Inventory Management with Attention-Based Meta Actions

Keisuke Izumiya* and Edgar Simo-Serra*
*Department of Computer Science and Communications Engineering
Waseda University, Tokyo, Japan
Email: k-izumiya@ruri.waseda.jp, ess@waseda.jp

***Abstract*—Roguelike games are a challenging environment for Reinforcement Learning (RL) algorithms due to having to restart the game from the beginning when losing, randomized procedural generation, and proper use of in-game items being essential to success. While recent research has proposed roguelike environments for RL algorithms and proposed models to handle this challenging task, to the best of our knowledge, none have dealt with the elephant in the room, *i.e.*, handling of items. Items play a fundamental role in roguelikes and are acquired during gameplay. However, being an unordered set with a non-fixed amount of elements which form part of the action space, it is not straightforward to incorporate them into an RL framework. In this work, we tackle the issue of having unordered sets be part of the action space and propose an attention-based mechanism that can select and deal with item-based actions. We also propose a model that can handle complex actions and items through a meta action framework and evaluate them on the challenging game of NetHack. Experimental results show that our approach is able to significantly outperform existing approaches.**

***Index Terms*—reinforcement learning, attention, NetHack**

I. Introduction

Designing autonomous agents to play video games can play an important role in game balancing, testing, and design. Furthermore, video games play an important role in developing robust Reinforcement Learning (RL) algorithms that can then be applied to other real-world situations. In this work, we tackle the challenging game of NetHack, a roguelike game based on procedurally generated content, and develop a general attention-based approach to handle unordered sets of actions, such as inventory management, with autonomous agents.

RL, a branch of machine learning, is a method that has the great advantage of being able to learn without directly using the label data. In particular, Deep Reinforcement Learning (DRL), which incorporates deep learning techniques, has been actively researched in recent years and is often applied to games such as Atari 2600 [1], Go [2], and StarCraft II [3]. Among games, roguelikes are suitable as a challenging target problem for DRL because they contain elements that current DRL methods have not entirely overcome, such as a huge state space, sparseness and delay of rewards, and the need for strategies. According to [4], roguelikes are turn-based games, and their grid-based environments are randomly procedurally generated each time the game begins. The variety of enemies and items makes the game complex enough that there are multiple ways to complete the game. The required tasks include resource management, combat with large numbers of enemies, and exploration.

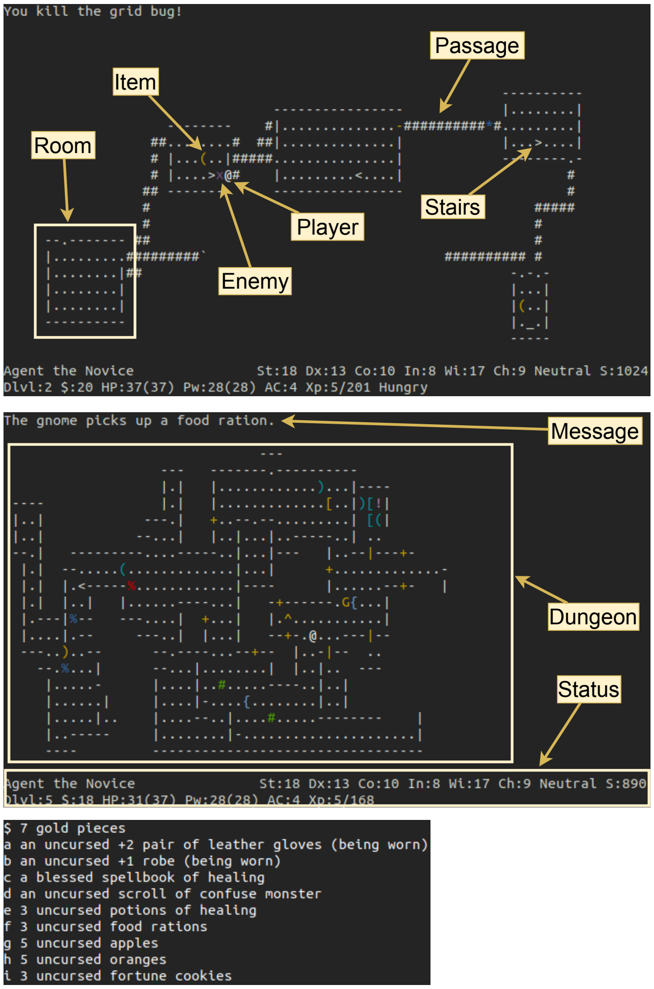
NetHack, which is the target problem of this research, is one of the most popular open-source roguelikes, and it is still being updated even though it is one of the earliest roguelikes. The game's objective is to search through over 50 levels of procedurally generated dungeons using various items to find the *Amulet of Yendor* and bring it back. It is incredibly challenging even for humans because there is a wide variety of enemies, items, and actions the player can take. Furthermore, the dungeons are not straight paths but rather branching paths that must be traversed back and forth. Examples of NetHack screens are shown in Figure 1. The player can use four main types of information: messages, dungeon, status, and inventory. Messages show events and confirmation messages. The dungeon shows the floor where the player @ is. Most floors consist of square rooms and passages # (top figure), but some floors do not (middle figure). An empty area indicates that nothing exists or the area has not been explored. The status indicates the player's attributes such as strength, experience, intelligence, and hunger. The inventory shows the items the player possesses (bottom figure).

As described in Section II, there are several studies on using DRL for roguelikes. However, most of these studies focus on constructing the environment itself or on learning in non-general situations. It is important that, to the best of our knowledge, there is no study that deals with items in the general situation that exist in most role-playing games, including roguelikes, and whose use is essential. Therefore, we propose a mechanism that handles items appropriately and show its effectiveness in experimental results[1].

