UNIVERSITY of ABERTAY DUNDEE

School of Design & Informatics

Session 2019/20

MAT501: Applied Mathematics & Artificial Intelligence

Rule Based System vs Reinforcement Learning

Daniel Zammit Student ID 1905316

MSc Computer Games Technology

Table of Contents

Introduction

The use of artificial intelligence (AI) techniques in video games enables developers to create an enjoyable immersive experience for their target audience. In this sector, the goal of the chosen AI approach needs to be centred around the player. The employed artificial intelligence required adjustments that allowed an experienced player to eventually beat it (Maass, 2019). The application of rule-based systems for designing game AI is a renowned approach. Early literature proposed the development of lightweight rule-based engines for the growing market of mobile games (Hall *et al.*, 2004) . It has also been implemented in popular strategy games such as Age of Empires and the Civilisation franchise (Jones, 2019).

In this paper, a scenario has been created to compare two AI techniques; rule-based system and reinforcement learning. The methodology illustrates the functionality and approach that was used to create both systems, together with the data provided during the exercise runtime. The pseudocode provided highlights the core logic that drives the rule-based system. The game engine of choice for both implementations is Unity. For reinforcement learning, the native ml-agents software development kit (SDK) provided by Unity was imported to the project (Juliani, 2017) . The results extracted from the environment are presented and their relevance is further discussed. The conclusion summarises the findings of the project and suggests future work for the implemented approaches.

Methodology

In order to demonstrate both AI techniques, the Unity game engine was the selection of choice. It is renowned for community support and prototyping ideas in a speedy iteration process. The programming language chosen for implementation is C#. Furthermore, the native SDK for reinforcement learning allowed for easy integration into the game engine. A hazardous scenario was designed for this work, were both AI agents learnt obstacle avoidance and benign object collection. The main level consisted of hazards and collectibles, which had been bound to confine both agents to the playing area. The objective for both agents was to maximise their lifetime in the game world. To achieve this goal, agents were tasked with collecting health items scattered around the level while avoiding hazards. In order to measure the success of each agent, a stopwatch timer represents the duration of the agent before health runs out. A hazard was denoted by a red spike and health was represented by a blue diamond. When the game was running, the life of active agents slowly decayed over time.

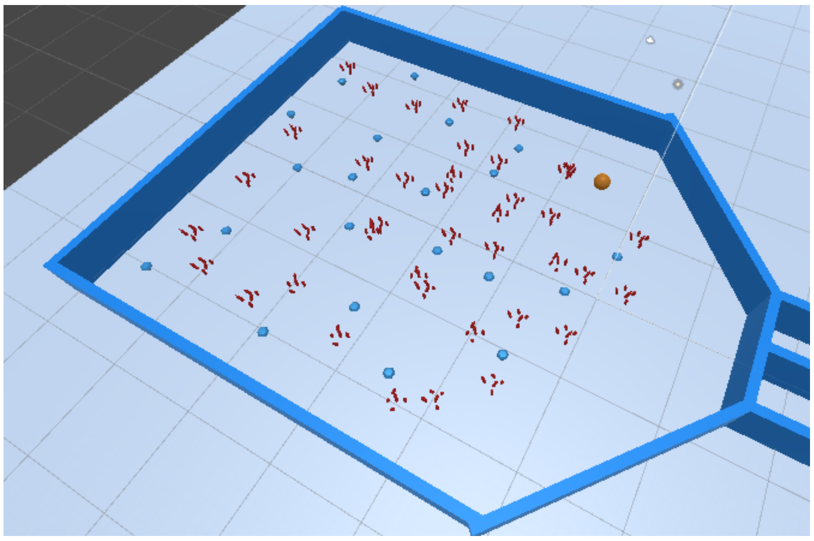


Figure 1 - Area layout of the game environment

Rule-based System

The first step in implementing a rule-based system (RBS) was identifying a set of rules that govern the behaviour of the agent. The rules were based on expert interaction with the game environment. The player controlled the agent by applying movement when a path was clear and changing direction when a hazard was blocking the way to a health collectible. The following pseudocode represents the rules that were implemented for the RBS controller in Unity.

```
Set hazardFlags to false
WHILE agent has health
    Move agent in relative forward direction
    IF collectibleList is not empty AND no hazardFlags are set to true THEN
        Set new direction and rotate towards nearest collectible item
    IF hazard detected on left side of agent THEN
        Change direction and rotation to reflected ray from hit surface
    IF hazard detected in front of agent THEN
        Change direction and rotation to reflected ray from hit surface
    IF hazard detected on right side of agent THEN
        Change direction and rotation to reflected ray from hit surface
ENDWHILE
```

