Reinforcement Learning - MiniHack

Tristan Le Forestier - 1835635
Michael Gomes - 1644868
Jesse Bristow - 1875955
Goolam Fareed Bangie - 1828201

November 10, 2021

Contents

1 Introduction **3**

2 Deep Q Learning (DQN) **4**
- 2.1 General Descriptions . . . 4
- 2.2 Deep Q Learning Implementation . . . 4
  - 2.2.1 The Developed Agents . . . 6
- 2.3 Hyper-parameters Used . . . 6
- 2.4 Result Plots of DQN Agents and Analysis . . . 7
  - 2.4.1 Movement Only Agent . . . 7
  - 2.4.2 Limited Actions Agent . . . 8
  - 2.4.3 Random Agent . . . 8
  - 2.4.4 Agent Sub-task Completion . . . 9
- 2.5 Deep Q Learning Conclusions . . . 9
- 2.6 Best Videos for each DQN Agent . . . 9

3 Advantage Actor Critic Algorithm (A2C) **10**
- 3.1 General Descriptions . . . 10
- 3.2 Actor Critic Implementation . . . 10
  - 3.2.1 The Developed Agents . . . 11
- 3.3 Hyper-parameters Used . . . 12
- 3.4 Result Plots of Actor Critic Agents and Analysis . . . 12
  - 3.4.1 Movement Only Agent . . . 12
  - 3.4.2 Limited Actions Agent . . . 14
  - 3.4.3 Random Agent . . . 15
- 3.5 Actor Critic Conclusions . . . 16
- 3.6 Link to Best Videos for each Actor Critic Agent . . . 17

4 Comparison Plots **17**
  - 4.0.1 Reward vs Episode Comparison . . . 17
  - 4.0.2 Sub-task completion Comparison . . . 18

5 Conclusion **18**

1 Introduction

Reinforcement learning is one of many fields in the broad subject of machine learning. It's all about taking the right steps to maximize your reward in a given situation. It is used by a variety of software and computers to determine the best feasible action or path in a given situation. In reinforcement learning we attempt to maximize rewards in any environment we tend to find the agent in. Reinforcement learning has been used successfully in healthcare sectors as well as engineering sectors, in this report we will investigate its effectiveness in the gaming world.

In this project, we will be making use of the Minihack framework which is built on top of both the Nethack [4] and Gym [5] frameworks. Minihack allows us to train Reinforcement Agents using a multitude of algorithms on a variety of environments that range in complexity.

The tasks that an agent can learn to perform include but are not limited to, navigation tasks and skill acquisition tasks. In this report the environment that our agent will attempt to learn on is the MiniHack Quest Hard environment, which involves first navigating a maze that changes in its structure followed by a pickup skill that is needed to pickup an object that will allow the agent to cross the lava river and a wand that the agent will need to use to kill a monster on the opposite side of the river and lastly navigate to the end of the dungeon, marked by the staircase. The link to the github repository containing the code can be found at `https://github.com/bongo1007/Reinforcement-Learning`

The two algorithms, which are deep learning in nature, that we have decided to use in order to train our agent are mentioned below:

- Deep Q Learning Algorithm (Value Based Method)
- Advantage Actor Critic Algorithm (Policy Based Method)

The following chapters will be covered in this report in the order that is represented below:

- Deep Q Learning (DQN)
- Advantage Actor Critic Algorithm (A2C)
- Comparison Plots
- Conclusion

2 Deep Q Learning (DQN)

2.1 General Descriptions

Q learning is an algorithm in reinforcement learning where the agent learns to take actions in the environment based on a value associated with each state-action pair, here the value will tell us the best action to take at each state. Prior to DQN's a tabular method was used to store the values for each state action pair however in this environment the number of states we would have to keep track of is infeasible and as such a function approximation technique would be needed to estimate the Q values for each state action pair. A DQN is a neural network which allows us to implement Q-learning in reinforcement learning where the state space and/or the action space is quite large. This allows us to use non-linear function approximation in order to determine the approximate Q values rather than updating a large number of Q values in a tabular format. Our goal is to find Q*. Since we don't have access to this we make use of the DQN to approximate the Q*.