Our main contributions are as follows:

- We propose an attention-based mechanism for handling an unordered set of items in a deep reinforcement learning framework.
- We develop a new agent model that significantly outperforms existing approaches in terms of in-game score on the challenging game of NetHack.
- We perform an in-depth evaluation of the proposed approach and compare with existing approaches.

[1] The source code is available at https://github.com/izumiya-keisuke/inventory-management



```
You kill the grid bug!
```

```
Agent the Novice              St:18 Dx:13 Co:10 In:8 Wi:17 Ch:9 Neutral S:1024
Dlvl:2 $:20 HP:37(37) Pw:28(28) AC:4 Xp:5/201 Hungry
```

```
The gnome picks up a food ration.
```

```
Agent the Novice              St:18 Dx:13 Co:10 In:8 Wi:17 Ch:9 Neutral S:890
Dlvl:5 $:18 HP:31(37) Pw:28(28) AC:4 Xp:5/168
```

```
$ 7 gold pieces
a an uncursed +2 pair of leather gloves (being worn)
b an uncursed +1 robe (being worn)
c a blessed spellbook of healing
d an uncursed scroll of confuse monster
e 3 uncursed potions of healing
f 3 uncursed food rations
g 5 uncursed apples
h 5 uncursed oranges
i 3 uncursed fortune cookies
```

Fig. 1. **Examples of NetHack gameplay screens.** The top two images show examples of the main in-game screen, while the bottom images shows an example of the inventory screen. Different parts of the screens are annotated for ease of understanding.

II. Related Work

A. Reinforcement Learning Algorithms

Mnih et al. [5] proposed a method that does not use experience replay but instead uses data collected by running multiple agents in parallel, represented by asynchronous advantage actor-critic (A3C). In addition to A3C, there are several other methods belonging to the actor-critic method, such as Actor-Critic with Experience Replay (ACER) [6], UNsupervised REinforcement and Auxiliary Learning (UNREAL) [7], which introduces auxiliary tasks in addition to experience replay, and importance weighted actor-learner architecture (IMPALA) [8], which can be extended to thousands of machines by elaborating on the data communication method and off-policy correction. Our approach is based on IMPALA with modifications te be able to do inventory management. The details of IMPALA are shown in Section III-A.

B. Reinforcement Learning for Games

Most of the modern RL algorithms are evaluated on Atari 2600 games provided by ALE [9] or the environments provided by OpenAI Gym [10] as a benchmark. For more popular games, some studies tackled perfect information games such as Go and Chess [2], [11]. There have also been studies on multiplayer imperfect information games, such as StarCraft II [3] and Dota 2 [12]. These methods combine the DRL methods with MCTS [13]. There is also the research [14] that used an environment based on Minecraft [15], [16], a game that uses items similar to NetHack. In this research we focus on roguelike game of Nethack.

C. Reinforcement Learning for Roguelikes

Some research has focused on tackling roguelikes with RL. To facilitate RL's application, highly customizable environments based on Rogue, the game that originated the roguelike genre and a simpler game than NetHack, have been created [17], [18]. With these environments, DRL techniques aimed at the exploration of the dungeon [17]–[21] (the exploration of the current floor and descending to the deeper floor) have been proposed. For NetHack, the subject of this study, some recent research has created environments based on NetHack and tackled various tasks. Campbell and Verbrugge [22] aimed at learning enemy combat with an abstracted state and action space. They also focused on finding hidden doors and passages, which is essential for dungeon exploration, and proposed a method using occupancy maps to achieve the goal efficiently. Küttler et al. [23] tackled several tasks such as scoring a game and collecting gold in a general situation by using random network distillation, which is one of the exploration facilitation techniques. Most of these methods do not take into account items, which are almost essential for completing the game, and this is one of the issues to be solved. In this research, we propose a technique which is able perform inventory management based on attention allowing for significant performance improvements over existing approaches.

D. Attention

Attention is a mechanism widely used in deep learning where the model can learn which parts of the data and features to pay attention to has been actively studied in natural language processing. In the past, attention had often used in combination with an RNN, as in [24], but many methods using attention alone have been proposed and showed better performance than existing methods since the appearance of Transformer models [25], which used self-attention without RNN. Now models that use self-attention are commonly employed in the field of image recognition, where Convolution Neural Network (CNN) have been commonly used [26]. In the field of RL, Berner et al. [12] used an attention mechanism to choose the target unit, and Zambaldi et al. [27] used self-attention to describe the relationship between entities. We focus on the position-independent property of attention and propose a mechanism to handle inventory using this property.

III. Method

A. Reinforcement Learning Background

RL is often modeled by Markov Decision Processes (MDP). The state space and the action space are denoted as $\mathcal{S}$ and $\mathcal{A}$, respectively. The state at the discrete time $t = 0, 1, 2, \dots$ is denoted as $S_t \in \mathcal{S}$. The action $A_t \in \mathcal{A}$ the agent takes at time t brings a reward R_t and a new state S_{t+1}. The goal of RL is to learn the policy $\pi(a \mid s) = \Pr\{A_t = a \mid S_t = s\}$ that maximizes the expected return $\mathbb{E}_\pi\bigl[\sum_t \gamma^t R_t\bigr]$, where γ is a discount factor.

