# Reinforcement Learning Assignment Report - MiniHack the Planet

Hlagala Lerato - 2130564
*School of Computer Science and Applied Mathematics*
*University of the Witwatersrand*
2130564@students.wits.ac.za

***Abstract*—The purpose of this report is to evaluate and compare 2 deep reinforcement learning algorithms when it comes to solving the MiniHack environment. The two algorithms used are the Deep Q-Network and the Advantage Actor Critic. MiniHack [1] is a powerful sandbox framework for easily designing novel Reinforcement Learning environments.**

## I. Introduction

Reinforcement learning is a machine learning training strategy that rewards desired behaviors while penalizing undesirable ones. A reinforcement learning agent, in general, Reinforcement learning is a subset of machine learning in which an AI agent attempts to maximize its rewards in its environment. This machine learning training technique rewards positive behaviors while penalizing bad ones. In general, a reinforcement learning agent can observe and analyze its surroundings, as well as act and learn through trial and error. It generates an action policy that results in the optimal outcomes based on the rewards or penalties it receives from the environment.can detect and comprehend its surroundings, act, and learn through trial and error.

This assignement will be looking at the DQN and A2C models, which we are using to solve the MiniHack environment. Code for the models will be found in *Github*

## II. Environment

MiniHack is a NetHack-based environment creation framework and accompanying suite of tasks. MiniHack is an open source project on *Github*. In order to give environment designers a simple way to access the game's richness for challenging real-world tasks, MiniHack makes use of the NetHack Learning Environment (NLE). More than 450 items, including weapons, wands, tools, and spell books, as well as more than 500 monsters from the game are included in this new sandbox. Each of these items has its own special traits and affects the environment's dynamics in a complex way. This framework enables RL practitioners to tackle more challenging skill-acquisition and problem-solving tasks as opposed to simple grid-world navigation tasks with constrained action spaces. NLE is built on NetHack 3.6.6 and is intended to provide a standard RL interface to the game. It also includes tasks that serve as a first step in evaluating agents in this new environment.

## III. DQN

A DQN is a method for approximating a state-value function that utilises a neural network in conjunction with a Q-learning framework. Deep Q-Learning learns in small batches using Experience Replay to avoid skewing the data set distribution of different states, actions, rewards, and next states that the neural network will see. Q-learning is a temporal difference (TD) control algorithm that is used to calculate the learned action-value function in order to directly approximate the optimal action-value function regardless of the policy being used [2]. The DQN code is largely based on Rail Lab's code and can be found at: *Github* The Neural network architecture was changed to accommodate less input to obtain the best results.

The e-greedy policy we used in our implementation used a threshold that was determined by the eps-start and eps-end hyper-parameters. Our agent would select a random action if the random value was found to be below the threshold; otherwise, the policy would choose the action. The agents' experiences are stored in the replay buffer and pooled across the many episodes to create experience replay. The standard replay buffer we used in our implementation was adapted from *Github*. The learning phase of the algorithm uses random samples from the experience replay thanks to the replay buffer, which helps to improve the policy by teaching actions that are more similar to the ideal action.

Every iteration, the optimization step is carried out, after which the new policy is trained using a random batch sampled from the replay memory. An ADAM optimizer was used for the optimization steps because it was discovered to perform better in DQNs than an RMSprop optimizer. We discovered that improved performance would be preferred over stability. The RMSprop optimizers have been used extensively in the literature for reinforcement learning implementations [2], most likely because of their improved stability over the ADAM optimizer. Based on the optimization, the target network computes the expected Q values. The policy network weights periodically update the target network weights based on the number of steps to keep the target network relevant.

The PyTorch framework was used to create a double-

DQN with replay memory. The MiniHack environment includes a number of observation spaces that can be used as input by a model. The DQN models use the glyph observation space, which is a 21 x 79 matrix with each glyph representing an entity on the map. This matrix is normalized before being used as input. For both the policy and target networks, a convolutional neural network (CNN) was implemented. The CNN architecture is made up of two 2D convolutional layers that employ the ReLU activation function and max pooling layers. Following the convolutional layers, two fully connected layers output an array the size of the action space.

