Shaomu Tan
s.tan3@students.uu.nl

Can Intelligent Agents play Minecraft?Developing a hierarchical reinforce-ment learning based automated agent to play Minecraft.

Shihan Wang
s.wang2@uu.nl
2021-11-24

1 Introduction

Deep Reinforcement learning has powerful capabilities to solve decision problems with complex environments. It has achieved good results in several well-known fields, such as AlphaGo, Atari, Starcraft II, etc. However, in complex environments with diverse decisions, sparse rewards, and hierarchical task structures, reinforcement learning requires large amounts of data and well-designed hierarchical models to perform the task well. Therefore, the completion of highly complex cascading MDP tasks relies heavily on model policy learning and high-quality trajectory or even expert data. Minecraft, as a representative of hierarchical strategy games, provides an excellent learning environment for reinforcement learning tasks due to its diverse environments and hierarchical task completion trajectories. Therefore, we aim to develop a reinforcement learning or imitation learning agent to play Minecraft games in this project. Importantly, we will examine the influence of different hierarchical structures on the learning performance and efficiency of RL algorithms.

Imitation learning and learning from demonstration are approaches to learning good strategies fast from large amounts of high-quality data to give intelligent agents a guide to complete their tasks. In this research project, we integrated imitation learning, a technique for learning and imitating decisions directly from data, with our RL algorithm to improve the learning efficiency and reduce the amount of experimental data. And we evaluated the performance of both behavior cloning and Learning from demonstration algorithms in the MineRL environment. The dataset and environment from the MineRL competition (a data contributor) will be used. The dataset includes recorded data of human players playing the Minecraft game. In addition, this internship project is a sub-project under a running collaborative project (aimed at developing an intelligent agent for supporting human players in Minecraft). Thus, a human-computer interaction interface was designed and implemented on the top of the intelligent agent for future development. The code and experiment details can be found here: `https://github.com/Smu-Tan/MineRL_HCI`

2 MineRL competition, dataset, and environment

MineRL competition[6] is a reinforcement learning competition launched by NeurIPS in 2019 for learning human player behavior in the Minecraft game environment to accomplish certain specific goals such as Obtaining Ironpickaxes, Diamonds, log items and etc. The competition also provides the largest dataset with over 60 million frames of recorded human player data, offering the experimental data and environment for modern reinforcement learning research. Specifically, the dataset provides recorded data for the following types of tasks for human play: 1. Navigate; 2. Treechop; 3. Obtain Iron Pickaxe; 4. Obtain Diamond; and etc(Figure.1).

To investigate the distribution of data for different types of tasks, we measured the number of episodes of data for different tasks, as well as the distribution regarding the length of episodes (Figure.2). In this case, An episode is one sequence of states, actions and rewards, which ends with a terminal state. While the length of an episode is the number of timesteps that the episode consists of, and when the episode has length is the number of timesteps in the episode, and the longer the length of the episode, the more time it spends on the task. Table 1 shows the number of episodes for *treechop* and obtains *ironpickaxe* tasks. Additionally, we found that the dataset is not perfect. Specifically, for the obtained iron pickaxe, only 77% of the episode data were actually obtained for the iron pickaxe. We also visualized the distribution of rewards for the different episodes of the iron pickaxe task (Figure.2), and we can see that it fits a long-tailed distribution.

Table 1: The number of episodes for MineRL dataset in the treechop and obtain ironpickaxe task

Task	Number of episodes [1]	Number of episodes obtained the sufficient item
Treechop	210	210
Obtain Iron pickaxe	234	182
Obtain diamond	122	73

[1] We consider one data record as a episode

These tasks also increase in hierarchy, as they range from simple to difficult, which means that complex tasks contain simple tasks. For example, in the tree-chopping task, the agent needs to cut down the corresponding number of trees to get 64 rewards. And, in the task of acquiring iron pickaxes and diamonds, in addition to simple, non-hierarchical tasks such as navigation, tree-chopping, the agent also needs to perform hierarchical actions on top of that, such as acquiring logs first, then making sticks, then crafting wooden pickaxe to mine cobblestone in order to obtain iron pickaxes. Moreover, when we refine the goal of the game into multiple subtasks, we get a subtask sequence, and this hierarchical structure poses another challenge: even human players do not take the same subtask sequence in the game. This is especially evident in the task of obtaining advanced items. Figure.3[6] shows the visualization of the subtask sequence for the MineRL competition dataset for the iron pickaxe and diamond tasks. It can be seen that there are transitions between subtasks, but in practice there are strong dependencies between many subtasks, which we will mention later in the methodology section.

In order to design an agent that can assist human players in the game, we will focus on the following two tasks:

Treechop: In this mission the agent will need to find trees in a forest environment and cut them down to get log items, each log item will get 1 reward point. when 64 rewards are obtained, the mission is over.

Obtain Pickaxe: In this task the agent will obtain the necessary materials to make a pickaxe in a complex environment (including lake/ocean, forest, desert). The process includes tasks such as moving, chopping down trees, making items, digging for stones, and the mission ends when a pickaxe is obtained. The general pickaxe task contains wooden, stone and iron pickaxe tasks.

Figure 1: Images of various stages of six total environments

Moreover, the MineRL environment allows us to train and test agents on different task environments. Each environment has its own defined maximum timestep limit, so we cannot have an unlimited number of training sessions in one environment, which means that after a certain number of actions, we need to reload the environment, which means to start interacting from a new random point.