We base our work on IMPALA [8], which is an asynchronous off-policy actor-critic method. As in IMPALA, we prepare multiple independent agents, actors, and environments and update the parameters of the learning agent, learner, using the data collected by these agents. We use an off-policy correction method called V-trace because each actor has its own local policy μ, which lags behind the learner's policy π. At the state s, the agent outputs the state value $V(s)$, which estimates the return from the state s, and a policy $\pi(a \mid s)$. The gradient of the loss function for the policy parameter is

$$\underbrace{\rho_t \nabla \log \pi(A_t \mid S_t)\bigl\{R_t + \gamma v_{t+1} - V(S_t)\bigr\}}_{\text{policy gradient term}} \underbrace{+ \beta \nabla \sum_{a \in \mathcal{A}} -\pi(a \mid S_t) \log \pi(a \mid S_t)}_{\text{entropy term}}, \tag{1}$$

where

$$v_t = V(S_t) + \sum_{s=t}^{t+n-1} \gamma^{s-t} \left(\prod_{i=t}^{s-1} c_i \right) \delta_s, \tag{2}$$

$$\delta_t = \rho_t \bigl\{R_t + \gamma V(S_{t+1}) - V(S_t)\bigr\}, \tag{3}$$

$$\rho_t = \min\left\{\overline{\rho}, \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)}\right\}, \tag{4}$$

$$c_t = \min\left\{\overline{c}, \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)}\right\}. \tag{5}$$

$\beta, n, \overline{\rho},$ and $\overline{c}$ are hyperparameters. The first term in (1) is a policy gradient, and the second one is entropy. The gradient of the loss function for the state value parameter is $\bigl\{v_t - V(S_t)\bigr\}\nabla V(S_t)$.

B. Baseline Model

As a baseline, we use a slightly modified version of the model used in [28], which is an unsurpassed model to our best knowledge. The diagram of the model is shown above the dashed line in Figure 2. The model can handle three of the four types of main information as described in Section I, all excluding the inventory. Since a message is a string of 256 characters in length, each character is embedded with its ASCII code, and then the features of the message are extracted by a 1D CNN. Although this is an outdated method in the field of natural language processing, we use it because it is one of the simplest models and message processing is not the focus of our experiments. Status is a vector consisting of numerical values such as *HP* and *strength*, and the feature is extracted by a multilayer perceptron (MLP). The dungeon's shape is 21×79, and each grid is a vector embedded with attributes such as its color and its kind. We extract the feature of the dungeon with a 2D CNN. In addition, the feature of 9×9 grids centered on the agent is extracted in the same way as the entire dungeon. This has been validated by [29] and [30] to help train the agent. These features are then concatenated and input to the MLP and Gated Recurrent Unit (GRU) in that order. $\boldsymbol{h}$ and $\boldsymbol{h}'$ represent the hidden state at the previous time and at the current time, respectively. Finally, the GRU output, which can be regarded as the feature of the current state, is input to two MLPs to obtain the state value and the policy.

C. Action Recursion

We add the embedding layer to the model to incorporate knowledge of the previous action.In this layer, the agent's previous action is embedded and concatenated with the main information's features. It is intuitively that a player may plan a sequence of actions. In particular, in NetHack, some actions are often repeated multiple times, and others consist of multiple actions. An example of the former is the search for hidden passages or doors. If there are hidden passages or doors in grids adjacent to the agent, it can find them with a certain probability by using the *search* action. Therefore, a common strategy is to repeat the *search* action a certain number of times to find them. An example of the latter is kicking and actions which use items. Kicking is composed of two actions: the *kick* action itself and a kicking direction, while the item-using action is composed of two actions: the context of the action such as *quaff* and *read* and the item to be used. Therefore, we need to input the actions that the agent has taken before into the model. Although it is theoretically possible for the model to learn to propagate this information through the hidden state of the GRU layer, we found that in practice, this effect did not occur and that it was beneficial to directly use the previous action as an input.

It is possible to treat all pairs of actions and their targets as separate actions to handle actions that require multiple inputs, but we do not do so because this increases the size of the action space and makes learning difficult. In NetHack, for example, there are more than ten actions using items and more than fifty types of items that can be held, so treating each potential command pair as an action makes the size of the action space beyond 500. Also, there is room for the number of actions input to the model, but we decide to use only the previous action because it should possible to consider any number of actions thanks to GRU. Note that we embed the action with the action space $\mathcal{A}'$, which is defined in Section III-F.

D. Meta Actions

In NetHack, specifying an item is limited to a few situations, such as selecting a potion to quaff immediately after selecting a *quaff* action and selecting an armor to wear immediately after selecting a *wear* action. In these situations, it is of primary importance to specify an item. Therefore, we add the

