Deep Reinforcement Learning for First-Person Shooter Games

Ekansh Mahendru*
2018csb1087@iitrpr.ac.in
Dept. of Computer Science & Engg.
Indian Institute of Technology Ropar

Hansin Ahuja*
2018csb1094@iitrpr.ac.in
Dept. of Computer Science & Engg.
Indian Institute of Technology Ropar

ABSTRACT

Training agents for first-person shooter (FPS) games is a well-known problem in reinforcement learning. FPS games are particularly interesting as they often involve rewards for conflicting objectives making reward-shaping a key aspect for the performance of the agent. In this paper, compare the performances of three well-known deep reinforcement learning methods, namely DRQN, C51-DDQN, and A2C-LSTM. We also analyze the different interesting strategies chosen by each agent. We train the agents in a partially observable environment provided by the ViZDoom AI research platform, which creates a semi-realistic 3D environment by simulating the classic video game Doom.

KEYWORDS

deep reinforcement learning, first person shooter games

1 INTRODUCTION

First-person shooter (FPS) is a video game genre primarily involving a protagonist or agent navigating a set of objectives from a first-person perspective while engaging in firearm-based combat with other adversarial agents, which may or may not be controlled by other players or other AI-based policies. The objectives may involve navigating a map or a maze, avoiding attacks from other agents, collecting health or weapons, etc. The episode or game terminates when either the protagonist dies by losing all of its health points as a result of attacks by other agents or other environmental interactions, or when a player completes a major objective as defined by the environment.

The major challenges or characteristics of FPS games make them particularly interesting problems to solve using reinforcement learning. As a direct consequence of the first-person perspective of the protagonist, the only information available to the agent is what is within the field of view of the agent. Information about the environment outside this field of view may be equally or even more important, given that adversarial agents or environmental dangers may be situated physically behind the agent. This limited frame of reference can be modeled as partially observable states for reinforcement learning problems.

The state representation of these problems also calls for intelligent state-encoding techniques using computer vision and deep learning. The environment may provide us with interesting non-visual heuristics like distance from the end-goal, intangible environmental factors, etc. However, the bulk of the state information provided to the agent is through its visual perspective. Converting this visual information to a computationally comprehensible format such as vector representations calls for employing feature extraction techniques used widely in the fields of computer vision and deep learning.

We use the ViZDoom AI research platform to provide a 3D environment to our agent. The platform comes with different configurations involving the agent navigating various mazes with expressly different objectives, as we will detail further. We compare the results of different deep RL techniques in these different circumstances and also discuss the different factors which necessitate intelligent reward shaping for each of these environments.

2 RELATED WORK

Employing deep reinforcement learning techniques to ViZDoom configurational setups is a well-studied problem. Efforts have been made to efficiently encode state information and learn from the agent-environment interactions. Here, we go over some of the methods to address the partially observable environment setting in general, and also the ViZDoom environment in specific.

Jaakkola et al [1] proposed and analyzed a learning algorithm to solve a class of non-Markov decision problems. Their algorithm applies to problems in which the environment is Markov, but the learner has restricted access to state information.

Spaan [2] presents the partially observable Markov decision process (POMDP) model by focusing on the differences with fully observable MDPs, and shows how optimal policies for POMDPs can be represented. The paper is a review of model-based techniques for policy computation, followed by an overview of the available model-free methods for POMDPs.

Shao et al [3] propose a method that utilizes the actor-critic with Kronecker-factored trust region (ACKTR), a sample efficient and computationally inexpensive DRL method. We present an approach to sensorimotor control in immersive environments.

Dosovitskiy et al [4] propose an approach that utilizes a high-dimensional sensory stream and a lower-dimensional measurement stream. The cotemporal structure of these streams provides a rich supervisory signal, which enables training a sensorimotor control model by interacting with the environment. Their method successfully generalizes across environments and goals. A model trained

*Both authors contributed equally to this research.

