make their games stand out and draw in players, such as with new rendering techniques made possible by advanced hardware technologies like the PlayStation 5.

According to Ed Thomas, an analyst from Technology Thematic Research , gaming is on the rise *"in the midst of a massive change from a product-oriented to a service-oriented business model New technologies such as 5G, cloud, and virtual reality will usher in a new era of innovation, while new business models such as in-game micropayments are already transforming the economics of gaming."* (GlobalData, 2019)

The gaming industry, which is expected to grow to \$300 billion in revenue by 2025, owes some of its success to the adaption of technological innovations and the invention of new business methods, both of which are developed domestically.

We believe that our AI agents will be an attractive and essential new technology the developers can leverage to create new experiences and make their game development process more resource efficient.

SWOT analysis

We will be using the SWOT analysis to identify the strengths, weaknesses, opportunities, and threats relating to the business prospects of the project.

Strengths

Strengths refer to aspects of the business that we can affect directly and leverage.

- A strength will be that employing reinforcement learning AI produces a more realistic result because of the way in which the AI learns about its environment as opposed to being programmed to do specific things in the video game, enhancing the player experience and making the video game more attractive to potential buyers (Statt, 2019).
- Another strength of the project is the ease in which a developer can implement an already trained AI into a new environment with minimal additional programming as opposed to manually programmed AI, which requires an AI to be manually coded for all their environments which can be time consuming and prone to errors.

Weakness

- A weakness of using reinforcement learning(RL) to train an AI is that reinforcement learning is very complex to learn and master (Kirk, 2017), which means that to effectively train and implement RL you need highly skilled individuals potentially increasing cost of development.
- The initial training of a Reinforcement learning agent requires significant computational resources, requiring the purchase of either expensive AI focused processors or significant cloud-based AI computational power from providers like Amazon Web Service and Microsoft Azure or both.

Opportunities

- Difficulty in applying reinforcement learning to train video games NPC raises the barrier of entry into the space, potentially limiting our competition.
- An increasing market for games will also lead to an increase in development of new games which will, in turn, make the prospect of small and medium developers outsourcing one of the most complex and essential part of a video game to focus on other parts of the game such as level design and character modelling , appear more appealing. This will enable these developers to create more immersive worlds that will attract players (Statt, 2019).
- We can list our AI agents on the online store fronts of the most popular game engines, like Unity asset store and Unreal marketplaces, to reach a wider potential audience of small and medium developers who can benefit from the ease of implementation and maintenance that our AI agents enables but do not have the resources or knowhow to create Reinforcement learning agents themselves.

- Applying RL to train a game's AI presents possibilities for new experiences in the video game world that standard AI cannot provide (Statt, 2019), creating new opportunities for game developers.

Threats

- Limited talent pool from which to hire more software developers with reinforcement learning expertise can constrain our ability to expand and capitalize on demand (Jarvis, 2020).

Conclusion

The SWOT analysis helped to bring aspects that will make the decisions clear for the future of the investment. The insertion of reinforcement learning AI products gives more realistic feeling for gamers and also reduces the time spent by other developers trying to program AIs manually (Statt, 2019), thus, they only need to implement our code in their environment with little to no change needed. Furthermore, the competition is limited and with the growth of the video game market more developers will get the opportunity to be more immersed in the modelling of complex gaming design, creating a more attractive game world for games.

The lack of skilled people with reinforcement learning expertise can be a hindrance to the business though, as this is an ongoing problem (Jarvis, 2020).

Technologies

To achieve the objectives set out by our team, we made use of the openAI Gym framework and algorithms (openai, 2022) and we also used , among others, VizDoom and a Unity ML agents video game environment to train and test our agents.

openAI gym is an open-source toolkit for developing reinforcement learning task and agents, it provides the necessary environments and toolkits that is cross compatible with other popular numerical computation library, like Google's TensorFlow making it relatively easy to import and export training agents (openai, 2022).

VizDoom on the other hand is an open-sourced video game environment based on the popular first-person shooter(FPS) video game, Doom. It was developed primarily to facilitate research in machine learning and reinforcement learning using visual information (Wydmuch, n.d.).

While Unity ML agents learning environment is a video game environment created by unity using the unity game engine for use as a template to for new environment or to test new ML algorithms (Unity Technologies, 2021).

We will use VizDoom and Unity ml agents to develop and test our algorithms , these algorithms can then be used to train AI agents in other game engines environments.

We will be wrapping the VizDoom environment with the openAI gym so our AI agents can learn how to play an FPS game in VizDoom using openAI toolkits.

Conclusion

In this paper, we explained how video games AI controlled NPC's can be improved using reinforcement learning as opposed to traditional programming techniques. By using our AI agents video game developers can significantly reduce or eliminate the amount of time spent programming and maintaining non-player-controlled characters, while also creating NPCs that behave and interact with their environment in a more convincing way than traditionally programmed AI, increasing customer immersion and enjoyment while decreasing development costs.

We will employ open-sourced tools to develop our algorithms to draw from a rich ecosystem of support and save on development cost.

The Video game field has a lot of growth potential and the area of video game development as traditionally been open to new technologies and methods of creating video games.

With the game industry estimated to grow to over 300 billion dollars by 2025, video games and the technologies that support them present a very attractive investment opportunity in 2022. Video games are very profitable business and there is a highly demand for this sector. The only thing needed to achieve the results aimed is a good idea and the help of AI.

Software Development Methodology

Choosing the best methodology to apply for the project needed an analysis on the purpose of the project, the team size and the deadline to have it done. The project needed a lot of team work and constant meetings in order to present progress and make improvements having into consideration the 3 months time to accomplish it.

An incremental Software Development Method such as Agile Framework based on Sprints weekly cycles was the best bet to guarantee the progression and success of the project

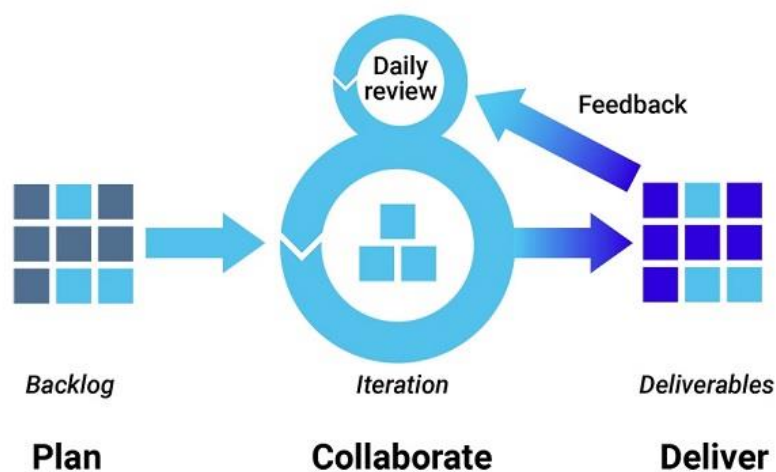


Figure 1 [Synopsys.com/blogs/](https://www.synopsys.com/blogs/)

Product Backlog

Defining the requirements/features of the project, is one of the most important phases, since they will establish the goals, we are aiming to and give a better scope where to start from. Having a list of them also helps to prioritise them so that working on them according to their importance

Product Backlog
Create and implement an open AI environment where our agent could train and learn
Train in different models such as PPO and DQN
Design a custom Unity environment
Models training
Test and evaluation
Hyperparameters tuning
Model tuning
Tuned model training
Re-testing and re-evaluation
Reports and Analysis

Sprints

After the backlog creation, the tasks have been released and assigned gradually to the team members by using Basecamp to establish the To-do activities and showing the progress (Figure 2).

Before the start of a sprint, our team held a sprint planning meeting. This meeting's goal is to decide the sprint plan and set a sprint target. The meeting is divided into two parts. The team evaluate the list of features in the first session and determines what needs to be produced in the following sprint. The following session entails determining which tasks must be completed in order to complete the build. The sprint planning meeting should result in the sprint goal and backlog.

Our team establishes and commits to an immediate goal, and then creates a set of user stories and supporting tasks that meet the team's Definition of Done for each requirement.

The sprint frequency set was weekly, since the best option was to gather before class.

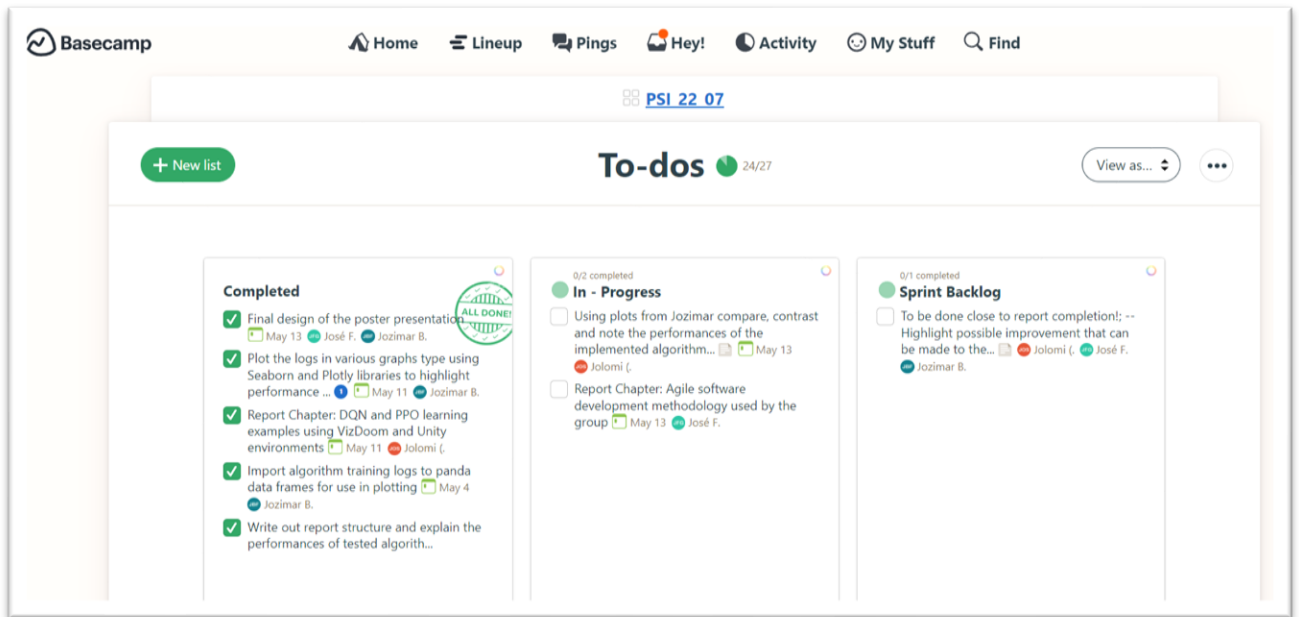


Figure 1 Sprint backlog

Sprint progress

We can identify 6 main sprints during the project management, which corresponds to the coding, testing, and evaluation process. By applying this methodology, we could have a milestones continuous progress that allow us to fix or adapt the code on time to aim the project goals.

Overlay												
Month	March						April				May	
Week	Week09	Week10	Week11	Week12	Week13	Week14	Week15	Week16	Week17	Week18	Week19	
Sprints			Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6				
BACKLOG	-General meeting to set the product's goals											
		Project planning, program installations and dependencies importation										
			Working on different Reinforcement learning models: - PPO - DQN - Custom Unity Videogame									
						Agents training using VizDoom Agents training using Unity						
								Error testing and evaluation				
								Hyper parameters tuning and Tuned agent training				
										Final report, graphs Analysis and project presentation		

Figure 3 Methodology Overlay

Reinforcement Learning

In machine learning, reinforcement learning is a type of model training in which the model, referred to as the agent, through trial and error, learns to perform tasks in an environment it has been placed in. The agent accomplishes this by receiving feedback from the environment in the form of values called rewards and observations, the agent then performs actions in its environment to try and maximize its rewards (Taylor, 2013).

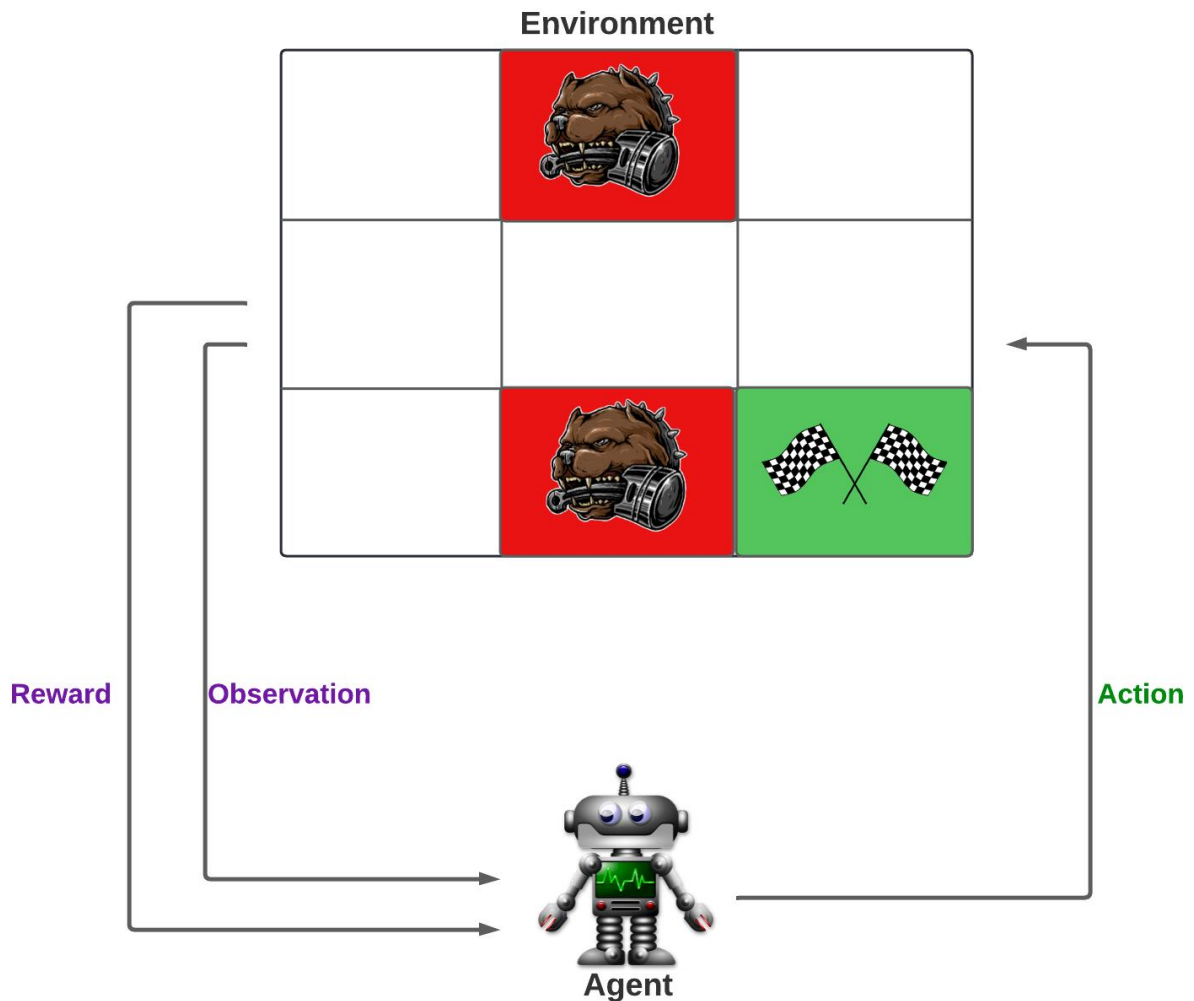


Fig 2.0 Agent – environment interaction

The agent's interactions with its environment are key; the agent performs an action in the environment which causes a change in the environment's state, the new state of the environment is passed to the agent as observations (openAI, 2022).

