Deep Reinforcement Learning in VizDoom First-Person Shooter for Health Gathering Scenario

Dmitry Akimov and Ilya Makarov
National Research University Higher School of Economics
Moscow, Russia
deakimov@edu.hse.ru, iamakarov@hse.ru

***Abstract*—In this work, we study the effect of combining existent improvements for Deep Q-Networks (DQN) in Markov Decision Processes (MDP) and Partially Observable MDP (POMDP) settings. Combinations of several heuristics, such as Distributional Learning and Dueling architectures improvements, for MDP are well-studied. We propose a new combination method of simple DQN extensions and develop a new model-free reinforcement learning agent, which works with POMDP and uses well-studied improvements from fully observable MDP. To test our agent we choose the VizDoom environment, which is old first person shooter, and the Health Gathering scenario. We prove that improvements used in MDP setting may be used in POMDP setting as well and our combined agents can converge to better policies. We develop an agent with combination of several improvements showing superior game performance in practice. We compare our agent with Recurrent DQN using Prioritized Experience Replay and Snaphot Ensembling agent and get approximately triple increase in per episode reward.**

Keywords–Deep Reinforcement Learning; VizDoom; POMDP; First-Person Shooter.

I. Introduction

The 3D shooter is a video game genre where a player controls the virtual combatant and tries to achieve some predefined goal, such as make their way trough the maze or capture the flag or fight other players with a ranged weapon. There are several game modes in which players can compete or cooperate with each other. Usually, such games are very demanding on player skill, reaction and cleverness.

3D shooters are challenging task for Reinforcement Learning (RL) algorithms. In terms of reinforcement learning concept, we can describe a 3D shooter as sophisticated environment with one general goal, for example, to maximize kill-death ratio during one game session or episode, and with many simpler tasks, such as map navigation or enemy detection. Learning behaviour policy in 3D shooters is difficult: rewards are extremely sparse and usually highly delayed. It means that an agent may receive reward signal for action it performed hundreds frames ago. Also, information received by the agent from the environment is incomplete: enemies positions are unknown and angle of view is limited by 90-110 degrees. Maps can be complex mazes, and the navigation inside is challenging for the agent as well. The only information that Deep RL agent can use for action choice is a game screenshot (image captured from rendered scene).

We decided to focus on First-Person Shooter (FPS) videogame, which we have previously simulated and studied related to intelligent path planning [1] [2] and building RL agent for weapon choice [3].

In this work, we choose VizDoom [4] as simulation environment. There are many maps (scenarios) in VizDoom, each with different goals and gameplay features. We choose the Health Gathering scenario to teach an agent to detect health packs and navigate inside a room with acid on the floor.

We combine several existing improvements for RL agents to propose a new model-free approach for general RL problems. We conduct experiments proving effectiveness of the proposed agent. We show that our agent outperforms well studied baseline methods, as well as their modifications.

The paper is organized as follows. Related work for deep reinforcement learning is overviewed in Section 2. We introduce basic concepts in Section 3 and describe chosen scenario for VizDoom in Section 4. Then, in Section 5–7, we describe proposed approach, experiments and analyze obtained results. Finally, in Section 8, we make a conclusion and describe future work.

II. Related Work

A. Rainbow

In the Rainbow paper [5] authors combined several of the DQN extensions. Particularly, authors propose to use Double Learning, Dueling Architecture, Multi-step learning, Prioritized Replay, C51 and Noisy Networks for exploration all together. Their combined agent learn up to 8 times faster than simple DQN. Also authors investigate effects of combining all the five out of these methods.

However, they tested combined agent on the Atari games, which were implemented in The Arcade Learning Environment [6], in MDP manner, and, thus, effects of this combination in environments with incomplete information is unknown.

