Reinforcement Learning with Large Language Model playing First Person Shooting Game

Tzu-Chieh Wei , Shi-Xin Fang , Shih-Yun Shang Kuan , Jui-Chiang Wei
National Taiwan University

Abstract

In this study, we introduce LLM to training FPS (First Person Shooting) game, in which a popular FPS game called Vizdoom we are using. Our approach commences with pre-trained skills, enabling agents to master complex maneuvers. The main part of our method focuses on improving the "manager network," which uses these trained moves. Instead of just learning from the game's rewards, our manager also learns better tactics from LLM (Large Language Models), making the gameplay smarter and more diverse. We found out that when using LLM , there's a significant improvement in the agent's performance. Moreover, as the influence of LLM decreases, the agent's performance tends to decrease as well.

1 Introduction

Learning diverse tasks in game worlds with aid of LLM is a significant milestone toward building capable agents. Recent studies in reinforcement learning with LLM have achieved great success in many domains like games and robotics. Among them, Minecraft, a popular open-world game with an infinitely large world size and a huge variety of tasks has also been successfully implemented with LLM reinforcement learning [1]. Because of the step by step task in Minecraft, it makes it an advantage using LLM to provide strategy or continuous tasks.

Therefore, we wonder if LLM can also provide useful informations for FPS (First Person Shooting) games such as Vizdoom [2], which also requires some smart "strategies". Previous works usually build policies in Vizdoom upon hierarchical learning [3]. However, in the landscape of FPS games, reinforcement learning agents often exhibit limited and static strategies, primarily focusing on maximizing enemy eliminations without showcasing tactical retreats or cover maneuvers. Addressing this challenge, our research delves into harnessing the "common sense" capabilities of large-scale language models (LLM) to engender more intricate and human-like strategic approaches within these agents. By infusing RL training with nuanced insights from these models, our objective extends beyond mere kill optimization to fostering intelligent tactics akin to human gameplay. Our contribution is, to our best knowledge, the first attempt to use LLM to train an agent on Vizdoom. Our results demonstrate a significant improvement in training efficiency.

Towards our goal, we first separately train our pre-trained skills, such as navigation, shooting, take cover, gathering health and gathering ammo. All trained in the same environment so as to decrease the uncertainty of different structure of maps. After training every skills we need, we start to train the manager, which is an high level policy deciding which skill should the agent take now. However, we found out that it's very difficult to train out a smart agent. Therefore, we introduce LLM to our training. LLM returns the sequence of skills, and adding reward to the manager if it takes the action suggested by LLM. This way, the manager can somehow learn how the LLM deals with different situations, making it much more smarter.

However, LLMs struggle to produce the correct action code consistently in one shot . To

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

address this challenge, we propose a two steps prompting mechanism that: (1) executes the generated program to obtain detailed observations from the game screen (2) use the detailed observations obtained previously as prompt and make GPT-4 generate a more reasonable suggestion of action sequences.

In conclusion, we successfully train out an agent with better performance using LLM comparing to the agent only uses hierarchical learning. We use the count of killed enemy and reward as method to determine the performance of an agent.

2 Related Work

2.1 Playing FPS with agent

Training an agent in FPS games has been attempted by early works. Clyde [4] applies the A3C [5] algorithm and uses an LSTM [6] network to create a basic intelligent agent in ViZDoom. To address the challenges of sparse rewards and reward delays, Arnold [7] uses reward shaping, while F1 [8] additionally uses curriculum learning and won the championship of ViZDoom AI Competitions 2016. StarNet [3] achieved first place in VDAIC 2018 Track(1) by using a hierarchical agent based on combined options with intrinsic rewards and a detection network. The Fittest Wins (TFW)[9] develops a multi-stage training framework using the Adaptive Strategic Control (ASC) algorithm combined with hindsight experience replay (HER)[10], achieving the SOTA result in the open ViZDoom AI Competition.

