Playing FPS Games With Environment-Aware Hierarchical Reinforcement Learning

Shihong Song[*] , **Jiayi Weng**[*] , **Hang Su** , **Dong Yan** , **Haosheng Zou** and **Jun Zhu**[†]
Institute for AI, Tsinghua University
Department of Computer Science and Technology, Tsinghua University
Beijing National Research Center for Information Science and Technology
Tsinghua Laboratory of Brain and Intelligence Lab
Center for Intelligent Connected Vehicles and Transportation, Tsinghua University
{songsh15,wengjy16,zouhs16}@mails.tsinghua.edu.cn; {suhangss,dcszj}@tsinghua.edu.cn;
sproblvem@gmail.com

Abstract

Learning rational behaviors in First-person-shooter (FPS) games is a challenging task for Reinforcement Learning (RL) with the primary difficulties of huge action space and insufficient exploration. To address this, we propose a hierarchical agent based on *combined options* with intrinsic rewards to drive exploration. Specifically, we present a hierarchical model that works in a manager-worker fashion over two levels of hierarchy. The high-level manager learns a policy over options, and the low-level workers, motivated by intrinsic reward, learn to execute the options. Performance is further improved with environmental signals appropriately harnessed. Extensive experiments demonstrate that our trained bot significantly outperforms the alternative RL-based models on FPS games requiring maze solving and combat skills, etc. Notably, we achieved first place in VDAIC 2018 Track(1)[1].

1 Introduction

First-person-shooter (FPS) games, e.g. Doom, provide an important yet challenging benchmark for deep Reinforcement Learning (DRL) [Li, 2017]. Different from the classic arcade games [Cook and Colton, 2011], FPS games have 3D graphics with partially observable states, which is a more realistic environment to study DRL. In general, FPS games directly simulate reality as humans perceive it (in a first-person point-of-view) and set novel, competitive goals such as navigation and shooting for agents, the overall managing of which is much desired for general intelligence. The availability of well-designed game environments (e.g., ViZDoom competition[2]) also accelerates the development of DRL algorithms [Lample and Chaplot, 2017; Wu and Tian, 2016].

[*]The authors contributed equally to this work.

[†]J. Zhu is the corresponding author.

[1]http://vizdoom.cs.put.edu.pl/competitions/vdaic-2018-cig/results

[2]Many participants submit pre-trained agents to fulfill a specified task and compete with multiple adversaries.

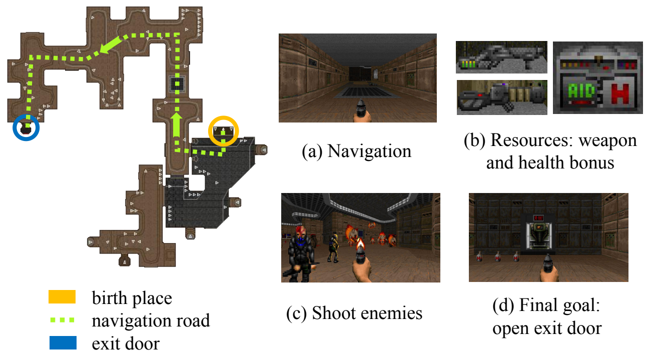
birth place
navigation road
exit door
(a) Navigation
(b) Resources: weapon and health bonus
(c) Shoot enemies
(d) Final goal: open exit door

Figure 1: Example of single-player scenarios in Doom (an FPS game). Left: a 2D map of this game. The green line denotes the fastest way to complete the game. Right: A wide range of cognitive abilities are generally required for an agent, including but not limited to navigation in a complex 3D environment (a) to open the door and accomplish the game (d), spotting and quickly aiming at multiple enemies (c), collecting resources when necessary (b), and team-based collaboration in some game modes. It may also introduce additional challenges of extracting visual information when only pixels on the screen are allowed. Demos are available at https://github.com/Trinkle23897/ViZDoom2018-Track1.

However, FPS games pose great challenges to DRL as an FPS agent is expected to orient and move in a 3D environment, collect resources when necessary, as well as locate and destroy multiple adversaries, as shown in Figure 1.

Most of the existing works are within the framework of end-to-end model-free DRL along with a CNN network, which learns to play directly from raw pixels by interacting with the game environments [Wu and Tian, 2016]. Although it can achieve human-like behaviors in basic scenarios, it still exhibits a multitude of challenges, especially the delayed rewards and long-term credit assignment as they usually receive rewards after a long sequence of actions. The problem is usually tackled by reward shaping [Lample and Chaplot, 2017; Dosovitskiy and Koltun, 2017] or curriculum learning [Wu and Tian, 2016]. However, it is generally non-trivial to design an appropriate reward shaping function or learning curriculum, which makes these methods usually hard to generalize to other tasks.