The main loop follows the following steps: We start by resetting the environment and initializing the state Tensor. Then we take a sample of an action and execute it We then observe the following screen and the reward and optimize our model once more. We restart the loop after the episode ends. The actions in the sample sample is taken from the gym environment and is picked either randomly or based on a policy. Every iteration, we save the results in the replay memory and run the optimization process. To train the new policy, Optimization selects a random batch from the replay memory. The expected Q values are computed using an older target net, which is updated to keep it relevant and correct for future use. [3]

2.2 Deep Q Learning Implementation

In order to try and complete the dungeon we used two separate DQN's. The one DQN was used to try and learn to pass the maze while the other DQN was used to try and learn to finish off the rest of the dungeon. During training, as soon as we find the end of the maze during an episode, we immediately switch from optimising and using the maze DQN to optimising and using the second DQN.

Our maze DQN would accept observations which are comprised of the colors crop observation available from the environment and a visited matrix of the same shape. This visited matrix is part of the reward shaping we did. It essentially is a matrix which holds information on which cells have already been visited by the agent and how recent those cells have been visited. The reason for using this matrix is to try promote exploration within the environment to try get to the maze exit. A cell in this matrix is treated as unvisited if its value is greater than 200 and otherwise the value represents how long ago the cell was visited. We

use the information of this matrix in our model and during our reward shaping.

For the reward shaping aspect of our maze DQN implementation, if the agent visits a cell which has been found to be previously unvisited, then it gets a reward of 1. It gets a reward of -0.3 if it visits a cell that has been visited recently and a reward of -0.7 if it goes into a wall. The reason for the reward of -0.7 for going into a wall is so that if it got stuck at a deadend the model would learn to prefer visiting the most recently visited cell instead of going into the wall. These reward shaping ideas above helped our agent explore more but was not sufficient as it would still get stuck sometimes. Thus, a reward between 0 and 0.3 was also given depending on how long ago the cell the agent just moved to was visited (if it moves to recently visited cell it gets a reward closer to 0 and if it moves to a cell that has not been recently visited then it gets a reward closer to 0.3). Finally to make the agent finish the maze more frequently, we set all the cells along the top and bottom row from half way all the way to the end to be very recently visited so that the agent had no incentive to waste it's time exploring those sections.

Every time a step is taken, then the visited matrix is updated. We crop this visited matrix around the agent and extract information so it can be passed to the network along with the color_crop observation to be propagated through the network. The information we extract from the visited matrix is which adjacent cells to the agent have been visited or not, which adjacent cell has been the most recently visited one and how long ago each adjacent cell has been visited. With this extracted information and the color_crop, our model can update the network correctly and try to get us to the next best action.

Our second DQN would accept observations which are comprised of the color_crop observation and the location of the agent. For the reward shaping aspect of our second DQN implementation, the agent would receive a reward of +1 every time it reached a new maximum x-coordinate. The reason for this was to incentivise the agent to go all the way to the right(which will take it to the end). We also gave the agent a reward of +5 for either killing the monster, reaching the end of the lava or reaching the final stairs.

We chose to use pytorch to implement the dqn. Both of the DQN's consisted of 3 convolutional layers with batch normalization between each layer and one fully connected layer to the output. The convolutional layers used a kernel of size 1 with a stride of 1 since this seemed to produce the best results with such a small sized input image. We first fed the color_crop images through the convolutional layers, then concatenated the extra state information to the flattened output, and then finally fed that vector through the fully connected layer. The best maze DQN model that was used to solve the maze had 4 output neurons, one for each of the movement actions, while the second DQN model that was used had an output neuron for each action in our chosen limited action space, although we

developed multiple agents that utilised different action spaces in order to analyse their performance.

2.2.1 The Developed Agents

We made 3 different DQN Agents:

- *Movement Only* - This agent's action space was solely limited to the navigation actions while it was in the maze and then limited to the limited actions described below after it finished the maze. We did this as we found when using the standard action space, the agent would learn to use actions like pray/sit as it would get a reward of 0 for using this action instead of the standard reward of -0.1 for taking a different action. Thus, we ensure that our agent actually does try navigate the environment instead of staying in the same block and hopefully it can at least solve the maze.
- *Limited Actions* - This agent's action space was limited to the navigation actions and a few others namely: apply, autopickup, cast, close, drop, eat, esc, fire, fight, invoke, kick, loot, look, open, pray, puton, quaff, read, remove, ride, rub, search, takeoff, takeoffall, tip, wear, wield and zap. We did this to see if an agent with an extended action space to the one above would still learn to navigate the maze and try complete other sub-tasks.
- *Random* - This agent purely just takes random actions from the movement actions.