B. Arnold paper

In 'Arnold' paper [7], authors develop an agent to play a deathmatch scenario in VizDoom environment. Authors used several useful learning tricks. They implemented augmentation of the agent with in-game features, which are accessible during training, and enemy-detection layer, which greatly speeds up training and helps agent to converge to a good policy. Also, authors did reward shaping presented as small reward signals for good actions, which do not directly correspond to the main objective, separate networks for action and navigation for faster leanring, and added dropout layer after convolutions, preventing neural network from overfitting and making agent's policy more robust to previously unseen in-game situations. Their final agent substantially outperforms build-in AI agent of the game and, surprisingly, humans. However, the 'Arnold' agent is

 ISBN: 978-1-61208-697-2

relatively simple: authors did not exploit Q-learning extensions that could significantly improve agent's performance.

C. DRQN with Prioritized Experience Replay, Double Q-learning and Snapshot Ensembling

Prioritized Experience Replay (PER) [8] is a smart way to speed-up training. In this method, the examples for Experience Replay are sampled with non-uniform probabilities, assigned to each tuple $\langle s, a, r, s' \rangle$ or $\langle o, a, r, o' \rangle$ in case of POMDP setting, according to loss value on this example. Examples with higher error values carry more information than ones with low error value, so we prefer to sample them more often. This approach has shown its potential and superior performance compared to DQN in Atari games. Authors of [9] combined PER with Double Q-learning and Snapshot Ensembling and tested their agent in VizDoom Defend The Center scenario, in which the agent stand at certain point and rotate in order to shoot the enemies closing to the agent. Also, authors integrated an enemy detector and trained it jointly with Q-function. We compare our results with [9] below.

Deep Reinforcement Learning improvements for MDP achieved super-human performance in many games. However, this improvements had not been considered in POMDP setting before. We think that it is essential to combine such simple heuristics from MDP with POMDP to push up state-of-the art methods considering several scenarios for first-person shooter (FPS) according to VisDoom Deep RL competition.

III. Deep Reinforcement Learning Overview

General reinforcement learning goal is to learn optimal policy for an agent, which maximizes scalar reward by interacting with the environment.

At each time step t an agent observes the state s_t of the environment, makes a decision about the best action a_t according to its policy π and gets the reward r_t. This process well known as *Markov Decision Process* (MDP) denoted as a tuple (S, A, R, T), where S indicates a finite state space, A indicates a finite action space, R is a reward function, which maps pair $(s, a) \in (S, A)$ into stochastic reward r, and last but not least T is a transition kernel: $T(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a)$.

Discounted return from state s_t is expressed by the following formula:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i},$$

where $\gamma \in (0, 1)$ is a discount factor reducing the impact of reward on previous steps. Typically, the choice of γ depends on the length of the game. An agent with the greater values of gamma concentrates on dilated rewards, which is important in the long game sessions, and an agent with small gamma concentrates on short-term rewards.

In order to choose the actions, an agent uses a policy $\pi = P(a|s)$. We will call a policy π *optimal* if it maximizes expected discounted reward and mark it with a star:

$$\pi^* = \operatorname*{argmin}_{\pi} \mathbb{E}_{\pi}(G_t)$$

There may be several optimal policies as well.

In this work, we consider *Q-learning*, a *value-based* method, because of its popularity and effectiveness. All required information about the methods is presented below.

A. Q-learning

To measure the quality of a given policy π one can use action-value function Q^{π} defined as:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[G | s_0 = s, a_0 = a \right] \tag{1}$$

Q is expected over all possible action and states reward that can be obtained by the agent with the policy π started from state **s** and performed action **a** and then following its policy. If the true Q-function (Q^*) is given for us, we can derive optimal policy by taking action a that maximizes Q for each state s: $a = \operatorname{argmax}'_a Q(s, a')$