2.2 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL), involving multiple layers of policies for decision-making at varying levels of abstraction, has long held promise for solving challenging tasks. In a hierarchical structure, where only the lower-level directly interacting with the environment, higher-level can be trained to plan over longer time scales. This approach abstracts complex problems into simpler subtasks. During the past years, HRL has become an active research area and been shown to outperform standard RL in several long-horizon problems such as continuous control [11, 12], long-horizon games [13, 14], robot manipulation, [15] and so on. Besides, the application of HRL to FPS games [3] has already demonstrated significant success.

2.3 LLMs with RL

Large language models such as GPT-3 [16] have been used to build agents capable of acting in interactive environments based on language instructions. [17] Recent works started focusing on using Large Language Models (LLMs) to guide agents' policies. SayCan [18], BOSS [19] and Code as Policies [20] used LLMs as high-level planners. Their LLM does not directly interact with the environment. Ahn et al. [18] had to use an external affordance function to generate feasible plans that guide an agent. Another work combines LLM with hierarchical reinforcement learning. Prakash et al. [21] instead of completely relied on LLMs, they used LLMs to guide a high-level policy, making learning significantly more sample efficient. Our work is closely related to [21].

3 Problem formulation

3.1 Environment

VizDoom is a research platform that integrates the classic video game Doom into a reinforcement learning framework. It provides a diverse environment with complex maps, adversaries, and scenarios, challenging agents to navigate, combat enemies, and accomplish various tasks within a first-person shooter setting.

The map we are using is called "D3_battle." It consists of darker scenes with relatively more corners and corridors compared to typical terrains. This design benefits the GPT model in understanding the structure of the visuals and generating more appropriate prompts. When viewed from a top-down perspective, as shown in figure 1 3.1, it resembles a maze-like structure. Throughout

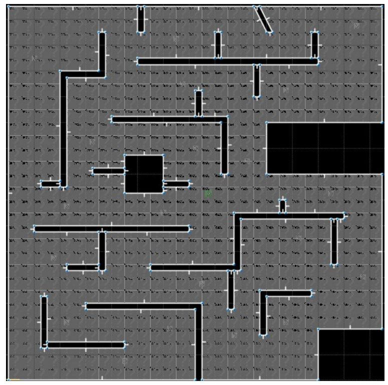

Figure 1: Top Down Image

Figure 2: Game Screen

the map, ammunition and health packs are scattered, and new supplies periodically appear to facilitate gameplay. The adversaries are creatures commonly found in shooting games, and they also spawn periodically, gradually increasing the game's intensity. The goal of this game is to kill as many enemies as possible, while stay alive as long as possible.

3.2 Agent Perspective

As a FPS game player, you have several actions at your disposal: MOVE_FORWARD, MOVE_RIGHT, MOVE_LEFT, TURN_LEFT, TURN_RIGHT, TURN_RIGHT, and attack. These actions can be combined and executed simultaneously. A straightforward approach would be to generate all possible combinations of these actions and represent each combination as a single action. However, this method becomes challenging as the number of possible combinations grows exponentially with the number of available buttons. Even with just six buttons, you'd have 2^6 = 64 potential outputs.

To address this combinatorial explosion in our action space, we opt to restrict certain combinations. For instance, activating both TURN_RIGHT and TURN_LEFT simultaneously doesn't make logical sense. Thus, we eliminate the following combinations: TURN_RIGHT and TURN_LEFT, as well as MOVE_RIGHT and MOVE_LEFT. Additionally, we restrict the ATTACK button to be used singularly. By implementing these constraints, we streamline our action set down to 18 distinct actions, a significant reduction from the potential 64 combinations.

Regarding the observation space, we adopt a multi-input method, defining it as a dictionary structured as follows: "image": Represents the game screen and is constrained by dimensions transformed from the original (3, 240, 320) to (100, 156, 3) 3.2. "vector": Contains game parameters like health and ammunition counts. We then concatenate the two vectors into one long vector and pass it to policy.

