 https://doi.org/10.3311/PPee.16540

Periodica Polytechnica Electrical Engineering and Computer Science, 65(2), pp. 160–166, 2021

A Simplified Pursuit-evasion Game with Reinforcement Learning

Gabor Paczolay[1*], Istvan Harmati[2]

[1] Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1117 Budapest, 2 Magyar Tudósok krt., Hungary
[2] Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1117 Budapest, 2 Magyar Tudósok krt., Hungary
[*] Corresponding author, e-mail: gpaczolay@iit.bme.hu

Received: 26 May 2020, Accepted: 10 July 2020, Published online: 04 March 2021

Abstract

In this paper we visit the problem of pursuit and evasion and specifically, the collision avoidance during the problem. Two distinct tasks are visited: the first is a scenario when the agents can communicate with each other online, meanwhile in the second scenario they have to only rely on the state information and the knowledge about other agents' actions. We propose a method combining the already existing Minimax-*Q* and Nash-*Q* algorithms to provide a solution that can better take the enemy as well as friendly agents' actions into consideration. This combination is a simple weighting of the two algorithms with the Minimax-*Q* algorithm being based on a linear programming problem.

Keywords

reinforcement learning, multiagent learning, pursuit-evasion

1 Introduction

Nowadays newer and newer robotics problems are surfacing. One of them is called "pursuit and evasion", when several robots are chasing another, trying to catch and stop it. This kind of task has already been researched called as predator-prey problem, however the only purpose of the previous algorithms, such as [1] and [2] is to catch a sole (possibly faster) agent by several (slower) agents, without taking collision into consideration. In our work we are proposing a solution to collision-free catching algorithm, and due to the flexibility of the solution, the possibility of application to greater state-spaces and the size of the actual problem, a reinforcement learning solution method was proposed.

Collision avoidance makes the standard predator-prey problem a lot more difficult. Meanwhile in the common predator-prey solution it is possible to relentlessly chase the enemy by all agents, the possibility of collision renders relentless chasing meaningless, it is better for the agents to find a solution that is less prone to accidental collisions and thus, they have to approach the enemy more tactically. In our experiment, we checked both possibilities for the agents of being able to communicate with each other or being only possible to rely on state information without further online communication.

By the nature of the problem, the agents have to take each other as well as the enemy agent into consideration, they cannot act as if other agents would not be present in the world. Friendly movements have to be taken care of to try to evade collision, and the enemy movement has to be watched to be able to take the enemy policy into consideration. This brings us to the problem group of multi-agent learning, which means learning in the presence of other agents in the world. This problem group, multi-agent learning, tries to solve the problems of stochasticity caused by other agents' actions in the world, mostly by explicitly taking them into consideration.

Littman [3] utilized the Minimax-Q algorithm first and applied it to a simplified version of robotic soccer game. Hu and Wellman [4] created the Nash-Q algorithm and used it on a small gridworld example to show its results. Bowling and Veloso [5] varied the learning rate of the training process to speed it up. Lowe et al. [1]

Cite this article as: Paczolay, G., Harmati, I. "A Simplified Pursuit-evasion Game with Reinforcement Learning", Periodica Polytechnica Electrical Engineering and Computer Science, 65(2), pp. 160–166, 2021. https://doi.org/10.3311/PPee.16540

created a continuous action-space solution for the predator-prey problem as well, utilizing a Multi-agent Deep Deterministic Policy Gradient algorithm. The continuous action space has also been researched by Li et al. [2], in this case in competitive environment via Minimax Deep Deterministic Policy Gradient.

Apart from gridworld, another usual problem in multi-agent environment to solve is micromanagement in strategy games, usually done by deep neural networks. Micromanagement means control of the units individually. Samvelyan et al. [6] proposed a benchmark for it called StarCraft Mutli-Agent Challenge. The full StarCraft II environment is also used for multi-agent learning by creating a headless version of the well-known game [7]. Foerster et al. [8] developed a counterfactual multi-agent policy gradient method with centralized critic and decentralized actors. Peng et al. [9] utilized Bidirectionally. Coordinated nets with a vectorized extension of actor-critic formulation. Rashid et al. [10] developed a value-based method that can train decentralized policies in a centralized way. Foerster et al. [11] tried to achieve better performance of deep reinforcement learning by stabilizing the experience replay part. There are some other types of multi-agent testbeds and solutions worth mentioning. Bard et al. [12] proposed the game of Hanabi, a cooperative and partially observable card game as a multi-agent testbed. For the Hanabi game, Foerster et al. [13] created a state-of-art solution called Bayesian Action Decoder. A first-person multiplayer shooter game was used by Jaderberg et al. [14] as a testbed, where human-level performance was reached by population-based deep reinforcement learning.