The state s is a representation for the environment, so when referring to the agent in the maze in fig 2.0, the state is the position (X,Y) of that agent in that environment.

The state can be abstract, like the continuous position an aeroplane that describe the position of the aeroplane , its velocity, angle, distance from the ground etc (Geek, 2018). The state is focus on what about the environment is changing at each timestep due to actions (or inactions) of our agent at each timestep. How the state change is referred to as the dynamics of the environment (Insights, 2018).

Policy

A policy is a mathematical function that takes states as inputs and returns probabilities as outputs.

The policy assigns some probability to each action in the action space for each state. It can be deterministic, meaning the probability of selecting on of the action is 1 and the other is 0 or it can be non-deterministic in which case the probability will be somewhere between 0 and 1. The policy is typically denoted by the Greek letter π (Insights, 2018).

Learning to beat the environment can then be described as the policy π that maximizes our total reward overtime by increasing the chances of selecting the best actions and reducing the chances of selecting the bad actions.

Reward

The reward is a way of telling the agent exactly what it is expected to do. These rewards can be positive or negative and are given to an agent for entering a certain state (Dickson, 2021).

Dynamics

The agent as no idea how its action affects it environment so we use a probability distribution to describe the dynamics. Dynamics can be represented as a probability distribution. Probability of getting state s and reward r given state s and action a . We will not know the value of this until we interact with the environment (Insights, 2018).

The agent also receives some rewards based on its action in the environment, there might also be situations where the agentys actions does not change the environment state or result in a reward.

The agents repeats this cycle where its takes an action and gets a reward and observation state in return. One such action-state/reward exchange is reffered to as a timestep (openAI, 2022), by repeating this timestep it learns about its environment, understanding what actions result in good or bad rewards and states.

This process enables the agent learn what action to take in its environment to maximize its reward.

Fully connected network

A fully connected network can be described as a neural network in which every neuron in one layer is fully connected to every neuron in the other network layer (Mahajan, 2020). Below is the code and image illustration of a fully connected PPO network used in this project.

We supply the searched network architecture, orth_init and activation function using policy kwargs

```
1 policy_kwargs2 = dict(  
2     ortho_init = True,  
3     activation_fn = th.nn.ReLU,  
4     net_arch = [dict(pi=[32, 32], vf=[32, 32])],  
5 )
```

Create our model using searched hyperparameters and policy kwargs

```
1 model = PPO("CnnPolicy", env, tensorboard_log=LOG_DIR, verbose=1,  
2     gamma = 0.9171858127315068, max_grad_norm = 1.5061594927147473,  
3     gae_lambda = 0.9973635408305496, learning_rate = 0.000459048881080786,  
4     ent_coef = 2.4386279564585447e-08, policy_kwargs=policy_kwargs2, n_steps = 1024)
```

Fig 2.1 Fully connected PPO layer code

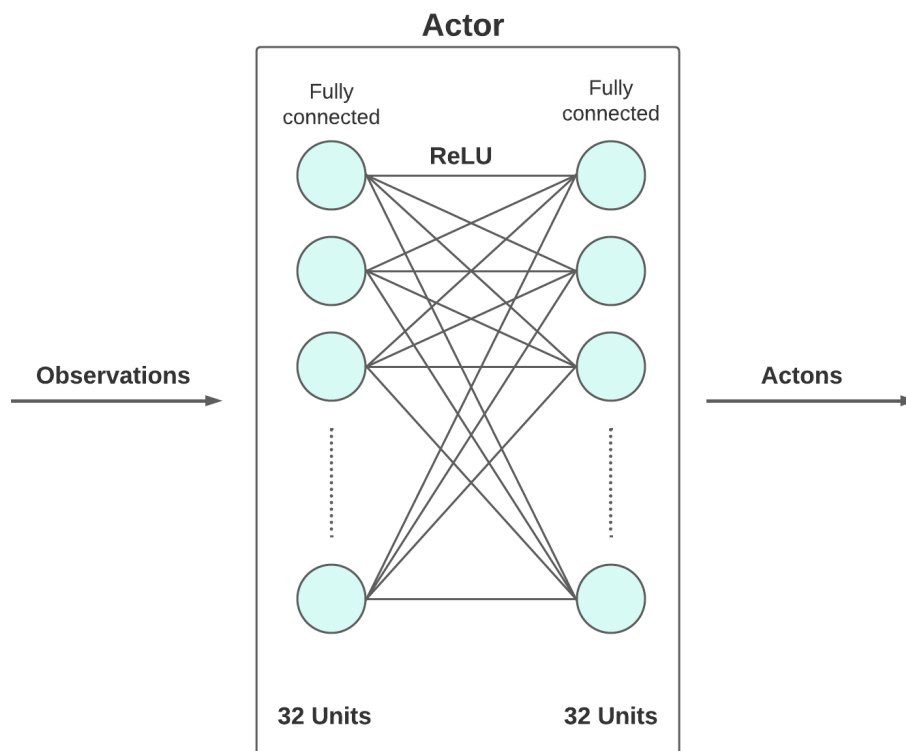


Fig 2.2 Fully connected Actor Network

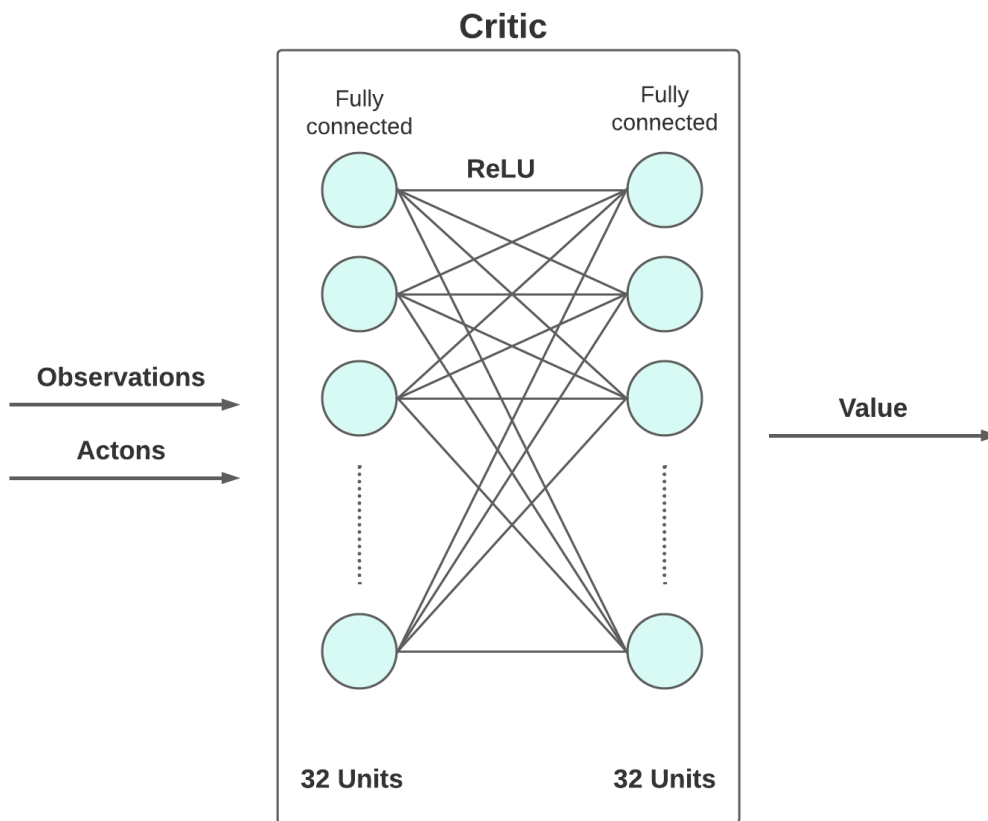


Fig 2.3 Fully connected Critic Network

Reinforcement learning Algorithms

This project primarily makes use of two algorithm to train our agents namely:

- Deep Q Network(DQN)
- Proximal Policy Optimization(PPO)

Deep Q Network(DQN)

Deep Q Network is a neural network that brings the advantages of deep learning to Q reinforcement learning (CHATURVEDI, 2021). To explain DQN we need to first explain Q learning.

Bellman Equation

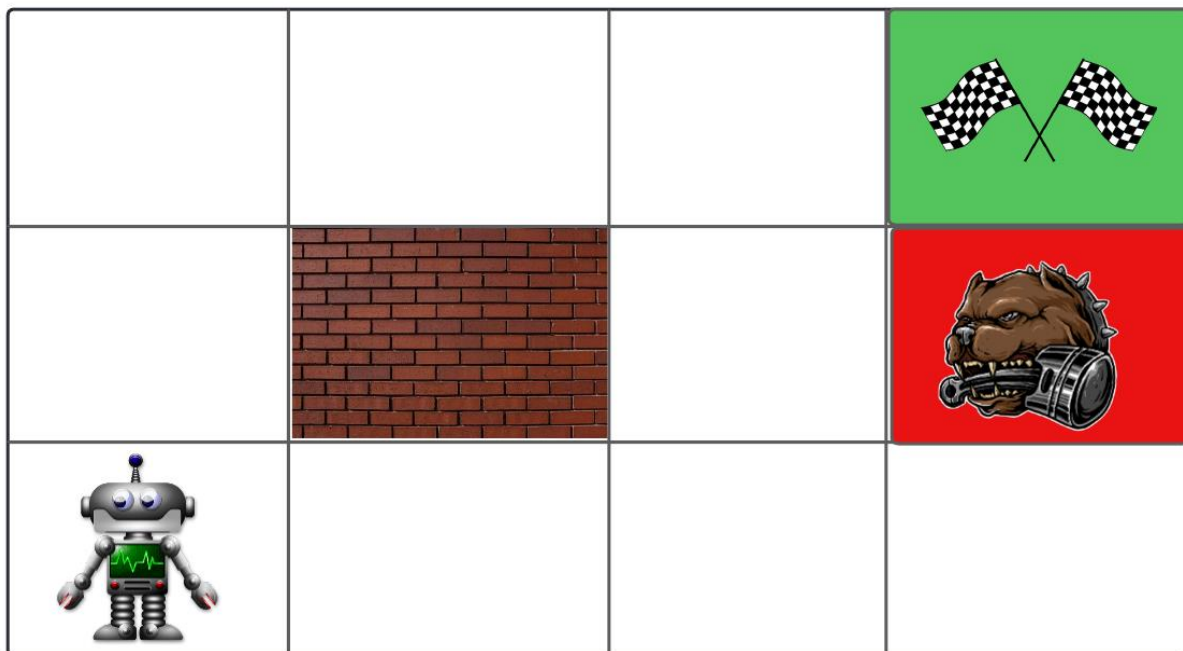


Fig 2.4

We will start with a simple maze in fig 1.1 above, the agent's goal is to reach the finish flag where it receives a reward of +1 which is desirable, the maze also has a hazard that gives the agent a reward of -1 and a wall which the agent cannot traverse.

The agent's objective is to select a series of actions that will get it to the finish and receive a reward while avoiding the hazard tile to maximize its reward.

If the agent ends up in the hazard, it receives a negative reward and the environment resets making the agents starts again, however if the agent ends in the finish tile and gets a reward, it tries to figure out how it got to the finish from the start. It can do this by examining its previous states(which are its previous locations in the maze) that led it to the finish flag. The agent can then assign values to those previous states that it can use to navigate to the finish as shown below.


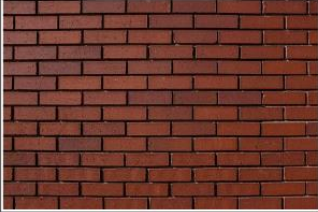


$V = 0.81$	$V = 0.9$	$V = 1$	 $V = +1$
$V = 0.73$		$V = 0.9$	 $V = -1$
 Agent $V = 0.66$	$V = 0.73$	$V = 0.81$	$V = 0.73$

Fig 2.5

The bellman equation gives the above values for each state the agent is in, the maximum is 1 and the agent will take action that leads to this value, we condition the agent to want to be in the max state which is one. In this environment the agent will have to explore the environment to discover these values.

The bellman equation is shown below:

$$V(s) = \max_a (R(s,a) + \gamma V(s'))$$

Fig 2.6 Bellman Equation

- V – is the value of being a current state
- S – is our current state
- $R(s,a)$ – is the reward we get for getting into a state s after taking action a
- $\gamma V(s')$ – the new state we get into , which has a value

If the agent is in state $V(s)$ it looks at all the possible values of $R(s,a) + \gamma V(s')$ maximize its reward it then takes action a that leads to the maximum of all those values.

The γ discounts the values of the states the farther we are from the goal. This helps avoid a situation where all the states leading to our goal is the same or 1, in this scenario if the agent starts in the middle of those states the direction to our goal will not be apparent.

Markov Decision Process(MDP)

Deterministic search

In a deterministic search if our agent decides to go a certain direction, left for example, it will always, with 100% probability, go left (O'Reilly Media, 2022).

Non-deterministic search

On the other hand, in a non-deterministic search, if an agent decides to go left, there is a probability, 10% for example, that it will go right, a 10% chance it will go up and an 80% chance it will go in the left direction it intended (O'Reilly Media, 2022). This is because we are dealing with a stochastic process.

The point is that, in reality, there is some degree of randomness that is not always in the control of the agent, and this is a way for us to represent and deal with that situation (Placeholder4).

A Markov process is a random process in which the future is independent of the past, given the present. Thus, Markov processes are the natural stochastic analogues of the deterministic processes described by differential and difference equations (Placeholder5).

And as stated on sciencedirect.com:

“Markov decision processes (mdps) model decision making in discrete, stochastic, sequential environments. The essence of the model is that a decision maker, or agent, inhabits an environment, which changes state randomly in response to action choices made by the decision maker. The state of the environment affects the immediate reward obtained by the agent, as well as the probabilities of future state transitions. The agent's objective is to select actions to maximize a long-term measure of total reward” (Littman, 2001)

We now modify the bellman equation to take into account the randomness of the environment and arrive with the following equation:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s') \right)$$

Maximum action in that state
The expected value of the new state it will be in
Reward agent gets by performing action a in state s Plus discount factor γ
Probability that when we are in state s we take action a to get to state s'
 Introduced to the Bellman equation to deal with randomness in the environment

Fig 2.7 Markov (Karunakaran, 2020)

Figure 2.5 shows the path out agent will take to its goal in a deterministic environment, however in a non-deterministic environment this will no longer be adequate as decisions made by the agent are no longer 100% certain.

Hence, we update fig 2.5 using the modified bellman equation above to arrive at the following:


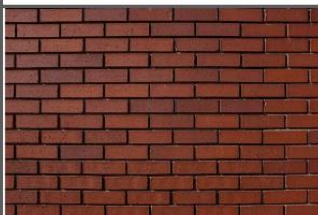


$V = 0.74$	$V = 0.74$	$V = 0.86$	 $V = +1$
$V = 0.63$		$V = 0.39$	 $V = -1$
 $V = 0.55$ Agent	$V = 0.46$	$V = 0.36$	$V = 0.22$

Fig 2.8

The value of each state in figure 2.8 drops compared to figure 2.5 even though we are in the same environment, because we are dealing with a non-deterministic scenario, the states close to the hazard have a percentage chance of ending up in the hazard tile regardless of the agent's intention.