The pseudocode illustrates the game loop for the agent. To implement situational awareness, ray casting checks were used to determine which rule to fire at the position in time. There are three rays in total for each representing a degree of freedom; left, centre and right direction. *hazardFlags* represent boolean operators that are changed during movement, depending on the object that hits the respective ray. The first condition was vital for the rule-based system as it creates a priority for the agent. A hazard in the path towards a collectible item has a higher priority than collecting the health object. Therefore, this rule ensured that even though a health item was within reach, the agent would not blindly traverse the path to its direction.

A list of health items, named *collectibleList*, was implemented to store the position of the nearest object that can be collected. The list is managed through trigger events provided by Unity. A health item that triggers the agent's area of detection is added to the list. It is removed only when consumed or out of vision. The logic that handles consumption and collision with hazards has been implemented within the collision event calls from Unity. Depending on the collision, health was either added or decreased to the agent's life bar.

Reinforcement Learning

The ml-agents SDK provides researchers with a tool to train realistic AI using heuristic, imitation and deep learning techniques. It was native to the Unity engine and available on Github. The inference engine provides researchers the ability to use pre-trained neural network models for their Unity project, as well as cross-platform support (Berges and Zioma, 2019). The principles behind this machine learning technique are states, rewards and actions. The agent needed to perform an action in the environment and depending on how the environment or agent was affected by this decision, a reward is given. The actions can change the state of either the agent or the environment, which means that to achieve the greatest reward, the agent would over time learn to perform according to an expected behaviour.

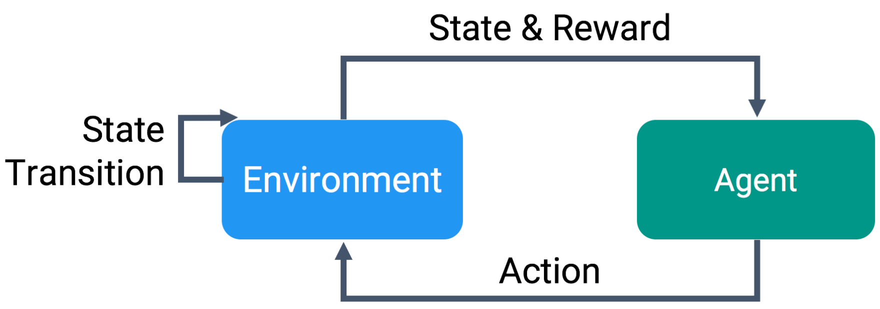


Figure 2 - Reinforcement training cycle (Juliani, 2017)

In practise, an ml-agents environment also requires observations. These were perceived attributes from the viewpoint of the agent. In the game environment, a total of fourteen observations were being collected at each game update. These observations were paired with actions taken in order to determine how well the agent was doing using the rewards as measurement. Each position vector has x, y and z components, and therefore each vector has a size of three observations.

Observation	Description	Vector Space Size
Target Position	Position of nearest health item	3
Agent Position	Position of the agent in world space	3
Spikes Position	Position of the recent two spikes in agent's detection area	6, 3 per spike
Agent's rigid body x component	The velocity value of the agent on the x component of the vector	1
Agent's rigid body z component	The velocity value of the agent on the z component of the vector	1

Table 1- Observations collected and supplied to the agent's brain

The information collected from the observations was sent to the brain. In Unity's approach for reinforcement learning, the brain was the component that contains the logic for the next action the agent must take. This was dependant on the observation and reward from the previous action. Finally, the academy oversees the operation of data gathering through observation and the next action for the agent. The main purpose of the academy was to send all the information to a Python API that handles machine learning algorithms. This was done through the external communicator that was found in the academy.