Table below shows the hyperparameters used to train the various DQNs in a variety of environments. After multiple training iterations, the best performing values were chosen. Because training a DQN can be resource-intensive, the number of episodes on which the model was trained varied depending on the environment.

| DQN Architecture | |
|---|---|
| conv_layer1 | Conv2d(in_channels=1, out_channels=20, kernel_size=(5,5)) |
| relu1 | ReLu() |
| maxpool1 | MaxPool2d(kernel_size=(2,2), stride=(2,2)) |
| conv_layer2 | Conv2d(in_channels=20, out_channels=50, kernel_size=(5,5)) |
| relu2 | ReLu() |
| maxpool2 | MaxPool2d(kernel_size=(2,2), stride=(2,2)) |
| fully_conn_layer1 | Linear(in_features=1600, out_features=500) |
| relu3 | ReLU() |
| fully_conn_layer2 | Linear(in_features=500, out_features=action_space.n) |

## A. *DQN Hyper-Parameters*

- **Replay-buffer-size:** 1e6: Size of the replay buffer
- **Discount-factor:** 0.9: The discount factor used to calculate an episode's discounted return.
- **The learning rate:** 0.01: is the rate at which the network is updated.
- **Target-update-freq:** the number of iterations that must occur between each target network update.
- **Batch size:** the number of transitions to optimize simultaneously.
- **Eps-fraction:** 0.4: the percentage of time spent annealing the Epsilon.
- **Eps-start:** 1.0: the e-greedy starting threshold
- **Eps-end:** 0.1: the e-greedy ending threshold

## IV. Advantage Actor Critic

The "Actor," which approximates the policy, and the "Critic," which determines the state value function, are the two components that make up the A2C implementation. Using a single neural network with shared bottom levels and independent output layers for the actor and critic, the Advantage Actor Critic is implemented in this assignment. Because both the policy and the value function estimations are parameterized using the same set of parameters (the network's weights), only one set of parameters needs to be modified during training.The advantage function, which is derived using the difference between the network's estimates and the actual reward obtained in the environment, is used to update the network parameters after this network is used to select actions in the environment (the "Actor") and to evaluate the state (the "Critic").

The A2C technique in this project was implemented using just one neural network. The model had to receive a state representation as input, produce a probability distribution across the set of actions (the policy), and estimate the value function of that state. These were the main criteria for constructing the model architecture. A number of observation spaces are supported by the MiniHack environment, including pixel, symbolic, and textual representations of the screen, the inventory, player statistics, and the onscreen messages. The original choice for the neural network's main input was the "glyphs" observation space, a 21x79 matrix encoding the map in which each glyph is represented by a different number between 0 and 5991. The matrix was reduced to a one-dimensional array and put into three hidden layers of a fully connected deep neural network (DNN), each containing 1024, 512, and 256 neurons. The network's last hidden layer provided inputs to two distinct fully connected layers, one of which had a single output neuron (for the value function estimate) and the other of which had many output neurons (one for each of the possible actions).Each of the hidden layers as well as the value function output layer were activated using the ReLU (as it is estimating a real-valued continuous variable). Given that it is a probability distribution over a discrete action space, the SoftMax activation function was chosen for the output layer of the policy estimates.

The activities during training were then sampled using this category distribution. While it was able to learn how to navigate in simple environment layouts, like in the environment Room-5x5, preliminary experiments using this model in the A2C algorithm on a number of MiniHack environments showed that it was unable to navigate through more complex environments, like Corridor-R2, or interact with objects as required by environments, like Eat. This was mostly caused by the loss of any spatial information that could have been obtained from the two-dimensional glyph representation when the state representation was flattened.

The onscreen messages were also incorporated as a distinct input to the model in order to provide more environmental feedback throughout the model training phase. A 256-dimensional vector containing the utf-8 encodings of the message prompts displayed on screen makes up the ”messages” observation representation from the MiniHack Environment. The output of this vector was concatenated with the CNN output before being supplied to the two distinct output layers. This vector was passed via a single, fully connected layer. To speed up model training, the glyphs and message inputs are both normalized before being sent to the network.

### A. A2C Hyper-parameters

- **Gamma:** 0.99: The discount rate used to determine the episode’s discounted returns.
- **max-episode-length:** 150 - 250: The A2C method stated above suggests that the process of choosing actions and utilizing the observed reward to adjust network parameters should continue until the environment achieves a terminal state. In reality, though, this is not always possible because an agent may travel millions of steps before ending. This parameter limits the number of steps an agent may take in an environment before the episode ends, preventing excessive training times.
- **The number of episodes:** 100 - 200: to be used for training.
- **Learning rate:** 0.02: that the ADAM optimizer uses while training a model.