CS533, Reinforcement Learning, Term Paper
IIT Ropar.

using the presented approach won the Full Deathmatch track of the Visual Doom AI Competition, which was held in previously unseen environments.

Chaplot et al [5] propose a model trained with deep reinforcement learning using an Action-Navigation architecture, which uses separate deep neural networks for exploring the map and fighting enemies. Furthermore, it utilizes a lot of techniques such as augmenting high-level game features, reward shaping and sequential updates for efficient training and effective performance.

3 DEEP REINFORCEMENT LEARNING

The goal of reinforcement learning is to maximize some return expression made up of the rewards at each time step. Formally, we express the return as:

$$G_t = R_t + \gamma * R_{t+1} + ... + \gamma^{T-t} * R_T \quad (1)$$

Here, gamma is the discounting factor used to avoid tremendous values in cyclic or non-finite state spaces. The expected value of this return starting from some state is known as the value of that state.

$$V(s) = E[G_t] \quad (2)$$

In practice, we do not estimate states' values. However, we estimate the value of a state-action pair, $Q(S, A)$, i.e., the expected return if the agent starts from a state S and takes an action A. Dynamic programming provides several solutions to estimate $Q(S, A)$, through exposure to the environment. However, sometimes, the number of state-action pairs is so large that it becomes infeasible to store all the values or even visit them. In these cases, we approximate the value function by using some state-dependent inputs and some parameters that change the function's shape.

$$Q(S, A) = f(x(S, A), \theta) \quad (3)$$

It makes sense to use neural networks as the approximating function f due to their high versatility and learning capacity. The use of Deep Neural Networks in Reinforcement Learning is known as Deep Reinforcement Learning.

3.1 Q-Learning

Q learning is a method to estimate the q-function using the Bellman optimality equation to calculate the loss/error.

$$loss = (Q(S_t, A_t) - R_t + \gamma * \max_{a'}(Q(S_{t+1}, a')))^2 \quad (4)$$

We can then use techniques such as stochastic gradient descent to minimize this loss value and hence, be as close as possible to satisfying the Bellman Optimality Equation.

3.2 Double Q-Learning

Q-learning suffers from the maximization bias, which is due to the use of overestimated values to calculate the target at the start of the learning, which delays the convergence. To tackle this problem, we create two value functions, one from which we are choosing actions and the other from which we calculate the estimated target. Periodically, we copy the first function to the second function.

$$loss = (Q_1(S_t, A_t) - R_t + \gamma * Q_2(S_{t+1}, \arg\max_a(Q_1(S_{t+1}), a))))^2 \quad (5)$$

3.3 Recurrent Q-Networks

Recurrent neural networks preserve the temporal relationships between the input sequence. These networks produce the required output and an internal state used for the next input in the sequence. We use these networks in reinforcement learning algorithms to encode states when the states are partially observable and time-dependent.

4 VIZDOOM

ViZDoom creates a semi-realistic 3D environment, and objectives vary from basic move-and-shoot tasks to more complex maze navigation scenarios. We must employ visual reinforcement learning techniques to navigate relatively realistic physics models in the ViZDoom environments [6]. The training is based on only the screen-buffer and other information available in the heads-up display of the player, like health points left, ammunition available, etc. The agent is rewarded by the game for both successful kills and finding the end to terminate the episode. The sooner the episode terminates, the fewer opportunities that the agent has to attack enemies. Hence, these intrinsic rewards are conflicting. This makes for interesting possibilities when it comes to constructing RL environments and the subsequent experiments with reward shaping. The ViZDoom environment has a lot of customisation capabilities, with users being able to define maps, rewards, goals, etc. In this paper, we look at two of these configurations.