This results in states by the hazard having lower values than they did previously making paths by them worse and results in the agent potentially taking a longer route to the finish to avoid states by the hazard tile.

Q-Learning

In Markov decision process we try to make decisions to maximize future rewards based on our current state and all future states moving forward, our previous states are not considered (Karunakaran, 2020).

With Q learning, we are looking at the value of each action to determine the best way forward instead of the value of each state that is used in MDP (Karunakaran, 2020). Q represents the value of the action and is represented by the formula below:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s,a,s') V(s'))$$

Reward for performing action in a given state
Expected value of the new state

Fig 2.9 (Karunakaran, 2020)

The agent gets a reward plus the *expected* value of the state it will end up in (because this is a stochastic process, we introduce probability).

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s,a,s') \max_{a'} Q(s',a'))$$

max of all Q values available in new state

Fig 3.0 Q learning (Karunakaran, 2020)

We end up with a recursive formula for the Q value. The agent does not look at the possible states, it looks at the possible actions and based on the Q value of the action, decide which action to take.

The Q value of an action is now the reward of the action plus the discount factor times the maximum of all the possible Q action in that state, and since this is a non-deterministic environment, all other possible states need to be examined.

Temporal Difference

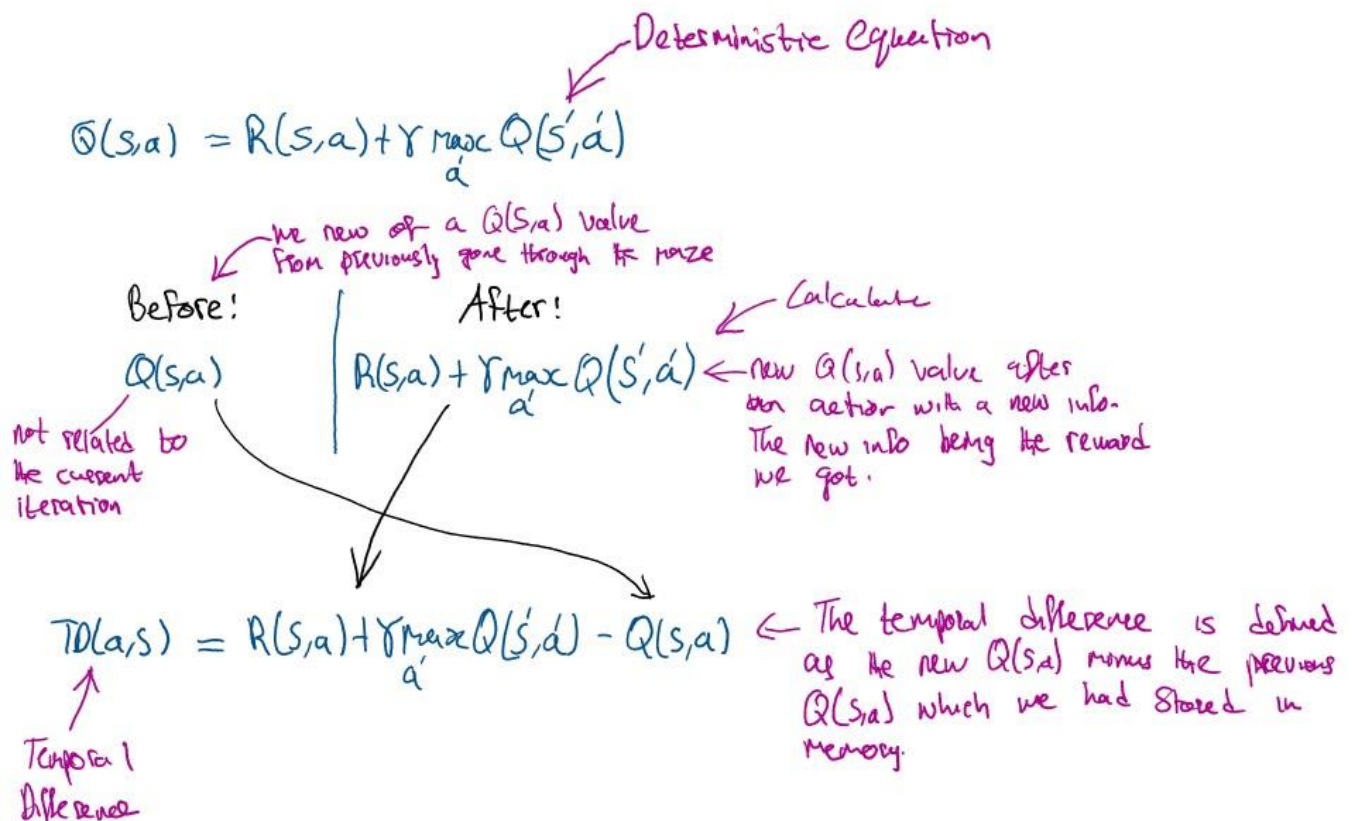


Fig 3.1 Temporal Difference (Violante, 2018)

In temporal difference we are effectively calculating the same thing as the Q equation, but the difference is *time* (Violante, 2018). We want to know if the randomness in the environment as resulted in a change over time and what is the change.

The temporal difference is defined as the new $Q(s,a)$ minus the previous $Q(s,a)$ which was stored in memory.

We do not want to replace our old value with the new value because the new value might be an outlier, in which case we will be discarding the value of the more consistent $Q(s,a)$ from previous states

We solve this by changing the q value gradually using temporal difference.

$$TD(a,s) = R(s,a) + \gamma \max_{a'} Q(s',a') - Q_t(s,a)$$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s)$$

Annotations for the above equation:

- $Q_t(s,a)$: current q
- $Q_{t-1}(s,a)$: Previous Q value
- α : Learning Rate α
- $TD_t(a,s)$: Temporal difference
- The entire equation is labeled as the "Main formula of Q-learning algorithm".

Amalgamation of both formulas

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha (R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a))$$

Annotations for the above equation:

- $Q_{t-1}(s,a)$: Previous Q
- α : Learning rate
- $R(s,a)$: Reward
- γ : Discount factor
- The entire equation is labeled as the "expanded Q-learning algorithm".

Fig 3.2 Full Q learning equation (Karunakaran, 2020)

From the formula in figure 3.2 if the learning rate α is 1 then the Q values will cancel each other, meaning that the previous Q will be replaced by the current Q which we want to avoid.

On the other hand, if α is 0, then the new Q value will also be zero so the current Q will be equals to the previous Q value which means the agent learned nothing. Hence α should not be zero or one, it should be something in between.

Ideally when our Q_t values eventually becomes 0, it means that our algorithm has converged and it is not necessary to continue updating our Q values, however in environments that constantly change we might still need to update the Q_t .

The agent learns the q-value function above, which gives it the expected total return for performing an action in a given state. Using this, the agent must try to maximize its reward.

As the game progresses, the values of Q will converge as rewards diminish in value (CHATURVEDI, 2021).

Deep Q- Learning(DQN)

In DQN the neural network will predict actions in a given state, but the neural network will compare not with what will happen after the action but to the exact value of that exact state in a previous timestep when the agent was in that exact state. In essence we are taking the neural network predicted value of Q and comparing it to a previously calculated value of Q in that exact state (Karunakaran, 2020).

In Q learning we learn through temporal differences, but in DQN agents learn through weights, so we adapted the concept of Q learning to how neural networks learn using weights.

As with Q-learning we want our loss to trend towards zero this means that the previously calculated states or target Q is the same as the predicted Q value. This indicates that the agent can navigate its environment (Karunakaran, 2020).

We will take the loss and using back propagation, pass the loss through the network and through stochastic gradient descent, update the weights of the synapses in the network and repeat the process (Karunakaran, 2020). This will be how our agent learns.

So next when we go through the network the weights will better describe the environment.

Convolution Neural Network(CNN)

CNN looks for features in images to be able to identify variances these features are compressed into convolutions. In convolution we have an input image and a feature detector(**kernel** or **filter**) which represents a matrix used to convolve the image, kernels are implemented to detect features that are integral to the supplied image (Saha, 2018).

To select features we take our kernel, overlay it on our input image and multiply, this gives us a feature map. The step at which we are moving our kernel is called a **stride** (Saha, 2018), in our code we have a stride of 2 pixels at the first layer.

An important aspect of a feature map is that it reduces the size of the image, the higher the stride, the more reduced the image will be, making the feature map even smaller (Saha, 2018). Making the image small is important because it makes it easier and faster to process the image.

In essence the feature map enables us to preserve only the important features of the supplied image while eliminating features that are unnecessary.

As seen in our code we hold multiple feature maps generated using different kernels and the NN decides through training which are important.

ReLU(Rectifier Linear Units)

This is a rectifier function applied on our convolution layer. This is applied on our input image to reduce non-linearity in our image (DeepAI, n.d.). The rectifier function is represented by the image below:

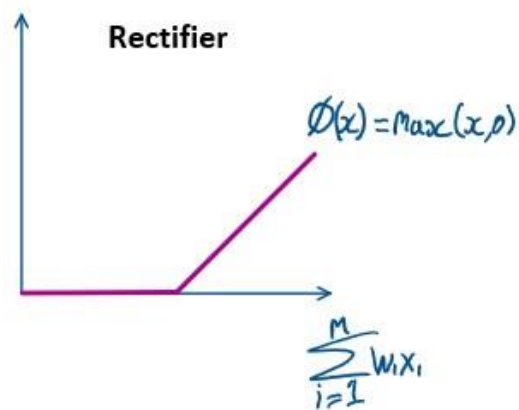


Fig 3.3 Rectifier Linear Units (DeepAI, n.d.)

We define a custom CNN using openAI BaseFeatureExtractor template with modifications made to suit our project as show in figure 3.4 below:

Defining custom DQN feature extraction CNN

```

1 class CustomCNN(BaseFeaturesExtractor):
2
3
4     def __init__(self, observation_space: gym.spaces.Box, features_dim: int = 256):
5         super(CustomCNN, self).__init__(observation_space, features_dim)
6         # We assume CxHxW images (channels first)
7         # Re-ordering will be done by pre-preprocessing or wrapper
8         n_input_channels = observation_space.shape[0]
9         self.cnn = nn.Sequential(
10             nn.Conv2d(n_input_channels, 32, kernel_size=3, stride=2, padding=0),
11             nn.ReLU(),
12             nn.Conv2d(32,32, kernel_size=2, stride=2, padding=0),
13             nn.ReLU(),
14             nn.Flatten(),
15         )
16
17         # Compute shape by doing one forward pass
18         with th.no_grad():
19             n_flatten = self.cnn(
20                 th.as_tensor(observation_space.sample()[None]).float()
21                 ).shape[1]
22
23         self.linear = nn.Sequential(nn.Linear(n_flatten, features_dim), nn.ReLU())
24
25     def forward(self, observations: th.Tensor) -> th.Tensor:
26         return self.linear(self.cnn(observations))
27
28
29 policy_kwargs = dict(
30     features_extractor_class=CustomCNN,
31 )

```

Fig 3.4 CNN feature extraction code

The code in fig 3.4 is graphically represented below:

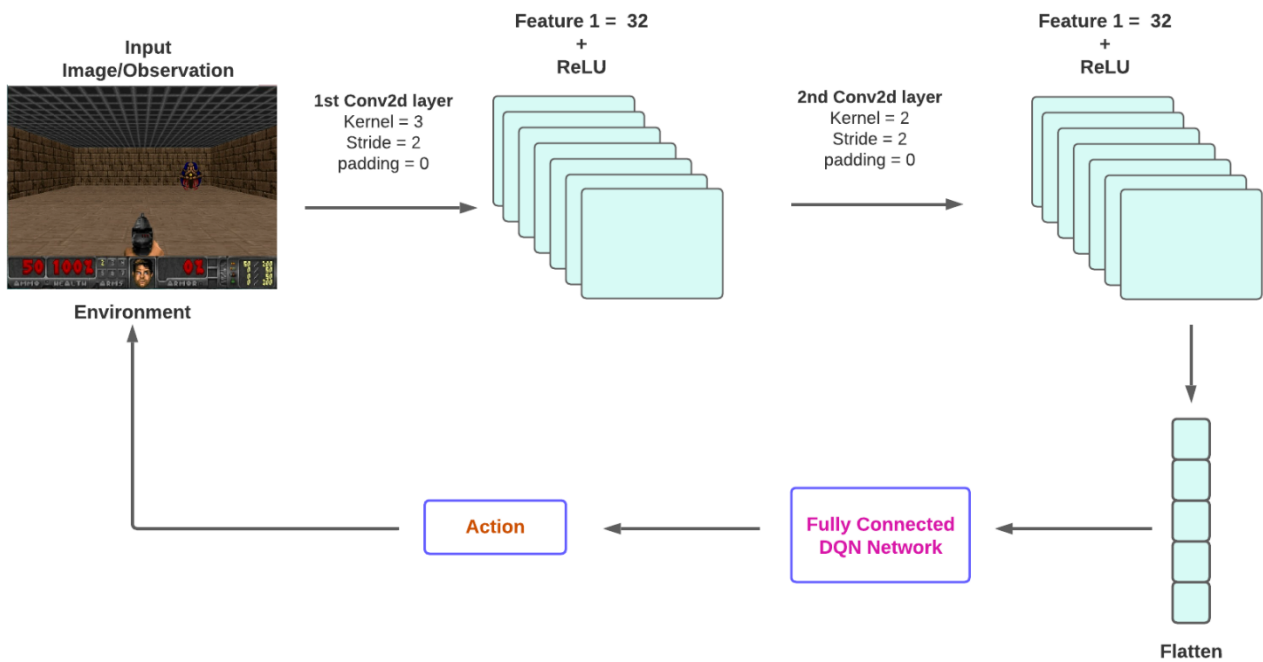


Fig 3.5 Feature extraction

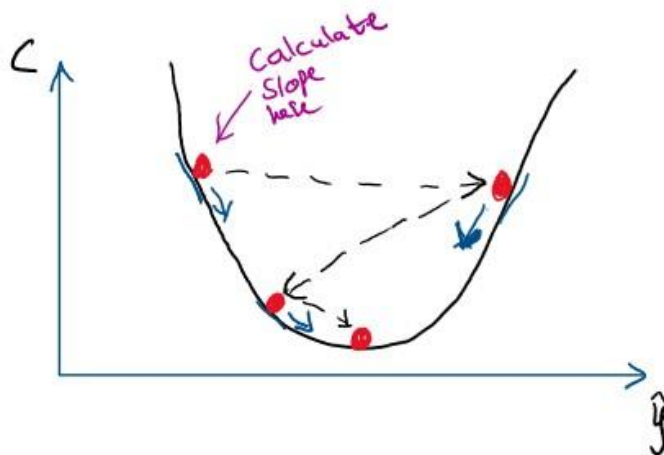
Action Selection Policy

We want to employ an action selection policy that will allow our agent not to get stuck in a local maximum. If the Q2 action above results in negative feedback that will force the neural network to explore and come up with a better action, in this case the environment is forcing the agent to learn. Hence, we want to Exploit(**Exploitation**) what we found but we also want to keep exploring(**Exploration**) if it is possible.

We want an action selection policy that will prevent an agent from being stuck in a local optimum.

Proximal Policy Optimization(PPO)

PPO is a policy gradient method that uses on-policy algorithm and trains with a stochastic policy.