Moreover, FPS games often have a vast action space due to the compositional nature of the actions, that is, several actions can be executed at the same time, e.g., moving around to avoid being attacked when shooting enemies. Most of the existing works for FPS games simplify the problem by forcing one action to be chosen at a single time step, for example, DoomNet [Kolishchak, 2018]. However, the unwarranted assumption may sacrifice the performance significantly. More recently, [Huang *et al.*, 2019] designs temporary-extended macro-actions to reduce the action space and trains Q-Learning networks over these actions, but this method still relies on substantial human design and reward shaping.

Finally, FPS games require the agent to play in a 3D environment, which introduces another challenge of extracting relevant information from raw pixels. Without a proper understanding of this information, the agent may experience quick death because of the enemy attack in Figure 1 (c), or cannot beat enemies due to unavailable weapons in Figure 1 (b). It, therefore, requires an agent to recognize the resources, enemies and be aware of the depth information to implement spatial reasoning about 3D environmental structures.

1.1 Our Proposal

To address the above issues, we formulate the decision process of FPS games as a combination of a Semi-Markov decision process (Semi-MDP) [Sutton *et al.*, 1999] and several MDPs. This formulation leads to a scalable hierarchical model that consists of a two-level hierarchy in a manager-worker fashion jointly optimized, following the terminology definitions of the manager and workers in [Vezhnevets *et al.*, 2017]. The top-level manager decides options (subgoals), and the low-level workers learn policies of the options. The workers are, therefore, intrinsically motivated, allowing them to explore new behaviors without the need to consider the overall task.

Technically, to alleviate the issue of reward delay and long term credit assignment, we use a hierarchical model combined with an intrinsic reward for each worker according to its utility, drawing inspiration from [Kulkarni *et al.*, 2016a]. The manager which lies in the center of the network, aware of the environmental information, learns a policy over subgoals or options [Sutton *et al.*, 1999] to maximize the extrinsic reward and then dispatches the workers to fulfill the subgoals accordingly. With the manager reasoning at a coarse temporal resolution, its learning process can be much accelerated.

To address the issue of huge action space, we take into account the different characteristics of the primitive actions and divide the action space into a few subspaces within which each worker acts. The action subspaces between workers are disentangled and orthogonal such that different workers can execute actions simultaneously, forming *combined options* across workers, contrary to the conventional mutually exclusive options in most existing works.

Finally, to make a more general awareness of the 3D environment, we introduce an environment-aware block to better extract and utilize environmental information. This block is a compressed representation of environmental semantic information, which is composed of enemies, resources as well as depth estimation. The extraction of this information is achieved by using detection networks, such as Yolov3 [Redmon and Farhadi, 2018], which facilitates the design for the intrinsic reward.

Taking Figure 1 as an example: we separate the whole task into four workers corresponding to four sub-tasks according to the orthogonality of the action space, i.e. motion for navigating, collecting resources, attacking for keeping safe, and using tools for opening the final door. When a combined option terminates, the manager chooses a new combined option which is composed of several basic options formed by the workers' policies.

Furthermore, our method with action division, combined options and environmental awareness on a hierarchical RL structure tackles the underlying challenges of all such single-agent multi-task environments as FPS games correspondingly and could be easily transferred to other single-agent multi-task scenarios, for example, domestic robots, mineral mining robots, etc. All of these potentially supported bots need to finish a sequence of, possibly concurrent, subgoals to complete their tasks. It's not hard to transfer our method to other tasks with the same general framework and redesign of the low-level workers.

Extensive experiments on FPS games show that our proposed method enables the FPS agent to converge to a reasonable policy, which significantly outperforms the alternative RL-based models with substantial evidence that we won the champion in the recent international competition. Our main contributions are as follows:

- We propose a novel hierarchical manager-worker agent acting with *combined options* to reduce the action space of FPS games. The combined options, together with our proposed intrinsic rewards, alleviate the original reward sparsity.
- We incorporate environmental awareness into our agent with detection networks and depth estimation for a better understanding of the 3D environment.
- We empirically validate our methods through extensive experiments, and won the first place in the ViZDoom 2018 competition Track.