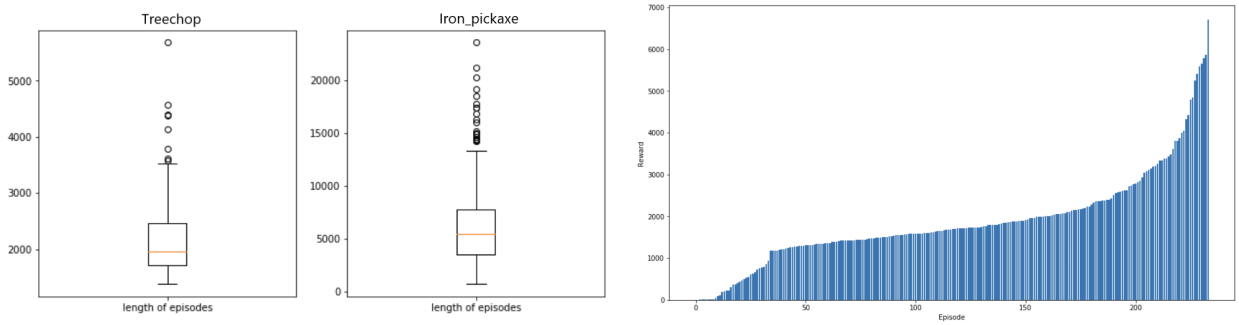

Figure 2: (a) Distribution of length of episodes for Treechopping and Obtain-Ironpickaxe task, y-axis is timesteps (b) Distribution of reward for Iron pickaxe task

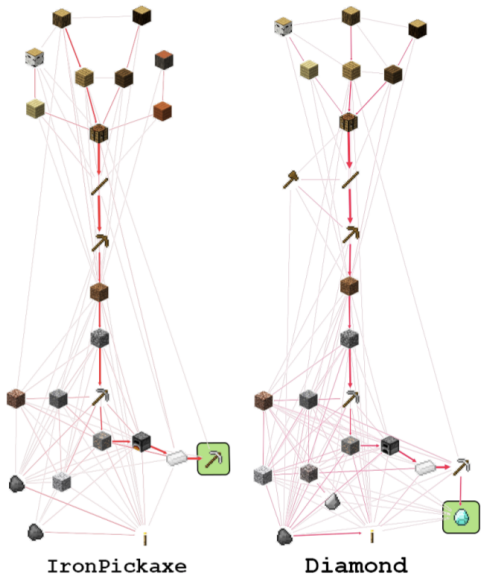

Figure 3: Item precedence frequency graphs for Obtain-Ironpickaxe and Obtain-Diamond

3 Methods

The MineRL Treechop and Obtain pickaxe tasks can be considered as a Markov Decision Process(S, A, P_a, R_a) where S stand for the state space, A is the action space available from the state S, $P_a(S, S') = Prob(S_t + 1 = S'|s_t = S, a_t = a)$ denotes the probability of transition from state S(at time t) to S'(at time t+1) while taking action a, and $R_a(s, s')$ denotes the expected immediate reward received by the environment after transferred from state S to state S' given action a. The perspective of our methods is to maximize the expected sum of rewards using the policy we learned as well as update the policy to the optimal one. We will focus on the imitation learning and learning from demonstration methods for the following sections.

3.1 Markov Chain

Since the Obtain Pickaxe in MineRL task has a complex hierarchical nature(see Figure.3), we first define this hierarchical route as a subgoal chain in order to keep consistency, and we call each subgoal in a hierarchical route as a subtask. Each episode in the data may have a unique subtask chain, which represents the policy of each human player. Therefore, we decided to use the Markov Chain model to extract the pattern of transition from each subtask to the next subtask and thus analyze the policies of the human players for the available data. However, since the markov chain is consistent with the markov assumption, i.e., the probablity of next state depends only on the state of the current state, we will use only the markov chain model to generate a suitable subtask chain and use it in future algorithms for reinforcement learning or imitation learning. The constraint in the markov chain is: we added the rules of transition between each subtask in the Minecraft game to generate the greediest subtask chain based on the markov chain, for example, only when agent has 3 planks and 2 sticks can be transitioned to the wooden pickaxe subtask.

3.2 Behavior Cloning

Imitation Learning is a framework for learning a behavior policy from demonstrations. Usually, demonstrations are presented in the form of state-action trajectories, with each pair indicating the action to take at the state being visited.

Definition 3.1. (Imitation Learning Problem) For a system with transition model with states $s \in S$ and controls(actions) $a \in A$, the imitation learning problem is to leverage a set of demonstrations $\Xi = \{\epsilon_1, ..., \epsilon_D\}$ from an expert policy π^* to find a policy $\hat{\pi}^*$ that imitates the expert policy.

One common way, known as Behavior Cloning (BC), treats the action as the target label for each state, and then learns a generalized mapping from states to actions in a supervised manner. Behavior cloning has been used in autonomous driving[4, 5], game environment[2, 3], and a range of RL applications[1, 10] to prevent the drawbacks of limited training samples and the high cost of attempts by RL algorithms. In other words, Behavior cloning is the use of supervised learning to mimic the behavior of expert data by accomplishing an optimization problem. As shown in the equation below, Behavior cloning approaches use a set of expert demonstrations $\epsilon \in \Xi$ to determine a policy that imitates the expert. π^* is the optimal policy in the demonstration data when state is x, $\pi^*(x)$ is the action under this policy, and $\pi(x)$ denotes the action derived from our trained policy, and our goal is to make the difference between them as small as possible to maximize the imitation of the expert's policy.

$$\hat{\pi}^* = \underset{\pi}{\operatorname{argmin}} \sum_{\epsilon \in \Xi} \sum_{x \in \epsilon} Loss(\pi(x), \pi^*(x))$$

Unlike other reinforcement learning methods such as Q learning, SARSA, and etc, behavior cloning process does not require interaction with the environment, and therefore significantly reduces training time and computational resources. However, the policy learned by behavior cloning will never be better than the optimal policy in the demonstration, because imitating behavior does not advance policies to the perfect policy as RL algorithms do, therefore we cannot expect that behavior cloning will have better results(accumulated expected reward) than the experimental data. Thus, behavior cloning requires high quality data to ensure that the learned policies will have a good performance. In addition, we may have many unseen states when the amount of experimental data is insufficient and not representative enough, which may lead to many stuck situations in the test.