- If a slope is negative at a specific point, we go right
- If the slope is positive, we go left

Fig 3.6 Gradient descent

Gradient descent requires that C above be convex, meaning it has a global minimum. If our C is not convex it looks like:

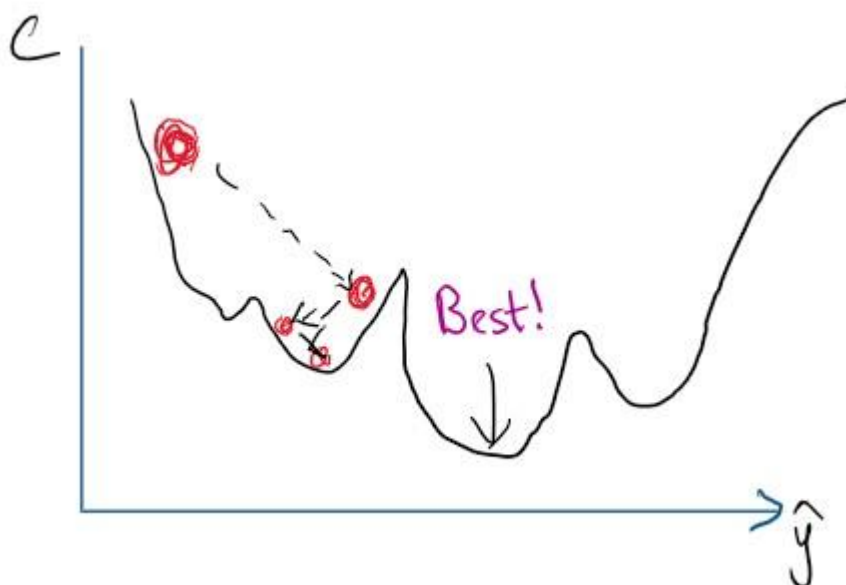


Fig 3.7 Stochastic Gradient descent

In figure 3.7 we could find a local minimum and not a global one. This happens during the training of one of our agents for the project and will be explained later.

There are 2 neural networks in PPO:

- The Actor or policy network: This is an actor who controls the actions of the agent
- The Critic or value network: This is the critic who tries to predict future occurrence from the current state. (Insights, 2018)

When an observation is gotten from the environment, it is passed through a feature extractor, the extractor used depends on if the observation is an image or a vector. If the observation is an image the process described in figure 3.5 above is applied, in the case that the observations are vectors the extractor is just a flatten layer (Stable Baselines3, 2020) as seen in figure 3.9 below. After the features are extracted, these are fed to a fully connected network that maps the extracted features to actions or values (Mahajan, 2020).

The feature extractor can be shared between the actor-critic network and the fully connected network can be partially shared between the networks (Stable Baselines3, 2020), but this project uses separate networks for both feature and the fully connected network.

```
1 policy_kwargs = dict(activation_fn=th.nn.ReLU,  
2 net_arch=[dict(pi=[32, 32], vf=[32, 32])])
```

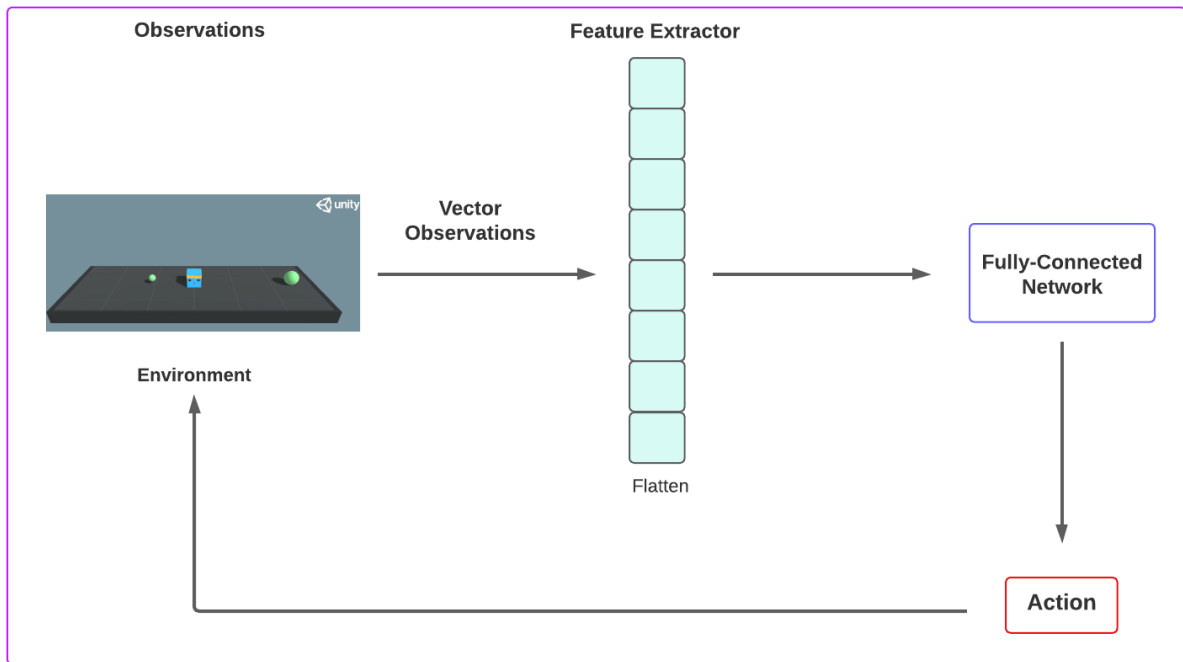
We now create our model using our custom neural network and policies

```
1 model = PPO("MlpPolicy", env, tensorboard_log=LOG_DIR, verbose=1, learning_rate = 0.0001, n_steps = 1024  
2 , policy_kwargs=policy_kwargs)
```

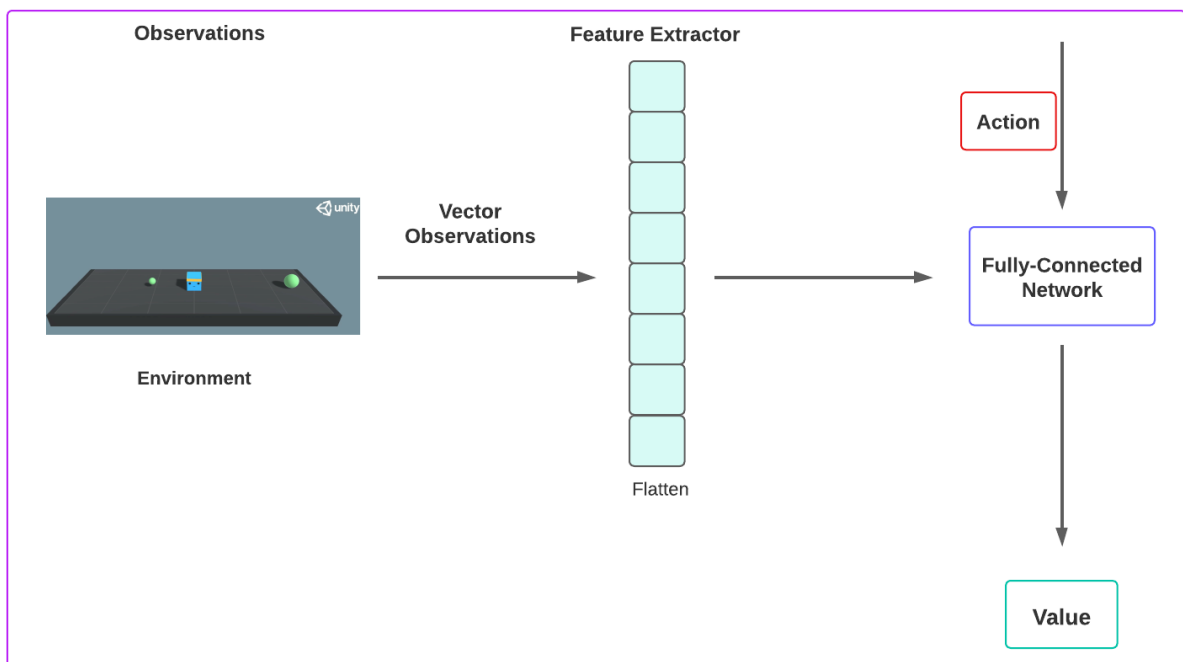
```
Using cuda device  
Wrapping the env with a `Monitor` wrapper  
Wrapping the env in a DummyVecEnv.
```

Fig 3.8 Actor-Critic Network

The code above shows a fully connected network with the policy and value network having 2 layers of 32 neurons, this is then used to create a model using policy_kwargs feature of openAI (Stable Baselines3, 2020) .



Actor Network



Critic Network

Fig 3.9 Actor-Critic Network illustration

In figure 3.9 above, the actor outputs a set of probabilities for which actions to take, this is referred to as the *log probability* .

At the same time the critic outputs the expected future returns from the current state. This is the prediction of the sum of all the rewards going forward. This is referred to as the *value* (Insights, 2018).

When an action is taken in the environment, the reward and the observation state 's' based on that action is returned. The reward is then calculated, and the advantages are computed, this will be used to update the policy (Insights, 2018).

If our agent performs better than expected, we incentivise it to take the same actions and vice-versa if it performs worse.

In actor critic methods it is not uncommon for the performance of the learning agent to suddenly fall after an update to the neural network which causes the agent to lose the ability to function properly in the environment and not recover from that fall (OpenAI, 2018).

This is because actor critic methods are sensitive to perturbations, small changes to the underlying parameters in our deep learning network(weights) can lead to large jumps in policy space that can result in "destructively large policy updates" (Kim, 2021).

PPO addresses this by limiting the updates to the policy networks, it does this in several ways, but the underlying principle is that it bases the update at each step on the ratio of the new policy to the old. We will also constrain this ratio to be within a specific range to ensure that the agent does not take huge steps in parameter space for our deep neural network.

We also must account for the advantage of the state as stated earlier. Taking the advantage can cause the loss function to grow too large so we address this by clipping the loss function and taking lower bounds with the minimum function.

We also observe during training of our agents for the project that the reward mean score may go upward for a while and then crash, this is because actor-critic methods can be brittle and are not the best solution for all cases (OpenAI, 2018).

To discuss policy gradient methods, we will start by defining the policy gradient loss as:

$$L^P(\theta) = \hat{E}_t [\log \pi_{\theta}(a_t | s_t) \hat{A}_t]$$

Expected \uparrow
 Advantage Function / hat term to estimate what the relative values of the selected action
 Policy Loss \swarrow
 Log(Probabilities) from the output of the policy network \swarrow

Fig 4.0 Policy Loss function (Kim, 2021)

To compute the advantage, we need the discounted sum of rewards and a baseline estimate

$$\hat{A} = \text{Discounted rewards} - \text{Baseline estimate}$$

Discounted sum of rewards or return!

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

discount factor \swarrow
 weight sum of all the rewards the agent got during each timestep of the episode

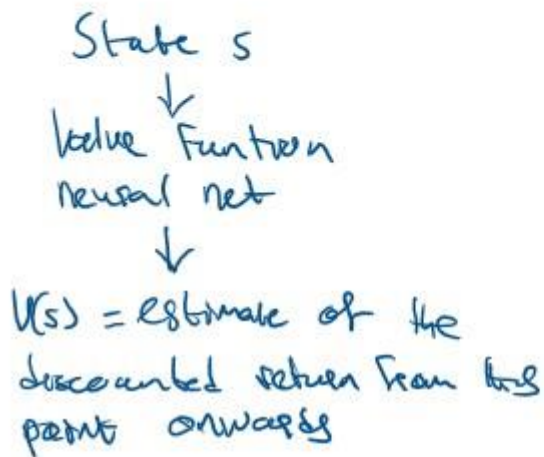
Fig 4.1 Advantage function (Insights, 2018)

Our discount factor gamma usually between 0.9 and 0.99 accounts for the fact that our agent cares more about the rewards it gets quickly over the same rewards it gets many timesteps from the present. The advantage is thus calculated after the episode sequence is collected from the environment, i.e., we know all the rewards (Insights, 2018).

The *value function* is trying to guess what the final reward will be in the current episode starting from the current state.

The advantage estimate is simply examining whether the action the agent took was better than expected or worse.

Observed States



$$\hat{A}_t = \text{Discounted rewards} - \text{Baseline estimate}$$

we know what happened

What did we expect would happen

Fig 4.2 Advantage function (Insights, 2018)

By multiply the log probability of our policy action with the advantage Function we get the final optimization objective used in policy gradient as seen in figure 4.0.

We can destroy our policy if we keep running gradient decent on a single batch of collected memories as explained earlier.

Trust region method (TRPO) solves this by ensuring that if we update our policy we will not move to far away from the old policy. This is what PPO is based on. To ensure that the updated policy does not move too far away from the current policy TRPO adds a KL constraint to the optimization objective. However, the KL constraint adds overhead to the optimization process and can sometimes lead to undesirable training behaviour (Insights, 2018).

PPO includes the KL constraint directly into the optimization objective to solve this (Insights, 2018).

Central optimization objective behind PPO

$R(\theta)$ is a probability ratio between a new updated policy outputs and the output of the previous version of the policy network.

Given a sequence of sample actions and states $R(\theta)$ will be larger than 1 if the action is more likely now than it was in the old version of the policy and it will be between 0 and 1 if the action is less likely now than it was before the last gradient step.

$$\boxed{L^{\text{PG}}(\theta) = \hat{E}_t [\text{Log} \pi_{\theta}(a_t | s_t) \hat{A}_t]} \rightarrow \text{vanilla policy gradient}$$
$$\boxed{\text{maximize}_{\theta} \hat{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right]} \rightarrow \text{Trust region methods}$$
$$\nabla_{\theta} = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$$

Fig 4.3 TRPO (Insights, 2018)

If we multiply the ratio $R(\theta)$ with the advantage function, we get the normal TRPO objective

$$L^{\text{CP1}}(\theta) = \hat{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] = \hat{E}_t [\nabla_{\theta}(\theta) \hat{A}_t]$$

Fig 4.4 TRPO (Insights, 2018)

The central objective function of PPO is as follows.

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

Normal Policy Gradient Objective
 ← Clipped version of normal policy gradient objective.
 ← Expectation operator (we compute this over batches of trajectories)

Fig 4.5 PPO (Insights, 2018)

The first term is $R()$ time the advantage estimate is the default objective for normal policy gradient which pushes the policy towards actions that yields a high positive advantage over the baseline.

The second term is like the first one except it contains a truncated version of the $r()$ ration by apply a clipping operation with the epsilon hyperparameter being how far our new policy can go from the old one while still profiting the objective (OpenAI, 2018), Then the min operator is applied to the two terms to get the final results.

The advantage function can be both positive and negative and changes the effect of the main operator

PPO differs from TRPO by not having a KL- divergence in the objective or a constraint, it relies on clipping in the objective function to prevent new policy from diverging to far from the old one (OpenAI, 2018).

PPO vs DQN

PPO is a policy gradient method. Unlike DQN that can learn from offline data, PPO learns online. It does not use a replay buffer to store past experiences but instead learns directly from what its agent encounters in the environment. Once a batch of experience has been used to do the gradient update, the experience is discarded, and the policy moves on (Insights, 2018).

This means that policy gradient methods like PPO are less sample efficient than Q learning methods because they only use the collected experience once for doing an update.

DQN and PPO learning Example Using VizDoom and Unity

This project primarily focuses on use of the DQN and PPO algorithms to train agents in different environments, however, additional algorithms are used for reasons of comparisons. The Unity environment saw only the use of the PPO algorithm because the DQN algorithm, unlike PPO, cannot train agents in environments with continuous actions and can also not train the box action space (Stable Baselines3, 2020), which was employed in this project.