We do 3 training runs of over 100 episodes each to ensure our agents are learning properly and to ensure we have data to make average reward plots over each run.

2.3 Hyper-parameters Used

Hyper-parameter	Value
Gamma Maze	0
Gamma Non-maze	0.999
Learning Rate	0.001
Epsilon	0.2
Target Network Update	5000
Replay Memory Size	5000

Figure 1: Agent Hyper-parameters

Gamma defines how discounted our rewards are going to be. *Learning Rate* defines the learning rate of our SGD optimizer we use in our network/model. *Epsilon* defines how much the agent explores compared to exploits. *Target Network*

Update defines how often the target network is updated with the policy network parameters. *Replay Memory Size* defines how many past state/action/reward/next state values we can store at a maximum.

2.4 Result Plots of DQN Agents and Analysis

Note that the results over the 100 episodes are generated from both DQN models. The maze DQN is used until the maze is completed in an episode and then the other DQN is used.

2.4.1 Movement Only Agent

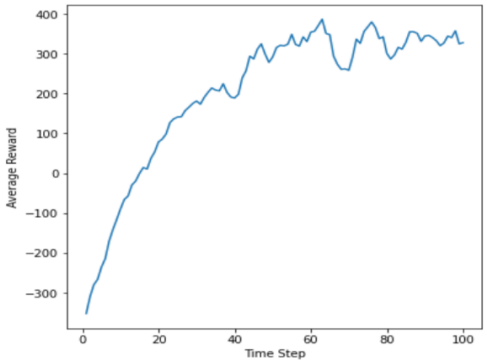

Figure 2: Rewards per Episode

The plot above (2) shows the reward for the episode averaged over 3 runs vs the episode number when only using the 4 movement actions. It has also been smoothed slightly with a sliding window of length 10. We can clearly see that the agent learns to achieve higher and higher rewards over time.

2.4.2 Limited Actions Agent

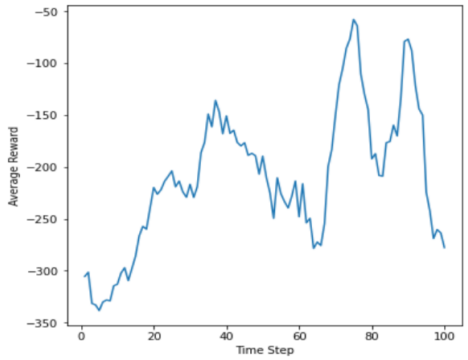

Figure 3: Rewards per Episode

The plot above (3) shows the reward for the episode averaged over 3 runs vs the episode number when using our limited action space. It has also been smoothed slightly with a sliding window of length 10. We can see that the agent learns over time although it doesn't learn to perform as well as the agent which only uses the movement actions. We can also see that the rewards that the agent achieves are fairly volatile. The reason for this instability and poorer learning is because it takes more time to learn and sufficiently explore a larger action space.

2.4.3 Random Agent

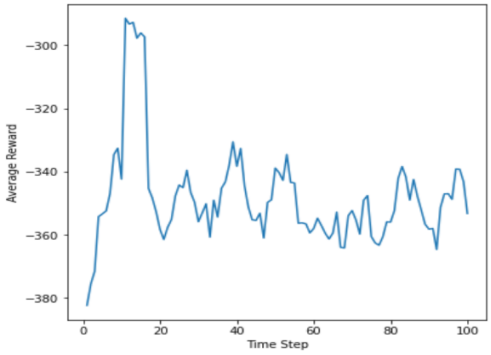

Figure 4: Rewards per Episode

The plot above (4) shows the reward for the episode averaged over 3 runs vs the episode number when only using random actions. It has also been smoothed

slightly with a sliding window of length 10. We can clearly see that the agent achieves very random and volatile rewards.