3.3 Deep Q learning from Demonstrations and forgER

Before mentioning the forgER algorithm, we first need to understand the algorithm at the bottom core of forgER: Deep Q learning from Demonstrations(DQfD)[7]. Overall, DQfD can be divided into two phases: 1. pre-training phase; 2. training step. the first part is very similar to behavior cloning, which optimizes the loss function using the expert data to simulate the behavior. However, in the training process, the agent will further interact with the environment on the basis of the pre-trained, and then learn and improve the policy. The Pseudo-code of DQfD is demonstrated in Algorithm.1[7]. It is worth noting that DQfD introduces a Prioritized experience replay buffer[8] to improve the data efficiency useage. This is reflected in particular in the training step: when updating the network, a portion of the data is sampled from expert data and self-generated data (new data generated when interacting with the environment) and then learned. In addition, DQfD has improved the loss function to optimize and prevent overfitting more effectively. Specifically, its loss term is composed of four terms: 1. imitation loss $J_E(Q)$; 2. 1-step TD loss $J_{DQ}(Q)$; 3. N-step TD loss $J_n(Q)$; 4. L2 regularization loss $J_{L2}(Q)$.

$$J(Q) = J_{DQ}(Q) + \lambda 1 * J_n(Q) + \lambda 2 * J_E(Q) + \lambda 3 * J_{L2}(Q)$$

Based on DQfD, the forgER algorithm[9] improves a lot on data usage efficiency and hierarchical structure and forgettable experience replay mechanism. Since the forgER team won the competition in 2019 and the results in their paper are the best published results of MineRL so far, we will reproduce the method in this paper in the newest 2021 dataset and environment in order to provide a high quality MineRL assistant. Figure.4[9] shows the structure of the forgER model, similar to DQfD, we can divide the whole process into two parts: 1.the imitating phase (pre-training phase in DQfD); 2.Forging phase (training phase in DQfD). In the imitating phase, the agent only learns demonstration data, and does not involve interaction with the environment. In the training phase, the agent learns sampled data from the experience replay buffer, which means that it learns both experimental data and self-generated data.

Moreover, based on DQfD, forgER adds a hierarchical design to achieve better results, which is largely learned from the subtask plan that human players have when playing Minecraft. In addition, forgER has improved the Prioritized experience replay mechanism into a forgettable feature. Specifically, during the training process, as time progresses, the agent will sample more data from self-generated data and less data

Algorithm 1 Deep Q-learning from Demonstrations

Inputs D^{replay} initialized with demonstration data set, θ: weights for initial behavior network (random), θ' : weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates.
for steps t $\in \{1, 2, ...k\}$ **do**
    Sample a mini-batch of n transitions from D^{replay} with prioritization
    Calculate loss J(Q) using target network
    Perform a gradient descent step to update θ
    if t mod $\tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**
end for
for steps t $\in \{1, 2, ...\}$ **do**
    Sample action from behavior policy a $\sim \pi^{\epsilon Q_\theta}$
    Play action a and observe (s', r).
    Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of n transitions from D^{replay} with prioritization
    Calculate loss J(Q) using target network
    Perform a gradient descent step to update θ
    if t mod $\tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**
    s $\leftarrow s'$
end for

from expert data, which theoretically helps the agent to imitate more expert behavior before surpassing it. We have attached the forgER pseudo-code (algorithm.2) to see it more clearly.

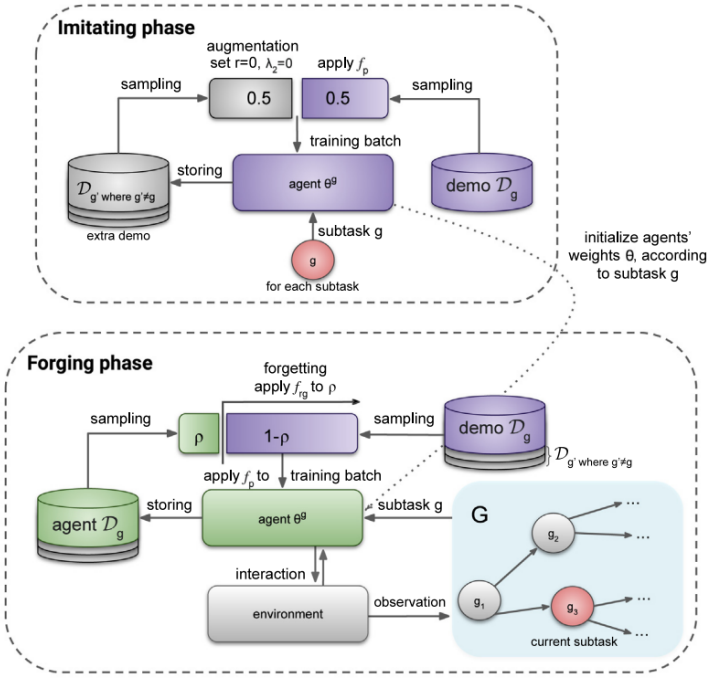

Figure 4: forgER model structure

4 Experiment

We conducted treechop and obtain-pickaxe experiments for both Behaviour cloning and forgER methods to investigate which method is more capable for obtaining better results as an agent for our HCI module. Besides, in order to improve these existing methods, we implemented the hierarchy structure that our markov chain generated as a subtask chain to both BC and forgER methods. All experiments are conducted on the envrionment of MineRLObtainDiamondDense-v0 since it provides the longest timestep limit.

Algorithm 2 Forgetful experience replay

Inputs: G: subtask graph, θ: weights for the initial sub-task network, θ': weights for the target sub-task network, τ: frequency at which to update the target network, k : the number of the imitating steps, D : structured replay buffer, f_p : goal-oriented pseudo reward function, $f_rg(t)$: forgetting ratio function
for g $\in V(G)$ **do**
    $t_g \leftarrow 0, \theta_g \leftarrow \theta, \theta'_g \leftarrow \theta'$
    $D_g^{demo} \leftarrow (o, a, f_p(o, r), o', a', 1, g) : d_i \in D$ where $g_i = g$
    $D_g^{extra} \leftarrow (o, a, 0, o', a', 0, g_i) : di \in D$ where $g_i \neq g$
    Imitating$(g, D_g^{demo}), D_g^{extra})$
end for
for episode k $\in \{1, 2, ...\}$ **do**
    forging$(g, D_g^{demo}), D_g^{agent}, f_p, k, f_{rg}(k))$
    Select next subtask g from G
end for