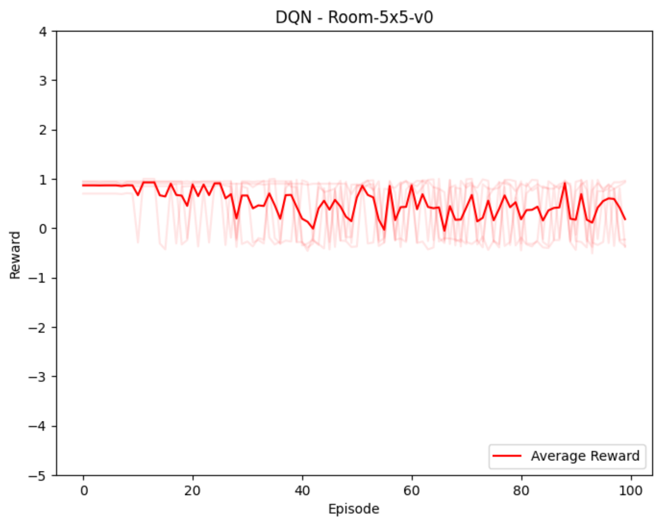
## V. Results

MiniHack offers a wide range of diverse environments for agent training and performance evaluation. Depending on the principal objective an agent must complete in each environment, multiple pre-built environments are included with the MiniHack package. In this assignment, the Room-5x5 environment functions as the initial learning environment and requires navigation from a defined start location to a fixed target position. In order to succeed in the second setting, the agent must acquire a certain skill. These skills require a series of steps to be taken, such as moving to a certain area, selecting how to interact with an object there, and then confirming your choice. The DQN model as well as the A2C model were both trained.

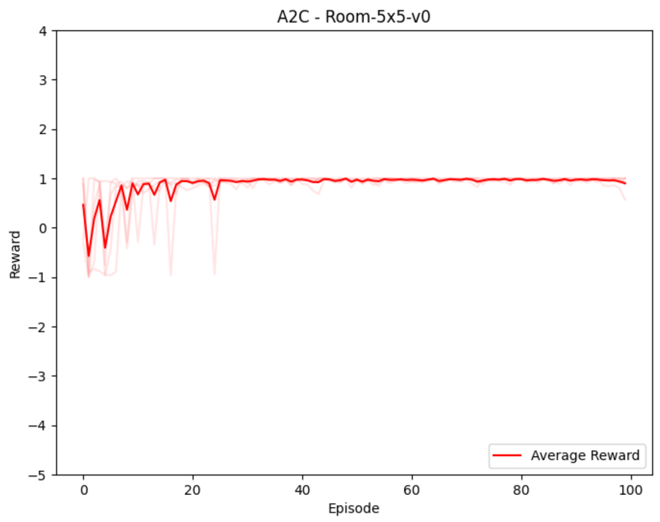
To make sure that the findings gained were not unique to the environment seed, five training iterations were done across various randomly seeded settings and summed. Additionally, this guarantees that the outcomes may be repeated. The five seeded runs are depicted in all of the graphs in this section with high opacity, and the averaged results are represented by a solid line. The Eat environment plots are presented in teal, the Quest hard plots are shown in orange, and the room 5x5 plots are shown in red.

The goal of the 5x5 room environment is to show how well the A2C agent and DQN can master the fundamentals of navigation. Only the cardinal directions North, South, East, and West, NorthWest, NorthEast, South West, and South East are present in this environment. The agent must locate the stairs down in the Room-5x5 environment. The agent is rewarded with 0.01 for touching the environment’s outer walls and +1 for making it to the staircase (the goal position). The Room 5x5 navigation challenge could be solved by both the DQN and A2C models.

The model below displays the average returns per episode over 100 episodes for the DQN. The DQN had a slow start. Nevertheless, the DQN continues to make errors, and the rewards still vary greatly.