2 Related Work

Training an agent in FPS games has been extensively studied recently. Facing with the sparse reward and reward delay problem, F1 [Wu and Tian, 2016] uses curriculum learning, while Arnold [Lample and Chaplot, 2017] uses reward shaping and Curiosity [Pathak *et al.*, 2017] uses intrinsic rewards, which introduce human-designed features and other signals to accelerate the convergence of the network. To use more compact representations of the 3D environment, [Alvernaz and Togelius, 2017] learns an autoencoder to compress high dimensional 3D data into low dimensional implicit but effective representation, [Yan *et al.*, 2019] reformulates the decision process in FPS games into a relational MDP problem which enables a compact representation of MDP and accelerates the learning speed, and [Bhatti *et al.*, 2016] uses SLAM to incorporate the structures and objects encountered. To make better temporal abstractions, [Kulkarni *et al.*, 2016b] extracts subgoals periodically and learns a network to satisfy these goals. There are other methods using different formulations of the problem, e.g., [Dosovitskiy and Koltun, 2017] which uses a supervised learning model, predicting the difference of the future measurements and current measurements to learn the action of the current step.

Hierarchical reinforcement learning has become an active research area for introducing various abstractions into problem-solving and planning. Inspired by the primarily work of [Dayan and Hinton, 1993], feudal hierarchical model [Vezhnevets *et al.*, 2017] presents an end-to-end network in a master-worker fashion which automatically recognizes goals for decision and makes long-term credit assignment more tractable; MaxQ architecture [Dietterich, 1998] decomposes the whole tasks by decomposing the value function; [Sutton *et al.*, 1999] introduces "options", a hyper action which the architecture chooses to act until it terminates, where each option contains a policy which can be learned or prefixed by a network; Following this work, [Bacon *et al.*, 2017] proposes a framework capable of learning policies of options and termination conditions together without other rewards and subgoals; Hierarchical controller learning [Van Hoorn *et al.*, 2009] applies hierarchical reinforcement learning architecture to FPS games such as Unreal Tournament 2004.

3 Methodology

In this section, we present a two-level hierarchical reinforcement learning architecture for FPS games. We start with an overview of this architecture, followed by the design of the manager and workers; we then incorporate environmental awareness and finally present the overall algorithm.

3.1 Overview

Considering the hierarchical nature of FPS games, we design our architecture consisting of a manager in the center and a few outer workers, which is a star-shaped network named **StarNet** in this paper. Concretely, the manager decides abstract sub-goals, or options [Sutton *et al.*, 1999], at a coarse temporal resolution for the workers. The workers in the outer network then execute primitive actions to fulfill the sub-goals received from the manager. Inspired by how humans play the game, we divide the primitive action space to facilitate *simultaneous* options instead of the usual mutually-exclusive ones. We further design intrinsic rewards for workers and incorporate environmental awareness, e.g., depth info to drive exploration.

Formally, the decision process of an agent in FPS games is formulated as a Semi-Markov decision process (Semi-MDP) [Sutton *et al.*, 1999] $M_o = (S, O, T, R_E, \gamma)$ over several basic MDPs $M_i = (S, A_i, T_i, R_i, \gamma)$, corresponding respectively to the manager and workers. Here S denotes the state space, A_i is the action subspace of M_i, T and T_i are respectively the transition probability of the Semi-MDP and MDP M_i. The manager maximizes the cumulative extrinsic reward R_E (e.g., game score) w.r.t. its policy over options

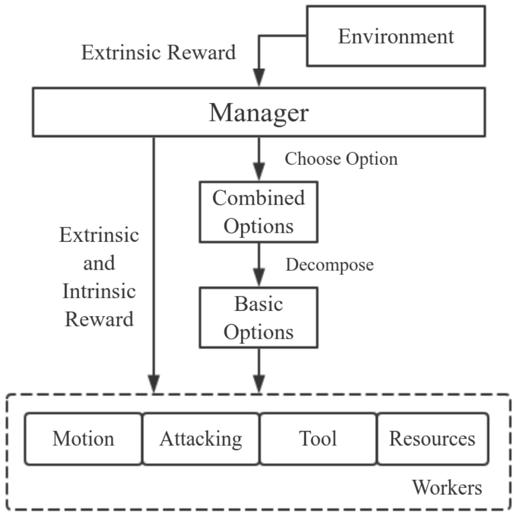

Figure 2: Our proposed StarNet architecture.

O with the discount factor γ. Oppositely, the workers maximize the cumulative reward of their own, R_i, w.r.t. its desired functionality, e.g., killing enemies, collecting resources, etc. Different from the manager, the workers are additionally motivated by the intrinsic rewards specified by the manager.

In particular, the manager chooses options at a lower temporal resolution, in which an option $o =< \pi, \mathcal{I}, \beta >$ denotes an extended behavior [Sutton *et al.*, 1999]. An option combines a policy $\pi(s, a)$ with a termination condition $\beta(s)$ and an initiation set $\mathcal{I} \subseteq S$. An option is available once $s_t \in \mathcal{I}$, and executes according to its policy π until the termination condition is satisfied. An option is selected conditioned on the different states, and the corresponding intrinsic reward is given to the workers to execute the option. The temporal abstraction of option largely alleviates the reward delay and the long-term credit assignment issue.