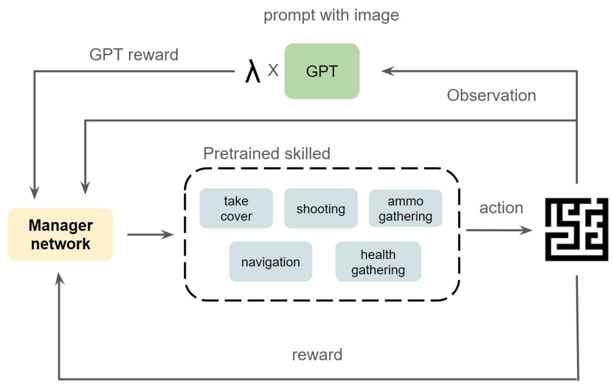

Figure 3: frame work of HRL with LLM

4 Method

4.1 Pretrained Skill

We divide our strategy into following skills: navigation, shooting, take cover, gather health, gather ammo, and I'm going to introduce how it works and how we train it one by one below.

(1) Navigation

Navigation, as the name suggests, allows our agent to navigate smoothly on the map. The action space utilized includes the actions mentioned earlier: MOVE_FORWARD, MOVE_RIGHT, MOVE_LEFT, TURN_LEFT, TURN_RIGHT, and TURN_RIGHT in various combinations. Among them, we also employ multiple reward shaping techniques to facilitate easier training. Firstly, since survival is paramount, we set the death penalty to 2, ensuring that the agent avoids contact with monsters as much as possible.

Furthermore, we utilize VizDoom's built-in label buffer and depth buffer to determine various objects and depths in the current scene. If the entire front is filled with walls, it signifies that the agent is continuously hitting the walls, thus imposing it as one of the penalties. As a result, our agent can navigate in the map smoothly without hitting any walls.

(2) Shooting

To train shooting in such a challenging environment, it's difficult to directly incorporate the entire action space into the training process. Therefore, we adopt an approach similar to Hierarchical Learning. Initially, we load a previously trained navigation model from a file and use it to create a function for detecting enemies. Specifically, when an enemy is detected, the model immediately switches to a shooting training mode. Through this method, we can train more distinct skills, making it easier to apply them in generating prompts for subsequent LLM tasks.

(3) Take cover

The 'take over' approach utilizes parameters within VizDoom that allow us to determine the current angle of the agent. Using these parameters, we can turn the agent to the desired direction using the 'turn_angle' function. With this function in place, when using LLM prompting, we can directly instruct the agent to provide recommendations on how many degrees to turn. Consequently, the agent can then move to a specific angle, achieving the objective of taking cover.

(4) Gather health/ammo

The skills 'gather health' and 'gather ammo' focus on collecting health packs and ammunition as rewards. By training, we can develop an agent specialized in navigating the map to collect health packs for healing or specifically replenishing ammunition. Compared to the previous skill, training for this particular skill can be achieved more effortlessly.

4.2 Manager

The framework of the manager is shown in Figure 3 4 as we can see, the manager takes two inputs including the observation and the GPT reward simultaneously. The observation is consist of a dictionary formed from the game screen, health status, and ammunition count, which will be concatenated into a vector in feature extractor. Our intention is for the manager to consider the current state of the game player based on not only the game screen, but also the players condition, thereby deciding whether to replenish health, gather ammunition, or engage in combat.

As for the GPT reward, we incrementally reward the manager each time it chooses an action consistent with the LLM strategy. This gradual approach ensures the manager learns progressively. Additionally, to prevent the manager from overly relying on the LLM strategy, we introduced an additional reward gamma. This means that the rewards for learning from the LLM decrease over time until they eventually reach zero.

Then, the manager chooses an action within the previously trained skills: navigation, shooting, take cover, and gather health or ammo, totaling five actions. Given its role at the highest policy level, we haven't employed extensive reward shaping methods. Instead, we have only incorporated the reward of enemy kills.