2.4.4 Agent Sub-task Completion

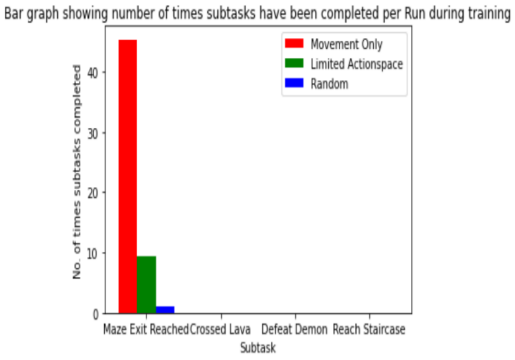

Figure 5: Sub-tasks Plot

The plot above (5) is a bar graph that shows the average number of times each sub-task is completed during training for each different agent. We can clearly see that the agent that only uses the movement actions learns to perform much better compared to the other agents on the maze, although none of them manage to perform well on the rest of the maze.

2.5 Deep Q Learning Conclusions

In conclusion, the Movement Only agent was our best performing actor critic agent as it completed sub-tasks the most out of the other agents. This is due to the fact that it actually learnt to find the maze exit compared to the Random agent which did it by accident and it learnt to finish the maze more consistently compared to the Limited Actions agent. None of the agents performed very well after finishing the maze.

2.6 Best Videos for each DQN Agent

One of each agent's best performances were recorded as a video and uploaded to the github page.

3 Advantage Actor Critic Algorithm (A2C)

3.1 General Descriptions

An Actor Critic algorithm is classified as a policy gradient based reinforcement learning algorithm. Policy gradient based reinforcement algorithms find optimal policies by finding some parameters (in our case weights of our model/network used) which parameterize a policy that maximize the expected rewards.

All reinforcement learning algorithms try to maximize the expected rewards, except in the case of policy based algorithms as the policy is essentially formulated by a model/network. The Actor Critic algorithm is advantageous in this regard for high dimensional problems as the policy is updated directly as opposed to more traditional reinforcement learning algorithms which find values depending on the state action pairs and then only update the policy. Due to us using the MiniHack-Quest-Hard-v0 [5] environment for this project, it makes sense for us to use an Actor Critic algorithm for this as an easier way to update the policy due to the sheer complexity of the environment such as the many possible combinations for observations and the hundreds of possible actions at every state [1].

Actor Critic algorithms are comprised of two important parts named the Actor and the Critic [1]. The Critic's role is to estimate the value or state-value function and inform the Actor how good the previous action was. The Actor's role is to update the policy via changing the policy distribution according to what the critic says and determine which action to take next. Another benefit of using an Actor Critic algorithm is that variance is reduced as opposed to more traditional policy gradient methods. Both the Critic and the Actor can be parameterized by the same model/network or two separate networks.

3.2 Actor Critic Implementation

For our Actor Critic implementations, we utilized an Advantage Actor Critic algorithm [1] with a singular convolutional neural network/model to parameterize both the Actor and Critic parts similar to [6] and [2]. Our network would accept observations which are comprised of the *colors_crop* observation available from the environment and a *visited_matrix* of the same shape. This *visited_matrix* is part of the reward shaping we did. It essentially is a matrix which holds information on which cells have already been visited by the agent and how recent those cells have been visited. The reason for using this matrix is to try promote exploration within the environment to try get to the maze exit. A cell in this matrix is treated as unvisited if its value is greater than 1000 and otherwise the value represents how long ago the cell was visited. We use the information of this matrix in our model and during our reward shaping.
For the reward shaping aspect of our implementations, if the agent visits a cell which has been found to be previously unvisited, then it gets a reward of 1.

It gets a reward of -0.3 if it visits a cell that has been visited recently and a reward of -0.7 if it goes into a wall. The reason for the reward of -0.7 for going into a wall is so that if it got stuck at a deadend the model would learn to prefer visiting the most recently visited cell instead of going into the wall. These reward shaping ideas above helped our agent explore more but was not sufficient as it would still get stuck sometimes. Thus, a reward between 0 and 0.3 was also given depending on how long ago the cell the agent just moved to was visited (if it moves to recently visited cell it gets a reward closer to 0 and if it moves to a cell that has not been recently visited then it gets a reward closer to 0.3).