To learn Q for the optimal policy we use *Bellman equation* (1):

$$Q^{\pi}(s, a) = r(s, a) + \gamma \max_{a'} Q^{\pi}(s', a') \tag{2}$$

It is proven in [10] that this sequential assignment will converge from any given Q to desired Q^* eventually if action and state spaces are finite and each pair of state and action are presented repeatedly. If we start learning procedure from some Q-function approximation we will learn nothing because of the $\max$ operator: agent has no will to explore environment, it will simply perform 'best' action according to its policy. So we must add *exploration* to the agent: we may sample actions with probabilities $p(a_i) = \frac{\exp Q(s,a_i)}{\sum_j \exp Q(s,a_j)} = softmax(a_i)$ (Boltzmann approach) or we can apply *epsilon-greedy* sampling: take random action with probability ϵ and take the best action with probability $1 - \epsilon$. It is recommended to start with epsilon close to 1 and gradually reduce it during training.

To tackle with infinite state spaces we can approximate Q-function with deep neural networks.

1) Deep Q-Network (DQN): The first attempt to use Deep Neural Networks to our knowledge was made in [11]. The authors presented an agent achieving super-human performance in several games. This work marks a milestone in deep reinforcement learning. To make it works, the authors propose to use 2-layer convolutional neural network (CNN) as feature extractor and stack two more fully-connected layers on top to approximate Q-values. This network takes a raw image as input of state and produces Q-values, one output per action.

Because of using a neural network as Q-function estimator, the authors of [11] can not directly apply the rule (2) and instead compute *Temporal Difference* error (TD):

$$TD = Q(s_i, a_i) - (r_i + \gamma \max_{a'} Q(s'_i, a')) \tag{3}$$

and then minimize square of it. Authors used an *online network* to estimate $Q(s_i, a_i)$ term and a *target network* to estimate $\max_{a'} Q(s'_i, a')$ term. The online network was trained via backpropagation, while a value produced by the target network is considered as a constant. The target network's weights were fixed during the online network training and were periodically updated to trained online network.

If we denote parameters of the online network as θ and parameters of the target network as $\tilde{\theta}$, then loss can be

 ISBN: 978-1-61208-697-2

expressed as:

$$L = \sum_i \left(Q(s_i, a_i; \theta) - (r_i + \gamma \max_{a'} Q(s'_i, a'; \tilde{\theta}) \right)^2, \quad (4)$$

where summation taken over a batch (defining the number of samples that will be propagated through the network).

To form a batch authors in [11] propose to use *experience replay*, which contains tuples $\langle s, a, r, s' \rangle$ of state, performed action, reward and next state. During training, authors uses estimation of $Q(s, a)$ not for all actions, but only for the performed ones, and, thus, backpropagate the error through neurons corresponding only to these actions.

Although DQN have shown good results in playing Atari video games, it has several problems, such as slow and unstable training [12], and overestimating of expected reward [13], from which we want to get rid off. There are several extensions to DQN, which can fix these drawbacks or boost up resulting performance.

2) Double Q-learning: Conventional Q-learning suffers from max operator in (2) and agent always overestimates obtained reward. There is simple solution called *Double Q-learning* [13] that is to replace $max_{a'} Q(s, a)$ with

$$Q(s', \operatorname{argmax}_{a'} Q(s'a); \theta) \quad (5)$$

leading to faster and more stable training process.

3) Dueling Networks: The *Dueling network* [12] is a specific architecture, which explicitly uses the Q-function decomposition: $Q(s, a) = V(s) + A(s, a)$, where $V(s)$ is *value* and $A(s, a)$ is advantage. In order to solve the problem of unidentifiability in the sense that given Q we cannot recover V and A uniquely, we present Q-function estimator now has two streams: one for $V(s)$ and one for $A(s, a)$ approximations, and Q-function is computed as following:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{N_a} \sum_j^{N_a} A(s, a_j) \quad (6)$$

Dueling network may result in a huge performance boost.

