Trying to Solve the NetHack Unicorn: A Series of Experiments in Reinforcement Learning

Nicolaas P. Cawood
2376182

Clarise Poobalan
383321

Shikash Algu
2373769

Byron Gomes
0709942R

I. Introduction

Reinforcement learning (RL) research has, up until this point, made progress using environments with deterministic and repetitive settings. The Arcade Learning Environment (ALE) [1] has exposed the fact that methods that perform well on most arcade games are unable to learn from environments with sparse rewards like *Montezuma's Revenge*.

The Nethack Learning Environment (NLE) [2], built around the popular terminal-based NetHack game, has recently been made available to drive future research towards developing methods for environments with sparse-reward systems and dynamic observations. The environment was developed to provide fast simulation, as state-of-the-art RL approaches make use of vast amount of samples to learn from.

A. Related work

Although the NLE environment has recently been published, it is similar to the *Rogueinabox* [3] environment based on the Rogue game. The work of Asperti and Cortesi [4], who experiment with the Rogueinabox environment using partitioned Asynchronous Actor-Ciritic-Agents (A3C), is closely related to the problem being solved in the NLE environment.

We also consider decisions and design choices made for the baseline model of the NLE environment, which implements two reinforcement learning models: Impala [5] and Random Network Distillation (RND) [6].

B. Problem Statement

Compared to the NLE environment, previously solved environments were deterministic and contained action spaces that are less complex. The NLE presents the following challenges:

1) Large observation space: The observation space includes many components that do not affect the agent's immediate setting. Therefore reducing the agents' view is a consideration, in order to have it learn from a more appropriate context.

2) Large action space: The action space contains a large set of actions, from which only a handful are appropriate depending on the agent's immediate setting and character statistics.

3) Procedurally generated: The environment is procedurally generated every time the game is reinitialised. This makes it difficult for agents to learn since the environment is dynamic.

4) Multi-goal domain: Although NetHack's ultimate goal is to retrieve the *Amulet of Yendor*, the game is regarded as complicated, especially for players who lack knowledge thereof. The gameplay involves multiple, stochastic sub-goals and it is therefore recommended for players to make themselves familiar with the Guidebook[1] before attempting play.

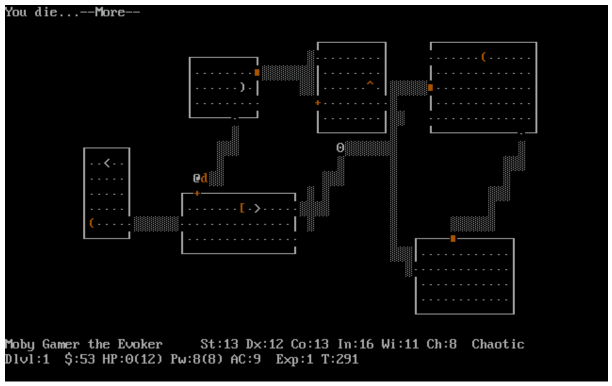
II. The Nethack Learning Environment

The NLE Environment is available as an OpenAI Gym environment, and although we do not consider the entire observation- and action spaces, we discuss features we believe are important for RL methods to consider.

A. Observation Space

The observation space includes numeric characters that represents different objects throughout the game, messages, and statistics indicating the player's progress. It was found that the most critical features are those of the *Glyphs* and *Statistics*, discussed briefly below.

1) Glyphs: The *Glyphs* are the classic symbolic representation of the gameplay in matrix form (21×79). The glyphs contain integer values ranging from 0 up to 5991 (representing monsters, items, dungeon features, etc.) Figure 1 depicts the glyphs of an explored dungeon level.



```
You die...--More--

Moby Gamer the Evoker     St:13 Dx:12 Co:13 In:16 Wi:11 Ch:8  Chaotic
Dlvl:1  $:53 HP:0(12) Pw:8(8) AC:9  Exp:1 T:291
```

Fig. 1. A visual representation of the *Glyphs* [2].

[1]https://www.nethack.org/v361/Guidebook.html

2) Statistics: Each state observation includes the character statistics that contains useful information regarding the characters' status. The statistics, found within the *blstats* dictionary of the observation space, are an array of 25 statistics in integer format. For convenience, the statistics are summarised in Table I below.