Algorithm 3 ForgER imitating phase

Function imitating
Inputs: g : current subtask, D_g^{demo}: initialized with demonstration data for current subtask g, D_g^{extra}: initialized with data from other subtasks
for steps t $\in \{1, 2, ...k\}$ **do**
    Sample a mini-batch of n transitions from D_g^{demo} and D_g^{extra}
    Calculate loss $L(Q^g)$ using target network θ'_g and perform a learning update to θ_g
    if t mod $\tau = 0$ **then** $\theta'_{cs} \leftarrow \theta_g$ **end if**
end for

Algorithm 4 ForgER forging phase

Function forging
Inputs: g : current subtask, D_g^{demo}: initialized with demonstration data for current subtask g, D_g^{agent}: initialized with data collect by the agent for current subtask g, t: current step, f_p: pseudo reward function, k: current episode, $f_rg(k)$: forgetting until episode
while subtask g not solved **do**
    Sample action from policy a $\sim \pi^{\epsilon Q_{\theta g}}$
    Play action a and observe (o', r).
    Replace (o, a, r, o') by (o, a, $f_p(o, r)$, o')
    Store (o, a, r, o',O,g) in D
    Sample a mini-batch of n transitions from D_g^{demo} and D_g^{agent} with forgetting rate $f_rg(k)$
    Calculate loss $L(Q^g)$ using target network and perform a learning update to θ_g
    if t mod $\tau = 0$ **then** $\theta'_{cs} \leftarrow \theta_g$ **end if**
    t $\leftarrow$ t+1
end while

4.1 Markov chain

We visualize the heat map(Figure.5) and markov model(Figure.6) about the subtask transformation of all ObtainIronPickaxe data. The start(S) and termination($\backslash S$) in the figure represent the start and end of an episode, respectively. We also generated a most greedy subtask chain based on the constraints of the minecraft game combined with the markov model, which supported our agent achieve better performance in subsequent tasks.

One thing worth noting is that the subtask chain here does not contain the number of dependencies, whereas in the subsequent markov_bc, markov_forger, and HCI modules, we include these number of dependencies by working backwards from the last subtask of the chain to the number of dependencies. Besides, in order to make our agent get as many rewards as possible, we adjusted the order of some subtasks, which is actually irrelevant when you collect a specified number of subtask items, e.g., we let the agent make the furnace first after completing the stone pickaxe in the follow-up experiment, and this process can immediately make the agent get rewards, but there is no dependency between making the furnace and finding the iron ore.

The greediest subtask chain generated by Markov model:

Start → log → planks → crafting_table → stick → wooden_pickaxe → cobblestone → stone_pickaxe → iron_ore → furnace → iron_ingot → iron_pickaxe → Termination

The greediest subtask chain generated by Markov model with item quantity constraint:

Start → log:6 → planks:24 → crafting_table:3 → stick:8 → wooden_pickaxe:1 → cobblestone:14 → stone_pickaxe:1 → iron_ore:3 → furnace:1 → iron_ingot:3 → iron_pickaxe:1 → Termination

Modified subtask chain that used in the later experiments:

Start → log:6 → planks:24 → crafting_table:1 → stick:8 → wooden_pickaxe:1 → cobblestone:14 → crafting_table:1 → stone_pickaxe:1 → furnace:1 → iron_ore:3 → iron_ingot:3 → crafting_table:1 → iron_pickaxe → Termination

4.2 Behavior Cloning

We first trained the Treechop agent using all treechop data and tested it in the MineRL Treechop environment. The whole training process does not involve interaction with the environment because behavior cloning is essentially supervised learning. After the training, we tested 100 episodes in the MineRLTreechop environment and visualized the distribution of the rewards and some statistical values, see: Figure.A.1 (1 reward means that a log item was obtained). For the Treechop task, the average value of rewards for the 100 episodes tested was 54, and there were some episodes that did not get any rewards or had rewards below 30. Besides, The average reward of expert data is 64, thats means we are approaching the performance of human players to some extent with evidence that average reward of our agent and human is 54 versus 64 .

In addition, for the obtain-pickaxe task, we also trained a cobblestone agent based on the behavior cloning algorithm and combined it with the most greedy subtask chain generated by the markov model, here we call it Markov_Behaviour cloning, to test the performance in the Obtain_pickaxe environment. And the Markov_BC model is also be compared to the pure behaviour cloning baseline provided by MineRL, Table.2 demonstrates the average reward two model earned. This proves that a good hierarchical policy is crucial, and it is difficult for a pure RL or IL algorithm to learn such a high-dimensional policy automatically from the available data.

Moreover, about the specific Markov_bc test results are shown in Table.4, where it obtained wooden pickaxe 43 times and stone pickaxe 11 times out of 100 episodes tested. One of the testing result demonstrates that in 100 episodes testing, our markov_forger had 67 episodes that obtained 14 cobblestone items. And the And for the simplest task of obtaining logs, only half of the episodes obtained a sufficient number, mainly because in many episodes the agent did not dodge the lake or the ocean and thus ended up.