B. Recurrent Q-learning

Simple Q-learning and extension works under assumption that we have full access to the environment state at any time. However, in practice we do not. In many cases, we can only *observe* part of the environment state, and observation may be incomplete or noisy. In such a case, it is better to use *Partially Observable Markov Decision Process* (POMDP) formalism. POMDP is a tuple (S, A, R, T, Ω, O), where first four items come from MDP, Ω indicates *observation* set and O indicates *observation function*: $O(s_{t+1}, a_t) = p(o_{t+1}|s_{t+1}, a_t), \forall o \in \Omega$. Due to no available knowledge of environment's state, the agent makes a decision by interacting with the environment and receiving new observations. The agent updates its distribution of belief in true state based on distribution of the current state.

1) Deep Recurrent Q-Network (DRQN): Since we do not have state s_t in POMDP, we could not estimate $Q(s_t, a_t)$ like in DQN. However, there is a simple solution to it. One needs to equip an agent with memory h_t and approximate $Q(s_t, a_t)$ by $Q(o_t, h_{t-1}, a_t)$. One of the most popular approaches is to use recurrent neural networks to solve the problem. Dependence on o_t can be eliminated by modelling $h_t = LSTM(o_t, h_{t-1})$ called long-short term memory block [14]. Such networks are called Deep Recurrent Q-Network (DRQN) [15].

Experience replay now contains tuples $\langle o, a, r, o' \rangle$ denoting observation, performed action, reward and next observation. It is essential to sample sequences of consecutive observations from experience replay to make use of agent memory: without such sampling it can not learn sequences. It is natural to learn only from several last observations, because agent has to form its memory from a few observations. All previously mentioned extensions of DQN can be integrated into DRQN model.

2) Multi-step learning: DQN is trained on a single time step, although DRQN trained on a sequence. One may use this in the loss construction as follows: to replace single-step temporal difference (3) by n-step temporal difference [16]:

$$TD(n) = Q(s_i, a_i) - \\ - (r_i + \gamma r_{i+1} + \ldots + \gamma^{n-1} r_{i+n-1} + \gamma^n \max_a Q(s_{i+n}, a')) \quad (7)$$

This also provides faster and more stable training, especially with delayed rewards, but n has to be properly tuned [17].

3) Distributional RL: Instead of learning Q-function, which is an expectation of discounted reward, it is possible to learn distribution of discounted reward [18] [19]. These distributions can be presented by probability masses, placed on a discrete support z with N atoms, $z_i = V_{min} + i * \Delta z, i = 0, \ldots, N$, where $\Delta z = (V_{max} - V_{min})/(N - 1)$ and (V_{min}, V_{max}) represent admissible value interval. Authors denote the value distribution as Z, then distributional version of (2) holds: $Z(x, a) \stackrel{D}{=} R(x, a) + \gamma * Z(X', A')$, which may be used for training. They proposed loss function and exact algorithm to learn such discounted reward distribution and name it Categorical 51 (C51), where 51 is the number of atoms z_i. It has been shown that an agent trained in distributional manner has richer expressiveness and usually converges to a better policy.

IV. Health gathering Scenario

In "Health gathering" scenario, the agent spawns in square room with lava on the floor. The agent has no access to its health and can only turn left, right and move forward (see screenshot of agent view at Fig. 1). There is a bunch of health kits, which increase agent's health and make it survive longer. We used the following reward shaping: agent receives +1 for every step, -100 if it dies, and + *amount of healing*, which is equal to $health(o_t) - health(o_{t-1})$ if this value is bigger than zero, where $health(o)$ represents the agent health in the game.

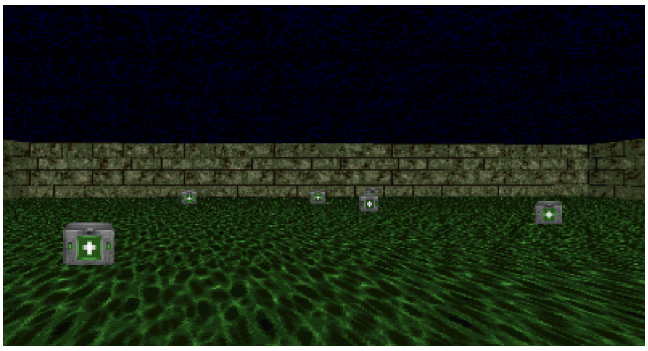


Figure 1. A Screenshot of Health Gathering Scenario

An episode ends if the agent dies or if 2100 tics done. We assume that such shaping will force agent to pick up health kits with some latency and do it only if it has been badly wounded. Agent trained with frameskip 4 and screen resolution of 108×60.