TABLE I
THE NLE STATISTICS OBSERVATION.

Index	Description	Index	Description
0	Position in longitude	13	Accumulated gold
1	Position in latitude	14	Energy level
2	Percentage of strength	15	Maximum energy
3	Maximum strength	16	Armor class
4	Level of dexterity	17	Monster level
5	Level of constitution	18	Experience level
6	Level of intelligence	19	Experience points
7	Level of wisdom	20	Time
8	Level of charisma	21	Hunger level
9	Accumulated score	22	Carrying capacity
10	Current health points	23	Other
11	Maximum health points	24	Other
12	Current dungeon depth		

3) Chars: An alternative to *Glyphs*, one might consider characters (*chars*). The chars, also in matrix form (21×79), provide a simpler representation of the environment in that, here only integers between 0-255 are used.

4) Message: Messages are revealed during gameplay that inform a player of impending obstacles, tasks etc in order to encourage the player to take appropriate actions. The message, contained within the *message* dictionary, is represented as an array of 256 integers, each element ranging between 0-255.

B. Action Space

The NLE environment has 93 available actions for a human player. This action space consists of 77 command actions and 16 movement actions. The movements might be made as a *single step* or until the agent runs into some entity by the *maximum step* actions. The command actions include eating, opening, kicking, reading, praying etc. A summary of the default NLE environment actions is provided in Table II.

TABLE II
THE DEFAULT NLE ACTION SPACE.

Action	Description	Action	Description
0	More	12	West max
1	North 1 step	13	North-East max
2	East 1 step	14	South-East max
3	South 1 step	15	South-West max
4	West 1 step	16	North-West max
5	North-East 1 step	17	Go up staircase
6	South-East 1 step	18	Go down staircase
7	South-West 1 step	19	Wait
8	North-West 1 step	20	Kick
9	North max	21	Eat
10	East max	22	Search
11	South max		

III. METHODS INVESTIGATED

A variety of methods were investigated in an attempt to find a model that solves the NLE environment resulting in the highest positive reward. The approach included investigating value-based, policy-based and combination methods, examples of which are shown below in Figure 2.

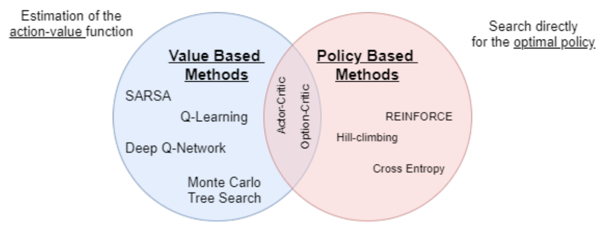


Fig. 2. Examples of RL methods classed as Value-based or Policy-based methods.

Value-based methods learn a value function in order to determine the optimal policy, whilst policy-based methods as the name suggests attempt to learn the optimal policy directly.

A. Deep Q-Network

Q-Learning is an off-policy TD method which approximates the Q-values from discrete action-value pairs. The interaction with the environment is carried by an ϵ-greedy policy on Q and the update rule evaluates a greedy policy which classifies the algorithm as an off-policy one [4]. Mnih *et al.* developed a Q-learning algorithm that that learns the Q-values using nonlinear function approximation using a deep neural network.

The Deep Q-Network (DQN) has achieved expert level scores to play Atari video games; however, the authors found that it still fails to learn from environments with larger temporally extended planning strategies like that of the NLE.

1) Experiment 1: We implemented a Double Deep Q-Network as proposed by Hasselt *et al.* [7]. We followed the deep convolution architecture and reduced observation representation as proposed by Asperti *et al.* [4] to learn from the Rogueinabox environment.

The Policy- and Target Network architectures are found using the following sequence of layers, each separated by Rectifier Linear Units (ReLU) :

- The first layer: a convolution layer, with 16 3×3 filters and a stride of 1.
- The first hidden layer : a convolution layer, with 32 3x3 filters, and a stride of 1.
- The second hidden layer : a fully connected linear layer, with an output length of 512.
- The output layer: a fully-connected linear layer, with a single output for each action.