Every time a step is taken, then the *visited_matrix* is updated. We crop this *visited_matrix* around the agent and extract information so it can be passed to the network along with the *color_crop* observation to be propagated through to update the Critic and in turn the Actor as well. The information we extract is which adjacent cells to the agent have been visited or not, which adjacent cell has been the most recently visited one and how long ago each adjacent cell has been visited. With this extracted information and the *colors_crop*, our model can update the policy correctly and try get us to the next best action. A reward of 100 is also given to the agent for reaching the maze exit coordinates to try encourage it to complete that sub-task. The agent can get that reward multiple times on the same episode so on the following episodes it tries to get to that coordinate faster.

3.2.1 The Developed Agents

We made 3 different Actor Critic Agents:

- *Movement Only* - This agent's action space was solely limited to the navigation actions. We did this as we found when using the standard action space, the agent would learn to use actions like pray/sit as it would get a reward of 0 for using this action instead of the standard reward of -0.1 for taking a different action. Thus, we ensure that our agent actually does try navigate the environment instead of staying in the same block and hopefully it can at least solve the maze.

- *Limited Actions* - This agent's action space was limited to the navigation actions and a few others namely: apply, autopickup, cast, close, drop, eat, esc, fire, fight, invoke, kick, loot, look, open, pray, puton, quaff, read, remove, ride, rub, search, takeoff, takeoffall, tip, wear, wield and zap. We did this to see if an agent with an extended action space to the one above would still learn to navigate the maze and try complete other sub-tasks.

- *Random* - This agent purely just takes random actions from the entire action space of the environment. It does not utilize any of the modified reward shaping above but instead the normal defined rewards from the environment. This is why the rewards for the random agent will be quite

low and close to zero compared to the agents above as it is not heavily penalized by the custom reward shaping.

We do 3 training runs of over 200 episodes each to ensure our agents are learning properly and to ensure we have data to make average reward plots over each run.

3.3 Hyper-parameters Used

Hyper-parameter	Value
Gamma	0
Learning Rate	0.001
Seed	np.random.randint(100)

Figure 6: Agent Hyper-parameters

Gamma defines how discounted our rewards are going to be. *Learning Rate* defines the learning rate of our Adam optimizer we use in our network/model. *Seed* defines the seed for each run. As you can see the seed is randomized for each run ensuring that our agent is not producing false results by just performing well on a singular predefined seed.

3.4 Result Plots of Actor Critic Agents and Analysis

3.4.1 Movement Only Agent

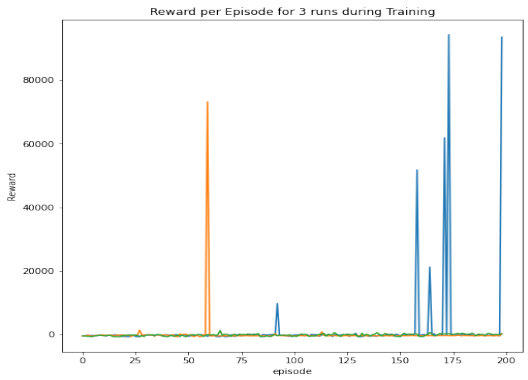

Figure 7: Rewards per Episode

From the plot above (7), we can see that the agent found the maze exit a few times on each run by seeing the visible spikes in rewards on the graph. The blue run found the maze exit quite a few more times than the others especially in its

later episodes which means it did learn to try get to the maze exit. Remember these rewards are elevated due to the custom reward shaping we used.

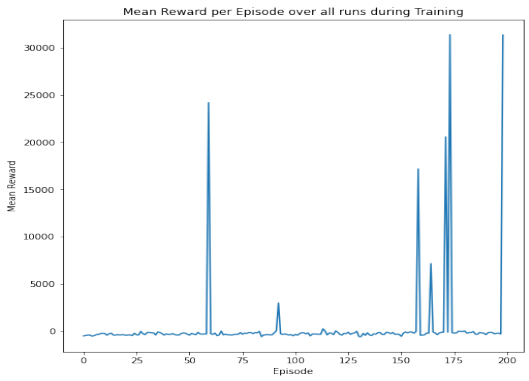

Figure 8: Mean rewards per Episode

The mean rewards (8) over all runs backs up that the agent found the maze exit quite a few more times in its later episodes compared to its earlier episodes which means it did learn to try get to the maze exit.