4.3 Large Language Model

4.3.1 Prompt

(1) Problems

Pricing : 0.03 $US/1k tokens, which is about 0.5 NTD/prompt(of 350 words), a single training of 10000 episodes would cost at least 5000NTD even if the agent is only prompted once per episode.

Inference time : GPT4-vision-preview inferencing a 150 words prompt with an image input of 240 * 320 is of average 20 seconds, which means the whole training process will have to halt and wait for the prompt, provided that we are using on policy algorithm PPO.

GPT restrictions : GPT will not response normally when it considers the prompt as involving any kind of violence or copyright infringement. Also it is restricted to not give gaming advice, so if it perceives the prompt as asking for gaming advice, it would respond abnormally too.

(2) Our solution

Sample from environment before training : To solve both problems of high budget of prompting while training and long inference time, we came up with the method of sampling the observation space (which is mainly the game image), with each image sampled representing a unique state in the game. Then we preprocess their prompting prior to training, thus the inference time won't be added on to the training time, and also we sample 100 images, which greatly lowers the cost compared to sampling 10000 or more images. Later during training, if an enemy is detected and it is required to prompt the game image to get a GPT plan in order to apply an additional reward, we simply just classify the current game image to be most identical to which of the 100 sampled images, assuming they are in similar states and can share the same reward.

Prompt twice : Prompting directly with GPT4 with vision is a more effective way, assum-

Table 1: Result

level	Without GPT($\lambda = 0$)	With GPT($\lambda = 0.01$)	With GPT($\lambda = 0.1$)
level 1	18.3	20.4	20.1
level 2	7.4	12.5	12.3

ing it is more budget saving since it can simply output the plan consisting of 5 actions in a format of only integers and it is the most robust model of the GPT family, this is our first straightforward but failed assumption. Analyzing the environment then planning the strategy is a bit too much a task for the model, its output was not stable in quality. Therefore we tried to lower the difficulty of the task by prompting twice, the first prompt 6 requesting for a detailed description of the environment in the image 6, and the second prompt 6 inputs the first output's description and requests GPT3.5 to formulate a strategy 6. The output is way more stable than prompting only once, and GPT4 can give a really detailed description (although not always 100% correct), and GPT3.5 can understand it then providing us with a feasible strategy.

4.3.2 Reward Shaping

(1) Strategy alignment : We hope the agents predictions of actions can align to GPT's advice (strategy). For example : if GPT thinks that attacking while the health is above a threshold is the best action, we want the agent to do the same. Thus we formulate the GPT additional reward by the whether the agent acts the same as GPT's strategy, while their is no punishment if it does not follow. We found that a strategy of 5 actions is its limit, it almost always assume the agent would eliminate the enemies in 5 actions, so for plans more than 5 actions, GPT would advice the agent to wander around which is not something we want to see.

(2) Advice Sample frequency : Sampling when away from battle is meaningless in our condition because it would need previous data to decide how to navigate, but our pre-sample method doesn't provide memory for GPT, so it is not possible for GPT to make really long horizon plans(e.g. how to navigate the whole map and hunt enemies down) in this case. So we would only activate the GPT reward when the agent encounters enemies (this would also be an reward for the agent to seek the enemies, since there are chances of receiving reward from GPT if any enemies are found and GPT rewards are enabled).

5 Experiment

5.1 Experiment Setting

We focus on the single-player game, where the agent's goal is to kill as many enemies as possible within the step limit while keeping alive by killing monsters, collecting health and collecting ammo.
Training Data
We use the 'D3-battle' maps as mentioned in part 3 and incorporate a step limit of 2100 to prevent excessively prolonged training.
Parameters
In each experiment, the learning rate for the manager network is set to 0.0001 with a batch size of 2. The surrogate loss is optimized every 3 epochs, and the environment is updated every 16 steps. The pretrained skills use a larger batch size and update steps, specifically 32 and 4096, while the learning rate remains unchanged.