Unlike previous literature, our paper proposes a method with collision as a focus point of the training process. With that, we also propose an algorithm based on the weightning of the Minimax-Q and the Nash-Q algorithms.

First, we introduce the theoretical background of our solution in Section 2. Afterwards, we show the formulated problem as well as the results of our experiments in Section 3. Finally, we discuss our results, conclude our work and signal some directions for future possibilities of improvement to our solution in Section 4.

2 Theoretical background

2.1 Markov decision processes

A Markov Decision Process is a mathematical framework for modeling of decision making. In a Markov Decision Process there are states, selectable actions, transition probabilities and rewards. At each timestep the process starts at a state s, and it selects an action a from the available action space. Then, it gets a corresponding reward r, and then finds itself in a state s' given by the probability of $P(s,s')$. A process is said to be Markovian if

$$P\left(a^t = a|s^t, a^{t-1}, \ldots, s^0, a^0\right) = P\left(a^t = a|s^t\right), \tag{1}$$

which means that a state is transitioning only on the previous state and the current action. Thus, only the last state and action are interesting regarding the decision for the next state. The notation a' means the action taken at time t, while s^t means the state at time t.

In a Markov Decision Process, the agents are trying to find a policy which maximizes the sum of discounted expected rewards. The standard solution for this is through iterative search method which searches for a fixed point of the Bellman equation:

$$v\left(s,\pi^*\right) = \max_a\left(r\left(s,a\right) + \gamma\sum_{s'}\left(ps'|s,a\right)v\left(s',\pi^*\right)\right), \tag{2}$$

where $v(s,\pi^*)$ is the value function for the optimal π^* policy, $r(s,a)$ is the reward taking action a at state s, and $p\left(s'|s,a\right)$ is the state transition probability from state s to state s' taking action a.

2.2 Reinforcement learning

When the state transition probabilities or the rewards are unknown, the problem of the Markov Decision Process becomes a problem of Reinforcement learning. In this group of problem the agent tries to make a model of the world around itself by trial and error. In most reinforcement learning methods, the agent tries to learn a value function that renders a value to the states or to the actions from states. These values correspond to the achievable reward from reaching a state or from taking a specific action from a state.

The most commonly used type of reinforcement learning is Q-learning, when the so-called Q-values are estimated for each of the state-action pairs of the world. These Q-values represent the value of choosing a specific action in a state, meaning how much reward could the agent possibly get by taking that action. The basic idea for Q-learning is that we can define a function Q such that

$$Q^*\left(s,a\right) = r\left(s,a\right) + \gamma\sum_{s'}\left(ps'|s,a\right)v\left(s',\pi^*\right). \tag{3}$$

By Eq. (2) we can deduce

$$v\left(s,\pi^*\right) = \max_a Q^*\left(s,a\right). \tag{4}$$

The equation for Q-learning for updating the Q-values of a state is:

$$Q(s',a) \leftarrow (1-\alpha)\cdot Q(s,a)+\alpha\cdot\left(r+\gamma\cdot\max_{a'}Q(s',a')\right), \quad (5)$$

where α is the learning rate and γ is the discount for the reward.The agent always selects an action that maximizes the Q-function for the state that the agent is in.

2.3 Matrix and Markov games

A Matrix game is a stochastic framework where each player selects an action and gets their immediate reward based on its and all other agents' action. They are called as Matrix games due to the fact that the games can be written as a matrix, with the first two player selecting action in the row and the column of the matrix. Unlike Markov Decision Processes, these games have no state.