Moreover, our model consists of a few workers to achieve different subgoals specified by the manager. By considering the different characteristics of the primitive actions for FPS games, we mainly design a set of workers based on the orthogonality of the actions. In other words, different workers can execute their actions simultaneously without affecting each other. In this paper, we divide the action space into four subspaces: action subspace for motion, attacking, tools, and resources. The workers learn their policies at a higher temporal resolution, acting according to options selected by the manager. In this case, each worker maintains a policy on a subspace of the whole action space given the state, which largely reduces the action number for inference and accelerates the exploration significantly.

To make a more comprehensive understanding of the environment, we use the recent network of Yolov3 [Redmon and Farhadi, 2018] to detect the entities, including the enemies, resources, etc. We further use a regression model to estimate the depth signal using the visual feature extracted by the CNN network. All this environmental information facilitates the design of the intrinsic reward, yielding a more reasonable and stable policy.

Figure 2 concisely describes our proposed two-level hierarchical reinforcement learning architecture. The high-level manager, receiving extrinsic reward from environment, chooses from the available combined options for some of workers to execute and dispatches intrinsic rewards plus extrinsic rewards to these workers; the low-level workers, selected by the manager, act according to their learned policies and update their strategies based on the received intrinsic and extrinsic rewards.

3.2 Manager Design

In our two-level hierarchical network, the manager acts as a high-level decision maker lying at the center of the StarNet. It constructs a set of intermediate goals for workers to achieve by maintaining a policy over options, which alleviates the reward delay and long term credit assignment issues.

Notably, the decision process of the manager is formulated as a Semi-MDP process over a fixed set of options. Given state s_t, the manager constructs available option set: O_s. With a network constructed over all the options, the manager can obtain a probability over available options using the current state as input. The manager samples an option according to the probability calculated until the option terminates and chooses a new option. Despite the function of choosing options, every time choosing an option, the manager sends the intrinsic rewards separately together with extrinsic rewards to each chosen worker. For instance, when the option of combining attacking and motion is selected, the intrinsic reward for motion plus the extrinsic reward are given to the motion worker, and the intrinsic reward for attacking plus the extrinsic reward is given to the attacking worker.

In this work, we adopt Advantage Actor Critic (A2C) [Mnih *et al.*, 2016; Dhariwal *et al.*, 2017] to train the manager, while in principle any off-the-shelf deep RL algorithm is applicable. We parameterize the manager as a two-head neural network with *option* output $\pi_m(o_t|s_t)$, value output $V_m(s_t)$ and overall weights θ_m. Its objective is defined as:

$$\max_{\theta_m} A_m(s_t, o_t)\log\pi_m(o_t|s_t) - (R_{E,t} - V_m(s_t))^2 + \alpha H(\pi_m), \tag{1}$$

where $R_{E,t} = \Sigma_{i=t}^{\infty}\gamma^{i-t}r_{E,i}$ is the cumulative *extrinsic* reward $r_{E,i}$ until episode termination, $A_m(s_t, o_t) = R_{E,t} - V_m(s_t)$ is the advantage value scalar without gradient computation, and α is the coefficient for the entropy bonus $H(\pi_m)$.

3.3 Worker Design

To reduce action number, we note that some of the actions in the whole action space are conflicting, i.e., they cannot be chosen simultaneously at a single step. So we design workers to be an option whose policy is a probability on a subspace of the whole action space given the state. Workers work on orthogonal subspaces. The design of workers naturally endows a semantic disentanglement of utilities between workers. The termination condition and initiation set of each option related to the worker are designed according to the semantic utility of the worker. To alleviate the sparse reward problem and accelerate the training process, we design intrinsic rewards to each worker according to the utility split.

Similarly as the manager, we also adopt A2C for the workers, with the notable difference that the neural nets have *action* output $\pi_{w_i}(a_t|s_t)$ instead of option, and their objectives incorporate both extrinsic and intrinsic rewards:

$$\max_{\theta_{w_i}} A_{w_i}(s_t, a_t)\log\pi_{w_i}(a_t|s_t) - (R_{E,t} + R^i_{I,t} - V_{w_i}(s_t))^2 + \alpha H(\pi_{w_i}), \tag{2}$$

where $R^i_{I,t} = \Sigma_{j=t}^{\infty}\gamma^{j-t}r^i_{I,j}$ is the cumulative *intrinsic* reward $r^i_{I,t}$ of worker i and w_i denotes each worker. We include both rewards following [Vezhnevets *et al.*, 2017].