(a) (b)

Figure 1: Screenshots from the (a) Defend the Center and (b) Deadly Corridor configuration

4.1 Defend the Center

The agent is fixed at the center of a circular hall where enemies appear at random times along the outer wall of the hall and are killed by a single shot at them. The objective of the agent is to prevent the enemies from reaching the center. Shots become more precise as the enemy gets closer. The death of the agent is inevitable as bullets are limited. So, the agent has to use each bullet wisely to increase its lifetime. We have ACTIONS: [TURN_LEFT, TURN_RIGHT, SHOOT] and GAME REWARDS: [+1 for killing an enemy, -1 for death].

4.2 Deadly Corridor

The agent spawns at one end of a corridor with three small circular rooms, each with two enemies, along the corridor. The objective is to reach the other end of the corridor where a vest is

placed which marks the successful end of the episode. This scenario adds an additional dimension to the previous scenario by allowing the agent to walk around and navigate. Survival along with a healthy level of exploration must be a priority for the agent. To ensure that the agent engages with the enemies instead of running straight for the vest, we increase the skill of the enemies to a level at which it is not possible for the player to obtain the vest by simply ignoring the enemies (doom level = 5). In other words, the player will die if s/he simply runs towards the vest without killing the enemies. We have ACTIONS: [MOVE_LEFT , MOVE_RIGHT, SHOOT, MOVE_FORWARD, MOVE_BACKWARD, TURN_LEFT, TURN_RIGHT] and GAME REWARDS: [Previous Distance from the Vest - Distance from the Vest, -100 for death].

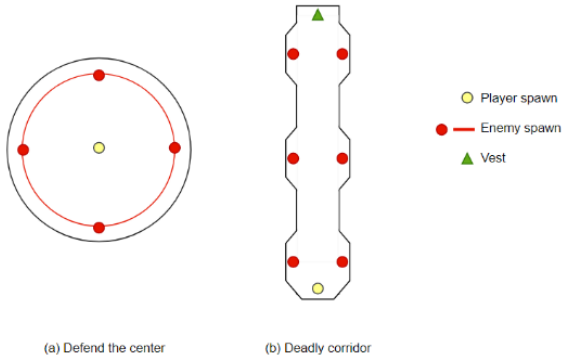

Figure 2: Maps for the two ViZDoom configurations used for experimentation in this paper

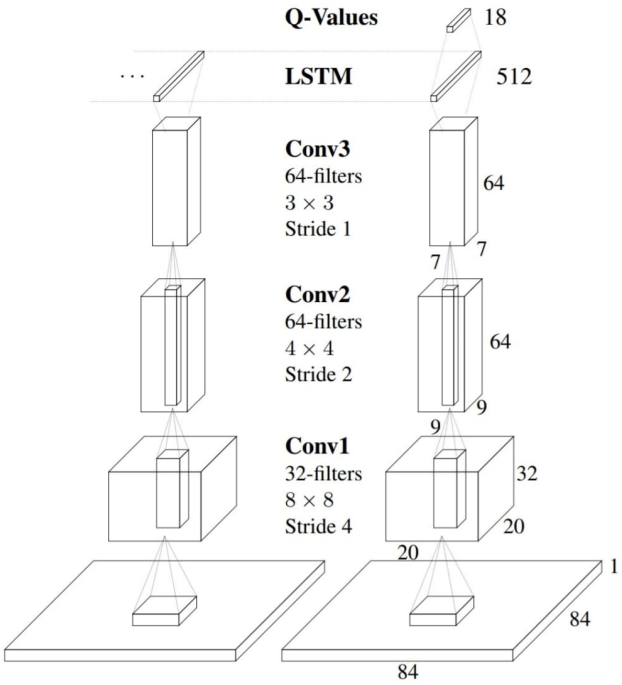

Figure 3: Architecture of the DRQN as proposed by Hausknecht et al

5 ARCHITECTURES

5.1 Deep Recurrent Q-Network

Hausknecht et al [7] proposed this architecture in 2015. The authors modify the traditional deep Q-network by replacing the fully connected layer with a recurrent LSTM layer. The authors aimed to inject recurrency and temporal context into the learning process. We modify the shapes and play around with the parameters of the network to suit it to the task at hand. The network accepts 4 stacked frames of dimensions 64x64x3, sends the input through 3 convolutional layers, a recurrent LSTM layer and then finally outputs a vector of Q-values for each action in the action space (3 for defend the center and 7 for deadly corridor). Additionally, we add a fully connected layer in between the LSTM layer and the final fully output layer. We also add some well-known RL techniques like experience replay [8] and double Q-learning [9].