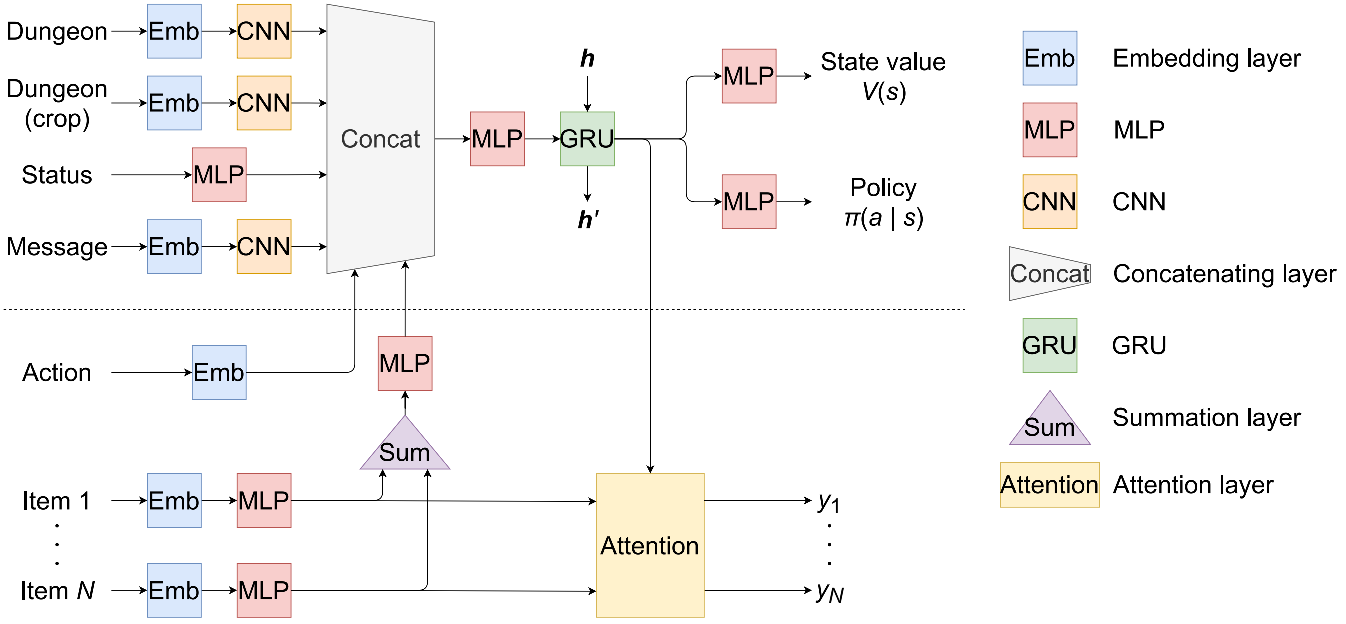


Fig. 2. **Overview of our agent model architecture.** The baseline model is shown above the dashed line. The whole model represents our proposed model where we use attention-based inventory management framework. We note that there are still some other differences in low-level between our full model and the baseline, such as the usage of meta actions.

meta action *use item* to the action space. When the agent chooses this action, it specifies the item according to the probability $\exp(y_i)/\sum_i \exp(y_i)$ where y_i is the ith item's score. The method of calculating the score is described in Section III-E.

The introduction of the meta action also has the advantage that items can be handled independently from normal actions. It is not easy for policy to incorporate items directly because the number of items is not constant. Using meta actions makes the size of the action space fixed and makes the implementation easier. Furthermore, we can handle the normal action space and the action space of inventory separately. We propose the computation of the entropy term in the loss function described in Section III-F as one example of separating. Besides, for the same reason as described in Section III-C, it can prevent the action space from becoming too large.

E. Attention-Based Inventory Feature Extractor

It is not easy to handle the inventory appropriately with neural networks because it is an unordered set of items. To express the unordered nature, we propose the attention-based mechanism to extract the feature of the inventory. First, each item in the inventory is transformed into a vector $\boldsymbol{x}_i$ using the embedding and the MLP. This embedding is done in a similar way to the embedding of each grid in the dungeon. Then, the sum of vectors of all items, $\boldsymbol{x} = \sum_i \boldsymbol{x}_i$, is input to another MLP, and the output is defined as the feature of the inventory. Note that the summation does not depend on the order of the items. The inventory feature is concatenated with other main information features and the embedded previous action, and the feature representation is processed in the same way as the baseline model. Then, to specify the items to be used, each item's score is calculated using the attention operation. Specifically, we denote the feature of the current state as $\boldsymbol{f}$ and prepare the matrices $\boldsymbol{W}_{\mathrm{Q}}$ and $\boldsymbol{W}_{\mathrm{K}}$ and the vector $\boldsymbol{w}_{\mathrm{V}}$, and calculate each item's score y_i by the following equation:

$$y_i = \frac{\boldsymbol{q}^\top \boldsymbol{k}_i}{\sqrt{d_k}} v_i, \tag{6}$$

where $\boldsymbol{q} = \boldsymbol{W}_{\mathrm{Q}}\boldsymbol{f}$, $\boldsymbol{k}_i = \boldsymbol{W}_{\mathrm{K}}\boldsymbol{x}_i$, $v_i = \boldsymbol{w}_{\mathrm{V}}^\top \boldsymbol{x}_i$, and d_k is the dimension of $\boldsymbol{q}$ and $\boldsymbol{k}_i$.

The feature vector is then used with the item meta actions to determine what item to use while being invariant to the order of the items.

F. Loss Function

The use of the meta action and the attention-based item selection policy necessitates modifying the loss function. First, we define modified action spaces and policies. We denote the *use item* action as b_0 and the action using the ith item as b_i. The virtual action space is defined as $\mathcal{A}_{\mathrm{v}} = \mathcal{A} \cup \{b_0\}$, the action space for the inventory is defined as $\mathcal{A}_{\mathrm{i}} = \{b_1, b_2, \ldots\}$, and the actual action space is defined as $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_{\mathrm{i}}$. Similarly, we define the virtual policy as $\pi_{\mathrm{v}} \colon \mathcal{S} \times \mathcal{A}_{\mathrm{v}} \to [0,1]$, the policy for the inventory as $\pi_{\mathrm{i}} \colon \mathcal{S} \times \mathcal{A}_{\mathrm{i}} \to [0,1]$, and the actual policy as $\pi' \colon \mathcal{S} \times \mathcal{A}' \to [0,1]$. Note that these policies satisfy the following relations for all $s \in \mathcal{S}$:

$$\pi'(a \mid s) = \begin{cases} \pi_{\mathrm{v}}(a \mid s) & \text{for } a \in \mathcal{A}, \\ \pi_{\mathrm{v}}(b_0 \mid s)\pi_{\mathrm{i}}(a \mid s) & \text{for } a \in \mathcal{A}_{\mathrm{i}}, \end{cases} \tag{7}$$

$$\sum_{a \in \mathcal{A}_{\mathrm{v}}} \pi_{\mathrm{v}}(a \mid s) = \sum_{a \in \mathcal{A}_{\mathrm{i}}} \pi_{\mathrm{i}}(a \mid s) = \sum_{a \in \mathcal{A}'} \pi'(a \mid s) = 1. \tag{8}$$