Action Space Division and Combined Options

The action space for FPS games is quite large, with various available actions to choose from. Directly learning policies by modeling the probability over all possible actions is difficult when the action number is quite large. However, reductions in action number can be done because of the conflicts existing between actions. To exploit this property, we divide the action space into several subspaces, where each worker is responsible for acting in one subspace. Formally, we divide $A = A_1 \times A_2 \times \cdots \times A_N$, where A is the whole action space, $\times$ denotes Cartesian product, A_i donates the action subspace for worker i, where actions inside A_i are mutually exclusive and N donates the number of workers. Note that from our definition on the Cartesian product, each A_i is a *split* of the original action vector with smaller dimensionality, but we still call it *subspace* from now on in this paper. With the division of the action space, the policy of the agent can be obtained from a hierarchical perspective: First, we choose the option o according to the manager's policy $\pi_m(o|s)$. Then, we act according to the policy according to o.

The general design of workers can vary from games to games, tasks to tasks. Here we give some common workers that are needed for most FPS games:

(1) Motion worker corresponds to the option of controlling the movement of the player.

(2) Attacking worker corresponds to the option of enabling the shooting of the player.

(3) Tool worker corresponds to the option of using tools in the game, for example, using a key to open the door.

(4) Resource worker corresponds to the option of collecting resources, for example, picking up guns on the ground.

These workers could be treated as options themselves if we construct action vectors by filling the sub-vectors not specified the worker with `no-op`. Conventionally, options are chosen *mutually-exclusively* by the manager. However, FPS game experts usually play in a multi-task fashion with the simplest example of turning around while shooting. Therefore, we propose to combine those workers:

Definition 3.1. ***[Combined Options]*** *Let $K \subseteq \{1, \cdots, N\}$ be a set of workers with action subspaces A_k and options $o_k =< \pi_k, \mathcal{I}_k, \beta_k >$ respectively. The combined option, $o_c =< \pi_c, \mathcal{I}_c, \beta_c >$, formed from K is defined as:*

$$\textit{Policy function: } \pi_c(a_c|s) \triangleq \prod_{k\in K}\pi_k(a_k|s), \tag{3}$$

$$\textit{Initiation set: } \mathcal{I}_c \triangleq \bigcap_{k\in K}\mathcal{I}_k, \tag{4}$$

$$\textit{Termination condition: } \beta_c(s) \triangleq \max_{k \in K} \beta_k(s), \qquad (5)$$

where a_c is constructed by filling a_k in corresponding action sub-vectors and filling $no{-}op$ in the other action sub-vectors.

Action probabilities are naturally the product of all included probability in K. Initiation sets are the intersection of the chosen workers' initiation sets since the option can only start if all initiation conditions are satisfied. Termination conditions are the max probability over the chosen workers' termination conditions since if there exists a worker with a termination probability of 1, the combined option should also be terminated. Combining N workers this way by enumerating all possible subsets of $\{1, \cdots, N\}$, we get 2^N combined options for the manager to choose from. In practical FPS games, $N = 4$, so the exponentiality is not an issue.

Intrinsic Reward

To alleviate the reward delay and sparse reward problem, we combine the hierarchical structure with intrinsic rewards designed with utilities of workers, which largely accelerates the exploration efficiency of each worker. Taking the environmental information into account, the intrinsic reward for each worker is designed as follows:

Motion. Motion worker is responsible for the agent to walk around the map and turn around. The intrinsic reward should motivate the agent not only to explore sufficiently but also assist the remaining workers, for instance, turning around to shoot enemies accurately. So the intrinsic reward for motion also contains the following components: positive reward for shooting correctly and negative reward for shooting to a wrong direction in scenarios $|\mathrm{En}| > 0$, where En denotes the enemies in the image, a positive reward for walking through resources or tools.

Collecting Resources. Positive reward for picking up resources while moving through resources, where Re denotes the resources in the image, $a = 0$ donates the action is not selected, and $a = 1$ donates the action is selected.

$$R_{\text{resources}}(a) = \begin{cases} 1, & \text{if } |\mathrm{Re}| > 0 \text{ and } a = 1 \\ -1, & \text{if } |\mathrm{Re}| > 0 \text{ and } a = 0 \end{cases}$$

Using Tools. Positive reward for using tools correctly, for example, opening doors with keys.

$$R_{\text{tools}}(a) = \begin{cases} 1, & \text{if use tools correctly and } a = 1 \\ -1, & \text{if use tools incorrectly and } a = 0 \end{cases}$$

Attacking. Positive reward for shooting in scenarios $|\mathrm{En}| > 0$, Negative reward for shooting when $|\mathrm{En}| = 0$.

$$R_{\text{attack}}(\mathrm{En}, a) = \begin{cases} 1, & \text{if } |\mathrm{En}| > 0 \text{ and } a = 1 \\ -1, & \text{if } |\mathrm{En}| = 0 \text{ and } a = 1 \end{cases}$$

3.4 Environmental Awareness

To grasp a better understanding of the 3D environment, we design an environmental block that can extract environmental information from raw pixel inputs. What environmental information we need is related to each FPS games according to human understanding of the game. Generally, we need the following components for environmental information: enemy En, resource Re, depth De, etc.. In this paper, we use Yolov3 [Redmon and Farhadi, 2018], a fast recognition network, to detect enemies and resources, which enables the agent to realize fast reaction.