We reduce the sparisty in the observation space by only using the *Glyphs* observations, and cropping them to a fixed box around the agent's position at each timestep. The cropped observations are pre-processed to only contain the following values:

0.5	Walls and doors
1.0	Corridors and gold
0.0	Everything else

The model is trained using an environment that rewards the agent at at each timestep according to the amount of new *Glyphs* explored added to three times the amount of new gold. We also reduce the action space so that the agent only has the option to move in the four cardinal directions.

The hyper parameters for this experiment are set out in Table III.

TABLE III
HYPERPARAMETERS FOR THE DQN MODEL.

Hyperparameter	**Value**	**Description**
Gamma	0.99	The discount factor for rewards.
Learning rate	0.02	The learning rate to update the model parameters.
Adam clip norm	10	The maximum norm of gradients.
Crop dimensions	15	The height and width of the cropped agent view.
Total steps	10×10^6	The number of steps used to train the model.
Batch size	32	The number of transitions per update step.
target network update	1,000	The number of iterations before each target network update.
initial exploration	1.0	Initial value for epsilon.
final exploration	0.1	The final value for epsilon.

B. REINFORCE

REINFORCE, a policy gradient method (a subclass of policy-based methods), estimates the optimal policy, parameterised by weights (θ), through gradient ascent [8]. The feed-forward neural network (NN), employed in the REINFORCE method, takes in as an input, an observation from the environment and returns a probability distribution action vector. During training of the NN, the reward received is used to update the policy (NN) weights as per Equation 1 below.

$$\theta \leftarrow \theta + \alpha R_t \bigtriangledown_\theta \, log\pi_\theta(a_t \mid s_t) \tag{1}$$

A series of experiments, described below, attempt to find the most suitable observation space to use in the REINFORCE method.

1) Experiment 1: The first experiment conducted for the REINFORCE method investigated the use of the glyphs observation space. A 14×14 cropped section of glyphs, around the agent, was used to reduce the observation representation presented to the NN, as proposed by [3]. Furthermore the cropped section of glyphs fed to the NN was scaled using the maximum glyph value available in the game. Likewise, the action space was also reduced to the 4 cardinal directions. The resulting NN architecture is as follows:

- Input Later: 196 input neurons.
- First Hidden Layer: fully connected layer, 256 neurons.
- Second Hidden Layer: fully connected layer, 128 neurons.
- Third Hidden Layer: fully connected layer, 64 neurons.

TABLE IV
HYPER-PARAMETERS USED FOR THE REINFORCE EXPERIMENTS.

Hyper-Parameter	**Value**	**Description**
Gamma	0.99	The discount factor for rewards.
Alpha	0.1	The learning rate to update \theta.
Max Steps per Episode	2000	Maximum number of steps allowed. per episode.
Number of Episodes	300	Number of episodes method was. trained for.
Crop Dimensions	14×14	Height and Width of the cropped view.

- Output Layer: 4 output neurons corresponding to the reduced action space.

The activation function used between each layer detailed above was the ReLu activation function. The output layer is activated using a Softmax activation function. Given the sparsity in the glyphs observation space, to prevent over-fitting a dropout regularization technique, with p = 0.5, was introduced between the input layer and first hidden layer.

2) Experiment 2: Similarly, another method investigated for the REINFORCE method observed the use of the chars observation space. Again, a 14×14 cropped section of chars, around the agent, was used and the action space was also reduced to the 4 cardinal directions. The resulting NN architecture is therefore identical to that detailed above (inclusive of the ReLu and Softmax activation functions and the use of the dropout regularization technique).

3) Experiment 3: Lastly the use of the message dictionary from the original observation space was considered. Seeing the length of the message vector was 256 elements, the message dictionary observation space was not reduced. Neither was the action space reduced. The resulting architecture is as follows:

- Input Later: 256 input neurons.
- First Hidden Layer: fully connected layer, 512 neurons.
- Second Hidden Layer: fully connected layer, 128 neurons.
- Third Hidden Layer: fully connected layer, 64 neurons.
- Output Layer: 23 output neurons corresponding to the default action space.

Here only Relu activation functions were used between layers and the output layer made use of a Softmax activation. The dropout regularization technique was not employed as the message features, after scaling were not as sparse.