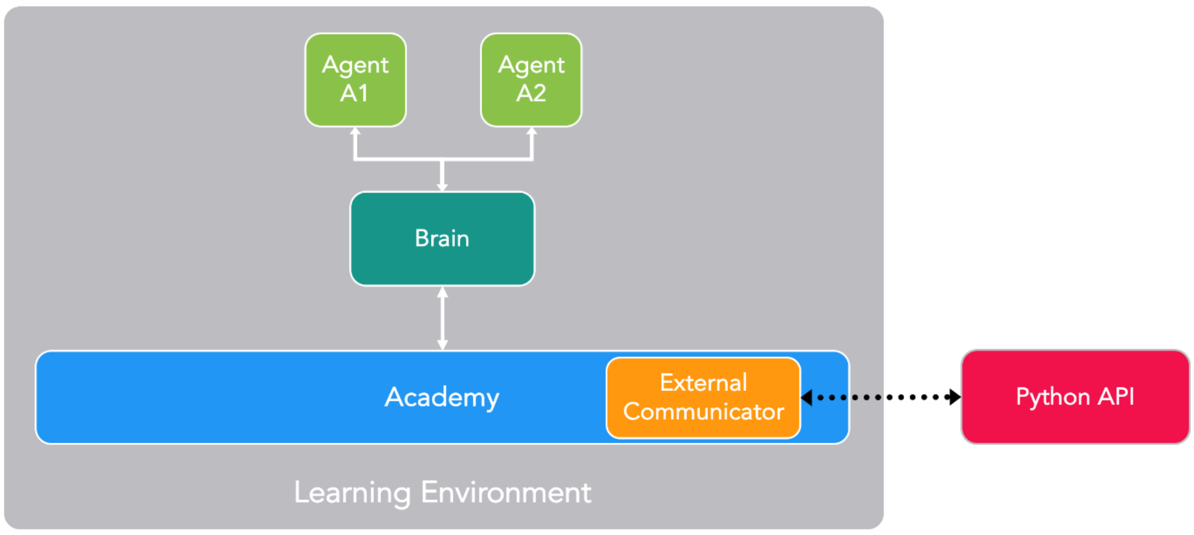


Figure 3 – Process flow of ml-agents (Juliani, 2017)

The external communicator sent data to Unity’s API that communicates with Tensorflow, an open source machine learning platform developed by Google. Tensorflow processes the observations and sent a decision back to the agent through the academy. The output of the learning process was a Tensorflow neural network model, that was supplied to a learning brain in the environment’s scene in Unity. The end goal for reinforcement learning was to learn a policy that will effectively optimise the decisions taken by maximising the reward earned for completing that action.

The reward system designed followed best practises from examples provided in ml-agents SDK. The agent was only rewarded when the all items had been collected, rather than providing a reward for each collected item. In addition, a negative reward was given when hitting walls and hazards. The reward for touching walls was larger than hitting hazards. Therefore, the agent was encouraged to explore the floor space and minimise the negative rewards. Also, it ensured that the current training episode was completed in order to allow the academy to progress to the next step of the simulation loop. Due to the complexity of the problem, a high number of max steps was required for the agent to find an optimal solution. In total, the training simulation lasted approximately three days.

Results

In order to test both AI techniques in the environment, a stop watch timer was implemented for each agent. The objective of the exercise was to collect as much health items as possible before life runs out. Life decays constantly every second during runtime and can only be increased when consuming health.

Run/Survival Time (in seconds)	Rule-based Agent	Reinforcement Agent
1	117	24.9
2	53	24.8
3	**93**	24.8
4	58	24.85
5	114	24.7
6	25	24.8
7	118	24.9
8	93	24.9
9	74	26.2
10	164	24.9
Average Survival Time	90.9	24.97

Table 2 - Survival time per agent in ten runs

The rule-based agent was able to traverse the level while bumping into hazards less often than the reinforcement agent. The physics ray casts that were used to determine the type of object near the agent gave it an advantage when compared to its adversary. On the other hand, the rule-based agent was unaware of health items that are outside the area of detection. During testing, it was noticed that although the agent took less damage in each run, it eventually starved to death as it was not able to locate the remainder of health items. On one occasion, marked in red in the results table, the agent was able to collect all health items. However, even though the maximum health was collected, during the run the agent received a substantial amount of damage to pursue its goal.