Algorithm 1 StarNet Training

```
Input: Initial policy for manager and workers $\pi_m$, $\pi_{w_i}$ and initial value function for manager and workers $V_m$, $V_{w_i}$
Output: Learned policy $\pi_m$, $\pi_{w_i}$
  for $i = 0, 1, 2, \cdots, max\_episode$ do
    repeat
      Manager constructs available option set $O_s$ given current state $s$
      Chooses an option $o_i$ using current policy
      repeat
        Act according to policy in $o_i$, record state $s_t$, action $a_t$ and reward $r_{E,t}$ from game engine and $r^i_{I,t}$(intrinsic reward) from manager
      until termination condition satisfied
      Update the parameters of the workers w.r.t. Eqn. 2
    until episode ends
    Update the parameters of the manager w.r.t. Eqn. 1
  end for
```

Unlike tasks in the 2D scenario, navigation in a 3D scenario focuses more on whether the scene has been seen before, which is much harder than it in the 2D scenario. Going straight forward may not gain any new information in 3D scenarios, which can even bring the agent into the corner of the room, making it hard to go out. So it is essential to extract depth information from the raw image inputs to guide the agent. To address this issue, we trained a vision network to classify if the agent is dashing towards a wall. For each step taken by the agent, we get the observation of the current frame and the depth estimation and use this network to check if the agent is dashing towards the wall. If it is, we heuristically guide it to turn a random degree while also introducing an intrinsic reward, which largely enhances exploration efficiency and makes the agent less likely to dash towards the wall when facing it.

3.5 Learning

The optimization of our architecture involves optimizing workers at a lower level and optimizing the manager at a higher level. The manager and workers are jointly optimized with the workers optimized every time an option terminates (so in practice the workers' cumulative rewards are bootstrapped from value heads) and the manager optimized every time an episode ends, as summarized in Alg. 1.

4 Experiments

In this section, we first introduce the experimental setup and analyze the training efficiency. We then investigate how each part of StarNet influences the overall performance, and finally demonstrate our results in the latest ViZDoom competition.

Agent	Level 1			Level 2			Level 3			Level 4			Level 5			Average		
	PMAT	AMAT	PR	PMAT	AMAT	PR	PMAT	AMAT	PR	PMAT	AMAT	PR	PMAT	AMAT	PR	PMAT	AMAT	PR
DoomNet	46.13	134.98	65%	40.44	105.33	75%	85.96	150.17	70%	67.64	125.73	75%	79.96	167.98	60%	63.53	136.84	69%
StarNetNoRI	54.19	54.19	**100%**	49.25	49.25	**100%**	43.57	43.57	**100%**	50.72	50.72	**100%**	42.69	56.06	95%	48.14	50.76	99%
FeUdal	70.86	254.17	20%	114.37	262.87	20%	88.56	225.99	35%	118.59	245.88	30%	191.66	289.17	10%	106.77	255.56	23%
StarNetNoDepth	74.72	212.10	40%	114.75	199.16	55%	57.76	216.80	35%	64.89	254.86	20%	**18.41**	246.69	20%	76.40	225.92	34%
StarNetNoExit	103.96	103.96	**100%**	74.63	120.15	80%	73.83	108.66	85%	81.46	92.38	95%	96.18	96.18	**100%**	86.95	104.27	92%
StarNet	**41.50**	**41.50**	**100%**	**30.49**	**30.49**	**100%**	**31.33**	**31.33**	**100%**	**34.59**	**34.59**	**100%**	27.43	**27.43**	**100%**	**33.06**	**33.06**	**100%**

Table 1: ViZDoom Single Player Task Result

All experiments are conducted on the ViZDoom platform [Kempka *et al.*, 2016] with PyOblige[3] map generator. Video demos and further experimental details can be found at https://github.com/Trinkle23897/ViZDoom2018-Track1.

4.1 Experiment Setting

We focus on the single-player game as Figure 1, where the agent's goal is to solve the maze to navigate to the exit while keeping alive by killing hostile monsters and collecting health, similar to the competition track we entered (Sec. 4.4).