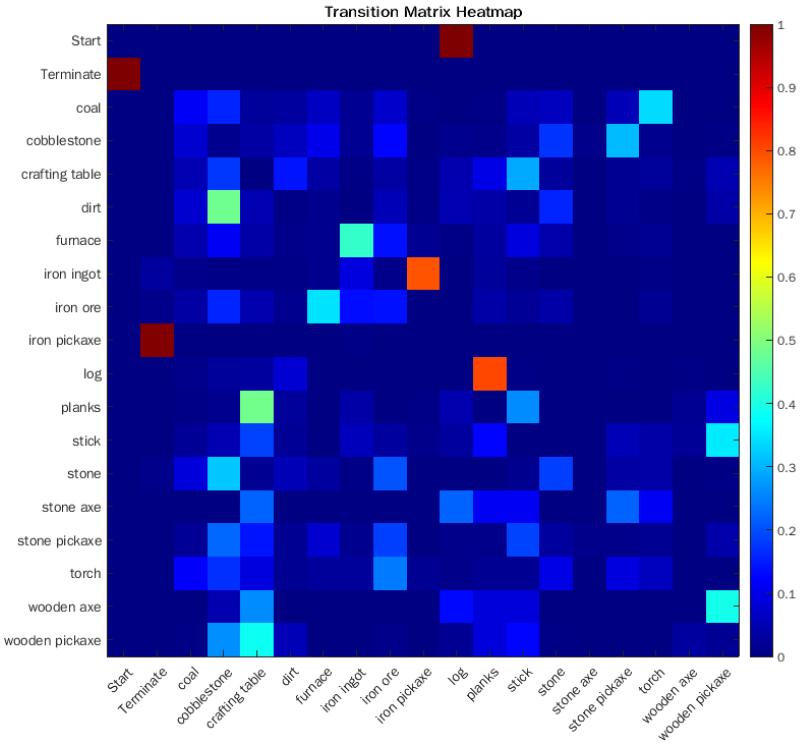

Figure 5: Heatmap of subtask transition

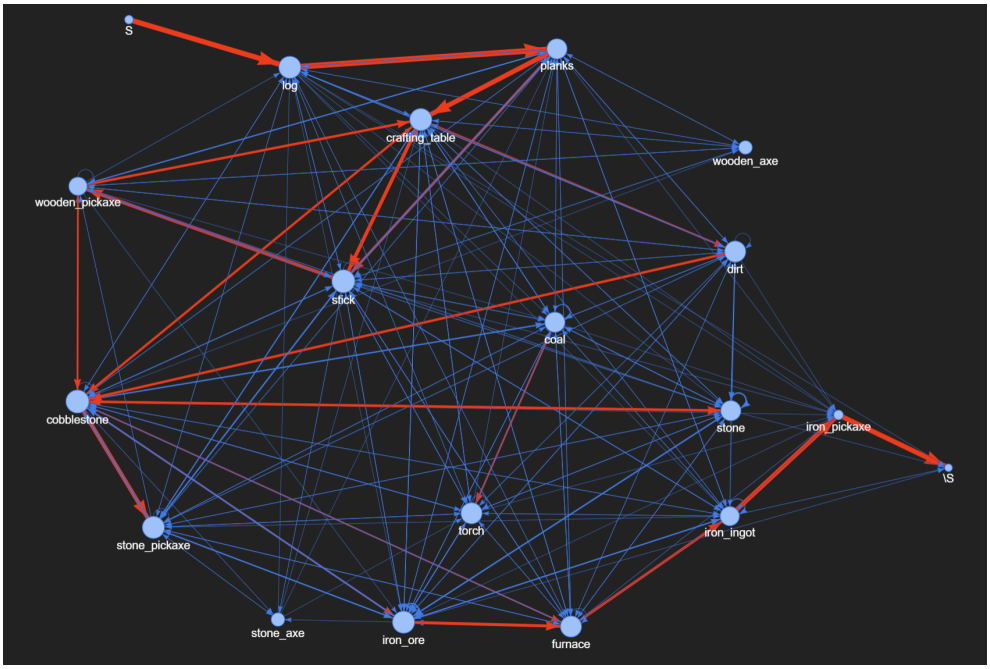

Figure 6: Visualization of markov model

Table 2: The average reward of BC algorithms in 100 episodes testing for Obtain-Pickaxe task

Name	MineRL Behaviour Cloning Baseline	Markov_Behaviour Cloning
Average Reward	98.70	116.85

4.3 forgER

We initially trained the original forgER agent for both tree-chopping and obtain-pickaxe tasks, however we found that we could not obtain the results claimed in the original paper. This is likely due to the experiments

Number of episodes obtained the sufficient item and Average acquisition time				
Item	Markov_BC	forgER	Markov_forgER	Human players
6 log	57(107.50 s)	87 (187.19 s)	88 (263.17 s)	-
14 cobblestone	16 (298.62 s)	41 (265.03 s)	67 (253.12 s)	-
1 wooden pickaxe	43 (81.524 s)	58 (127.71 s)	73 (169.42 s)	- (93.9 s)
1 furnace	11 (348.81 s)	41 (261.88 s)	67 (264.30 s)	-
1 stone pickaxe	11 (348.70 s)	38 (259.1 s)	64 (264.94 s)	- (171.9 s)
1 iron pickaxe	0	0	1 (537.6 s)	- (334.8 s)

Table 4: Obtain-pickaxe testing result

in the original paper were based on the 2019 dataset and environment, which has now changed. For tree-chopping task, we tested 100 episodes in the MineRLTreechop environment and visualized the distribution of the rewards and some statistical values, see: Figure.A.2 (1 reward means that a log item was obtained). In terms of average rewards, the forger method obtained slightly better results than behavioral cloning (58 versus 54 respectively). This is most likely due to the fact that forger, in addition to mimicking the behavior in the expert data, we also let the agent interact with the environment to update the Q-network.

Further we trained the original forger and the forger mixed with the subtask chain generated by our knowledge-based hierarchical strategy of markov model for obtain-pcikaxe task. Table.3 shows the average rewards achieved by these two methods in a test with 100 episodes. markov_forger achieves 300 more rewards than forger in terms of average rewards. And if we focus on the number of pickaxe that can be obtained by the two different methods, using markov_forger increases the chance of obtaining wooden pickaxe and stone pickaxe by 25% and 64%, respectively. This is basically because of the limitation of the original subtask chain that the forgER has, such as the agent need to find the crafting it previous placed to craft a new item. In addition, We computed the average reward of human expert data in MineRL Obtain-Diamond dataset. And because we conducted the obtain-pickaxe experiments with the ObtainDiamondDense-v0 since it provides the longest timestep limit, We only measured the episodes of human expert data up to the point where they acquired ironpickaxe. To see the details of the experiment results, please find it on `https://wandb.ai/tan3/MineRL_pickaxe/runs/3iiosxzl?workspace=user-tan3`.