5.2 C51 - Double Deep Q-Network

A distributional perspective on reinforcement learning was proposed by Bellemare et al in 2017 [10]. This is in contrast to the common approach to reinforcement learning which models the expectation of this return, or value. The idea is that an expectation of the return can lead to extremely bad results if the probability distribution which dictates this expectation is multi-modal in nature. Additionally, even unimodal distributions with different variances need to be evaluated in a distinct manner. However, the traditional Q-learning approach fails to identify any differences between such distributions. So, the modified distributional version of the recursive relation becomes:

$$Z(s, a) = r + \gamma * Z(s', a') \tag{6}$$

Here, $Z(s, a)$ denotes the probabilistic distribution of the return at the state-action pair rather than the expectation of it. The equality sign here represents two equivalent distributions. In order to construct computationally feasible loss functions, we discretize these distributions into 51 buckets, hence the name C51. The distance between two distributions is taken to be the cross-entropy loss, which can now be easily calculated given the discrete nature of distribution.

The network now involves an input layer of size 64x64x3, followed by 3 convolutional layers and a fully connected layer. Instead of one vector representing the Q-values of each action, we output a 51-dimensional vector for each action in the action space, representing the discretized probability distribution of each action.

Additionally, we employ experience replay and double Q-learning here as well.

5.3 Advantage Actor-Critic - LSTM

We experiment with a synchronous version of the A3C algorithm proposed in 2016 [11]. The traditional A2C algorithm suffers from not being able to track agent mobility in a dynamic environment and a version of the algorithm was proposed in 2020 which incorporates long short-term memory into the algorithm [12]. The network takes as input 4 stacked frames with dimensions 64x64x3, passes it through 3 convolutional layers and then an LSTM layer. The network then forks out into the actor stream and the critic stream, each of which is a single fully connected layer.

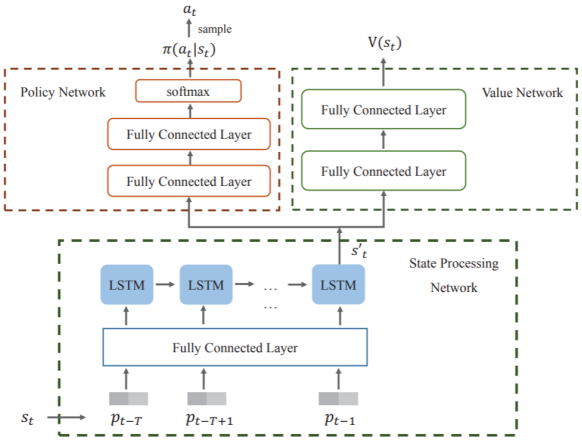

Figure 4: Architecture of the A2C-LSTM as proposed by Wang et al

6 EXPERIMENTS

First, we employ the 3 algorithms on the "defend the center" configuration. The skillfulness of the enemies is established by a variable called "doom skill," which we set to 3. Apart from the rewards returned to the agent by the ViZDoom environment itself (which we've discussed previously), we also provide a +1 reward for killing an enemy, and a reward of -0.1 for using ammunition or losing health points.