VizDoom and Unity learning environments

VizDoom and unity environments were used in this project to train agents. Screen shots of both environment running in our program is shown below:



Fig 5.0 VizDoom

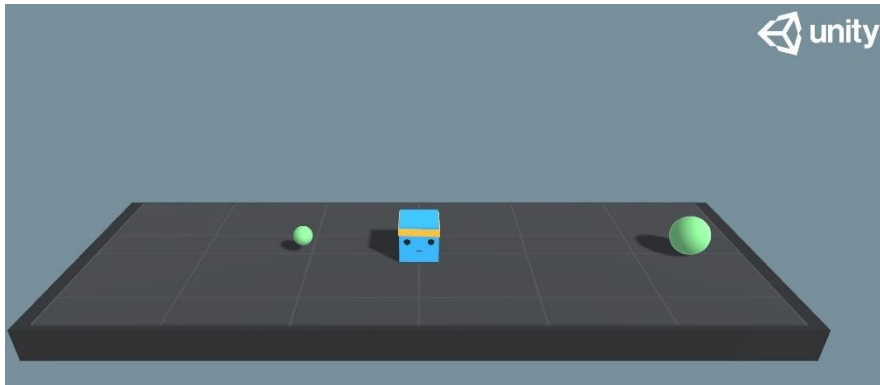


Fig 5.1 Unity environment

VizDoom

In the VizDoom environment used the agent takes up the role of a player whose objective is to kill monsters in the environment when they spawn. The agent has no knowledge of its environment and objective when it starts out and must figure this out on its own.

VizDoom environment information

- Agent living penalty = -1
- Monsters kill reward = +101
- Shoot and miss penalty = -5
- Timeout = 300 seconds

The episode ends when a monster is killed or on timeout.

Unity

In the unity environment, the agent must move left or right to get rewards, however, both directions offer rewards, but one offers more than the other. There is also a living penalty and the agent spawns closer to the lesser of the two rewards.

Hence, our agent must employ an appropriate action selection policy to exploit, as well as explore its environment so as not to get stuck in a local maximum.

Unity environment information

- Agent living penalty = -0.01 at each step
- Inferior reward = +0.1
- Superior reward = +1.0

Observations on Example results

Unity Basic

We ran the PPO algorithm through the unity basic scenario to rate its performance, and see if it can solve the environment, the results are as follows:

Unity_basic PPO learning rewards

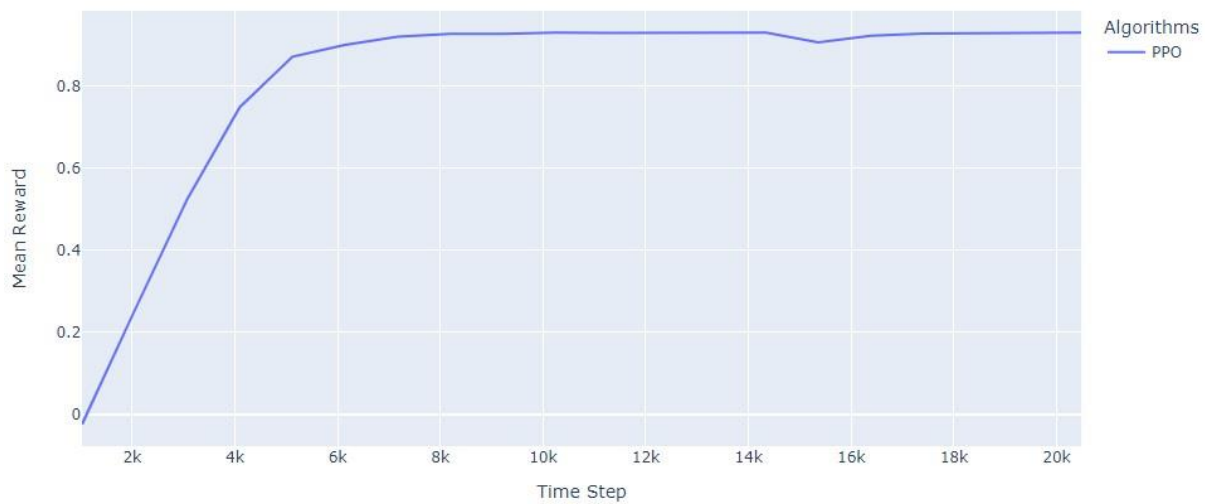


Fig 5.2 Unity Basic reward mean

The agent was run in the environment over 20 thousand timesteps and achieved a high and stable return on rewards after only 6 thousand timesteps.

Unity_Basics PPO Value Loss

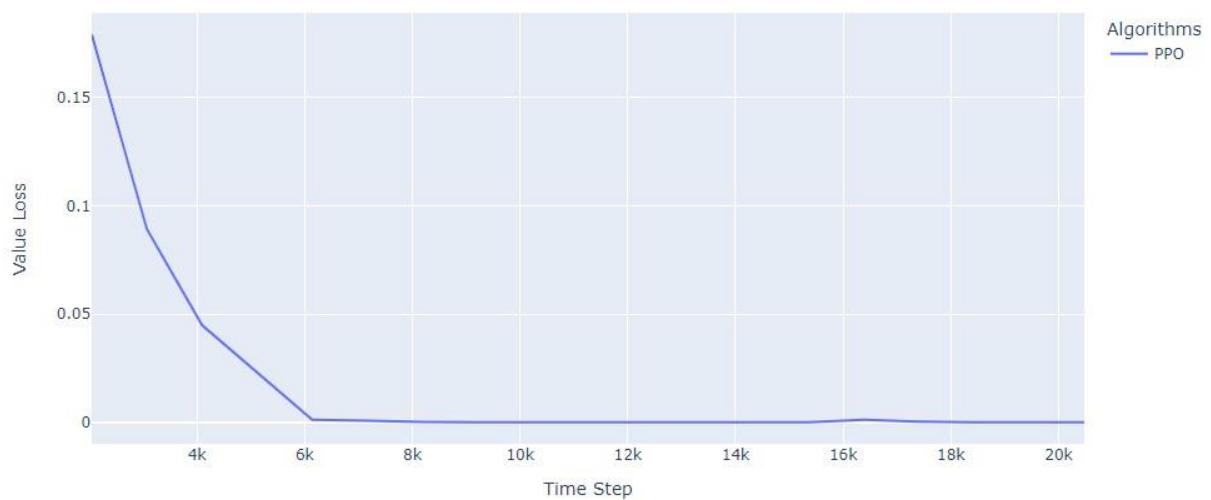


Fig 5.3 Unity Basic Value loss

The value loss tells us how well the policy network can predict future return on rewards, this should increase as the agent learns and decrease once rewards become stable (AurelianTactics, 2018). As seen the value quickly trends towards zero and stays there, upon further investigation we found that this was because the agent had found a local optima and not a global one. The agent still attained a high score, 0.93 out of a possible 0.94 but this behaviour highlights one of the shortcomings of the PPO algorithm (OpenAI, 2018).

VizDoom Basic

Finally, we ran three algorithms, PPO, DQN and A2C in the VizDoom basic environment to analyse the performance of each.



Fig 5.4 VizDoom Basic Reward mean

As seen from fig 5.4, both DQN and PPO algorithms were able to maximize their rewards while the A2C algorithm was not.

PPO did this quickly while DQN was eventually able to maximize its rewards.

Doom_basic algorithms Value Loss

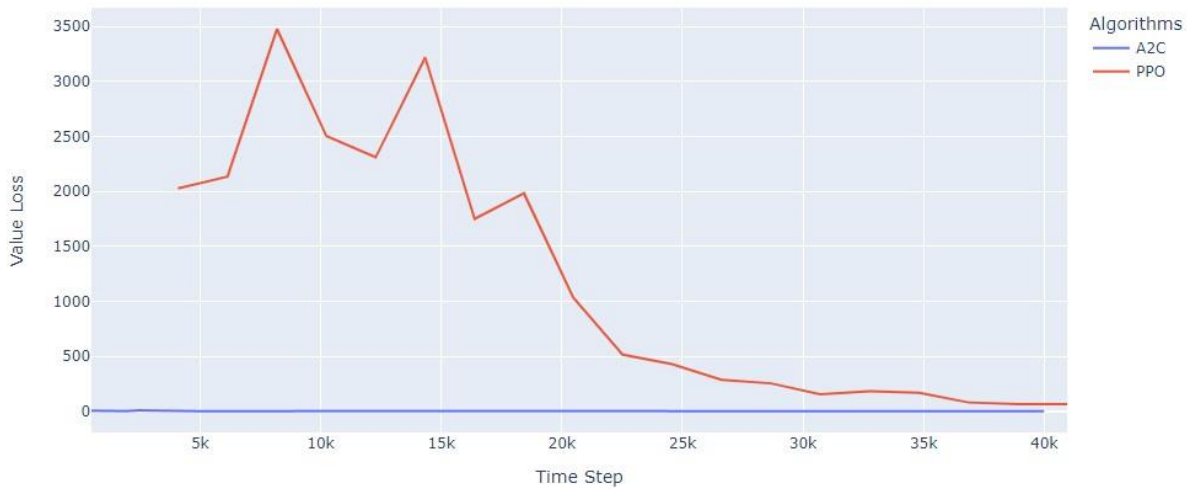


Fig 5.5 VizDoom Value loss

Exploring the value loss, which only applies to PPO and A2C as actor-critic methods, the PPO algorithm loss rose as it learnt and decrease towards zero when it began understanding its environment, the A2C algorithm on the other hand never rose above zero, indicating that the agent never learnt anything at all, this could be because there was not enough experience in the environment and the agent was overfitting to a specific scenario, or the fact that A2C can be inherently unstable (Stable Baselines3, 2020).

Doom_basic explained variance

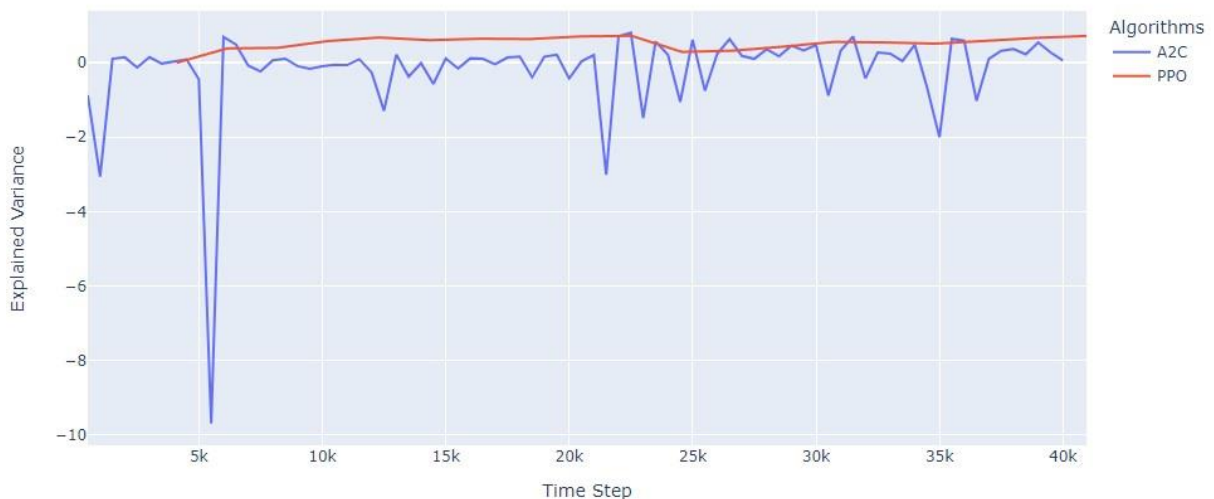


Fig 5.6 VizDoom explained variance

The explained variance tells how well the value network is able to predict the future sum of rewards (AurelianTactics, 2018), in other words, how well our value network understands its environment.

We want this to increase, which happens with the PPO algorithm, but spikes in the A2C algorithm indicating that the value network is not able to predict future rewards.

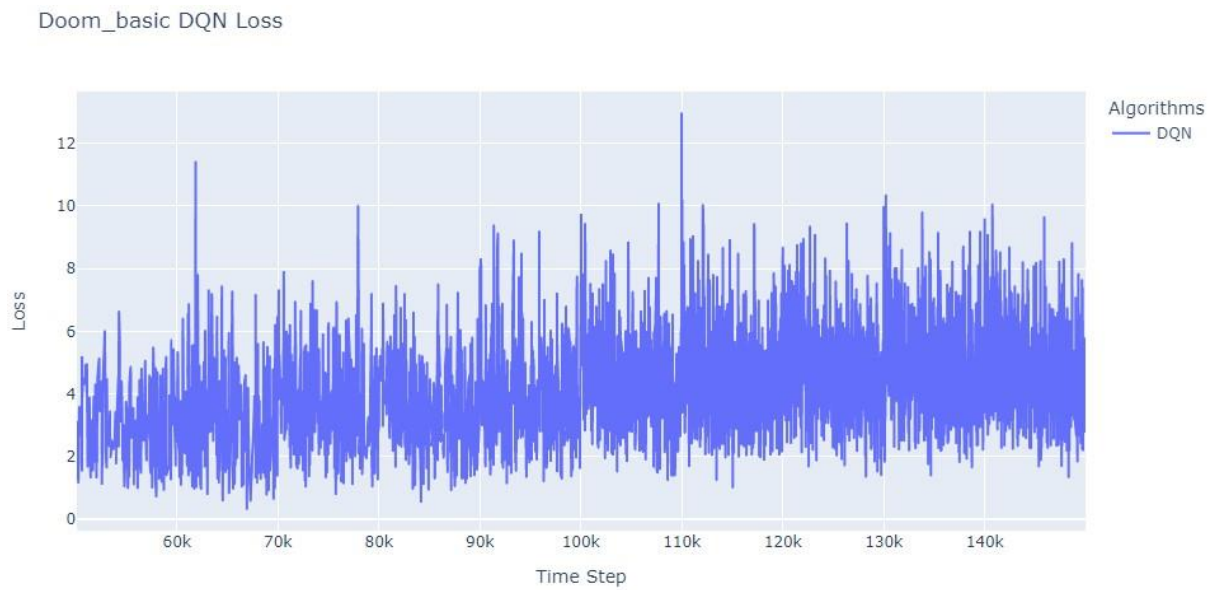


Fig 5.6 VizDoom DQN Loss

Finally, we have the loss of the DQN algorithm, this also tells us how well the agent is able to predict future sum of rewards (Karunakaran, 2020). Ideally this should be close to zero with spikes as the as the network compares its target Q value with its predicted Q (Karunakaran, 2020), this indicates that the agent is able to predict its future rewards well enough.

Conclusion

In conclusion, one of the main advantages of the PPO algorithm is that it can work in a range of action and observation spaces, with both discrete and continuous actions (Stable Baselines3, 2020), this makes it easier to implement and use. However, PPO can be brittle as mentioned earlier and is also prone to finding localized optima (OpenAI, 2018). DQN is more experience efficient and less prone to overfitting if fitted with an appropriate action selection policy (baeldung, 2021), however DQN only works with discrete actions and the openAI implementations does not support the box action spaced used for the Unity environment in this project.

Thus, we see that our agents indeed can learn how to perform tasks in their given environment without being explicitly programmed to do so, but the outcome of the learning depends on the environment and the algorithms used.

Misc.