Training Data

We use PyOblige to generate 100 maps at five different difficulty levels[4], yielding 20 maps per level as our training set. These maps have lots of resources together with scary monsters, posing significant challenges for the agent to complete the game. The higher the number of configuration, the harder for the agent to finish the game.

Parameters

In each experiment, the learning rate for the manager network is 10^{-3} with batch-size 32. The motion worker adopts the SNAIL [Mishra *et al.*, 2018] network architecture, with learning rate 10^{-2} and batch size 32. All the other workers' architectures are the same as the manager.

Metrics

We use the following metrics for evaluating:

- **PMAT (Passed Maps' Average Time)** which denotes the agent's average time consuming on its passed maps.
- **AMAT (All Maps' Average Time)** which denotes the agent's average time consuming on all of the maps.
- **PR (Pass Rate)** which denotes the probability of this agent to finish this level's game.

An agent is better if it completes maps at a higher rate (**PR**) within a shorter time (**PMAT** and **AMAT**). If it fails to finish a game, the time is recorded as the maximum limit 300s.

4.2 Efficiency of Training Process

For the stability of training, we first use the intrinsic reward to train each worker separately at different scenarios as an initialization before the joint-training process to let them converge to a more stable policy. The training procedure is terminated after 600 epochs.

[3]https://github.com/mwydmuch/PyOblige/

[4]https://github.com/mwydmuch/PyOblige/blob/master/pyoblige/wad_configs.py

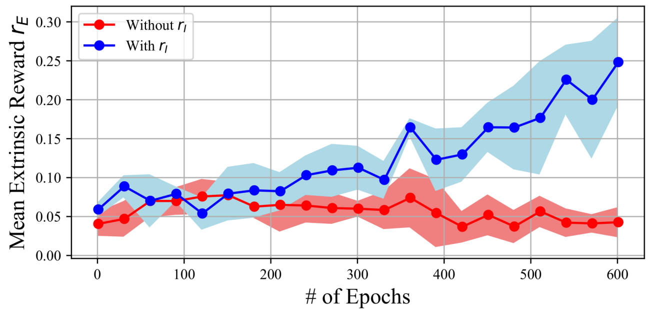

Figure 3: The StarNet training curve: we compare the mean extrinsic reward r_E each epoch it receives during training. StarNet can learn a reasonable behavior more effectively with intrinsic reward r_I.

Figure 3 presents the training process. The extrinsic reward is a large positive reward when the agent successfully finishes the game, and a small negative reward when the agent dies or the game is timeout. From this figure, we can draw the conclusion that with intrinsic reward r_I appropriately set, our StarNet improves its performance quickly in the training maps. We also investigate the policy of the manager and find it tends to choose options related to its observation, like activating the resource worker when noticing a weapon. There are two interesting things we'd like to mention: (1) At the beginning the average extrinsic reward it receives is positive, due to the pre-trained workers contribute to some of the reasonable behaviors, regardless of the random policy of the manager. (2) We early-stop the training process at about 600 epochs because its performance would drop down quickly after the peak.

4.3 Performance Evaluation

For evaluation, we generated another 100 maps using the same difficulty configuration as mentioned above. Table 1 shows all the performance of the agents on these 100 maps, where StarNet achieves the best score among these agents. Compared with other agents, it can finish all of the unseen games in the most efficient way. We present a detailed analysis in terms of different aspects as follows.

Two-Level HRL vs Non-Hierarchy Structures

To find out how the hierarchical architecture contributes to the overall system, we compare our agent with **DoomNet** [Kolishchak, 2018] and **StarNetNoRI** with StarNet. DoomNet is trained end-to-end by PPO [Schulman *et al.*, 2017] and won second place in ViZDoom 2018 Track(1). It has complex

Team	Ours	DoomNet	VIPLAB	ddangelo
Total Time (min)	**25.34**	29.86	31.54	37.33
# of Best Record	**8**	4	3	4

Table 2: VDAIC 2018 Track(1) Competition Result

reward-shaping functions and takes a long time to converge. StarNetNoRI is a version without intrinsic reward through joint-training.

On one hand, even if DoomNet achieves an excellent result on these unseen maps with over 60% pass rate, it always performs poorly and is soon killed when facing a large number of enemies as the difficulty level increases. In contrast, our manager will cleverly bypass these enemies to avoid being attacked. Thus the two-level hierarchical architecture is necessary for solving long-term decision-making problems. On the other hand, without intrinsic reward, the StarNetNoRI can achieve nearly 100% pass rate owing to the pre-trained worker, but using intrinsic reward in the joint-training process can shorten StarNet's pass time without loss of pass rate, which primarily enhances its performance as is illustrated in Figure 3. Therefore, the intrinsic reward is crucial to alleviate the sparse reward and reward delay problem.