V. Proposed Approach

In this section we describe our baseline models, as well as combined agent architectures. We use two agents as a baseline to compare with: simple DQN and DRQN with LSTM. Also we use two modified versions: first is DRQN with Dueling architecture and Double Q-Learning and dropout with keeprate=0.5 (D4RQN), and second in addition with C51 and Multi-step (C51M).

We decide to use simplified neural network architecture for all agents in this scenario. Also, all the agents preprocess game screenshot by convolutional feature extractor with the same architecture, presented in Table I. CR denotes Convolution + Relu, n denotes number of convolution filters, k denotes kernel size, s denotes stride.

TABLE I. CONVOLUTIONAL FEATURE EXTRACTOR

Input size	Basic	Input size	Other
$30 \times 45 \times 1$	CR, n=8, k=6, s=3	$60 \times 108 \times 3$	CR, n=32, k=8, s=4
$9 \times 14 \times 8$	CR, n=8, k=3, s=2	$14 \times 26 \times 32$	CR, n=64, k=4, s=2
$4 \times 6 \times 8$		$6 \times 12 \times 64$	CR, n=64, k=3, s=1
		$4 \times 10 \times 64$	

After convolutions, the feature map is reshaped to vector form and is feeded into next layers. DQN network has dense layer with 512 units , with Relu activation in both cases. DRQN has LSTM cell with 128 and 512 units respectively. Both DQN and DRQN has one more dense layer at the end with *number of actions* units and linear activation.

D4RQN and C51M has dropout layer after convolutions with keep rate = 0.5. After this they both have LSTM layer with 512 units. D4RQN splits computation into two streams: *value* stream and *advantage* stream by dense layers with 1 and *number of actions* units with linear activation. Outputs from streams are combined by formula (6) and targets during optimization are picked according to (5) thus combining Double Q-learning and Dueling Networks.

There are atoms in C51M algorithm supporting discounted reward distribution. Each atom has the probability, calculated as softmax over atoms. We split LSTM output into two streams: *value* stream with *number of atoms* units and *value* stream with *number of actions* linear layers, each with *number of atoms* units. For each atom these streams combined by formula (6) and then softmax function applied. To compute Q-value from this distribution we use formula: $Q(s,a) = \sum_i^{n\_atoms} z_i p_i(s,a;\theta)$, where z_i and p_i is the i-th atom support and probability and θ represents network parameters. An illustration of the network architecture for C51M algorithm is presented in Fig. 2.

We choose 51 atoms as in the original algorithm for our scenario. It is crucial to pick right values for V_{min} and V_{max} in atom support. These values must represent lowest and highest possible discounted reward. Indeed, if we choose V_{min} and V_{max} too close to each other, agent will not have rich enough expressiveness, but if we choose the gape too much then only few atoms will be used during training and agent degrades down to DQN.

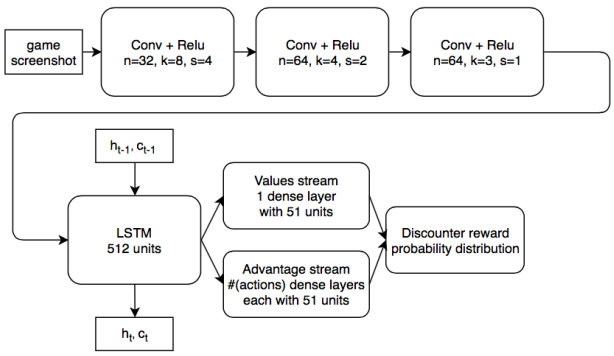


Figure 2. Neural Network for C51M Architecture

In Health Gathering scenario we set these values to -5 and 195. The maximum possible reward equals to maximum episode length which is 2100, and can not be precisely precomputed for shaped reward because of environment randomness. We also decide to focus on small rewards for the agent and significantly reduced range of rewards.