Environment training by Nicholas Eruba

Note:

*The following section was written and analysed by group member Nicholas Eruba, Conclusions and analysis made here **DOES NOT** represent that of the above report as this was presented a few hours before the submission deadline and could not be cross examined. That being said, Nicholas and Fernando of the group worked for weeks, as stated in the journal below, to create custom environments that the group could use to train agents in but these environments were not ready in time for testing.*

Custom Environments

What is a Custom Environment

We have explored training and evaluating various reinforcement learning algorithms to beat scenarios in the unity and vizdoom gym environment adapted from the DOOM game,

A custom environment is a reinforcement learning environment that is created to reflect the unique challenges of a specific scenario(Willies Ogola, Building a Reinforcement Learning Environment using OpenAI Gym, 2022)

An example of a custom environment could be a manufacturing factory floor where a trained agent must map and avoid obstacles, while completing a course to deliver work tools from A to B, this scenario can be simulated and trained on a computer as a custom environment

Though reinforcement learning and machine learning models in general are known for their highly generalizable nature, training an algorithm, and structuring its reward scheme on an environment replicated to be most like its test destination is likely to produce better results

In this chapter we explore an example scenario where a customer has requested an agent be trained to navigate a custom environment based on a game,

We build a custom reinforcement learning environment using OpenAI Gym.

All code files including learned weights for the model can be found at this link

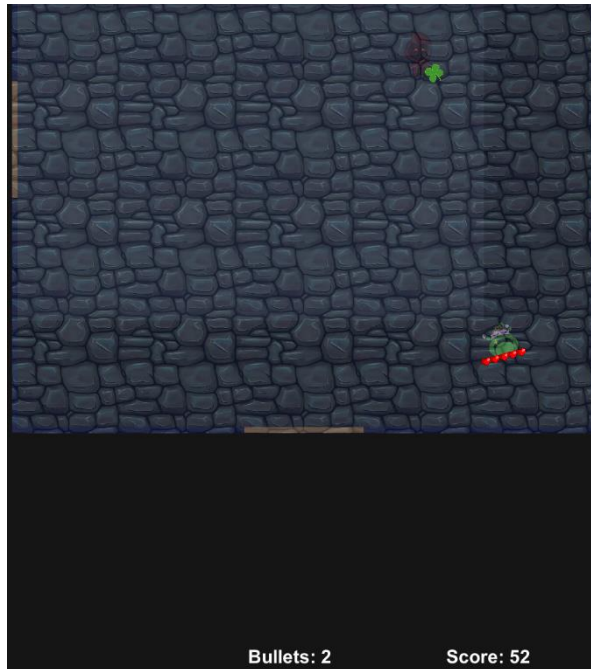
https://drive.google.com/file/d/1fFpMW7MBcS_1NmwtRANqqqcMvI91eUtC/view?usp=sharing

Github:

https://github.com/Jolomi2k9/RL_Game_AI/blob/nicholas/leprechaun_environment.zip

Description Of the Environment

The environment is based on a unity game Capstone2D which was built by our team,



Capstone2D game capture

The aim of the game is to navigate a leprechaun to shoot at bugs without colliding with them, The leprechaun must survive in the environment for as long as possible will shooting as many bugs as possible

1. The leprechaun must survive in the environment for as long as possible
2. The episode terminates if the leprechaun collides with a bug,
3. There are ammo packs which the leprechaun can collect to refill bullets to a fixed number max_bullets, here we chose 50

Elements Of the Environment

There are 5 elements in the environment

- Leprechaun-protagonist of the game and trainable agent
- Bug-antagonist of the game, moves in a straight line down the screen, dies with one bullet
- Ammo- represents a pack of max_bullets bullets
- Bullet- shown below as the shamrock is the bullet fired by leprechaun
- Bg-the blue stone background image to the game on which every other element is placed



The 5 elements

they are defined by 5 classes, which all inherit from the point base class

The Point Base Class

The point base class defines the basic unit of an element in the environment

```
class Point(object):
    def __init__(self, name, x_max, x_min, y_max, y_min):
        self.x = 0
        self.y = 0
        self.x_min = x_min
        self.x_max = x_max
        self.y_min = y_min
        self.y_max = y_max
        self.name = name

    def set_position(self, x, y):
        self.x = self.clamp(x, self.x_min, self.x_max - self.icon_w)
        self.y = self.clamp(y, self.y_min, self.y_max - self.icon_h)

    def get_position(self):
        return (self.x, self.y)

    def move(self, del_x, del_y):
        self.x += del_x
        self.y += del_y

        self.x = self.clamp(self.x, self.x_min, self.x_max - self.icon_w)
        self.y = self.clamp(self.y, self.y_min, self.y_max - self.icon_h)

    def clamp(self, n, minn, maxx):
        return max(min(maxn, n), minn)
```

It initializes the element by accepting the allowed coordinates for the element to be within the environment, and defines functions to position and move the element within the environment

```

class Leprechaun(Point):
    def __init__(self, name, x_max, x_min, y_max, y_min):
        super(Leprechaun, self).__init__(name, x_max, x_min, y_max, y_min)
        self.icon = cv2.imread("lep2.png")
        self.icon = cv2.cvtColor(self.icon, cv2.COLOR_BGR2GRAY)
        self.icon = self.icon[:, :, np.newaxis]
        self.icon_w = 32
        self.icon_h = 32
        self.icon = cv2.resize(self.icon, (self.icon_h, self.icon_w), interpolation=cv2.INTER_CUBIC)
        self.icon=np.reshape(self.icon, (self.icon_h,self.icon_w,1))

class Bug(Point):
    def __init__(self, name, x_max, x_min, y_max, y_min):
        super(Bug, self).__init__(name, x_max, x_min, y_max, y_min)
        self.icon = cv2.imread("bug.png")
        self.icon = cv2.cvtColor(self.icon, cv2.COLOR_BGR2GRAY)
        self.icon = self.icon[:, :, np.newaxis]
        self.icon_w = 32
        self.icon_h = 32
        self.icon = cv2.resize(self.icon, (self.icon_h, self.icon_w), interpolation=cv2.INTER_CUBIC)
        self.icon=np.reshape(self.icon, (self.icon_h,self.icon_w,1))

```

the individual element classes inherit from the point base class, here the element icon is read converted to grayscale and resized

Creation Of the Environment

The first consideration when designing a custom environment is to determine what the observation and action space will be

- The observation space defines the environment, it can either be discrete or continuous, an example of a discrete observation space would be that of a card game where the environment state is determined by the values of all cards in play, an example of a continuous observation space would be an environment where the agent's position is determined by real valued coordinates or pixel location on screen (Jacob Wilson, What is observation space in Openai gym? – Tech-QA.com, 2022)
- The action space can also be either continuous or discrete and defines the possible actions our agent can take, an example of a discrete action space is a simple platformer game where actions correspond to 0:jump, 1:move forward and 2:move backward but the action does not quantify the outcome, an example of continuous action space could be a rotary cannon that shoots golf balls at a target, the shoot action also quantifies how much power the ball is shot with and the left and right actions also quantifies by how many degrees the cannon swivels (Jacob Wilson, What is observation space in Openai gym? – Tech-QA.com, 2022)

Landscape Class

This class inherits from Gym.env and contains all the code to create and run the environment, it has 3 main functions, `_init_`, `reset` and `step`

`_init_` function

We start by defining the observation and action space

The observation shape is a continuous 500 X 500 X 1 canvas where the "1" dimension is a single grayscale colour channel dimension ranging from 0 to 255, using grayscale or black and white imagery as opposed to an RGB image will exponentially reduce training time as the neural network has significantly less information to interpret

All elements in the environment will be drawn on this canvas and the canvas will represent the state of the environment

The Action space is a set of discrete values ranging from 0 to 6 representing respectively the actions

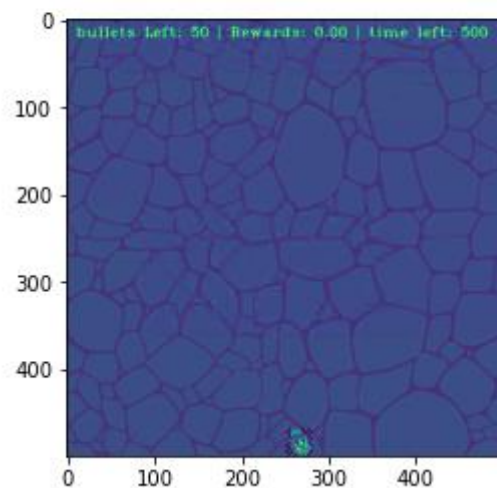
```
{0: "Right", 1: "Left", 2: "Down", 3: "Up", 4:"shoot", 5: "Do Nothing"}
```

Reset Function

This function is responsible for resetting our environment at the end of each episode, here we redefine all variables, clear the canvas and draw the leprechaun on it for a fresh episode

```
env = LepScape(show_status=True)
obs = env.reset()
screen = env.render(mode = "rgb_array")
plt.imshow(screen)
```

<matplotlib.image.AxesImage at 0x1d40214c070>



a reset canvas

The reset function also defines and reset variables that keep track of the environment state

`ep_length` : keeps track of the length of the episode in timesteps, one timestep represents one frame of the game in which all elements of the environment take a single action, it is set to "Episode_length" configuration and here we have chosen steps

`bullets_left`: keeps track of how many bullets the leprechaun has left, ranges from 0 to `max_bullets`

`ep_return` : keeps track of the total reward of the episode

`self.bug_count` : number of bugs in this episode

`self.ammo_count` : number of ammo packs in this episode

these variables below are returned as information by the step function, they are used to evaluate the model's performance eventually

`ep_kill_count`: how many bugs were killed in this episode

`ep_bullets_used`: how many bullets were used in this episode

`ep_died`: true if the leprechaun died and false if the episode_length ran out

ep_ammo_collected: how many ammo packs was the leprechaun able to collect

The step Function

The step function accepts an action applies it to the agent and updates all other elements in the environment, it advances the environment by one frame, at the end of which it returns the environment state, any rewards accrued and whether the episode is done.

We start by asserting that the action is valid and contained within the action space

Once that is done, the action is implemented to the leprechaun as shown below

```
# apply the action to the leprechaun
if action == 0:
    self.leprechaun.move(0,5)
elif action == 1:
    self.leprechaun.move(0,-5)
elif action == 2:
    self.leprechaun.move(5,0)
elif action == 3:
    self.leprechaun.move(-5,0)
elif action == 4:
    if self.bullets_left>0:
        # Spawn a bullet
        spawned_bullet = Bullet("bullet_{}".format(self.bullets_left), self.x_max, self.x_min, self.y_max, self.y_min)
        self.bullets_left -= 1
        self.ep_bullets_used+=1

        bullet_x, bullet_y = self.leprechaun.get_position()
        bullet_x=bullet_x+int(self.leprechaun.icon_w/2)
        spawned_bullet.set_position(bullet_x, bullet_y)

        # Append the spawned bullet to the elements currently present in Env.
        self.elements.append(spawned_bullet)
        spawned_bullet.move(0,-5)
        #small penalty for firing a bullet
        reward+=self.config["BULLET_USE_REW"]
    elif action == 5:
        pass
```

Implement action to leprechaun

New bugs and ammo are spawned at the top of the screen based on their respective probabilities which are 5% and 1% chance respectively

Next, check for collisions, first between bugs and bullets in which case both elements are removed from the environment and rewards are updated

Next, update the rest of the elements, if the leprechaun has collided with a bug the episode is ended, and rewards are updated, else all bugs are moved downward by delta 5

If the leprechaun has collided with an ammo pack the pack is removed, bullets left is reset to max bullets and rewards updated,

All elements that have reached the bottom of the screen are removed

Next, check if the episode has run out of time steps where we end the episode

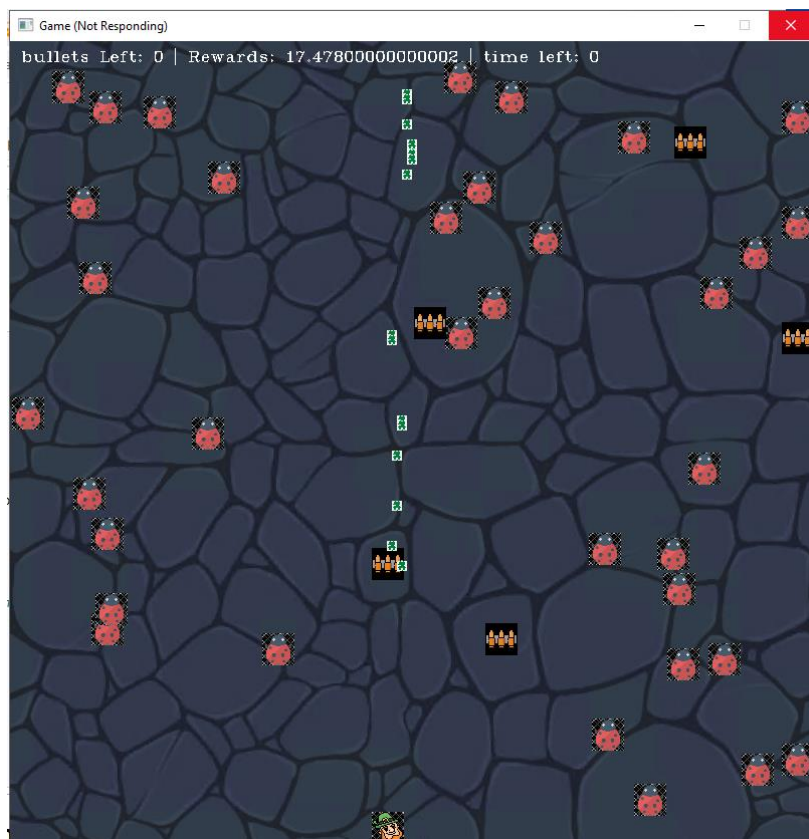
```

#return the environment rewards as well as some useful information about the episode
return self.canvas, reward, done, {
    "ep_kill_count":self.ep_kill_count,
    "ep_bullets_used":self.ep_bullets_used,
    "ep_died":self.ep_died,
    "ep_ammo_collected":self.ep_ammo_collected,
    "ep_length":self.ep_length
}

```

Step info

At the end of the step function, information from the step is returned as shown above



A working environment

Testing the environment for 10 episodes produced the following result

Total Kill	Total Bullets Used	Bullets Per Kill	Total Ammo Picked	Episode Reward Mean	Episode Length Mean	Number of Deaths
31	435	14.0	4	1.1	156.9	8

Training

Configuration Of the Environment and Reward Structure

```
#configuration variables can be tweaked to manipulate difficulty, reward structure and appearance of the environment
CONFIG={
  "OBSERVATION_W":500,#screen width
  "OBSERVATION_H":500,#screen height
  "BUG_SPAWN_PROB":0.05,
  "AMMO_SPAWN_PROB":0.01,
  "EPISODE_LENGTH": 500,
  "MAX_BULLETS": 50,
  "STEP_REW":-0.001,
  "BUG_KILL_REW": 7,
  "LEP_COLLISION_REW": -10,
  "AMMO_COLLECT_REW": 2,
  "BULLET_USE_REW":-0.3
}
```

The environments parameters are stored in a config variable to ensure flexibility in training.

This config variable is used to determine dimensions of the screen, probability of bugs and ammo packs spawning on any given step and the reward for each action.