Divided vs Whole Action Space

To investigate how action space division contributes to the overall performance improvement, we compare our agent with the most recent HRL framework of **FeUdal** [Vezhnevets *et al.*, 2017]. The reward is the same as the sum of StarNet's intrinsic and extrinsic reward, and it receives the StarNet's environmental block's information as part of the input. However, it has a more extensive action space of size $2^7 = 128$ from the combination of 7 orthogonal actions: move left/right/forward, turn left/right, attack, and open the door.

The vast action space brings FeUdal's performance with an unsatisfied result of $\sim$20% pass rate. However, using our action space division method, even the StarNetNoRI could reach a satisfying result. Thus, the action space division make great contributions to the overall performance significantly.

Aware vs Non-Aware of Environment

To analyze the function of environment-aware block, we compare agent **StarNetNoDepth** and **StarNetNoExit**. StarNetNoDepth is a version without a depth perception network. The motion worker cannot receive useful information about the depth information. StarNetNoExit is a version without "Exit Label" perception. This label will appear on the halfway to the final goal.

Without exit label perception, the agent StarNetNoExit still has more than 90% probability of finishing the game. However, the average pass time is nearly three times longer than the StarNet's. Besides, the performance of StarNet drops dramatically without depth perception. The results further prove that the comprehension of environments largely contributes to the overall performance.

4.4 ViZDoom Competition

We took part in ViZDoom AI Competition 2018 Track(1)[5] to evaluate our proposed StarNet performance. In this competition, all of the participants submitted an agent to finish the game on ten unseen and more difficult maps. Our method achieved first place in both public-rank and private-rank leaderboards. Table 2 summarizes a part of the official results, where our agent achieves the best total time among these maps. Moreover, it gets 8 of 10 rounds of the best record. These impressive results again prove the effectiveness of our proposed method.

5 Conclusion

In this paper, we propose StarNet, an environment-aware hierarchical reinforcement learning model to play FPS games. The designed hierarchical structure with intrinsic rewards alleviates the sparse reward and reward delay problems. Further, the action spaces division enhances the performance through combined options together with a proper understanding of the 3D environment. Experimental results show that this architecture enables our agent to play significantly better than other methods in FPS games.

Acknowledgements

This work was supported by the National Key Research and Development Program of China (No. 2017YFA0700904), NSFC Projects (Nos. 61620106010, 61621136008, 61571261), Beijing NSF Project (No. L172037), DITD Program JCKY2017204B064, Beijing Academy of Artificial Intelligence (BAAI), Tiangong Institute for Intelligent Computing, NVIDIA NVAIL Program, and the project from TOYOTA.

References

[Alvernaz and Togelius, 2017] Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *IEEE Conference on Computational Intelligence and Games*, 2017.

[Bacon *et al.*, 2017] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[Bhatti *et al.*, 2016] Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N Siddharth, and Philip HS Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

[Cook and Colton, 2011] Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *IEEE Conference on Computational Intelligence and Games*, 2011.

[Dayan and Hinton, 1993] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NeurIPS*, 1993.

[5]https://www.crowdai.org/challenges/visual-doom-ai-competition-2018-track-1

[Dhariwal *et al.*, 2017] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[Dietterich, 1998] Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, 1998.

[Dosovitskiy and Koltun, 2017] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *ICLR*, 2017.

[Huang *et al.*, 2019] Shiyu Huang, Hang Su, Jun Zhu, and Ting Chen. Combo-action: Training agent for fps game with auxiliary tasks. In *AAAI*, 2019.

[Kempka *et al.*, 2016] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, 2016.

[Kolishchak, 2018] Andrey Kolishchak. Doomnet. https://github.com/akolishchak/doom-net-pytorch, 2018.

[Kulkarni *et al.*, 2016a] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, 2016.

[Kulkarni *et al.*, 2016b] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[Lample and Chaplot, 2017] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, 2017.

[Li, 2017] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[Mishra *et al.*, 2018] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[Pathak *et al.*, 2017] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[Redmon and Farhadi, 2018] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Sutton *et al.*, 1999] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

[Van Hoorn *et al.*, 2009] Niels Van Hoorn, Julian Togelius, and Jurgen Schmidhuber. Hierarchical controller learning in a first-person shooter. In *IEEE Conference on Computational Intelligence and Games*, 2009.

[Vezhnevets *et al.*, 2017] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.

[Wu and Tian, 2016] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *ICLR*, 2016.

[Yan *et al.*, 2019] Dong Yan, Shiyu Huang, Hang Su, and Jun Zhu. Learning to assign credit in reinforcement learning by incorporating abstract relations. In *AAAI Workshop on Reinforcement Learning in Games*, 2019.