The RL algorithm used in this study is IMPALA [8]. Among the gradient of the loss function for the policy parameter, as can be seen in (1), two modifications are applied: all π are replaced with π' in the policy gradient term, and the entropy term is replaced with

$$\nabla \sum_{a\in\mathcal{A}_{\mathrm{v}}} -\pi_{\mathrm{v}}(a \mid S_t)\log \pi_{\mathrm{v}}(a \mid S_t) + \lambda \pi_{\mathrm{v}}(b_0 \mid S_t)\nabla \sum_{b\in\mathcal{A}_{\mathrm{i}}} -\pi_{\mathrm{i}}(b \mid S_t)\log \pi_{\mathrm{i}}(b \mid S_t), \quad (9)$$

where λ is a hyperparameter. When $\lambda = 1$, it means that all actions belonging to $\mathcal{A}'$ are treated equally in the entropy calculation, and (9) is equal to $\nabla \sum_{a\in\mathcal{A}'} -\pi'(a \mid S_t)\log \pi'(a \mid S_t)$.

IV. Experiments and Results

A. Experiment Settings

In this study, we used the in-game score as a reward, which shown in the status bar at the bottom as seen in Figure 1. In NetHack, the in-game score can be earned by various events, such as descending to a new level, defeating enemies, or getting gold. We used this setting because it contains many elements necessary to complete the game and considers combat with enemies, which is often done with the use of items, for which our model is explicitly designed to handle. Furthermore, previous research described in Section II restrict the actions that agents can take to movement only or movement and a few other actions. In this study, however, we used an action space covering most of the actions directly related to the game.

We compared against the approach of Küttler et al. [28], which is based on IMPALA [8] and TorchBeast [31] and uses a limited amount of actions. In particular, only movement in 8 directions, climbing up/down, reading messages, eating, searching, and kicking are allowed for a total 14 different actions. Our baseline extended the amount of actions by considering an additional 11 actions (*apply*, *drop*, *pickup*, *puton*, *quaff*, *read*, *takeoff*, *throw*, *wear*, *wield*, and *zap*) in addition to meta actions. Our proposed approach extended the baseline with action recursion, meta action, and attention-based inventory feature extraction. We trained agents for all approaches for one billion steps. In addition, we conducted the tests over ten episodes with the same seed after training.

Most evaluation on NetHack was done using the character as `mon-hum-neu-mal` (indicating that the role is monk, the race is human, the alignment is neutral, and the gender is male). The game content in NetHack varies greatly depending on the characters, especially on the roles. Therefore, in addition to `mon-hum-neu-mal`, two other characters with different characteristics were used in the experiment. The additional characters were `val-dwa-law-fem` (valkyrie, dwarf, lawful, and female) and `tou-hum-neu-fem` (tourist, human, neutral, and female), and the characteristics of the used characters are as follows:

TABLE I
Average score over ten episodes of testing. We show results for different approaches and three different character scenarios. Best results are shown in bold.

	Monk	Valkyrie	Tourist
[28]	807.1	645.4	42.2
Baseline	1431.2	686.2	56.8
Ours w/o Inventory	1348.4	840.9	191.1
Ours w/o Action Recursion	1500.1	890.5	153.0
Ours	**2345.0**	**906.7**	**283.7**

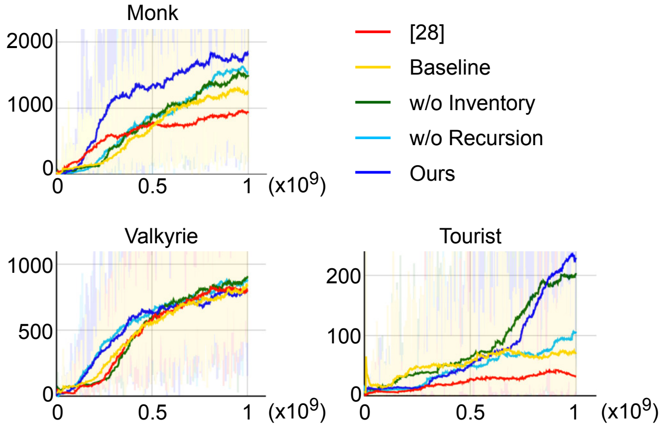


Fig. 3. **Average return during training.** The horizontal axis shows the training step, and the vertical one shows the average return of all agents.

- **monk:** strong at the beginning of the game, with various items and high combat performance, but more challenging to conquer from the middle of the game.
- **valkyrie:** has very few items at the beginning of the game, but the character has high combat performance.
- **tourist:** has various items but is very weak at the beginning of the game, so tactics specific to this character is required.

We evaluated all approaches independently in each of the three character settings.

B. Comparison against Existing Approaches

We compared against existing approaches and testing results are summarized in Table I and the training evolution is shown in Figure 3. We can see that the proposed approach is able to significantly outperform existing approaches. In particular, we can see the largest relative increase in the most challenging class *Tourist*, which does not start out with any combat skills and must rely on inventory usage to stay alive.

C. Ablation Study

We compared four models' performance: the baseline model, the model without action recursion, the model without the attention-based mechanism for the inventory, and the full proposed model. Figure 3 shows the average return during training, and Table I shows the average return in the test. We can see that all the different components of our approach play an important role in obtaining good results.