One famous Matrix game is Matching Pennies. This game is played between two players, Even and Odd. Both players have a penny in their hand, which is secretly turned to Heads or Tails. Then, these pennies are shown to the other as well. If the pennies match, Even keeps all pennies, if they do not match, so one is Heads and the other is Tails, Odd keeps them. As one player wins when the other loses, and wins or loses the same amount as the other one loses or wins, this game is called zero-sum.

A famous example of non-zero-sum Matrix game is Prisoners' dilemma. In this game, two prisoners are kept away from each other and have the possible actions of remaining silent or betraying the other one. If they both remain silent, they become sentenced to a short period. If one betrays the other meanwhile the other one remains silent, one is set free and the other is sentenced for a long period. If they both try to betray the other one, they are sentenced to an even longer period. This example has been analyzed by game theory to show why two rational individuals might not cooperate.

Markov games, or Stochastic games are an extension of Markov Decision Processes with multiple agents. Also, it can be thought of as an extension to Matrix games with multiple states. In a Markov game, each state has a payoff matrix for all of the states. The next state is determined by the joint action of the agents and fixed transition probabilities. The probabilities mus satisfy the constraint

$$\sum_{s'}\left(ps'|s,a^1,\ldots,a^n\right)=1. \quad (6)$$

A game is Markovian if

$$P\left(a_i^t=a_i\,|\,s^t,a_i^{(t-1)},\ldots,s^0,a_i^0\right)=P\left(a_i^t=a_i\,|\,s^t\right), \quad (7)$$

so the next state depends only on the current state and the current actions taken by all agents. We can define Nash equilibria for a Markov game, which is a joint policy π^* such that for all s and i

$$V^i_{\pi^{i*},\pi^{-i*}}(s)\geq V^i_{\pi^i,\pi^{-i}}(s) \text{ for all } \pi^i, \quad (8)$$

so the value for the optimal joint policy is higher or equal to all other policies in all states.

2.4 Multi-agent learning methods

In Subsection 2.4 we go through some of the methods for learning the agents with several other agents present in Markov games with the agents being either strictly enemy or friendly.

The easiest method to solve a world with several agents is the consideration of other agents as if they were part of the environment, this is called Independent Q-learning. In this case, the environment is considered to be stochastic. Although this method is a very coarse simplification, many times it still performs well enough to evade using more complex algorithms.

One subgroup of multi-agent learning reflects to zero-sum games. In this case, the reward of one agent equals to the penalty of the other one, the sum of their reward is always equal to zero, hence the name zero-sum. From game theory, zero-sum games have a solution called Minimax:

$$\overline{v_i}=\min_{a_{-i}}\max_{a_i}v_i\left(a_i,a_{-i}\right), \quad (9)$$

which means the action taken should be the one that maximize the reward when the other agent is trying to maximize its own reward, and with that, it is trying to minimize ours. This method is utilized in **Minimax *Q*-learning,** where each state contains a payoff matrix which is then solved by the minimax method [3].

When the sum of the rewards is not strictly zero, this type of game is called general-sum game. In this case, agents are not strictly competitive, cooperative behavior is also allowed. In general-sum games the minimax strategy does not work due to the agents not seeking the worst strategy for the other one(in strictly competitive scenarios, their reward function is even the same as the other agents' reward function. In general-sum games, the best strategy for all agents is to follow a Nash-equilibrium strategy, which is the best response. In a multiagent scenario, the optimal Q-value is based on the current reward and future rewards when all agents play specified Nash equilibrium from the next period instead of single-agent scenarios, where the optimal value is from current reward and future rewards by playing the optimal strategy from nex period by the single agent.

The nash equilibrium is used by **Nash-*Q* learning**, where the payoff matrix is searched for Nash equilibrium to determine the policy. The Nash-*Q* algorithm utilizes a distinct *Q*-table for all agents, with separate entries for all states and all actions taken by all agents [4].

3 Experiments and results

The testbed was an 5 × 5 grid, where the agents were able to only move one step horizontally or vertically. As shown in Fig. 1, the gridworld contained three catchers and one fleeing agent, with the formers placed in the corners of the grid and the latter positioned in the center. The catchers, depending on the problem, were either able to communicate with each other or not. The number of the states were 25^4 even in this smaller world, this explains the size of the gridworld as well as the limited number of agents.