The step_reward is the reward for executing a step in the environment. As shown above, has been set to a very low negative number this has the effect of incentivizing the agent to sacrifice living longer for being more aggressive towards shooting at the bugs, there is a large penalty for colliding with the bug and an equally large reward for killing a bug, there is a small penalty for firing a bullet and reward for collecting ammo pack to prevent wastage and consequently running out of bullets and to encourage the agent to collect ammo packs

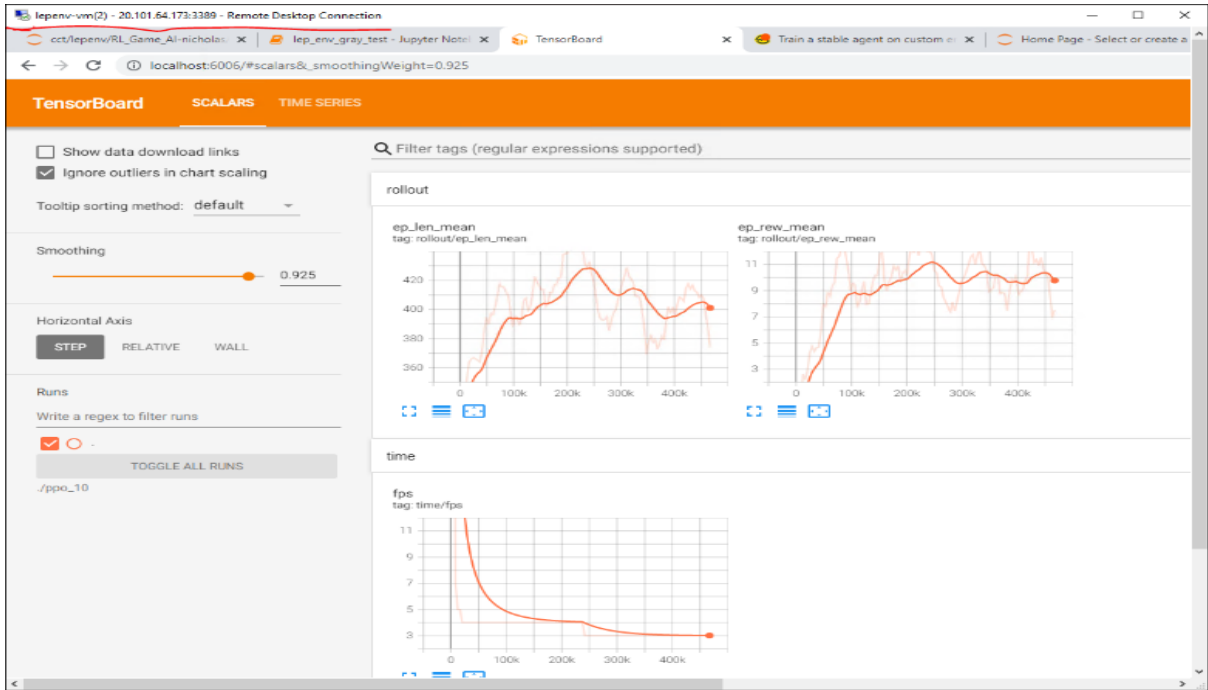
Preparation for Training

A Few actions were taken to prepare the environment for training

The environment is first scaled down from the original size of 800X800X3 to a 500x500X1 canvas to increase the convergence time for the model

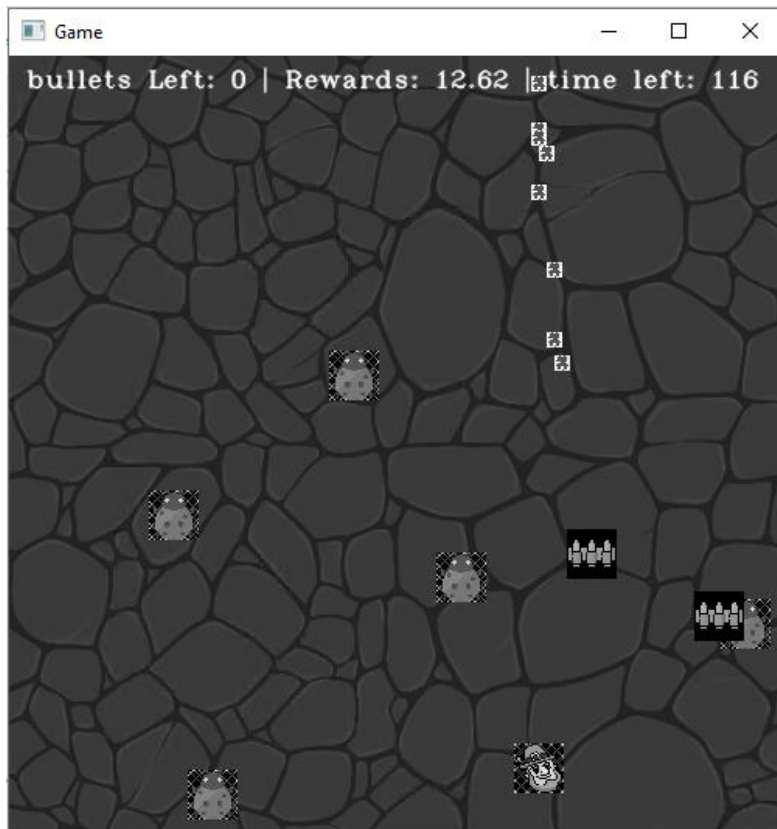
all images in the environment are transformed from 3-color channel RGB image to a single channel grayscale image to reduce training time

A virtual machine shown below is setup for training the environment due to the resource intensive nature of reinforcement learning



Tensorboard running in vm

A callback function is set for the model to log training data, and save the weights learned by the model every 10000 timesteps, these training logs will later be used to plot training metrics on tensorboard



Environment downsized and greyscaled

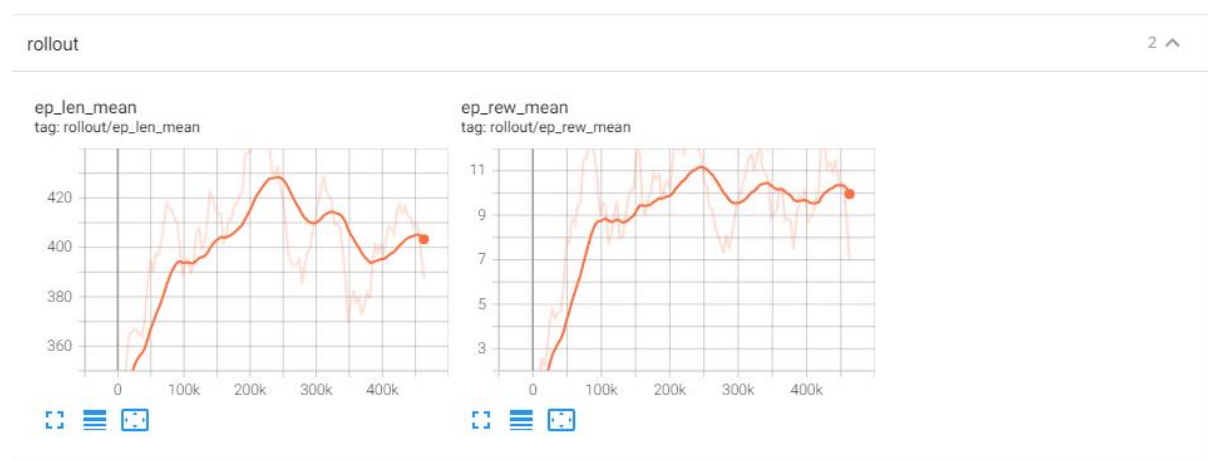
The model is then trained with the PPO CNN policy as shown below for 500,000 Timesteps

```
# Non rendered environment
env = LepScape(show_status="training")
```

```
#create the model
model = PPO('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=1, learning_rate=0.001, n_steps=4096)
```

```
Using cpu device
Wrapping the env with a `Monitor` wrapper
Wrapping the env in a DummyVecEnv.
Wrapping the env in a VecTransposeImage.
```

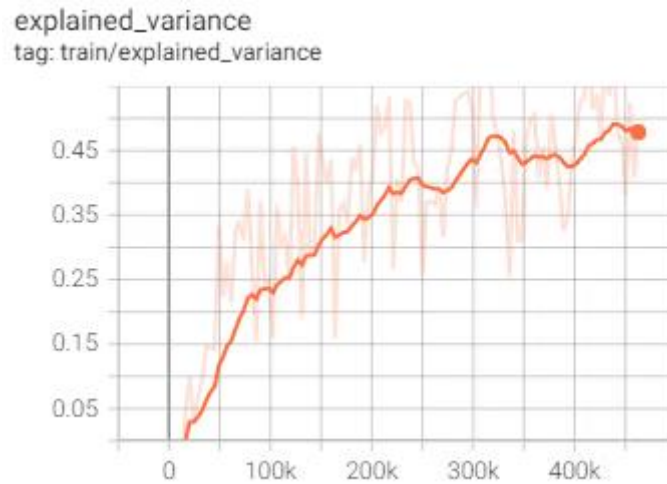
Training Process



Length and reward mean

Based on the logs shown above that the agent lived steadily longer in the environment until about timestep 250k where it peaked at 444 out of 500 steps per episode, this tracks well with the increasing rewards, however the model started to sacrifice living longer for being more aggressive by going after the bugs causing it to die early a lot more resulting in a disproportionate decline of the episode length with respect to the episode reward

Explained variance is a measure of how much of the rewards gained in the episode is explained by decisions taken by the model as opposed to random, the value steadily increases to a peak of about 60% at the 450k timesteps



Explained variance

This metric is important because it shows that the model is learning

Results Metrics

```
# Import eval policy to test agent
from stable_baselines3.common.evaluation import evaluate_policy
```

```
# Reload model from disc
model = PPO.load('./train/train_basic/best_model_6_230000')
```

Model 230k is loaded

After training completed the environment was tested for 100 episodes, on a random agent and the trained model 230000 agent and the results are shown below

	Total Kill	Total Bullets Used	Bullets Per Kill	Total Ammo Picked	Episode Reward Mean	Episode Length Mean	Number of Deaths
Random agent	296	4569	15.4	30	1.5	126.8	57
Trained Agent	320	3489	10.9	36	8.0	100.7	43

Reflection on Results

Total Kill: the trained agent killed a lot more bugs than the random agent despite living shorter on average which shows evidence of learning to prioritize kills over just surviving in the environment

Total Bullets Used: the trained agent used less bullets than the random agent which shows evidence of conserving bullets and not firing sporadically

Bullets Per Kill: this metric is an important one and shows evidence of targeting behaviour, however though the agent uses 5 less bullets than the random agent per kill it is still wasting a lot of bullets, an ideal scenario here would be a 1:1 ratio, training the model for much longer epochs will likely improve the targeting behaviour

Total Ammo Picked: the trained agent picked 6 more ammo packs than the random agent despite living for a shorter period on average, the difference is not much but this is not necessarily a bad thing as the model may have learned to only attempt to pick an ammo pack when the shoot action stops producing bullets due to running out of ammo

Episode Reward Mean: here we see a clear difference with the random agent which shows that overall, the model is adapting to the aims set out for it with the reward structure.

Episode Length Mean: the random agent lived longer on average than the trained agent, further investigation revealed that the reason for this was due to the small negative step reward of -0.001.

The model learned it incurred penalties by simply avoiding bugs, so it would go after them which will result in the leprechaun getting killed due to a combination of sometimes poor targeting, attempting to shoot when it had run out of bullets or shooting accurately at the bug but sometimes the collision does not register due to an unfixed logical error within the underlying check collision code, interestingly in later epochs the model tried to fix this by firing multiple bullets at the bug, the reverse effect of this is that the bullets per kill ratio increases and it incurs multiple penalties for wasting bullets

Overall, a better reward structure might have been to give the agent a small positive step reward for surviving in the environment also the performance can be improved by making all collisions register without exceptions

Number of Deaths: on the opposite side of active targeting a lower number of deaths relative to the random agent showed that the leprechaun learned to dodge bugs

Applications of Custom Environment

Automatic Difficulty

Custom environments can be used to implement automatic difficulty in games by introducing longer trained agents as the players ability improves

The benefits of this are that it automates not just the labour of manually creating an algorithm corresponding to various levels of difficulty in the game, but it also automates the creativity of how these agents increase the challenge to the player

Instead of the developer conceptualizing how to increase the challenge to the player they can rely on patterns learned in the environment by the model over time to gradually improve in complexity and skill

(Watcharasatharpornpong, Nirach & Kotrajaras, Vishnu. (2009). Automatic Level Difficulty Adjustment in Platform Games Using Genetic Algorithm Based Methodology. 10.5176/978-981-08-3190-5_482.)

Special Agents

By manipulating the reward structure in a custom environment, special Agents can be introduced who are incentivized to achieve a particular task in the environment, in the LepScape environment for example a custom “bullet chaser” bug can be introduced and trained to closely follow ammo packs and thus confuse the leprechaun into colliding with the bug to while trying to collect the ammo pack,

While it is reasonable to assume that a model trained to oppose the leprechaun might learn this behaviour over time, the main aim of designing and training special agents is to explore the creativity of the developer without having to manually implement a new algorithm that codes for the special characters unique behaviour and how it will become more challenging as the players skill improves

(Sung and Cho, 2012)

Appendix

Reflective Journal

Jozimar

In my journey through this project, I have many talks to my pairs. In our first week as group, we were assigned to bring project ideas for Problem Solving for Industry subject. I came up with AI in farms and how robots could help in plantations and control production and plagues. After a discussion with my groupmates, we decided to go with video games instead of my previous idea, therefore, we required a drone to do the tests and it would be out of our reach at that point. After we sorted the business idea for our project, we divided our tasks for the week and we started to work on it during the first weeks. I was responsible for the business case application and the market research, also researched as contributor for the SWOT analysis for the project. I was responsible to help into the instruction and conclusion for the first part of the project.

After we completed the paper part, we started to work on the code. I was assigned to train a stable agent using PPO algorithm in a deadly corridor scenario, a scenario where the agent should walk towards a corridor to defeat enemies armed. I had to train the agent in 5 different level because each level added a more difficult challenge for the agent. It took me several hours to achieve the final result. My computer wasn't helping too, and it made my task take longer to be completed. After finalise the deadly corridor, I was working on Defend the center, basically the agent should shoot in the center to kill the enemies.

I left my colleague in charge of the coding part for a while, because I was designated to implement and describe how we used Agile Scrum in our project. I made research and introduced the concept to our work, as we were already working on Basecamp, we had all the necessary information to work on. At the same time, I was creating the post presentation, implementing idea and the design. I created the prototype that were modify to accomplish the standard desired by our lecture.

After finalising with the poster presentation, my new challenge was to plot logs from TensorBoard in different graphs to highlight performance and to compare algorithms in implemented scenarios. I created a plotline graph to show the performance between PPO and DQN algorithms. At first, I created different graphs with the algorithms a part, but after a suggestion from my teammate Jolomi, I changed the graphs and plotted on graph but with both algorithms together.

To conclude to the project, I collaborate to the final introduction and created a conclusion for our SWOT analysis.

Fernando Gonzalez Anavia

Working through this project, we had meetings every week before class in order to assign activities, show progress and discuss any questions or concerns.

At the begining, my first duty was training an agent (VizDoom) using a DQN algorithm. That was to test if the model worked, but not further analysis or parameters modification was implemented. After that, my main task was to develop a custom Unity Game Environment from scratch to apply the future model training. Unfortunately, The VideoGame was done for human interaction but It could not be finished to be adapted to the models we worked into.