TABLE II
AVERAGE NUMBER OF TIMES THE AGENT TOOK THE ACTION OF USING ITEMS WITHIN AN EPISODE. RESULTS ARE SHOWN FOR THE POLICY LEARNED WHEN PLAYING WITH THE CLASS OF *Monk*. ACTIONS THAT HAVE NOT BEEN TAKEN EVEN ONCE ARE OMITTED.

	drop	*eat*	*quaff*	*read*
Baseline	0.0	4.0	1.0	0.2
Ours w/o Inventory	0.0	6.8	2.6	0.0
Ours w/o Action Recursion	0.0	4.0	2.7	0.0
Ours	0.6	6.2	2.7	1.7

D. Effect of Difficulty

We can see that the difficulty of the game plays a fundamental role in the total score obtained. As *Monk* does not rely on items and obtains strong intrinsic abilities when leveling up, it shows the highest performance overall. *Valkyrie* and *Tourist* rely heavily on items to stay alive and show lower performance overall.

E. Analysis of the Policy

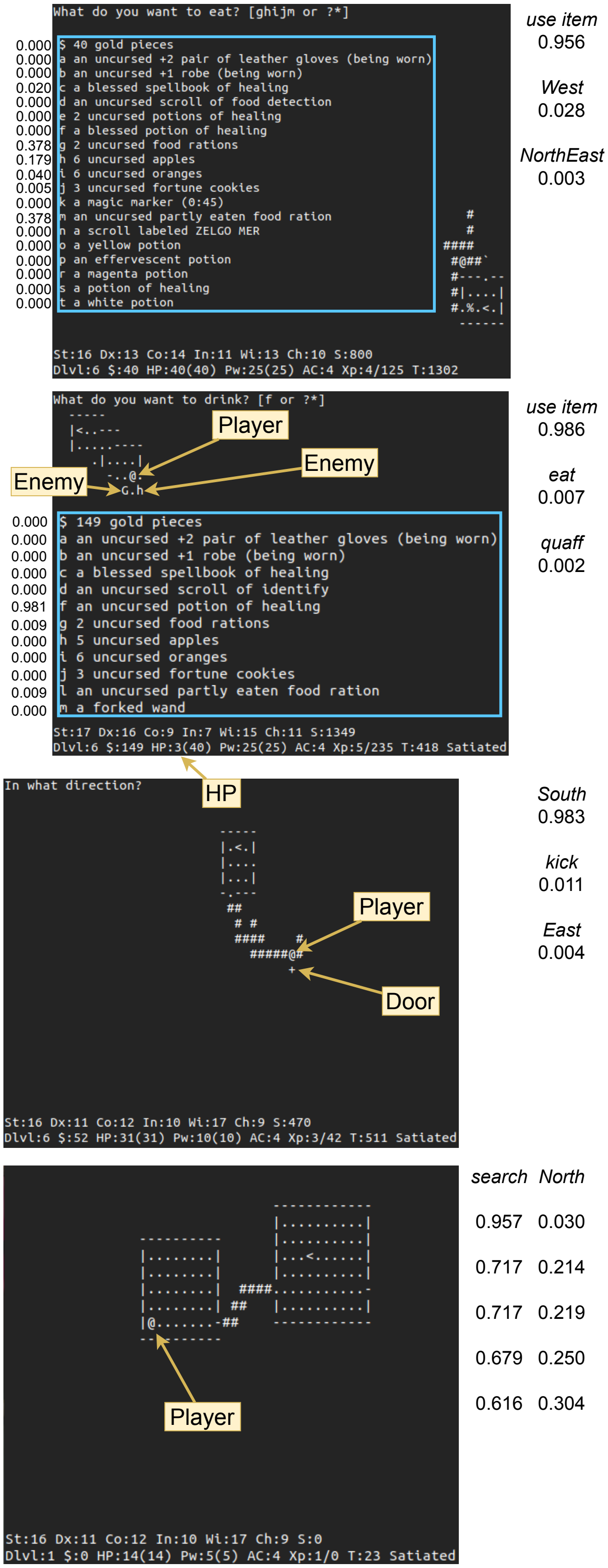
We show characteristic examples of the learned policy of our model in Figure 4. The figures with black background show the state of the game, and the probabilities on the right side show the actions the agent will take in this state and their probabilities. In the top three figures, the three actions with the highest probability in the action space $\mathcal{A}_v$ are shown. In the bottom figure, the transitions of probabilities of the two actions with the highest probability, *search* and *North*, are shown until the agent takes *search* action five times in a row and finds the hidden door. In addition, the inventory screens are shown in the blue boxes and the probability of using each item, or π_i, is shown to the left of each item in the top two examples because the action with the highest probability is *use item*.

The first example shows the screen immediately after the agent takes the *eat* action, and the message at the top of the screen asks the agent which item to eat. In the second example, the agent is in the state immediately after taking the *quaff* action and is asked the item to quaff. The third example shows the state immediately after taking the *kick* action, and the agent is asked in which direction to kick. The fourth example shows the screen immediately after the agent moves to the left by the *West* action.

We can see that in many cases, the agent is able to learn to use items depending on the game state to stay alive. We can see that the attention is able to understand the different usage of the types of items and is invariant to the order of the items.