VI. Experiment design

In this section we provide training procedure details. We set experience replay size to 10^5 . It is important to sample sequences of consecutive observations from experience replay, such that only the last observation may be terminating. We decide to do it by simply checking if any of observations in a sequence is terminal, except the last one, and if there is one, we resample until there are no such cases. By doing so, we greatly reduce total amount of terminate observations to train on. It is important in Health Gathering scenario, because agent receives huge penalty at the last observation if it dies. We set batch size and sequence length to 128 and 10, respectively. We use first four observations to update agent's memory and last six to train on. We set number of steps for gradient descent per epoch to 8000 and number of steps before sampling from experience replay to 15. We also reduce learning rate to 0.0002.

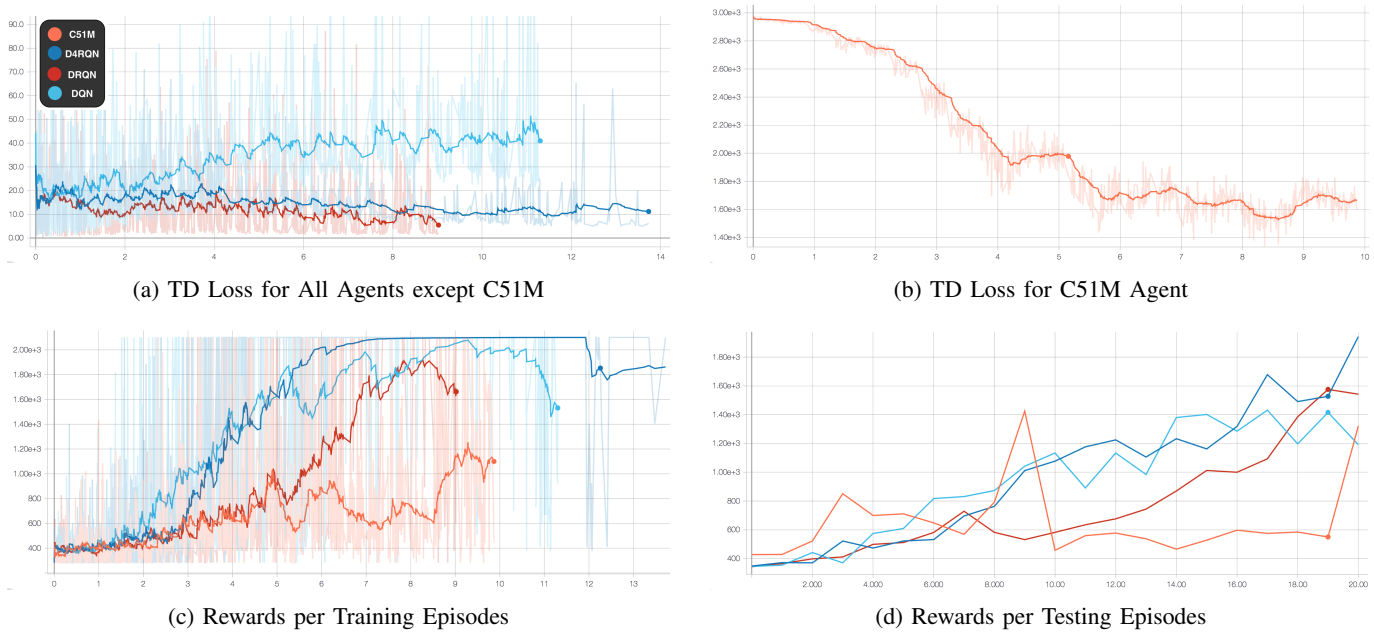
VII. Results

In this section we describe our experimental results and compare performance of all the described agents. For each scenario, we compare learning stability by measuring TD loss at each gradient descent step, learning speed by measuring per-episode reward changes during training and performance of agents after each training epoch. For Fig. 3 the visualization is the following: C51M (dark-blue), DQN (red), DRQN (light-blue) and D4RQN (orange). For TD loss plots we visualize C51M separately, so that we could see the difference between this model and other agent architectures.