The hyper-parameters used for all 3 REINFORCE experiments are summarised below in Table **??**.

C. Actor-Critic

The Actor-Critic method combines ideas from policy gradient and value function methods. In this methodology we approximate both the policy and the value function, and use the latter to help us improve the former.

The Actor approximates the policy $\pi_\theta(s, a)$, which is used at each time step to provide a random action to be taken. The new state and reward are then used to update the Critic which approximates the value function $V_\omega(s)$, as well as the Actor, using the temporal difference (TD) error to do so. This process is shown diagrammatically in Fig. 3.

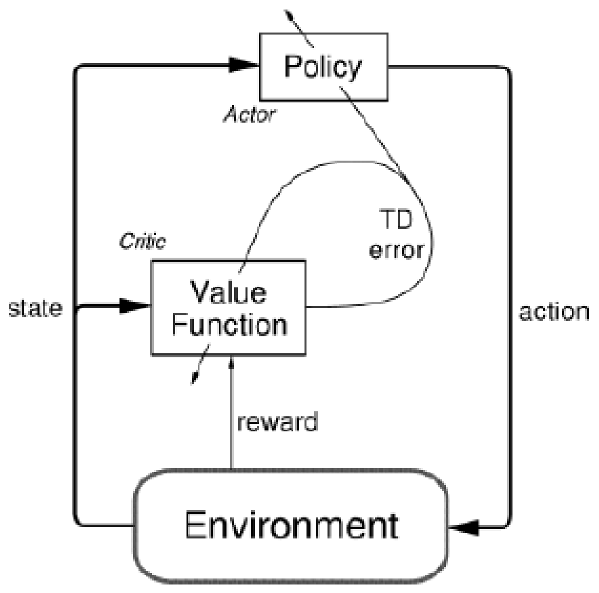


Fig. 3. Diagram of the actor-critic algorithm.

1) Experiment 1: For the first experiment an Actor-Critic method is implemented in PyTorch as two neural networks with a shared body which is similar to that used in the Policy Gradients lab and described in [9] with some modifications.

The body of the network has an input layer which is comprised of a 15×15 grid of *Glyphs* around the agent in the game, plus the x and y co-ordinates, hunger, and health stats from the *Statistics* provided by the environment. Then the rest of the body had four hidden layers reducing in size in equal amounts. The first layer has 203 nodes, the second has 152, the third has, 100 and the fourth has 49. Each layer is separated by ReLU functions.

From here the network splits into two, one for the Actor, and one for the Critic. The Actor network takes the last hidden layer of the body with 49 nodes as an input layer, then has one hidden layer of the same size, and an output layer with a node for each of the 23 actions in the NLE environment. The Critic network also takes the last hidden layer of the body, has its own hidden layer with 49 nodes, and then has one output node representing the value of that state $V_\omega(s)$. Again each layer is separated by a ReLU function while the final output layer makes use of the Softmax function.

Training this model was done in several stages, starting with limiting the agent to a maximum number of steps for a number of episodes. The reasoning behind this was to encourage the network to find rewards as soon as possible before increasing the number of steps it is allowed to play. At the end of the training the number of steps is so large as to not matter as the agent usually dies before reaching the maximum. A summary of the maximum steps and number of episodes trained with that limit is provided in Table V.

TABLE V
TRAINING REGIME FOR THE ACTOR-CRITIC MODEL.

Training Session	Maximum Steps	Number of Episodes
1	100	100
2	200	100
3	500	200
4	750	200
5	1000	200
6	2000	200
7	5000	500
8	10000	500

The hyper parameters for the Actor-Critic model are set out in Table VI.

TABLE VI
HYPERPARAMETERS FOR THE ACTOR-CRITIC MODEL.

Hyperparameter	Value	Description
Gamma	1	The discount factor for rewards.
Beta	0.001	The learning rate for the Actor.
Zeta	0.001	The learning rate for the Critic.
Batch size	10	The number of transitions per update step.

2) Experiment 2: For the second experiment we followed the deep convolution architecture and reduced observation representation as proposed by Asperti *et al.* [4] to learn from the Rogueinabox environment. The network extends the design of the Asyncrhonous Advantage Actor Critic (A3C) and includes a Long-short-term memory (LSTM) layer to process temporal dependencies.