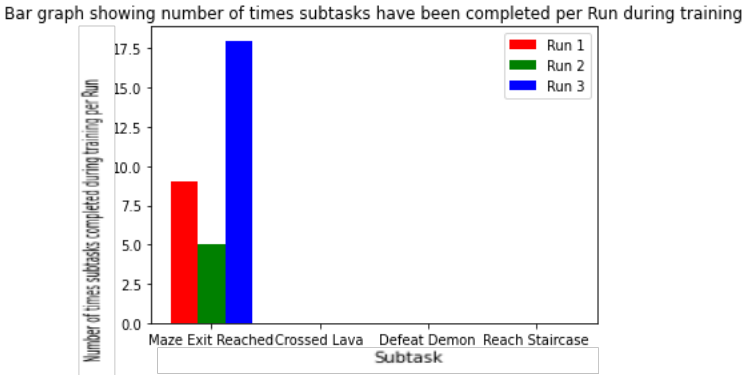

Figure 9: Sub-tasks Plot

The plot above (9) confirms that the blue run was the best run as it found the maze exit in 18 of its episodes but more importantly it shows that the movement only agent did learn to complete the maze of the environment. The agent did not have the action space available to try learn to complete the other sub-tasks (completed 0 times) which is why there are no graphs visible for the other sub-tasks.

3.4.2 Limited Actions Agent

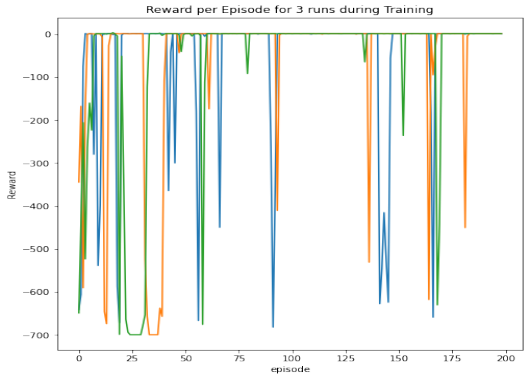

Figure 10: Rewards per Episode

From the plot above (10), we can see that the agent never found the maze exit as the rewards are only below 0. This is probably due to the fact that the agent had to many actions at its disposal to actually learn how to complete the maze or it constantly used an action like sit/pray which would give it a reward of 0. Remember these rewards are like this due to our custom reward shaping.

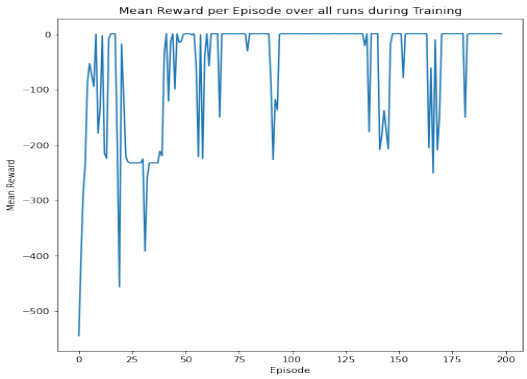

Figure 11: Mean rewards per Episode

The mean rewards over all runs just confirms the ideas put forward above (11).

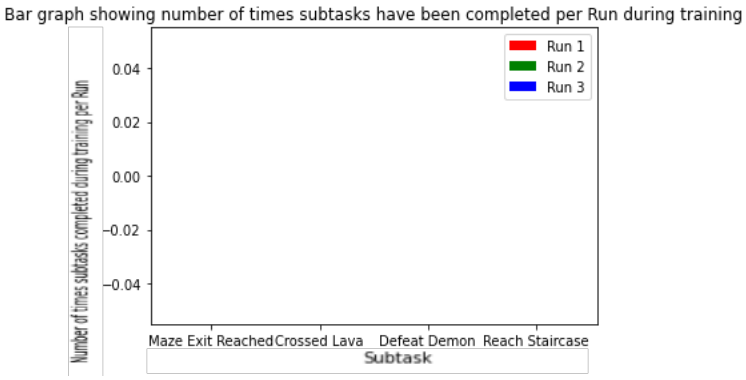

Figure 12: Sub-tasks Plot

The plot above confirms that this agent learned to do no sub-tasks at all (12).