A. Health Gathering

TD loss for all the agents can be seen at Fig. 3(a) and 3(b). Again, all agents, except DQN, show stable learning process. Rewards obtained during training can be observed at Fig. 3(c) (plot is constructed in relative time-scale). D4RQN obtains max score more than in half of training episodes, but other agents do not show stable performance. Test rewards presented at Fig. 3(d). C51M has lower performance while D4RQN unexpectedly has highest reward while training than while testing with turned off dropout. We further study agents' performance with and without dropout at testing time.

 ISBN: 978-1-61208-697-2



(a) TD Loss for All Agents except C51M

(b) TD Loss for C51M Agent

(c) Rewards per Training Episodes

(d) Rewards per Testing Episodes

Figure 3. Health Gathering Scenario Comparison

We noticed that D4RQN detects health packs very well and goes directly to them. With this strategy agent is able to survive all 2100 frames, which is length of the episode, and dies rarely. However, if there are multiple health packs in different sides and approximately on the same distance, agent can not decide where to go and jiggles screen some time in attempt to go both directions at once before deciding which direction it wants to go. DQN does not detect health pack as well as D4RQN and may just move into a wall and die. DRQN plays pretty well, but agent's behaviour was not interpretable in terms of similarity to any reasonable humans' behaviour.

C51M is good at detecting health packs, too. But it scores lower than D4RQN and DRQN have, because it tries to wait several frames before pick up a health pack. We believe that such behaviour occurs, because of our reward shaping and experience replay design. If agent has low health it will obtain bigger reward than if it will pick up health pack will full health. Agent trains on terminal observations rarely due to our sampling method. Also, we believe that human players tend to do the same: it is optimal to pick up health pack when you have only, for example, 50 health points, instead of 90 or more. In situations where health is not observable, skilled players will wait optimal time, learned after several deaths. Alas, C51M did not learn it in our experiment and usually waited too long.

B. *Summary results*

Since Health Gathering scenario forced to end after 2100 steps during training, it is interesting to test agents without this forced ending. So, we modified config file and set episode length to 10000 and call it Health Gathering Expanded. We add this version of scenario in Table II that contains final performance of all trained agents in order: DQN, DRQN, D4RQN dropout off, D4RQN dropout on, C51M dropout off, C51M dropout on. Values in Table II equal to $mean(R) \pm std(R)$, where R is non-shaped rewards over 100 episodes.

From Table II we can observe that dropout at *testing time* may increase agent's performance: Health Gathering scenario

TABLE II. RESULTS COMPARISON

Model	HG	HG Expanded
DQN	1262.0 $\pm$ 631.0	1469.6 $\pm$ 1257.0
DRQN	1578.1 $\pm$ 665.9	3173.5 $\pm$ 2842.3
D4RQN	1890.0 $\pm$ 434.3	4781.7 $\pm$ 2938.8
D4RQNd	2078.8 $\pm$ 144.5	9291.4 $\pm$ 2231.2
C51M	1451.7 $\pm$ 708.9	2466.4 $\pm$ 1766.5
C51Md	1593.7 $\pm$ 702.3	4138.4 $\pm$ 3634.6

for both D4RQN and C51M. Our results are available via link https://github.com/DEAkimov/vizDoom

VIII. Conclusion

In this work, we were the first to present a new model-free deep reinforcement learning agents in POMDP settings for 3D first-person shooter. The presented agents drastically outperformed baseline methods, such as DQN and DRQN. Our agent successfully learned how to play several scenario in VizDoom environment and show human-like behaviour. We aim to further compare such an agent with our other deterministic intelligent agents developed for imitating human behavior [20] [21].