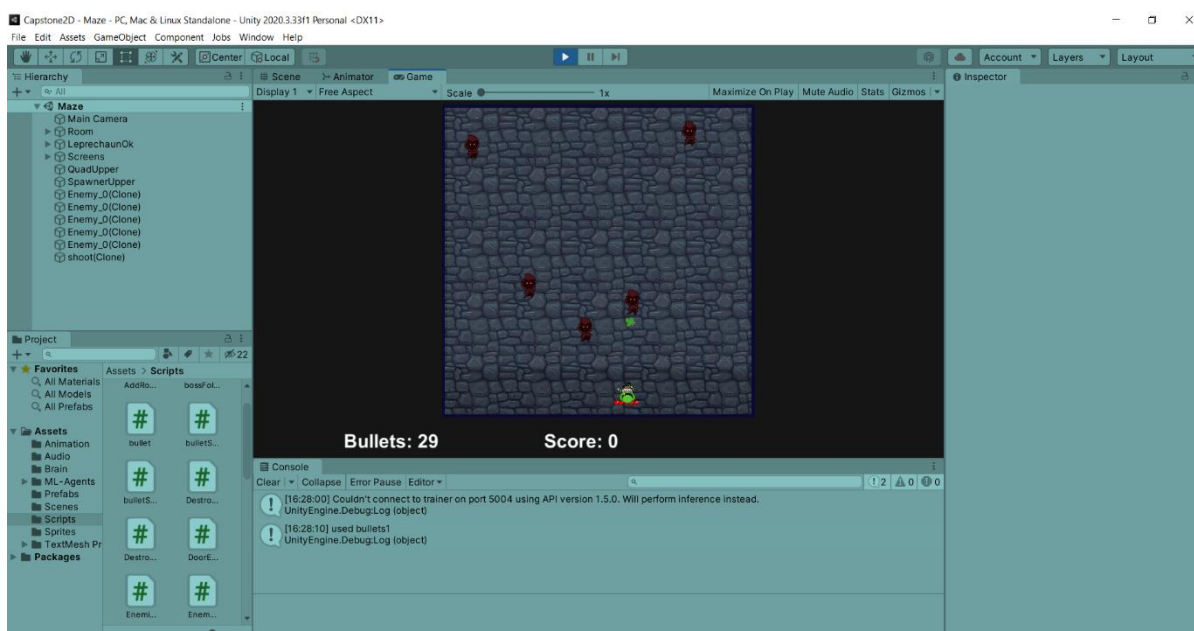


Figure 2 Videogame Environment

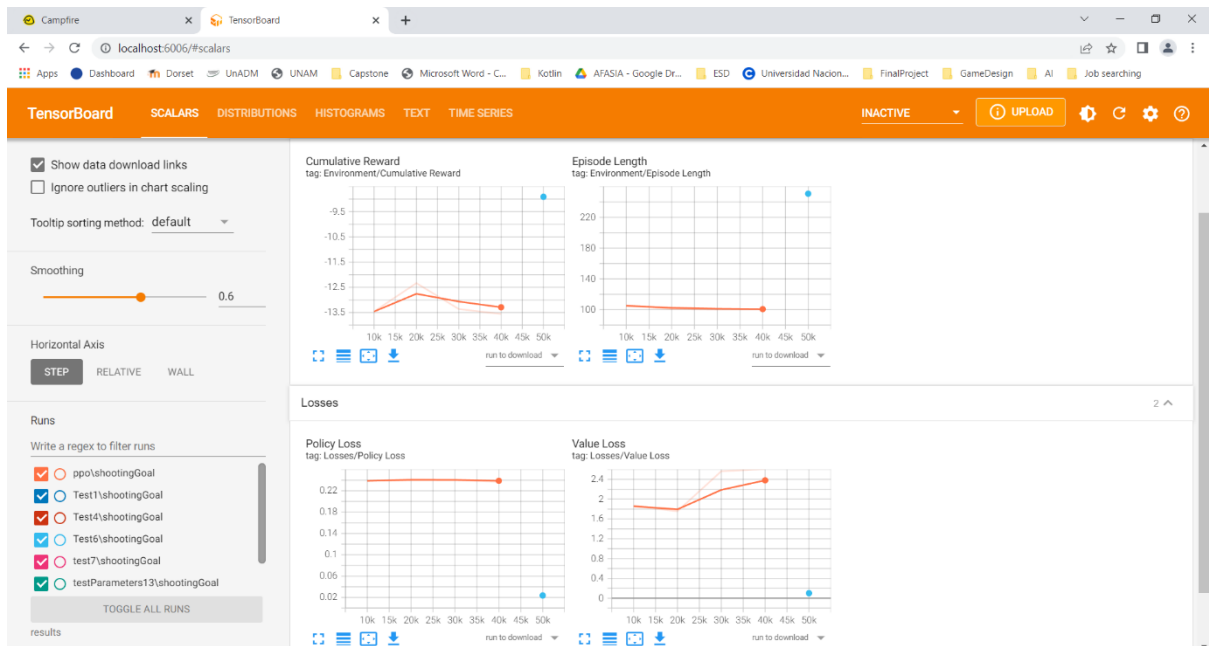


Figure 3 Scalars graphs in Tensorboard

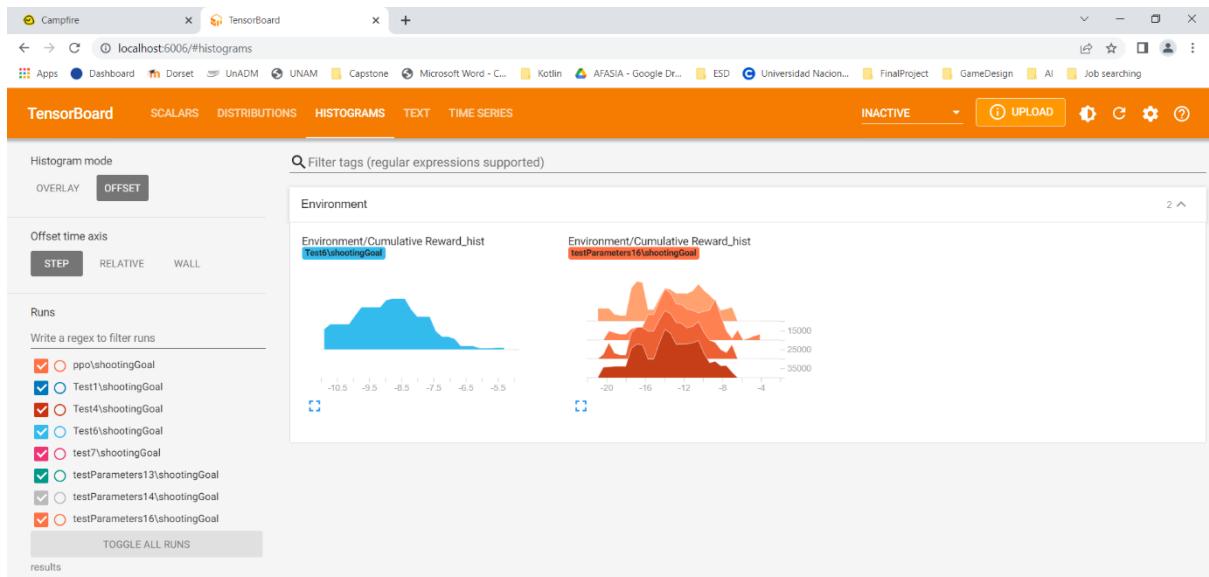


Figure 4 Histograms (Cumulative rewards) in Tensorboard

Lastly, I wrote the Agile Software Development chapter we used during the previous weeks.

Although, my main duty was not successful I will keep working on that by my own since the application of Machine Learning on Unity is not only useful for VideoGames, but also to animate and demonstrate the interaction of models in a better way. On the other hand, I have learnt how a team is affected by getting stuck on an activity since everyone depends on each other.

Jolomi

My primary task was to organize the group and try to move the project forward by assigning task every week for each group member to attempt, we agreed on tasks in class, and I issued them each week throughout the duration of the project.

I was responsible for the documentation and implementation of the algorithms used in the main project and also the coding solutions, and collaborated with Jozimar on the graphs used in the project.

I wrote the Algorithm and Example section of the report and collaborated with Jozimar on the Introduction section of the Report.

I also created the interface used for the Unity environments, created custom unity environments for testings algorithms on and ran tests on trained agents using the various algorithms in the Unity and VizDoom environments.

I trained other agents in custom unity environments, which are in the code, but the two stated in the report proved sufficient.

Finally, I explained the solution code with a ReadMe on Github , and put the final project together on behalf of the group for submission.

Nicholas

Week	Activity
March 2	I helped the team proposal by creating a PowerPoint slide comparing open source vs proprietary technology
March 16	Trained a stable agent in the vizdoom deadly corridor scenario
March 21	Trained a stable agent in the predict position scenario
April 6	Attempted to implement open ai gym in unity environment using the ML_Agents library
April 19	Wrote a chapter on what are convolutional neural networks

April 27	Created custom environment
May 4	Convert custom environment to grayscale to reduce training time
May 11	Trained custom environment to 40000 epochs on virtual machine
May 16	Trained custom environment to 500 000 timesteps Wrote chapter on custom environments Made presentation on custom environments

References

- baeldung, 2021. *Epsilon-Greedy Q-learning*. [Online]
Available at: <https://www.baeldung.com/cs/epsilon-greedy-q-learning>
[Accessed 16 May 2022].
- AurelianTactics, 2018. *Understanding PPO Plots in TensorBoard*. [Online]
Available at: <https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2#:~:text=Value%20Loss%20%E2%80%94%20The%20mean%20loss,decrease%20once%20the%20reward%20stabilizes.>
[Accessed 16 May 2022].
- BOSCH, 2022. *Artificial intelligence in automated driving*. [Online]
Available at: <https://www.bosch.com/stories/artificial-intelligence-in-cars/>
[Accessed 16 May 2022].
- CHATURVEDI, M., 2021. *What Are DQN Reinforcement Learning Models*. [Online]
Available at: <https://analyticsindiamag.com/what-are-dqn-reinforcement-learning-models/>
[Accessed 20 April 2022].
- Clement, J., 2021. [Online]
Available at: https://www.statista.com/topics/868/video-games/#dossierContents_outerWrapper
[Accessed 02 March 2022].
- DeepAI, n.d. *What is a Rectified Linear Unit?*. [Online]
Available at: <https://deepai.org/machine-learning-glossary-and-terms/rectified-linear-units>
[Accessed 16 May 2022].
- Dickson, B., 2021. *Reinforcement learning improves game testing, EA's AI team finds*. [Online]
Available at: <https://bdtechtalks.com/2021/10/04/ea-reinforcement-learning-game-testing/>
[Accessed 12 March 2022].
- Geek, S. t., 2018. *Continuous Action Space Actor Critic Tutorial*. [Online]
Available at: <https://www.youtube.com/watch?v=kWHS2HgbNQ>
[Accessed 16 May 2022].
- GlobalData, 2019. *Video games market set to become a \$300bn-plus industry by 2025*. [Online]
Available at: <https://www.globaldata.com/video-games-market-set-to-become-a-300bn-plus-industry-by-2025/>
[Accessed 11 March 2022].
- Insights, A., 2018. *An introduction to Policy Gradient methods - Deep Reinforcement Learning*. [Online]
Available at: <https://www.youtube.com/watch?v=5P7I-xPq8u8&list=PLUpJwqe3sul4suHI-86H69-buvXXufdKi&index=36>
[Accessed 14 05 2022].

Jarvis, D., 2020. *The AI talent shortage isn't over yet*. [Online]
Available at: <https://www2.deloitte.com/us/en/insights/industry/technology/ai-talent-challenges-shortage.html>
[Accessed 5 March 2022].

Karunakaran, D., 2020. *Deep Q Network(DQN)- Applying Neural Network as a functional approximation in Q-learning*. [Online]
Available at: <https://medium.com/intro-to-artificial-intelligence/deep-q-network-dqn-applying-neural-network-as-a-functional-approximation-in-q-learning-6ffe3b0a9062#:~:text=Error%20or%20loss%20is%20measured,value%20and%20prediction%20Q%20value.>
[Accessed 16 May 2022].

Karunakaran, D., 2020. *Q-learning: a value-based reinforcement learning algorithm*. [Online]
Available at: <https://medium.com/intro-to-artificial-intelligence/q-learning-a-value-based-reinforcement-learning-algorithm-272706d835cf>
[Accessed 16 May 2022].

Kim, T., 2021. *Understanding Proximal Policy Optimization (Schulman et al., 2017)*. [Online]
Available at: <https://towardsdatascience.com/understanding-and-implementing-proximal-policy-optimization-schulman-et-al-2017-9523078521ce>
[Accessed 14 05 2022].

Kirk, M., 2017. Thoughtful Machine Learning with Python - A Test-Driven Approach. In: M. Loukides & S. Cutt, eds. *Thoughtful Machine Learning with Python - A Test-Driven Approach*. California: O'Reilly Media, Inc, pp. 130-143.

Littman, M., 2001. *Markov Decision Processes*. [Online]
Available at: <https://www.sciencedirect.com/topics/computer-science/markov-decision-process>
[Accessed 30 April 2022].

Mahajan, P., 2020. *Fully Connected vs Convolutional Neural Networks*. [Online]
Available at: <https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5#:~:text=A%20fully%20connected%20neural%20network%20consists%20of%20a%20series%20of,be%20made%20about%20the%20input.>
[Accessed 16 May 2022].

O'Reilly Media, 2022. *Deterministic and Nondeterministic Functions*. [Online]
Available at: <https://www.oreilly.com/library/view/sql-in-a/9780596155322/ch04s01s01.html#:~:text=Functions%20can%20be%20either%20deterministic,same%20input%20values%20are%20provided.>
[Accessed 16 May 2022].

OpenAI, 2018. *Proximal Policy Optimization*. [Online]
Available at: <https://spinningup.openai.com/en/latest/algorithms/ppo.html>
[Accessed 14 05 2022].

openAI, 2022. *API, Initializing Environments*. [Online]
Available at: <https://www.gymlibrary.ml/content/api/>
[Accessed 19 April 2022].

openai, 2022. *Gym*. [Online]
Available at: <https://gym.openai.com/>
[Accessed 05 March 2022].

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [Online]
Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
[Accessed 16 May 2022].

Stable Baselines3, 2020. *A2C*. [Online]
Available at: <https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html>
[Accessed 16 May 2022].

Stable Baselines3, 2020. *Custom Policy Network*. [Online]
Available at: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html
[Accessed 16 May 2022].

Stable Baselines3, 2020. *DQN*. [Online]
Available at: <https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html>
[Accessed 16 May 2022].

Stable Baselines3, 2020. *PPO*. [Online]
Available at: <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>
[Accessed 14 05 2022].

Statt, N., 2019. *HOW ARTIFICIAL INTELLIGENCE WILL REVOLUTIONIZE THE WAY VIDEO GAMES ARE DEVELOPED AND PLAYED*. [Online]
Available at: <https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning>
[Accessed 2 March 2022].

Statt, N., 2019. *HOW ARTIFICIAL INTELLIGENCE WILL REVOLUTIONIZE THE WAY VIDEO GAMES ARE DEVELOPED AND PLAYED*. [Online]
Available at: <https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning>
[Accessed 5 March 2022].

Taylor, M. E., 2013. *Reinforcement learning agents providing advice in complex video games*. [Online]
Available at: <https://www.tandfonline.com/doi/full/10.1080/09540091.2014.885279>
[Accessed 19 April 2022].

Unity Technologies, 2021. *Example Learning Environments*. [Online]
Available at: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning->

Environment-Examples.md

[Accessed 03 May 2022].

Violante, A., 2018. *Simple Reinforcement Learning: Temporal Difference Learning*. [Online]

Available at: <https://medium.com/@violante.andre/simple-reinforcement-learning-temporal-difference-learning-e883ea0d65b0>

[Accessed 16 May 2022].

Witkowski, W., 2021. *Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic*. [Online]

Available at: <https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990>

[Accessed 2 March 2022].

Wydmuch, M., n.d. *VIZDOOM*. [Online]

Available at: <http://vizdoom.cs.put.edu.pl/>

[Accessed 5 March 2022].