3.4.3 Random Agent

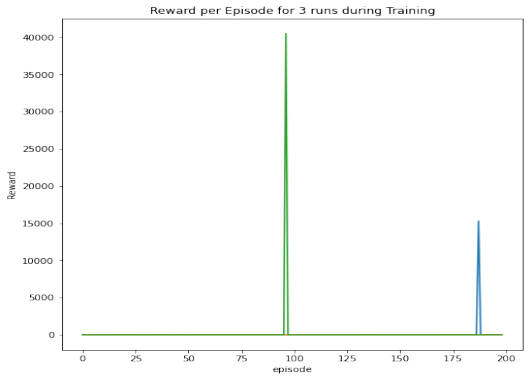

Figure 13: Rewards per Episode

From the plot above (13), we can see that the random agent found the maze exit a few times on each run by seeing the visible spikes in rewards on the graph which is quite impressive. One major difference compared to the Only Movement Agent is that the frequency of reaching the maze exit did not increase near the last few episodes which means it found the maze exit by chance and not due to the fact that it learnt to find the exit. Remember these rewards are elevated due to the custom reward shaping we used.

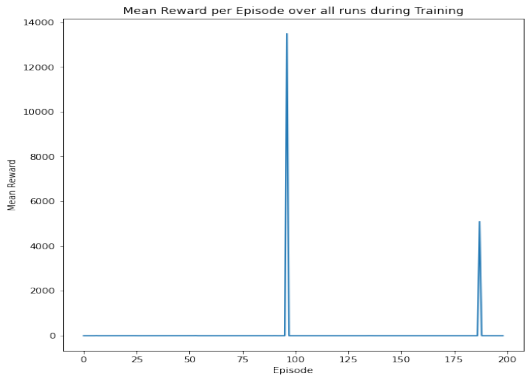

Figure 14: Mean rewards per Episode

The mean rewards over all runs just confirms the ideas put forward above (14).

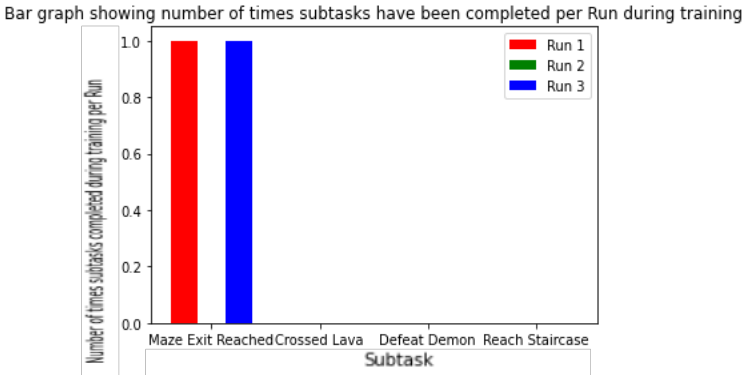

Figure 15: Sub-tasks Plot

The plot above confirms that the blue and red run were the best runs for this agent as they both found the maze exit once. The also displays that the agent did not complete any other sub-tasks randomly as well (15).

3.5 Actor Critic Conclusions

In conclusion, the Movement Only agent was our best performing actor critic agent as it completed sub-tasks the most out of the other agents. This is due to the fact that it actually learnt to find the maze exit compared to the Random agent which did it by accident and the Limited Actions agent which did nothing. We also know that the Movement Only agent did not have the action space necessary to complete the other sub-tasks which is why we will suggest some ideas to anyone who may want to continue to try improve upon the agents we have created for this project. We would suggest that different agents designed to

complete each different sub-tasks are created with limited but necessary action spaces. For example, we used a Movement Only agent and it learnt to explore and find the maze exit. The idea would then to train another agent for the next sub-task of crossing the lava with the movement actions and actions such as puton, pickup, search, wear, wield and zap in the hope it could learn to complete this sub-task.