We aim to present our other experiments on Basic and Defend the Tower scenarios in Vizdoom, as well as use our agent as backbone architecture for more challenging task, like Deathmatch scenario, which is exactly our plan for future work. Moreover, our agent could be easily combined with Action-specific DRQN [22], Boltzmann exploration [16], Prioritized Experience Replay [8], and can be modified to use in-game features as well as separate networks for action and navigation to improve further.

Acknowledgements

The work was supported by the Russian Science Foundation under grant 17-11-01294 and performed at National Research University Higher School of Economics, Russia.

 ISBN: 978-1-61208-697-2

REFERENCES

[1] I. Makarov and P. Polyakov, "Smoothing voronoi-based path with minimized length and visibility using composite bezier curves." in AIST (Supplement), vol. 191. Ceur WP, 2016, pp. 191–202.

[2] I. Makarov, P. Polyakov, and R. Karpichev, "Voronoi-based path planning based on visibility and kill/death ratio tactical component," in AIST (Suppl). Ceur WP, 2018, pp. 129–140.

[3] I. Makarov and et al., "Modelling human-like behavior through reward-based approach in a first-person shooter game," in EEML, vol. 1627. Ceur WP, 2016, pp. 24–33.

[4] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in Computational Intelligence and Games (CIG), 2016 IEEE Conference on. IEEE, 2016, pp. 1–8.

[5] M. Hessel and et al., "Rainbow: Combining improvements in deep reinforcement learning," arXiv preprint arXiv:1710.02298, 2017, pp. 1–14.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," in Proceedings of the 24th International Conference on Artificial Intelligence, ser. IJCAI'15. AAAI Press, 2015, pp. 4148–4152. [Online]. Available: http://dl.acm.org/citation.cfm?id=2832747.2832830

[7] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning." in AAAI, 2017, pp. 2140–2146.

[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015, pp. 1–21.

[9] C. Schulze and M. Schulze, "Vizdoom: Drqn with prioritized experience replay, double-q learning, & snapshot ensembling," arXiv preprint arXiv:1801.01000, 2018, pp. 1–9.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3, May 1992, pp. 279–292. [Online]. Available: https://doi.org/10.1007/BF00992698

[11] V. Mnih and et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, 2015, pp. 529–533.

[12] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," arXiv preprint arXiv:1511.06581, 2015, pp. 1–15.

[13] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in AAAI, vol. 16, 2016, pp. 2094–2100.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, 1997, pp. 1735–1780. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[15] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," CoRR, abs/1507.06527, 2015, pp. 1–9.

[16] R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine learning, vol. 3, no. 1, 1988, pp. 9–44.

[17] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998, vol. 1.

[18] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," arXiv preprint arXiv:1707.06887, 2017, pp. 1–19.

[19] I. Makarov, A. Kashin, and A. Korinevskaya, "Learning to play pong video game via deep reinforcement learning," in Proceedings of AIST'17. Ceur WP, 2017, pp. 236–241.

[20] I. Makarov, M. Tokmakov, and L. Tokmakova, "Imitation of human behavior in 3d-shooter game," in AIST2015 Analysis of Images, Social Networks and Texts. Ceur WP, 2015, pp. 64–77.

[21] I. Makarov and et al., "First-person shooter game for virtual reality headset with advanced multi-agent intelligent system," in Proceedings of the 24th ACM International Conference on Multimedia, ser. MM '16. New York, NY, USA: ACM, 2016, pp. 735–736.

[22] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for pomdps," arXiv preprint arXiv:1804.06309, 2018, pp. 1–7.

 ISBN: 978-1-61208-697-2