Secondly, we employ the algorithms on the "deadly corridor" configuration. As discussed earlier, doom skill is set to 5 to disincentive making a run for the vest by ignoring the enemies. We implement 3 reward shaping techniques. The first one exactly similar to the one employed in the previous scenario, the second one with the kill reward being increased to +10, and the third one as described by Boubnan and Ayman [13]. We divide the system reward by 5, award the agent +100 for every kill, -0.5 for using ammunition, -5 for losing any health points, and then finally divide the reward by 100. We rationalise these decisions in the next section.

7 RESULTS

7.1 Defend the Center

The average number of kills for 20 games is used as the measure of performance for each model. The results are plotted in Fig. 5. DRQN and C51-DDQN quickly learn to get 10+ kills in a couple

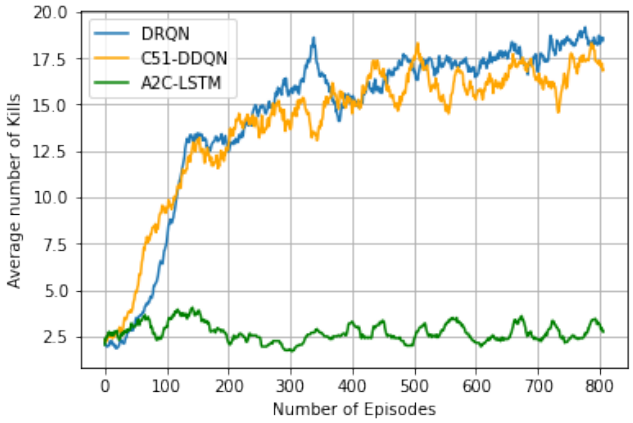

Figure 5: Average number of kills for each model after each episode in Defend the Center

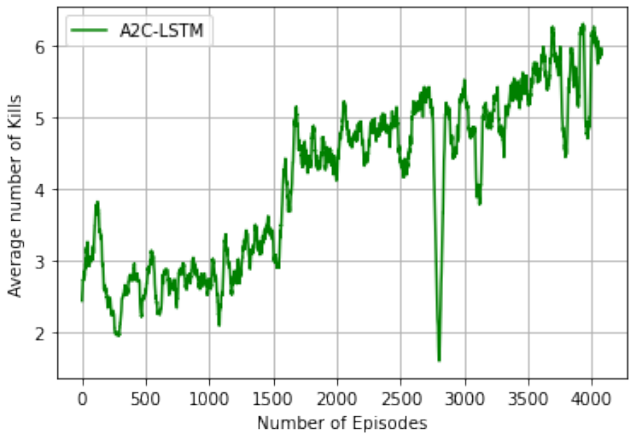

Figure 6: Average number of kills after extended training of A2C-LSTM in Defend the Center

hundred episodes. Although both these methods achieve comparable performance, DRQN mostly stays above the C51-DDQN model. Since, even after 800 episodes of training, A2C-LSTM did not show any improvements, we trained it further till about 4500 episodes and it was still able to achieve only 5-6 kills on average which is far less than what the other two models were achieving in a fraction of the number of training episodes. This can be attributed to the lack of experience replay and a less stable target due to updates in the same network which is used for target estimation in A2C-LSTM, making it a slow learner.

Strategy analysis:

DRQN learned to finish all the targets on the screen by moving in both directions to aim. The main weakness of this strategy is that enemies from behind can come as close as possible, if the agent takes more time to finish all the targets on screen, even if they are far. C51-DDQN model learned a similar strategy but did not learn how to shoot accurately making it waste ammo on mis-attacks.

This model often dies before DRQN by the lack of ammo. A2C-LSTM model chooses to move mostly in one direction and shoot only when the enemies are very near, reducing the buffer time to respond to other attackers. It faces difficulties whenever more than one targets are in close proximity.