We used this same network architecture in an Advantage Actor Critic (A2C) method, and the network is constructed of the following sequence of layers in the shared body, each separated by ReLU activation:

- The first layer: a convolution layer, with 16 3×3 filters and a stride of 1.
- The first hidden layer : a convolution layer, with 32 3x3 filters, and a stride of 1.
- The second hidden layer : a fully connected linear layer, with an output length of 256.
- The third hidden layer : an LSTM layer, with an output length of 256.
- The output layer: a fully-connected linear layer, with a single output for each action.

We reduce the sparsity in the observation space by only using the *Glyphs* observations, and cropping them to a fixed box around the agent's position at each timestep. The cropped observations are pre-processed to only contain the following values:

0.5	Walls and doors
1.0	Corridors and gold
0.0	Everything else

The model is trained using an environment that rewards the agent at at each timestep according to the amount of new *Glyphs* explored added to three times the amount of new gold. We also reduce the action space so that the agent only has the option to move in the four cardinal directions.

The hyper parameters for this experiment are set out in Table VII.

TABLE VII
HYPERPARAMETERS FOR THE A2C WITH LSTM MODEL.

Hyperparameter	Value	Description
Gamma	0.95	The discount factor for rewards.
Alpha	0.99	The alpha term for the RMSprop omptimiser.
Epsilon	0.1	The epsilon term for the RMSprop omptimiser.
Learning rate	7×10^{-4}	The learning rate to update the model parameters.
RMS clip norm	40	The maximum norm of gradients.
Crop dimensions	10	The height and width of the cropped agent view.
Steps per episode	500	The maximum number of training steps per episode.
Batch size	1	The number of transitions per update step.

D. Model Based

Given the sparse environment and large state space of the Nethack environment, a Model Based RL method was attempted. Model based RL is a combination of planning and learning [10]. Model based RL methods interact with the environment to learn a model [11], [12]. Model based RL learns the optimal behavior by sampling the environment. A sampling based approach was used to search through the action space. The environment was sampled to learn four random policies, i.e. ”Learn message”, ”Can move”, ”Cant move” and ”Gives reward”.

1) Experiment 1: To get an understanding of the environment, four policies were constructed. The first policy deals with messages and how to best act when a message is encountered. If a message is encountered, the best action is drawn from a ”tree”. The root of the tree is the action that was taken to generate the message and the leaves are the actions to react to the message. The action linked to the highest reward is used as a selection criteria with an ϵ-greedy approach to take the agent ”out of the message”.

The second and third policies deal with Movable and Not Movable areas respectively. Based on a random agent and the observation space, if the agent takes an action and moves from the current location to the next location, the ”Glyph” value of the next location is saved as movable. Similarly, if the agent takes an action to move and does not move due to a restriction, the ”Glyph” value of the location attempted to move to is saved as not movable.

The fourth policy deals with rewards. If an agent encounters a ”Glyph” and gets a positive reward, the value of the ”Glyph” gets stored as a reward.

If a saved state appears in more than one policy, the policies are updated accordingly. This event happens due to rewards moving around the room.

Fig. 4. shows how these policies were joined together to navigate the dungeon. If there is a reward in the room, the first prize is to collect the reward and the agent will navigate to this reward following an appropriate path . If a message is encountered, the appropriate action is taken to resolve this message. When navigating a room, the agent tries to take a path that has not been explored before. If all immediate neighbours have been explored, the agent will randomly explore until a new path is found. Navigation is based on where the agent can and cant move according to the policies.

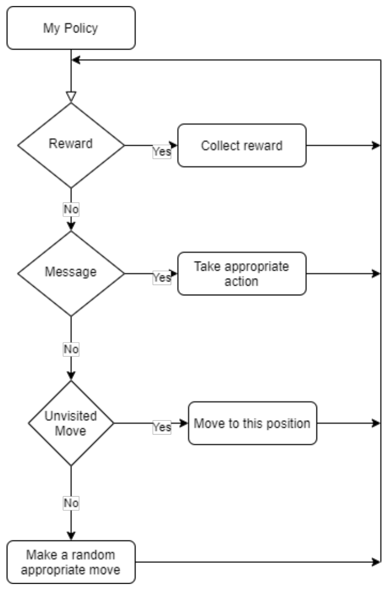


Fig. 4. Policy flow of agent.