V. DISCUSSION AND CONCLUSION

We have presented an attention-based approach that is able to handle unordered sets, such as item inventories, in reinforcement learning models. We evaluate on the challenging benchmark that is the roguelike game NetHack and have shown that our proposed inventory management framework, in combination with our handling of meta actions and action recursion, our approach is able to significantly outperform existing approaches.



```
      What do you want to eat? [ghijm or ?*]

0.000 $ 40 gold pieces
0.000 a an uncursed +2 pair of leather gloves (being worn)
0.000 b an uncursed +1 robe (being worn)
0.020 c a blessed spellbook of healing
0.000 d an uncursed scroll of food detection
0.000 e 2 uncursed potions of healing
0.000 f a blessed potion of healing
0.378 g 2 uncursed food rations
0.179 h 6 uncursed apples
0.040 i 6 uncursed oranges
0.005 j 3 uncursed fortune cookies
0.000 k a magic marker (0:45)
0.378 m an uncursed partly eaten food ration                #
0.000 n a scroll labeled ZELGO MER                          #
0.000 o a yellow potion                                  ####
0.000 p an effervescent potion                            #@##`
0.000 r a magenta potion                                  #---.--
0.000 s a potion of healing                               #|....|
0.000 t a white potion                                    #.%.<.|
                                                           ------

      St:16 Dx:13 Co:14 In:11 Wi:13 Ch:10 S:800
      Dlvl:6 $:40 HP:40(40) Pw:25(25) AC:4 Xp:4/125 T:1302
```

```
      What do you want to drink? [f or ?*]
        -----
        |<..---
        |.....----
           .|....|
             -..@.
              G.h

0.000 $ 149 gold pieces
0.000 a an uncursed +2 pair of leather gloves (being worn)
0.000 b an uncursed +1 robe (being worn)
0.000 c a blessed spellbook of healing
0.000 d an uncursed scroll of identify
0.981 f an uncursed potion of healing
0.009 g 2 uncursed food rations
0.000 h 5 uncursed apples
0.000 i 6 uncursed oranges
0.000 j 3 uncursed fortune cookies
0.009 l an uncursed partly eaten food ration
0.000 m a forked wand
      St:17 Dx:16 Co:9 In:7 Wi:15 Ch:11 S:1349
      Dlvl:6 $:149 HP:3(40) Pw:25(25) AC:4 Xp:5/235 T:418 Satiated
```

```
In what direction?


                                -----
                                |.<.|
                                |....
                                |...|
                                -.---
                                 ##
                                  # #
                                  ####    #
                                    #####@#
                                         +









St:16 Dx:11 Co:12 In:10 Wi:17 Ch:9 S:470
Dlvl:6 $:52 HP:31(31) Pw:10(10) AC:4 Xp:3/42 T:511 Satiated
```

```

                                      -----------
                                      |.........|
                    ---------         |.........|
                    |.......|         |...<.....|
                    |.......|         |.........|
                    |.......|  ####...........-
                    |.......| ##      |.........|
                    |@.......-##      -----------
                    ---------













St:16 Dx:11 Co:12 In:10 Wi:17 Ch:9 S:0
Dlvl:1 $:0 HP:14(14) Pw:5(5) AC:4 Xp:1/0 T:23 Satiated
```

search	*North*
0.957	0.030
0.717	0.214
0.717	0.219
0.679	0.250
0.616	0.304

Fig. 4. **Examples of the policy output of the proposed model.** The right side shows the probabilities of taking actions that belong to $\mathcal{A}_v$ with high probabilities. In the top two figures, the blue boxes represent the inventory screens, and the probabilities to the left of the box represent π_i. In the bottom figure, the transitions of the probabilities are shown.

TABLE III

The item specified by the action that uses items. Up to three items are shown in order of the number of times they have been used. The number in parentheses indicates the average number of times the item were used. Results are for the class of *Monk*.

Action	Used Items
drop	*unlabeled scroll* (0.5), unknown *potion* (0.1)
eat	*food ration* (3.0), *apple* (1.5), *orange* (1.0)
quaff	*potion of healing* (2.7)
read	*spellbook* (1.3), unknown *scroll* (0.2), *unlabeled scroll* (0.2)

As shown in Figure 3 and Table I, the proposed model outperforms the existing model. The first example in Figure 4 shows that the agent was able to use items appropriately because it selected edible items as targets to eat with a very high probability at the right time. The bottom two examples show the effect of action recursion. In the second example, the agent was fighting two enemies and had very little *HP* left. Therefore, the possible effective tactics are to flee or quaff a *potion of healing* to recover *HP*, and this example shows that the agent chose the latter. In the third example, the agent was able to take kick action appropriately that requires two consecutive actions: *kick* and the direction to kick. The door + on the south of the agent @ can be opened only by kicking at the stage because the door is locked and the agent does not have a key, so kicking is one of the appropriate actions to take here. In the last example, some rooms likely exist on the dungeon's left side because it is unexplored there, but it is difficult to imagine a path to the left side based on the currently observable dungeon. Therefore, it is natural to assume that there is a hidden door somewhere in the leftmost part of the explored area where the agent @ is now, and it is appropriate to search for it. In fact, the agent found the hidden door on the left side after five *search* actions. Also, since the search action searches around the agent, the agent cannot find the door if it exists in the room's upper left area. Therefore, one of the natural strategies is to search upward again if the door is not found after a certain amount of search actions. The probability transitions of the *search* and *North* actions indicate that this strategy is being taken.

The ablation study shown in Table II indicates that it is necessary to incorporate both of mechanisms to use items appropriately. The most frequently performed actions were *eat* and *quaff*; both are directly related to increasing the agent's survival time. The character we used, *Monk*, always has some food and three *potions of healing* at the beginning of the game. In NetHack, a hunger level progresses with actions, and high hunger level makes the agent's combat power weak and eventually causes death by starvation. Therefore, eating food is essential for the agent to survive for a long time and to be able to get a higher reward. A potion of healing also helps the agent's survival since quaffing it restores *HP*. Besides, the proposed model also took *read* actions. Monk starts the game with a random scroll and is trained to read it if it is a *scroll of enchant armor*. This is because reading it enhance the agent's *AC*, *i.e*. defense and evasion, which is advantageous in combat, where most agent deaths occur. The list of items used is given in Table III.

On the other hand, other actions of using items were not learned. Many of them have a temporary disadvantage when taken or use items that the *Monk* does not have at the beginning of the game, *i.e*., items that need to be picked up to use. In addition, the identity of items that the agent does not have at the beginning of the game is unknown. It can only be revealed by limited actions such as using the item or using a specific item. However, some items have disadvantageous effects, and the *Monk* does not have identifying items initially, so learning these actions is difficult.