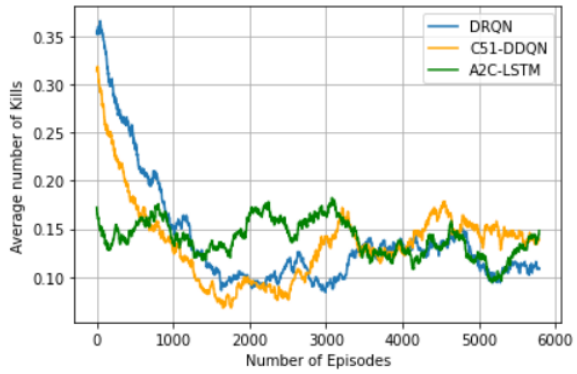

Figure 7: Average number of kills for each model after each episode in Deadly Corridor with more weightage to distance reward

7.2 Deadly Corridor with heavier intrinsic rewards

First, we train the algorithms on the exact same reward structure as for the previous scenario. We notice that none of the algorithms are able to perform well. The average number of kills consistently remains a fraction less than 1 for almost 6000 episodes, and no trend for convergence can be seen either. On visualising the agents' performances, it can be seen that all of them are making a run for the vest without attacking the enemies, and dying fairly quickly. We believe that this is because the agent receives a fairly heavy award for the distance that it travels as compared to the kills that it achieves. The agent doesn't have a compelling incentive to spend time killing the agents when it can achieve a significant reward by just closing the gap between itself and the vest.

Hence, we increased the kill reward to +10. We chose only a slight increase (as compared to our second increase), because we didn't want our agent's primary objective to be killing the enemies. The reward for procuring the vest still ought to be the highest reward it can achieve, and all the preliminary rewards must be designed to channel towards this outcome. However, the agent's policy remained the same with this change, i.e. it still ignored the enemies and made a dash for the vest.

7.3 Deadly Corridor with heavier kills

Finally, we implemented a substantially different reward structure. We realised that the agent will be free to procure the vest only if it kills the enemies. These weren't conflicting objectives, i.e. the agent need not settle priorities between covering distance and killing enemies. Killing the enemies was an action channelled towards achieving the final objective. In fact, it was absolutely necessary to achieve the final objective, given our parameter of doom skill = 5. So,

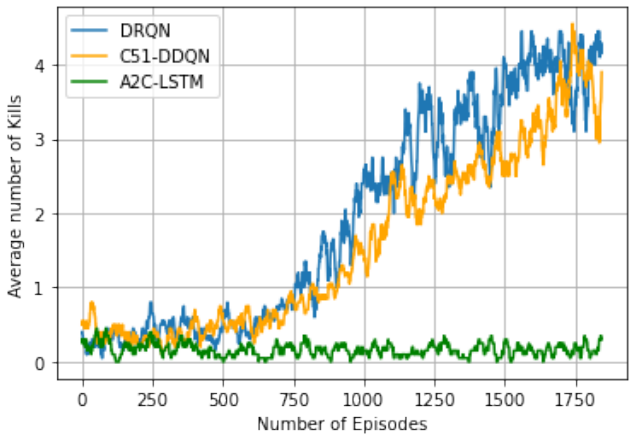

Figure 8: Average number of kills for each model after each episode in Deadly Corridor with more weightage to kill reward

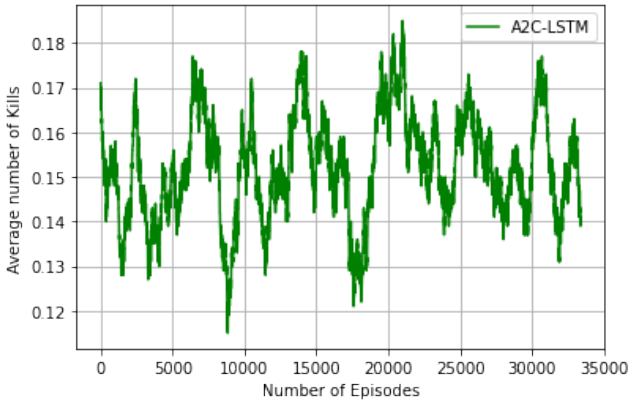

Figure 9: Average number of kills for A2C-LSTM after each episode in Deadly Corridor with more weightage to kill reward

we scaled down the distance reward by 5 to dwarf it in comparison to the additional kill reward, which we assigned the value of +100. Using ammunition was to be penalised, but not severely, as that would be a deterrent for achieving more kills. Losing health points was assigned an increased negative return of -5, because prolonged survival was a necessity for reaching the vest. Finally, we divide the reward by 100 to scale everything to the range of [-1, 1].