IV. TRAINING RESULTS

In this section, we compare the results for all of the proposed methods. We implemented all algorithms in Python and we evaluate our trained models using seeds 1 up to 5 for a total of 10 runs. The source code for all implemented methods is available on GitHub [2].

A. DQN Results

Our DQN results confirm the findings of [3], [13] that faced problems with agents being stuck at a wall once they have reached them. We note that during the training of a million episodes, the losses oscillate for each update and the model never finds an approximation for the value function.

B. REINFORCE Results

To determine which observation space would be best to use for the REINFORCE method, the 3 experiments detailed in Section III.B were conducted. The result of each experiment conducted is presented in Table VIII.

As can be seen, the REINFORCE method trained on the message observations produced the best result for this method. Therefore for the remainder of this study, the REINFORCE method presented refers to the method trained on the message observation space.

[2]https://github.com/Pieter-Cawood/Reinforcement-Learning

TABLE VIII
PRELIMINARY RESULTS OBTAINED FOR REINFORCE METHOD

Observation Used	Average Reward
Glyphs	-11.00
Chars	21.02
Message	30.76

Figure 5 below show the rewards returned after training the the REINFORCE method for 300 episodes capped at 2000 steps per episode. Figure 5 reveals the method learns as much as it can within the first 50 episodes, thereafter the learning curve plateaus indicating the REINFORCE algorithm is unable to adapt to the continuously changing environment.

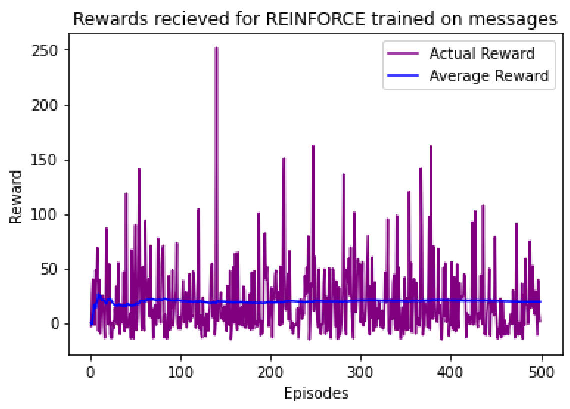


Fig. 5. The learning curve for the REINFORCE method.

C. Actor-Critic Results

For experiment 1, the learning curve for the last step of training the Actor-Critic network from training session 3 (see Table V) is presented below in Fig. 6.

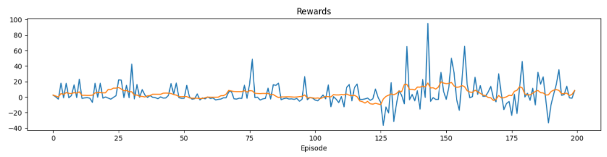


Fig. 6. The learning curve for the Actor-Critic method.

This training session showed the most significant increase in the rewards. The learning curves from previous and subsequent training sessions remained noisy and mostly flat. The model performs moderately well in the environment, but the amount of training required to achieve these results appears to be quite expensive compared to the other models we implemented.

For our second experiment, we trained the model using $1,000$ seeds, similar to the training process used for the baseline model of the NLE. The training was done for a total of $10,000$ episodes and we note that more samples are required to analyse the model's true capability.

Of the two experiments performed for Actor-Critic methods, the first experiment performs better on an average of 100 episodes. The worst, average, and best scores for each model based on these 100 episodes are presented in Table IX below, and the best results are marked in bold.

TABLE IX
ACTOR-CRITIC MODEL COMPARISON

	Experiment 1	Experiment 2
Worst score	**-18.23**	-17.06
Average score	**30.65**	20.80
Best Score	**384.15**	171.48

D. Model Based Results

"Can move", "Can't move", "Gives reward" and "Learn message" polices were trained on a random agent. For the "Can move", "Can't move" and "Gives reward" policies, the agent ran for 200 episodes. The Message policy was trained on a random agent for 100 episodes. Each episode is based on a random seed.