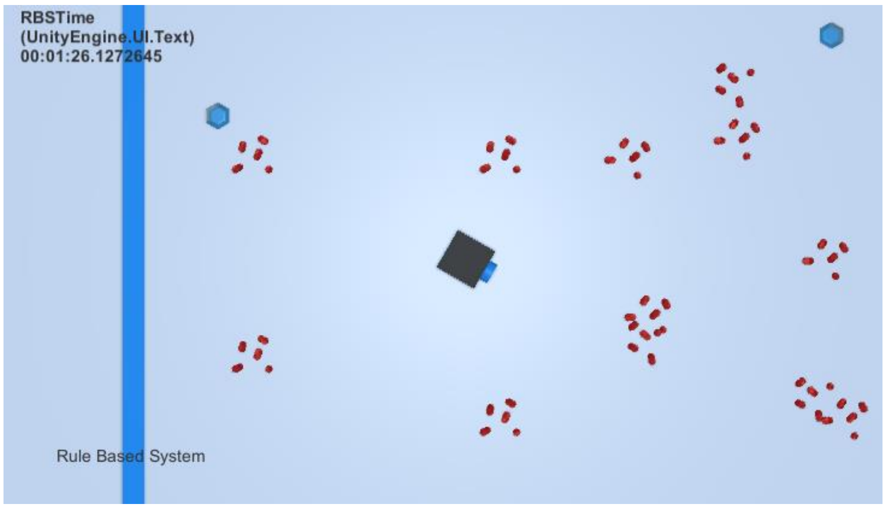
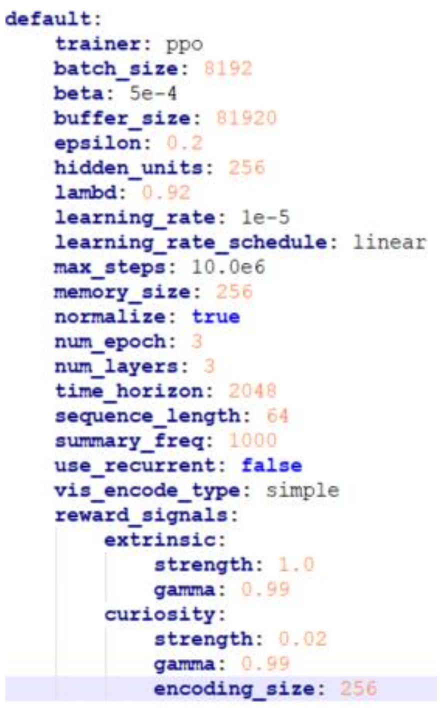