Finally, to further validate the markov_forger model, we tested the items it could obtain in 500 episodes, please refer to `https://wandb.ai/tan3/test_minerl/runs/22why0qw?workspace=user-forthespecificvisualizationresult` Overall, there is a 68% chance that our agent can get wooden pickaxe and more than a half chance that it can get stone pickaxe. Besides, we measured the three pickaxe items' average aquisition time in our dataset. To compare the average aquisition time between human players and our best agent, We still can observe a huge difference, that is, our best agents on average acquire these pickaxes for 100s longer than human play parents.

Table 3: The average reward of forgER algorithms in 100 episodes testing for Obtain-Pickaxe task

Task	forgER	Markov_forgER	Human
Average Reward	688.94	1009.40	2689.81

5 Human-Computer Interaction Module

Furthermore, we have designed a human-machine interface for the Minecraft game. It can be used in future experiments to give advice and guidance to human players with policies learned by the intelligent agent. The model diagram of the HCI module is shown in the Figure.F.3. Briefly, in the experiment, The real-time inventory information and first-person view pictures from human players will be input to the HCI module, and then it derives the recommended subtask, special action(s) and basic action(s) recommendations by the intelligent agent's operation, and finally output the textual recommendations through a simple dialogue system to the minecraft game. Overall, our intelligent agent can provide three different recommendations:

1. subtask recommendation
    - Subtasks can be obtaining a certain number of log, cobblestone, stick items etc. Subtasks sequence can be learned by an intelligent agent or hand-crafted.
2. special action recommendation

- It provides instructions for synthesizing items, placing items, and other operations, such as crafting wooden pickaxes.
- Craft – [Planke, stick, crafting_table, wooden_pickaxe, etc]: converting raw material to tools using crafting table.
- Place – [item]: placing item(s)
- NearbyCraft – [item]: Craft item(s) when furnace nearby
- Equip – [item]

3. basic action recommendation:

- Basic actions recommendation provides some basic operational guidance to the human player, and we only provide it to the player on the task of obtaining logs and cobblestone.
- Move – [forward, back, left, right]
- Camera – [up, down, left, right]
- Combo – [forward+jump]
- Attack

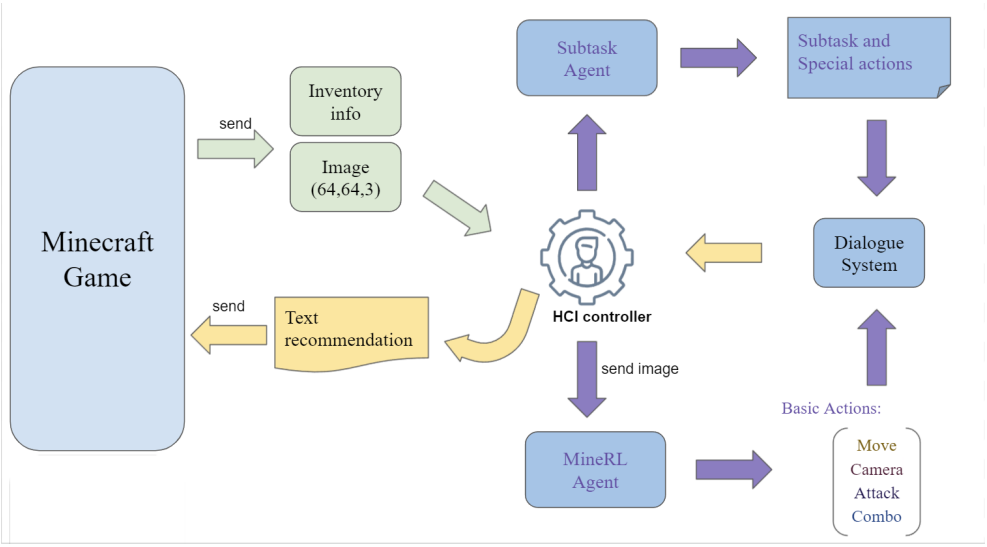

Figure 7: Human-Computer Interaction model structure

It is worth noting that we have designed HCI modules for two different experiments: 1. obtain stone pickaxe; 2. obtain iron pickaxe. For the stone pickaxe experiment, we provide a basic action recommendation interface for giving guidance to inexperienced players in cutting trees and getting cobblestone. Note: the basic action recommendation feature can be turned off during the experiment. However, for the iron pickaxe experiment, we will only provide subtasks and special action recommendations to human players. The code and corresponding documentation can be found at `https://github.com/Smu-Tan/MineRL_HCI/tree/main/MineRL_HCI_Assistant`. The subtask sequence learned by the intelligent agent and the full recommendation template (no basic action recommendation) for both stone pickaxe and iron pickaxe tasks can be found in Appendix.H and I.

Conclusion

In this report, we conducted an in-depth study on the hierarchical reinforcement learning methods on the area of MineRL. First, we proposed and implemented a markov model in an attempt to explore the hierarchy structure using MineRL human player's data for the obtain-pickaxe task. The markov model calculates the state transition probability, and can be used to generated subtask chain in a simple and intuitive way when adding Minecraft constraints on it. Besides, the generated hierarchy structure can be combined with different reinforcement or imitation algorithms to improve their performance largely in the later experiments.

In addition, we implemented a total of four algorithms to try to explore MineRL's tree-chopping and obtain-pickaxe problems, including behavior cloning baseline, markov_behaviour cloning, forger and markov_forger.

It was shown that chains generated by the markov model would substantially improve the performance of intelligent agents, as shown by the average reward and the types of pickaxe that could be obtained. However, the performance of intelligent agents still remains incomparable to that of human experts. The possible reason for this deficiency comes from the imperfect dataset, the restricted MineRL environment and the difficulty for the RL agent to learn high-dimensional policies such as the order of subtasks, but only sub-policy like how to chop trees or dig cobblestone.

Lastly, we have developed an interface for human-computer interaction on our best model (markov). In general, even though it is difficult for our intelligent agents to reach a similar level to human players from a certain point of view, our HCI interface can provide diverse policies, i.e., high-dimensional planning policies and low-dimensional execution policies corresponds to subtask, special action recommendations and basic action recommendations. Such diverse policies can be studied in relation to human decision making reasoning when playing minecraft, and the developed markov_forger, HCI interface can also be reused, providing possible research directions and help for future research in psychology and cognitive science.