The model based method was run over 10 random seeds. Each run consisted of 10 episodes and the Agent was left to explore until the game ended (done=True). The action space was limited to the 4 cardinal directions, 2 inter-cardinal directions, Up, Down and Kick. The results for this experiment can be seen in Table. X.

Compared to a random agent, the random agent with message policy results in an improvement of 59.88%. When considering rewards, the model improves by 125.15% compared to the original random agent. By considering messages, rewards and navigation, the model improved by 195.8% over the initial random agent. By adding various policies, an improvement in the model can be seen.

TABLE X
AVERAGE REWARD FOR POLICIES

	Average Reward	Improvement
Random Agent	16.7	-
Random Agent with Message	26.7	59.88%
Random Agent with Reward	37.6	125.15%
My Policy	49.4	195.80%

Adding search to the model allowed the agent to explore new areas, unlocking new observations. Fig. 7. Shows the area explored for a random seed. The agent was successful in finding new rooms and tunnels.

V. EVALUATION RESULTS

In Table XI. we compare the performance of the best models from each of the previous sections. We exclude the DQN model as it did not perform well in training. Each of the models were run for 20 episodes generated using 5 seeds (0, 3, 22, 42, and 88) for the NLE environment, giving a total of 100 episodes per model. We recorded the worst, average, and best scores over these episodes.

The results from the model comparison indicate that the Model Based RL model had the least variance in rewards and the highest average score. This makes it the preferred model

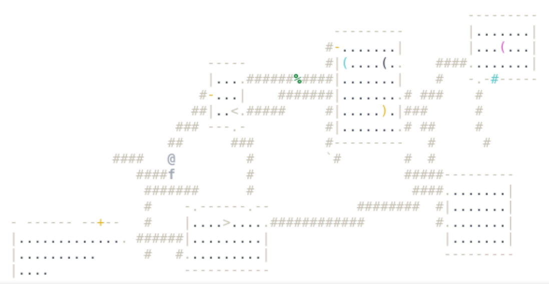


Fig. 7. Area Explored from Model Based approach

TABLE XI
MODEL PERFORMANCE RESULTS

	REINFORCE	**Actor-Critic**	**Model Based RL**
Worst score	-36.99	-18.23	**-10.16**
Average score	22.46	30.65	**35.00**
Best Score	752.26	384.15	**155.10**

for this environment as it was able to more consistently deliver higher scores than the other models.

The REINFORCE model had the highest variation in rewards and the lowest average score, making it the least preferred model for NLE. The Actor-Critic model performed somewhere between REINFORCE and Model Based RL.

VI. CONCLUSION

In this project several RL methods were explored to train an agent to learn to interact and navigate the NLE environment. Some methods proved to be less than useful, while others showed some ability to learn some aspects of the game. A limitation in training the models was the computational power required. Training the models for a longer period of time may improve the results.

Furthermore, throughout the process of this project, it became more apparent that hierarchical reinforcement learning (HRL) methods are more suitable to the NLE environment as they allow for the decomposition of sparse problems into a hierarchy of sub-tasks. As future work, exploring a hierarchical approach is suggested.

REFERENCES

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[2] H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, "The nethack learning environment," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[3] A. Asperti, C. De Pieri, and G. Pedrini, "Rogueinabox: an environment for roguelike learning," *International Journal of Computers*, vol. 2, 2017.

[4] A. ASPERTI and D. CORTESI, "Reinforcement learning in rogue."

[5] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[7] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[8] R. J. Williams, "Simple statistical gradient following algorithms for connectionist reinforcement learning." *Machine Learning*, vol. 8, pp. 229–256, 1992.

[9] DataHubbs, "Two-headed a2c network in pytorch," 2017, https://www.datahubbs.com/two-headed-a2c-network-in-pytorch/, (visited 2020-11-30).

[10] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," 2020.

[11] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," 2019.

[12] S. Ray and P. Tadepalli, *Model-Based Reinforcement Learning*. Boston, MA: Springer US, 2014, pp. 1–6.

[13] A. Asperti, C. Pieri, M. Maldini, G. Pedrini, and F. Sovrano, "A modular deep-learning environment for rogue," *WSEAS Transactions on Systems and Control*, vol. 12, 2017.