Figure 4 - Rule-based agent in the environment

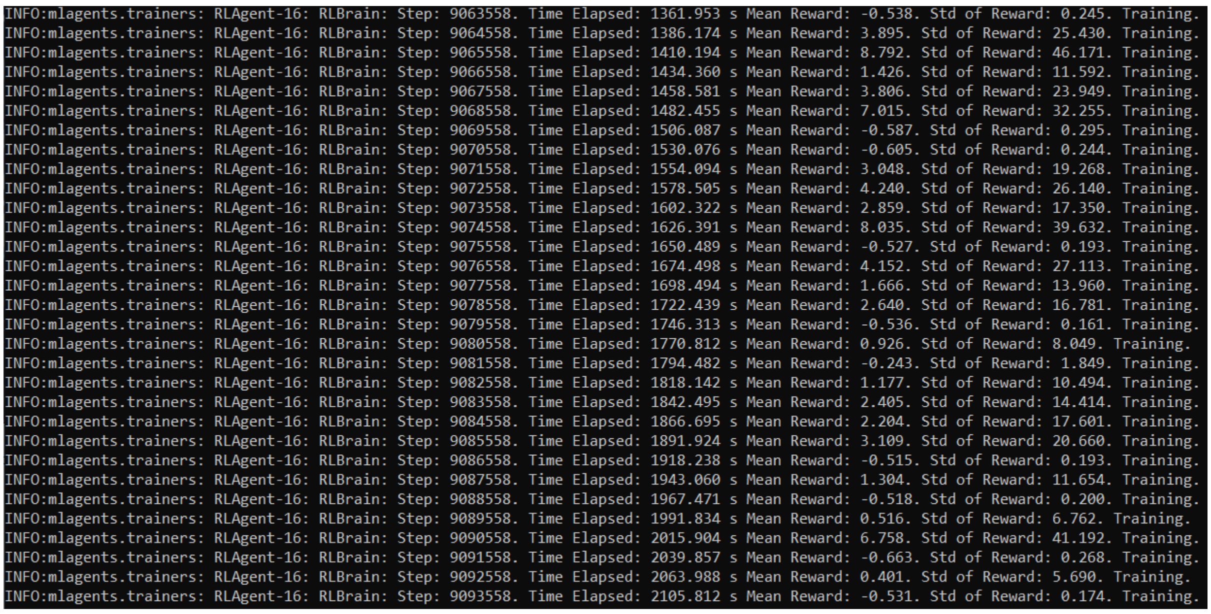
The reinforcement agent results represented from the table above have been generated using the proximal policy optimisation (PPO) machine learning algorithm. The neural network used in this algorithm closes on the ideal function that guaranteed the best action the agent can take from observations given the current state in time. During runtime, the policy was updated to reflect the rewards given for each action (Schulman *et al.,* 2017). An advantage of PPO is the ease of implementation when compared to the quality of the result that can be achieved. At its core, reinforcement learning suffered from hyperparameter sensitivity and maximum steps initialisation. PPO enabled ease of tuning for these hyperparameters to modify values and continue the training process from a saved state, without the need to start the process again. The hyperparameters loaded from the configuration file used for the agent's training can be found below.



```
default:
    trainer: ppo
    batch_size: 8192
    beta: 5e-4
    buffer_size: 81920
    epsilon: 0.2
    hidden_units: 256
    lambd: 0.92
    learning_rate: 1e-5
    learning_rate_schedule: linear
    max_steps: 10.0e6
    memory_size: 256
    normalize: true
    num_epoch: 3
    num_layers: 3
    time_horizon: 2048
    sequence_length: 64
    summary_freq: 1000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

Figure 5 - Hyperparameters configuration file for reinforcement agent

The fundamental parameters that were changed for this scenario were batch size, buffer size, beta, learning rate, maximum steps and time horizon values. In addition, the curiosity module was toggled to reward the agent for exploring the actions space. The first two values changed correspond to the number of observations, actions and rewards that were collected before updating the policy model. The beta value ensured that the agent properly explores the action space, with a higher value resulting in more random actions. The learning rate was used as the value of the weight updates when applying the next stochastic gradient descent for the neural network (Brownlee, 2019). Lastly, ten million steps were used for this simulation given the complexity of the problem. Time horizon corresponded to total steps of experience that were considered before adding to the experience buffer.