References

[1] Amro Awad and Yan Solihin. "Stm: Cloning the spatial and temporal memory access behavior". In: *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2014, pp. 237–247.

[2] Miroslav Bogdanovic et al. "Deep apprenticeship learning for playing video games". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[3] Brian Chen et al. "Behavioral Cloning in Atari Games Using a Combined Variational Autoencoder and Predictor Model". In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, pp. 2077–2084.

[4] Felipe Codevilla et al. "Exploring the limitations of behavior cloning for autonomous driving". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9329–9338.

[5] Wael Farag and Zakaria Saleh. "Behavior cloning for autonomous driving using convolutional neural networks". In: *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE. 2018, pp. 1–7.

[6] William H Guss et al. "MineRL: A large-scale dataset of Minecraft demonstrations". In: *arXiv preprint arXiv:1907.13440* (2019).

[7] Todd Hester et al. "Deep q-learning from demonstrations". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[8] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[9] Alexey Skrynnik et al. "Forgetful experience replay in hierarchical reinforcement learning from expert demonstrations". In: *Knowledge-Based Systems* 218 (2021), p. 106844.

[10] Zihao Wang et al. "Two-stage Behavior Cloning for Spoken Dialogue System in Debt Collection." In: *IJCAI*. 2020, pp. 4633–4639.

Appendices

A Testing result of both BC and forgER algorithms for Tree-chopping task

Figure.A.1 and figure.A.2 show the testing result of two different algorithms on the tree-chopping task. And the visualization of reward of BC training in the tree-chopping environment can be found at: `https://wandb.ai/tan3/Treechop/runs/3nyclwi9?workspace=user-tan3`

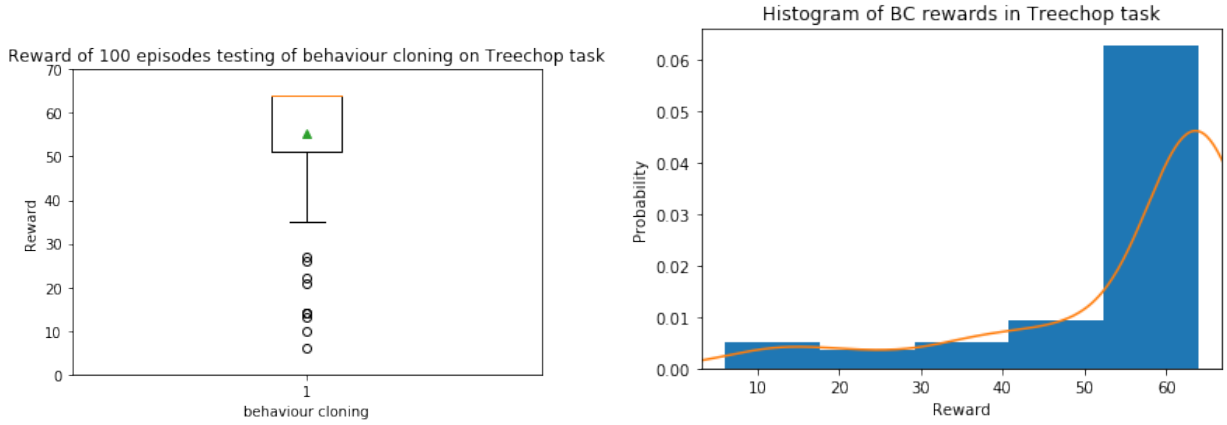

Figure A.1: (a) Boxplot of the reward of 100 episodes testing of Behaviour cloning for Tree-chopping task(b) Histogram of 100 episodes testing Of Behaviour cloning for Tree-chopping task

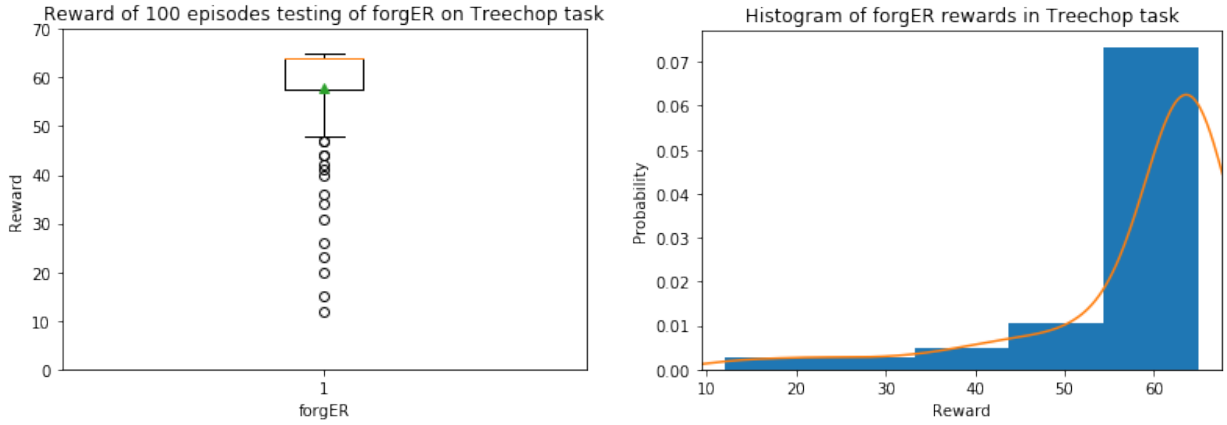

Figure A.2: (a) Boxplot of the reward of 100 episodes testing of forgER for Tree-chopping task(b) Histogram of 100 episodes testing Of forgER for Tree-chopping task

B Code and result details of Markov model

Code and result details of Markov model can be found at: `https://github.com/Smu-Tan/MineRL_HCI/tree/main/MineRL_Markov`

C Testing result of BC algorithm for obtain-pickaxe task

Visualization of testing results of BC method for the oobtain-pickaxe task can be found at: `https://wandb.ai/tan3/BC_test_pickaxe/runs/3ht1ij9u?workspace=user-tan3`