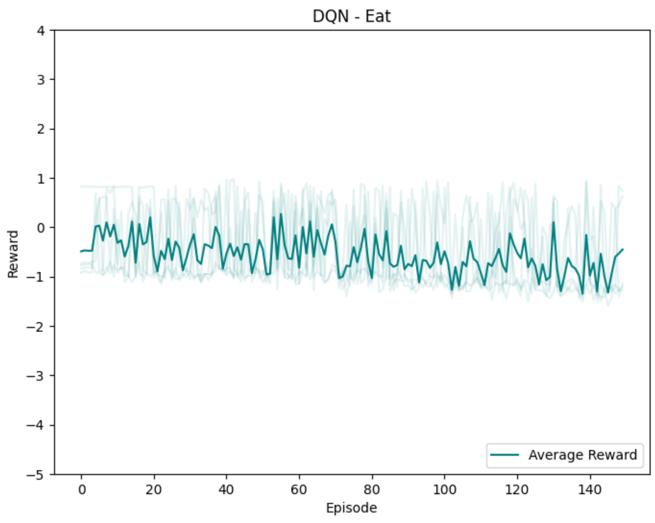


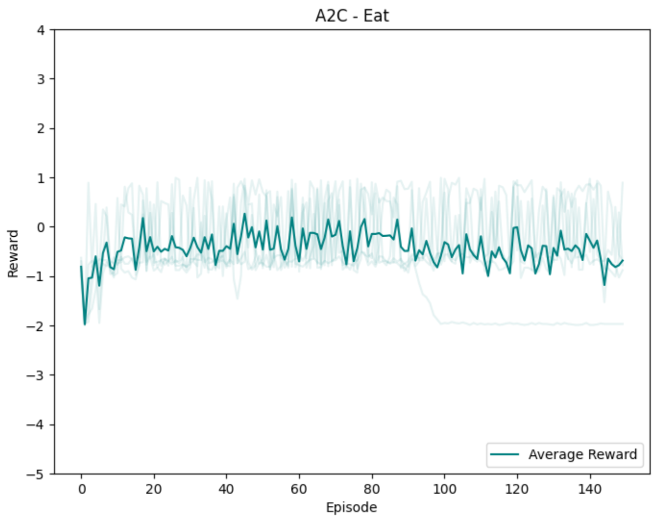
Because the A2C model could converge more quickly, it was only trained for 100 episodes. Figure below depicts the episode reward plot for the A2C model completing the Room 5x5 navigation challenge. The model finds it difficult to complete the job in the first 10 episodes, but after just 20 episodes, the average reward starts to converge to 1, the model is able to complete the task with ease.



An agent must figure out how an object interacts with its activities in the second environment, the Eat environment. The agent must find an apple, decide whether to eat it, and then confirm his decision. Every episode randomly selects the apple's location and the agent's beginning position. Again, the agent gets rewarded with 0.01 for striking the environment's boundary walls, but it only earns a +1 reward if it successfully completes the series of steps necessary to consume the apple. Only the EAT command and the four cardinal directions are allowed in the action space.

The Eat assignment proved impossible for the DQN model to complete. The plot for the episode-return is depicted in Figure below. The negative overall trend of the mean reward indicates that learning was not successful. The model performs better in the beginning since it explores more and initially chooses arbitrary actions.





The episode reward plot of the A2C model Above demonstrates how it failed to complete this assignment. Compared to the navigation task, there is greater variation in the rewards, indicating that this challenge was more challenging for the model to learn. And it does continue to struggle with solving the task.

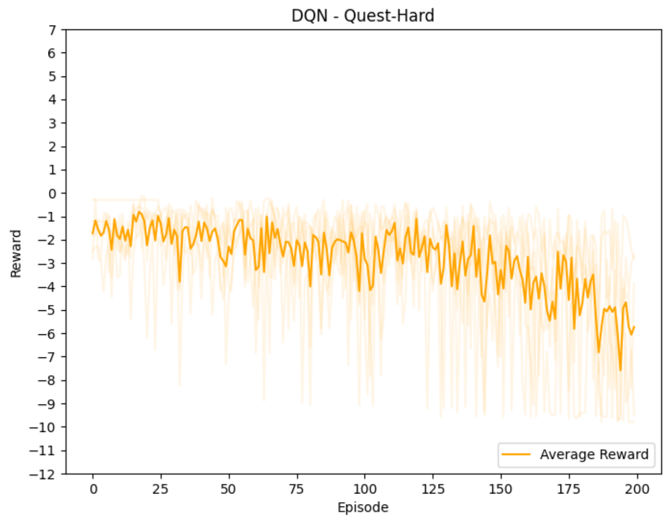
The third environment, Quest-Hard, demands the agent to carry out a significant number of consecutive tasks before returning a single, tiny reward for success at the episode's conclusion. This involves finding your way out of a randomly generated maze, picking up something, utilizing it to cross a lava river, killing a minotaur, and getting to the stairs below. Reward shaping and action space constraints were included in order to make learning such a complex job easier.