3.6 Link to Best Videos for each Actor Critic Agent

The ttyrec files and the videos will also be available on the GitHub repository as well as in the submission file. The links below will redirect you to a Youtube page where the videos of the respective agents can be found.

- *Movement Only* - `https://www.youtube.com/watch?v=caMugKbVEL8&ab_channel=MichaelGomes`
- *Limited Actions* - `https://www.youtube.com/watch?v=BjmBjcDINJU&ab_channel=MichaelGomes`
- *Random* - `https://www.youtube.com/watch?v=1939JvGOoVU&ab_channel=MichaelGomes`

4 Comparison Plots

4.0.1 Reward vs Episode Comparison

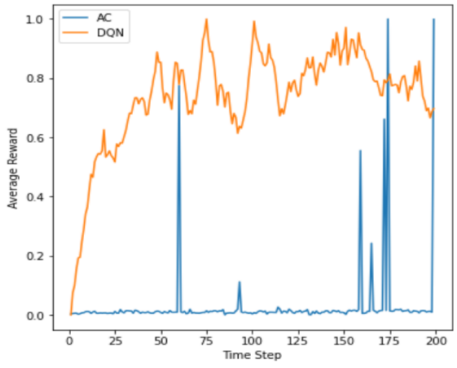

Figure 16: Sub-tasks Plot

The plot above (16) shows the normalised reward for the episode averaged over 3 runs vs the episode number for the best DQN agent and the best AC agent. The rewards have been normalised to values between 0 and 1 in order to better compare the agents since they use different reward functions. Although both

agents do learn, it can be seen that the DQN agent starts to reach it's maximum reward slightly faster than the AC agent.

4.0.2 Sub-task completion Comparison

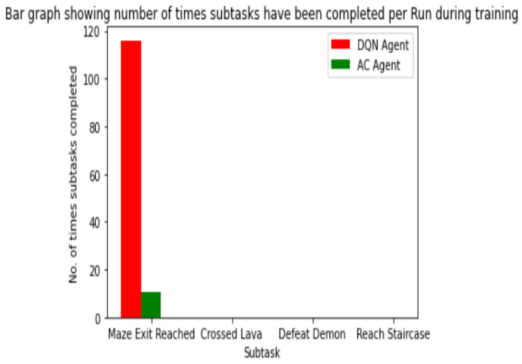

Figure 17: Sub-tasks Plot

The plot above (17) is a bar graph that shows the average number of times each sub-task is completed during training for the best DQN agent and the best AC agent. We can see that the DQN agent learns to perform better compared to the other agent on the maze, although none of them manage to perform well on the rest of the maze. This mainly due to the improvements made to the DQN agent that allow it to finish the maze more frequently.

5 Conclusion

In conclusion, we can see that reinforcement learning can be used to train agents to not only help healthcare and engineering professionals, but it can also be used to play games that humans struggle to play, trust me playing these minihack games manually was not a piece of cake.

The two algorithms mentioned above performed well and allowed our agent to learn to navigate the different configurations of the maze in the MiniHack-Quest-Hard environment. Although the agents don't learn to complete the dungeon, we can see that they are capable of learning and with more time and adaptions a complete solution to this dungeon could be found, perhaps this could be thought of as future work.

References

[1] Y Chris. *Understanding Actor Critic Methods and A2C*. 2019. URL: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (visited on 11/07/2021).

[2] B Craig and V.N Benjamin. *a2c*. https://github.com/raillab/a2c. 2018.

[3] Deeplizard. *Deep Q Learning)*. 2018. URL: https://www.youtube.com/watch?v=wrBUkpiRvCA&t=248s (visited on 10/23/2021).

[4] Heinrich Küttler et al. "The NetHack Learning Environment". In: *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*. 2020.

[5] Mikayel Samvelyan et al. "MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*. 2021. URL: https://openreview.net/forum?id=skFwlyefkWJ.

[6] W Sarah. *RL_NetHack_2020*. https://github.com/Sarah-wookey/RL_NetHack_2020. 2020.