The model without the action recursion or inventory handling mechanism successfully took actions of using items a certain number of times. As mentioned earlier, an action to use an item usually consists of multiple inputs, so it seems impossible for a model that does not use action recursion to learn this kind of action. However, a message often indicates what input is required for the second action, as shown in Figure 4. Therefore, although it is not easy to learn messages, which are natural language, it is possible to learn actions that require multiple inputs without action recursion. However, we observed that the model often failed to take appropriate these actions in the test. Also, the use of items is learned even though the model does not use the inventory feature. In NetHack, items are specified by the alphabetic characters shown on the left of the inventory screen in the bottom figure in Figure 1. Therefore, it is possible to learn using items that the agent has at the beginning of the game without the inventory feature if their alphabets are used to specify standard actions.

Figure 3 and Table I show that the proposed method's effectiveness is large for the *Monk* and the *Tourist* and small for *Valkyrie*. The *Monk* and *Tourist* have many items at the beginning of the game, but the *Valkyrie* has few items. As mentioned earlier, it is not easy to learn to use items that it does not possess at the beginning of the game, which is the reason why the proposed method is not very effective for *Valkyrie*.

Although our model allows for inventory usage, the agent still has issues with long-term planning and is unable to learn to equip new equipment. This is also partially due to the fact that in NetHack, equipment is not always a strict upgrade, it can be cursed and give large penalties in many situations. Detecting if equipment is cursed or uncursing it is a hard endeavour which discourages the agent from learning such complex behaviours. We hope that with further developments in reinforcement learning that it may be possible to overcome such issues.

In conclusion, we have shown that our proposed model is able to properly use items, which are essential elements in role-playing games, including roguelikes. As a result, the in-game score increased significantly. On the other hand, the proposed model could not sufficiently learn the use of items that were not possessed at the beginning of the game because their identities were unknown at first. Also, our proposed model did not consider spells, which is a common element in many of role-playing games. Spells often have the same feature: players can cast multiple spells, and casting them consumes a shared resource. The proper handling of these remains as future work.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” in *NIPS Deep Learning Workshop*, 2013.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] International Roguelike Development Conference, “Berlin interpretation,” 2008, accessed: March 14th, 2021. [Online]. Available: http://www.roguebasin.com/index.php?title=Berlin_Interpretation

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, Jun 2016, pp. 1928–1937.

[6] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, “Sample efficient actor-critic with experience replay,” in *International Conference on Learning Representations*, 2017.

[7] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, “Reinforcement learning with unsupervised auxiliary tasks,” in *International Conference on Learning Representations*, 2017.

[8] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, B. Yotam, F. Vlad, H. Tim, I. Dunning, S. Legg, and K. Kavukcuoglu, “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures,” *35th International Conference on Machine Learning, ICML 2018*, vol. 4, pp. 2263–2284, 2018.

[9] 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*, 2013.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[11] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[12] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.

[13] R. Coulom, “Efficient selectivity and backup operators in monte-carlo tree search,” in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[14] S. Frazier and M. Riedl, “Improving deep reinforcement learning in minecraft with action advice,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, no. 1, 2019, pp. 146–152.

[15] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin *et al.*, “The MineRL competition on sample efficient reinforcement learning using human priors,” *NeurIPS Competition Track*, 2019.

[16] W. H. Guss, M. Y. Castro, S. Devlin, B. Houghton, N. S. Kuno, C. Loomis, S. Milani, S. Mohanty, K. Nakata, R. Salakhutdinov, J. Schulman, S. Shiroshita, N. Topin, A. Ummadisingu, and O. Vinyals, “Neurips 2020 competition: The MineRL competition on sample efficient reinforcement learning using human priors,” *NeurIPS Competition Track*, 2020.

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

[18] Y. Kanagawa and T. Kaneko, “Rogue-gym: A new challenge for generalization in reinforcement learning,” in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.

[19] A. Asperti, C. De Pieri, M. Maldini, G. Pedrini, and F. Sovrano, “A modular deep-learning environment for rogue,” *WSEAS Trans. Syst. Control*, vol. 12, pp. 362–373, 2017.

[20] A. Asperti, D. Cortesi, and F. Sovrano, “Crawling in rogue’s dungeons with (partitioned) a3c,” in *Machine Learning, Optimization, and Data Science*, G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, Eds. Cham: Springer International Publishing, 2019, pp. 264–275.

[21] A. Asperti, D. Cortesi, C. De Pieri, G. Pedrini, and F. Sovrano, “Crawling in rogue’s dungeons with deep reinforcement techniques,” *IEEE Transactions on Games*, vol. 12, no. 2, pp. 177–186, 2020.

[22] J. Campbell and C. Verbrugge, “Learning combat in NetHack,” in *Thirteenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2017)*, October 2017, pp. 16–22.

[23] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” in *International Conference on Learning Representations*, 2019.

[24] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, “Structured attention networks,” in *International Conference on Learning Representations*, 2017.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[26] H. Zhao, J. Jia, and V. Koltun, “Exploring self-attention for image recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[27] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia, “Deep reinforcement learning with relational inductive biases,” in *International Conference on Learning Representations*, 2019.

[28] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, “The NetHack Learning Environment,” in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[29] F. Hill, A. Lampinen, R. Schneider, S. Clark, M. Botvinick, J. L. McClelland, and A. Santoro, “Environmental drivers of systematicity and generalization in a situated agent,” in *International Conference on Learning Representations*, 2020.

[30] C. Ye, A. Khalifa, P. Bontrager, and J. Togelius, “Rotation, translation, and cropping for zero-shot generalization,” in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 57–64.

[31] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, “TorchBeast: A PyTorch Platform for Distributed RL,” *arXiv preprint arXiv:1910.03552*, 2019.