D Testing result of forgER algorithm for obtain-pickaxe task

Visualization of testing results of forgER method for the oobtain-pickaxe task can be found at: `https://wandb.ai/tan3/test_minerl/runs/1lsluick?workspace=user-tan3`

E Testing result of Markov_forgER algorithm for obtain-pickaxe task

Visualization of testing results of Markov_forgER method for the oobtain-pickaxe task can be found at: `https://wandb.ai/tan3/test_minerl/runs/22why0qw?workspace=user-tan3`

F Average acquisition time of three different pickaxe items of Human players

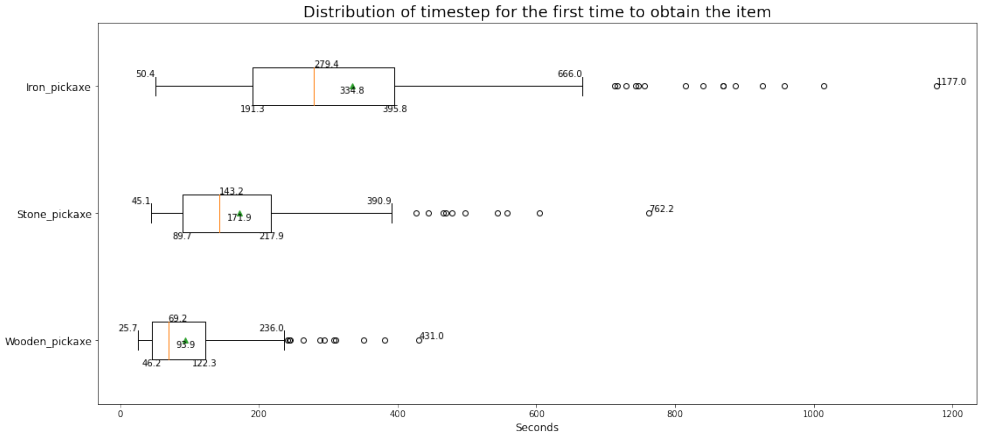

Figure F.3: Average acquisition time of three different pickaxe items of Human players

G Code and documentation of HCI module

Code and documentation of HCI module can be found at: `https://github.com/Smu-Tan/MineRL_HCI/tree/main/MineRL_HCI_Assistant`

H subtask sequence used in HCI module

We have modified the most greedy route generated by the markov model to better fit the HCI module. Specifically, we adjusted the number of crafting able to be synthesized for the first time from 3 to 1. This is essentially the same as the original Markov model's chains, but this approach allows for as little raw material to be used at once as possible, taking into account that some inexperienced players may synthesize some tools and use more raw material.
subtask sequence:

Start → log:6 → planks:24 → crafting_table:1 → stick:8 → wooden_pickaxe:1 → cobblestone:14 → crafting_table:1 → stone_pickaxe:1

Start → log:6 → planks:24 → crafting_table:1 → stick:8 → wooden_pickaxe:1 → cobblestone:14 → crafting_table:1 → stone_pickaxe:1 → furnace:1 → iron_ore:3 → iron_ingot:3 → crafting_table:1 → iron_pickaxe → Termination

I Recommendation template generated by HCI module

1. Stone_pickaxe(without basic action recommendation):
    - ”Welcome to the game! I’m your personal AI assistant, I will give you suggestions to help you play this wonderful game! Now try to collect 6 log.”
    - ’Now you can collect 24 planks, trust yourself! In order to do that, you have to do action ”craft planks”.’
    - ’Now you can collect 24 planks, trust yourself! In order to do that, you have to do action ”craft planks”.’
    - ’Next, I suggest you to find 8 stick item(s). In order to do that, you have to do action ”craft stick”.’
    - ’Good job, now perhaps try to get 1 wooden_pickaxe. In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft wooden_pickaxe”.’
    - ’Good job, now perhaps try to get 1 wooden_pickaxe. In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft wooden_pickaxe”.’
    - ’Now you need to obtain 14 cobblestone item(s). In order to do that, you have to do action ”equip wooden_pickaxe”.’
    - ’Next, you need to get 1 stone_pickaxe item(s). In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft stone_pickaxe”.’
    - ’Congrats! You have finished the process!’
2. For Stone_pickaxe with basic action recommendation, the whole recommendation will be like the following form:
    - 1). welcome sentence(if current stage is first time output recommendation) 2). Subtask and special action recommendation sentence. 3). Basic action recommendation sentence.
    - some templates of basic action recommendation:
    - Next, I suggest you go forward. Next, I suggest you turn your camera left. Next, I suggest you attack.
3. Iron_pickaxe:
    - ”Welcome to the game! I’m your personal AI assistant, I will give you suggestions to help you play this wonderful game! Now try to collect 6 log.”
    - ’Now you can collect 24 planks, trust yourself! In order to do that, you have to do action ”craft planks”.’
    - ’Now you can collect 24 planks, trust yourself! In order to do that, you have to do action ”craft planks”.’
    - ’Next, I suggest you to find 8 stick item(s). In order to do that, you have to do action ”craft stick”.’
    - ’Good job, now perhaps try to get 1 wooden_pickaxe. In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft wooden_pickaxe”.’
    - ’Good job, now perhaps try to get 1 wooden_pickaxe. In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft wooden_pickaxe”.’
    - ’Now you need to obtain 14 cobblestone item(s). In order to do that, you have to do action ”equip wooden_pickaxe”.’
    - ’Next, you need to get 1 stone_pickaxe item(s). In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft stone_pickaxe”.’
    - ’Next, I suggest you to find 1 furnace item(s). In order to do that, you have to do action ”nearbyCraft furnace”.’
    - ’Try to collect 3 iron_ore from now on. In order to do that, you have to do action ”equip stone_pickaxe”.’
    - ’Collect 3 iron_ingot item from now on! In order to do that, you have to do action ”place furnace”, then ”nearbySmelt iron_ingot”.’
    - ’Try to collect 1 iron_pickaxe from now on. In order to do that, you have to do action ”place crafting_table”, then ”nearbyCraft iron_pickaxe”.’
    - ’Congrats! You have finished the process!’