```
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9063558. Time Elapsed: 1361.953 s Mean Reward: -0.538. Std of Reward: 0.245. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9064558. Time Elapsed: 1386.174 s Mean Reward: 3.895. Std of Reward: 25.430. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9065558. Time Elapsed: 1410.194 s Mean Reward: 8.792. Std of Reward: 46.171. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9066558. Time Elapsed: 1434.360 s Mean Reward: 1.426. Std of Reward: 11.592. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9067558. Time Elapsed: 1458.581 s Mean Reward: 3.806. Std of Reward: 23.949. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9068558. Time Elapsed: 1482.455 s Mean Reward: 7.015. Std of Reward: 32.255. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9069558. Time Elapsed: 1506.087 s Mean Reward: -0.587. Std of Reward: 0.295. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9070558. Time Elapsed: 1530.076 s Mean Reward: -0.605. Std of Reward: 0.244. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9071558. Time Elapsed: 1554.094 s Mean Reward: 3.048. Std of Reward: 19.268. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9072558. Time Elapsed: 1578.505 s Mean Reward: 4.240. Std of Reward: 26.140. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9073558. Time Elapsed: 1602.322 s Mean Reward: 2.859. Std of Reward: 17.350. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9074558. Time Elapsed: 1626.391 s Mean Reward: 8.035. Std of Reward: 39.632. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9075558. Time Elapsed: 1650.489 s Mean Reward: -0.527. Std of Reward: 0.193. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9076558. Time Elapsed: 1674.498 s Mean Reward: 4.152. Std of Reward: 27.113. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9077558. Time Elapsed: 1698.494 s Mean Reward: 1.666. Std of Reward: 13.960. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9078558. Time Elapsed: 1722.439 s Mean Reward: 2.640. Std of Reward: 16.781. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9079558. Time Elapsed: 1746.313 s Mean Reward: -0.536. Std of Reward: 0.161. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9080558. Time Elapsed: 1770.812 s Mean Reward: 0.926. Std of Reward: 8.049. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9081558. Time Elapsed: 1794.482 s Mean Reward: -0.243. Std of Reward: 1.849. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9082558. Time Elapsed: 1818.142 s Mean Reward: 1.177. Std of Reward: 10.494. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9083558. Time Elapsed: 1842.495 s Mean Reward: 2.405. Std of Reward: 14.414. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9084558. Time Elapsed: 1866.695 s Mean Reward: 2.204. Std of Reward: 17.601. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9085558. Time Elapsed: 1891.924 s Mean Reward: 3.109. Std of Reward: 20.660. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9086558. Time Elapsed: 1918.238 s Mean Reward: -0.515. Std of Reward: 0.193. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9087558. Time Elapsed: 1943.060 s Mean Reward: 1.304. Std of Reward: 11.654. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9088558. Time Elapsed: 1967.471 s Mean Reward: -0.518. Std of Reward: 0.200. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9089558. Time Elapsed: 1991.834 s Mean Reward: 0.516. Std of Reward: 6.762. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9090558. Time Elapsed: 2015.904 s Mean Reward: 6.758. Std of Reward: 41.192. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9091558. Time Elapsed: 2039.857 s Mean Reward: -0.663. Std of Reward: 0.268. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9092558. Time Elapsed: 2063.988 s Mean Reward: 0.401. Std of Reward: 5.690. Training.
INFO:mlagents.trainers: RLAgent-16: RLBrain: Step: 9093558. Time Elapsed: 2105.812 s Mean Reward: -0.531. Std of Reward: 0.174. Training.
```

Figure 6 - Reinforcement Agent progressing through training

Several tweaks were required to obtain the desired result. During testing, these parameters provided the best learning rate over time. The reinforcement agent was able to gather all resources in training mode. However, given no perception or situational awareness, the agent hit several hazards during the learning process. When the agent was placed in the environment, these issues became more evident in the initial actions taken. As a result, the reinforcement agent did not perform well in this scenario.

Conclusions

In conclusion, the results achieved for the reinforcement agent show that in this scenario, the rule-based approach was far more superior. For future work on the project, it would be recommended to implement a mechanism for the rule-based agent to find the second nearest item closest to its position in the case were agent starts looping between two blocking directions. The reinforcement agent relied on a total of sixteen vector observations from the environment, however given the complexity of the scenario, future work can enhance the agent's performance by adding both local perception, similar to the work conducted in Lai's research (LAI, Xiliang and ZHANG, 2019), and also an increase in the number of simulation steps.

References

Berges, V. and Zioma, R. (2019) *Unity ML-Agents Toolkit v0.7: A leap towards cross-platform inference.* Available at: https://blogs.unity3d.com/2019/03/01/unity-ml-agents-toolkit-v0-7-a-leap-towards-cross-platform-inference/ (Accessed: Nov 2, 2019).

Brownlee, J. (2019) 'How to Configure the Learning Rate When Training Deep Learning Neural Networks', *Machine Learning Mastery,* -01-22T18:00:54+00:00. Available at: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ (Accessed: Nov 4, 2019).

Hall, L., Gordon, A., James, R. and Newall, L. (Sep 2, 2004) *A lightweight rule-based AI engine for mobile games.* . June 2004. ACM, pp. 284.

Jones, M.T. (2019) 'Machine learning and gaming', *IBM Developer,* -06-04. Available at: https://developer.ibm.com/articles/machine-learning-and-gaming/ (Accessed: Nov 1, 2019).

Juliani, A. (2017) *Introducing: Unity Machine Learning Agents Toolkit.* Available at: https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/ (Accessed: Nov 1, 2019).

LAI, J., Xiliang, C. and ZHANG, X. (2019) 'Training an Agent for Third-person Shooter Game Using Unity ML-Agents', *International Conference on Artificial Intelligence and Computing Science,* ICAICS 2019. doi: 10.12783/dtcse/icaic2019/29442.

Maass, L.E.S. (2019) *Artificial Intelligence in Video Games.* Available at: https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22 (Accessed: Oct 31, 2019).

Schulman, J., Klimov, O., Wolski, F., Dhariwal, P. and Radford, A. (2017) *Proximal Policy Optimization.* Available at: https://openai.com/blog/openai-baselines-ppo/ (Accessed: Nov 3, 2019).