As we can see in the graphs, performance of the models is similar to their performance in the first experiment, i.e., Defend the Center scenario. DRQN and C51-DDQN achieve comparable performance with DRQN just slightly better. However, in this case, A2C-LSTM does not seem to learn from the environment but shows similar performance all throughout training even when trained till 35000 episodes (See Fig. 9).

Strategy analysis:

Both DRQN and C51-LSTM learned to shoot one enemy from the

blind spot of the previous room where the enemy cannot see the agent. This makes it much easier to enter the next room as the agent just has to kill one enemy without the fear of dying from a shot from the other one. A2C-LSTM did not learn that it is essential to kill enemies to reach the vest but it tries to avoid enemy fire by directly sprinting towards the vest which is not possible due to the increased Doom Skill of 5 in this scenario.

8 CONCLUSION

In this paper, we employ different deep reinforcement learning strategies to tackle the problem of navigating a first-person shooter game like Doom. We experiment with different reward shaping techniques and learn that some algorithms are able to generalise and perform better, and we try and put forth explanations for why that might be. These algorithms can be used as foundations for even more challenging configurations in the ViZDoom environment such as Deathmatch. We can also try and examine the applicability of these algorithms to more exploratory configurations (like "My Way Home") and more defensive configurations (like "Take Cover").

9 ACKNOWLEDGEMENT

Thanks to Felix Yu for his helpful Github repository [14]. We sourced a lot of code from his Keras implementations. Also, thanks to Dr. Shashi Shekhar Jha for helping us out throughout this course.

REFERENCES

[1] Jaakkola, Tommi, Satinder P. Singh, and Michael I. Jordan. "Reinforcement learning algorithm for partially observable Markov decision problems." Advances in neural information processing systems. 1995.
[2] Spaan, Matthijs TJ. "Partially observable Markov decision processes." Reinforcement Learning. Springer, Berlin, Heidelberg, 2012. 387-414.
[3] Shao, Kun, et al. "Learning battles in vizdoom via deep reinforcement learning." 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2018.
[4] Dosovitskiy, Alexey, and Vladlen Koltun. "Learning to act by predicting the future." arXiv preprint arXiv:1611.01779 (2016).
[5] Chaplot, Devendra Singh, and Guillaume Lample. "Arnold: An autonomous agent to play fps games." AAAI. 2017.
[6] Kempka, Michał, et al. "Vizdoom: A doom-based ai research platform for visual reinforcement learning." 2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2016.
[7] Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." arXiv preprint arXiv:1507.06527 (2015).
[8] Lin, Long-Ji. "Self-improving reactive agents based on reinforcement learning, planning and teaching." Machine learning 8.3-4 (1992): 293-321.
[9] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." arXiv preprint arXiv:1509.06461 (2015).
[10] Bellemare, Marc G., Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." arXiv preprint arXiv:1707.06887 (2017).
[11] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. 2016.
[12] Li, Rongpeng, et al. "The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility." IEEE Communications Letters 24.9 (2020): 2005-2009.
[13] Boubnan, Mehdi and Chaouki, Ayman. "Deep reinforcement learning applied to Doom." Github repository (2019). https://github.com/Swirler/Deep-Reinforcement-Learning-applied-to-DOOM
[14] Yu, Felix. "Implementation of Reinforcement Learning Algorithms in Keras tested on VizDoom." Github repository (2017). https://github.com/flyyufelix/VizDoom-Keras-RL