The agent in Quest-Hard spawns in a procedurally created maze and is only able to observe the cells in its immediate surroundings. More blocks are made visible when the agent navigates the maze's unknown locations. The number of disclosed cells in the current state was compared to the number of revealed cells in the previous state in order to develop a specific reward that would stimulate maze exploration. The agent is rewarded positively and is encouraged to expose as much of the map as he can if the current state has more disclosed cells than the previous one had.

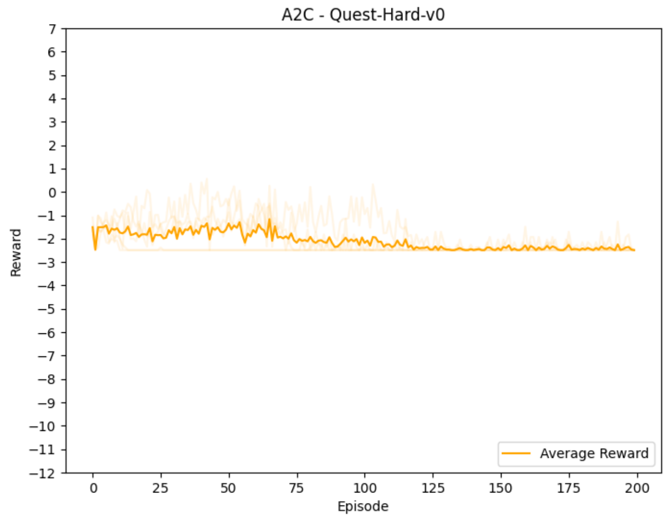
The coordinate reward, which involves receiving a reward for achieving a certain coordinate, is another helpful technique for creating rewards. As it does in each episode of Quest-Hard, a satisfying prize was placed at the conclusion of the maze. Another subtask that the agent must finish in order to achieve Quest-Hard also received a favorable reward: killing the minotaur. Only the following actions were permitted in the action space: North, South, East, West, North-West, Pickup, Apply, Fire, Rush, Zap, Puton, Read, Wear, Quaff, and Pray.

Due to insufficient computer hardware, the maximum number of steps per episode was limited to 250. and the model failed because it did not provide enough information for the agent to successfully navigate the maze.

The quest-hard environment was too challenging for the DQN to understand how to navigate. As the model trains become worse, the episode keeps on, as can be seen in the illustration below. This could be because the DQN investigates and selects arbitrary actions more frequently during the beginning of training. As fewer random actions are selected, the model's performance degrades and it is unable to learn.



Using the above-mentioned reward shaping, the A2C model was able to learn how to explore more of the procedurally created maze at the start of Quest-Hard, but was unable to exit the maze and hence could not accomplish the remaining tasks necessary to overcome the environment.



## VI. Conclusion

MiniHack is a great sandbox environment for RL-based activities, although it has several drawbacks. To begin, the MiniHack framework documentation is plagued with mistakes, provides inaccurate implementation details, and lacks thorough examples of how to utilize the various functionalities. This problem, together with the enormous amount of processing power and time necessary to train RL agents to finish the more difficult settings, may explain why the suggested models could not achieve the desired outcomes in the Quest Hard scenario.

We would propose researching prioritized experience replay to potentially improve the outcomes and minimize the calculation time necessary to train the DQN. In 41 of 49 Atari games, a prioritized experience replay buffer, as reported by Schaul et al. [3], outperformed a uniform replay. We anticipate that by adopting prioritized experience replay, the DQN agent will learn more effectively since it will be able to prioritize events based on their relevance in the past more often than with the uniform buffer utilized in our implementation.

This study describes the implementation and results gained by two separate RL agents in MiniHack scenarios. The eating environment, the policy-based technique, A2C, beat the value-function-based method, DQN in the 5x5 room setting. Some potential future proposals for enhancing the two agents have been explored. Unfortunately, even after implementing environment adjustments such as reward shaping, action space reduction, and raising the maximum number of training repetitions, both agents were unable to execute in the quest-hard environment.

## References

[1] M. Samvelyan et al., “Papers with Code - MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research,” MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research — Papers With Code, Sep. 2021. [Online]. Available: https://paperswithcode.com/paper/minihack-the-planet-a-sandbox-for-open-ended

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, pp. 529–33, 02 2015.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2016.

[4] B. Budler, T. Packirisamy, K. Nair, and N. Raal, “Brentonbudler/deep-RL-minihack-the-planet: Learning minihack environments using DQN and actor-critic architectures,” GitHub, 09-Nov-2021. [Online]. Available: https://github.com/BrentonBudler/deep-rl-minihack-the-planet.