5.2 Performance Evaluation

For evaluation, we utilize the same map as in training but with two different levels. Level 1 corresponds to the training level, albeit with an increased step limit of 10,000. Level 2 poses a greater challenge to the player, featuring more enemies and fewer resources.

Table 2: Result

level	100 steps	1000 steps	2000 steps	3000 steps
level 2	11.4	12.35	12.6	9.1

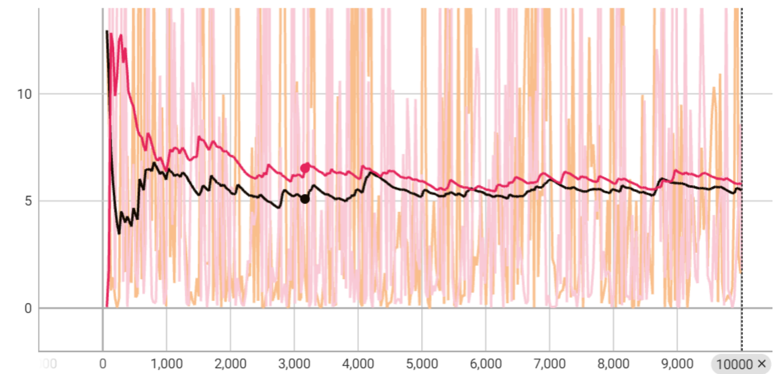

Figure 4: training loss

Table 1 1 compares hierarchical architectures without LLM and those with different GPT rewards, all trained for 2000 epochs, with the model achieving the highest reward selected. The pretraining skills chosen are navigation and shooting, two commonly used skills. Across maps of different levels, it is evident that the addition of GPT rewards significantly increases the number of enemy kills. This indicates that the suggestions provided by GPT are highly beneficial for manager training.

Converting the game screen into a GIF to observe the actual situation, agents without GPT assistance tend to choose the same skill, resulting in frequently shooting when there are no enemies or easily missing enemies. On the level 1 map, the kill count of the manager without GPT is only slightly lower than that with GPT, as these agents can generally survive beyond the step limit of 10,000 steps on this map. In the level 2 environment, a more noticeable difference among three agents is observed. Additionally, we found a positive correlation between the number of kills and survival time, leading to significant variability in each episode. Therefore, evaluating the results' quality requires visual inspection of the actual gameplay.

In addition, the reward for GPT decreases linearly within 1000 steps. In other words, GPT only provides rewards in the early stages of training, so the presence of GPT assistance does not make a significant difference in training time.

Table 2 2 compares the training results over 10,000 episodes with different rates of GPT reward decay. It can be observed that the effectiveness of GPT reward is achieved rapidly with minimal impact, and even with a slow decay, the results are comparable. However, from Figure 4 5.2 , it is evident that a longer decay leads to a smoother decline in loss. Yet, it is important to note that a longer decay does not necessarily result in a loss. Additionally, it is observed that incorporating GPT reward significantly increases the difficulty of convergence. In the end, we attempted to train a manager capable of controlling four pretrained skills: navigation, shooting, health gathering, and ammo. As shown in Table 3, the results were not as promising as those achieved by agents with only two skills. However, there was significant improvement in survival time. We hypothesize that the inclusion of these two defensive skills might have reduced the inclination towards aggressive behavior. Additionally, it is possible that the plan of the GPT did not take into account player conditions such as health and ammunition, leading to generated recommendations that do not clearly distinguish between the timing of using navigation and using these two skills.

6 Conclusion

LLMs can generate effective strategies from game screens through a two-stage prompt. In addition, prompting GPT4 before training greatly reduce the training cost. The addition of GPT generated reward is very helpful for manager training.Experimental results show that this architecture enables our agent to play significantly better than other methods in FPS games.

Besides the above improvement of the agent, our research can also serve as a case of generating expert data by LLMs to apply in regions that expert data isn't sufficient (DOOM game is quite old and does not have a large player community, while Minecraft has lots of resources and large player community's support), and imitate learning may possibly be applied with the LLMs generated gaming/task data.

References

[1] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[2] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2016.

[3] Shihong Song, Jiayi Weng, Hang Su, Dong Yan, Haosheng Zou, and Jun Zhu. Playing fps games with environment-aware hierarchical reinforcement learning. In *IJCAI*, pages 3475–3482, 2019.

[4] Dino Stephen Ratcliffe, Sam Devlin, Udo Kruschwitz, and Luca Citi. Clyde: A deep reinforcement learning doom playing agent. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[5] 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 *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[8] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *International Conference on Learning Representations*, 2016.

[9] Shuxing Li, Jiawei Xu, Honghua Dong, Yu Yang, Chun Yuan, Peng Sun, and Lei Han. The fittest wins: a multi-stage framework achieving new sota in vizdoom competition. *IEEE Transactions on Games*, 2023.

[10] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[11] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[12] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[13] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[14] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[15] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[17] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[18] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[19] Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. *arXiv preprint arXiv:2310.10021*, 2023.

[20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[21] Bharat Prakash, Tim Oates, and Tinoosh Mohsenin. Llm augmented hierarchical agents. *arXiv preprint arXiv:2311.05596*, 2023.

Appendix

First prompt

"This image is a screen of a gameplay in the first person shooting game 'DOOM', containing information of its location and the environment and the enemies. Based on the specific situation shown in the image, please give me a detailed description of the environment the agent is seeing now (no need to describe the environment behind the agent), and please focus more on describing 1.the wall's and obstacle's position and orientation 2.enemy's position(if there is any) 3.item's position(please list them all if there is any, and there are 2 kinds of items available : (a)health package, a white box chararcterized with a bit of red on its right and green on its left, (b)ammo, a brown colored shell that is relatively smaller in size), also please use the the angle notation of left or right theta degrees to the agent to specify the directions."

First prompt sample output

"This image showcases a scene from the game 'DOOM,' where the agent, seen in the first-person view, is within a gray-walled environment with a relatively dark ambiance. 1. Walls and obstacles:- Directly ahead, there is a continuous stone wall with an alternating block pattern, which extends beyond the right edge of the view.- To the left about 45 degrees from the center of the agent's field of view is the corner of a wall, indicating a passage or opening to another area outside the immediate field of view. 2. Enemy's position:- There is an enemy at a location roughly to right 30 degrees from the center of the agent's field of view. It appears to be a 'Cacodemon,' a floating, monstrous creature associated with the 'DOOM' franchise. The enemy is located at a moderate distance, just off the right side of the wall that extends from the view.3. Item's position:- No items resembling health packages or ammo are visible in the field of view shared in the image.the limitations stated, this is all the information that can be drawn from the environment as presented in the picture."

Second prompt

"The below text enclosed in brackets is the description of the screen of a gameplay in the first person shooting game 'DOOM', containing information of its location and the environment and the enemies. First prompt's output Based on the specific situation described, please imagine you are a pro player and form a plan of 5 consecutive actions listed below to do at this moment so that you can optimize in navigating through the map and eliminating enemies. Please perceive that all enemies are able to deal long range attacks like shooting, so you need to consider the risk when collecting a health package in front of an enemy. Please output the number (an integer) and its direction (in degree format, let 90 degrees to the right be 0 and 90 degrees to the left be 180) that represents the action only, and further explanation is not needed to be included in your output. For example, the output should be like : [1(30),3(60),2,4,6]. Below are the actions you can choose from :1.Aim and shoot 2.Take cover behind the wall or obstacle in direction of 0, 30, 60, 90, 120, 150, 180 degrees (pick from this 7 directions) 3.Collect health package within sight 4.Collect ammo within sight 5.Navigate around the map"

Second prompt sample output

"[2(30),1,5,2(60),1]"