The enemy had several distinct strategies: either it was moving randomly, or it was fleeing from the other agents with special attention not to get stuck in the corners of the field. When the agent would get stuck in the corner, it would select an action randomly instead.

In our first experiment, all three agents were able to communicate with each other, realizing a centralized communication. In this case, a centralized minimax controller controlled the movement of all three agents concurrently. in this realization the action space grew to 4^3, with the action space of the enemy staying at 4.

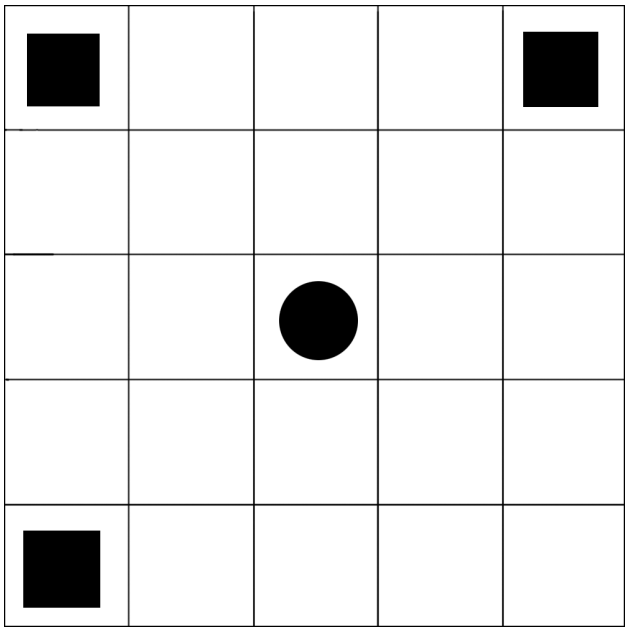

Fig. 1 The simulation environment. The squares the controlled agents, meanwhile the circle represents the fleeing enemy. The goal is to catch the enemy by moving horizontally or vertically.

In the second experiment the agents were unable to communicate with each other during the iteration, but the training was processed in a centralized way - a centralized training and decentralized execution was implemented.

During the second experiment, a mixture of the Minimax-*Q* and Nash-*Q* algorithms was implemented. The algorithm can be seen in Algorithm 1. All agents are playing a zero-sum game against the enemy agent, as well as all agents are playing a general-sum game with all other agents for collision avoidance, but also considering the enemy player's actions - regarding the enemy actions, the reward was always considered to be equal to the negative of the sum of other agents' rewards. The mixure of the two algorithm was created in such a way due to the fact that in this case, the algorithm shows a two-level hierarchy with a higher-level game against the enemy. In the Nash-*Q* part of the algorithm, the policy to the Nash equilibrium was selected by the following process: A multi-dimensional array was constructed for all of the states of the gridworld space, so the sum of the dimensions for each state in the gridworld space was equal to $N \times A^{N+1}$, where A is the number of possible actions that can be taken by the agents, and N is the number of the friendly agents. The $N + 1$ number is due to our consideration of the enemy agent in the Nash-*Q* algorithm.

In the Nash-*Q* algorithm, the Nash*Q* value was calculated as the following:

$$\text{Nash}Q^i = \pi_1 \cdot \ldots \cdot \pi_N \cdot Q^i. \tag{10}$$

In our work, the scalar product was carried out by outer products of the policies with an element-wise product of the *Q*-table for the specific state-agent pair.

The Minimax part of the algorithm utilized two steps to find the required value and *Q*-values. The value of a state in a Markov game is

$$V(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s,a,o)\pi_a \tag{11}$$

and the quality of an action against action in state is

$$Q(s,a,o) = R(s,a,o) + \gamma \sum_{s'} T(s,a,o,s')V(s'). \tag{12}$$

As noted in Algorithm 1, the Minimax part of the algorithm used linear programming to find values for the value function and the policy. The linear programming part is further explained in Algorithm 2. In that case, $x_0 \ldots x_n$ are the unknown variables, and $Q(s,a,o)$ are the elements of the *Q*-table.

Algorithm 1 Combination of the Minimax-Q and the Nash-Q algorithm for the decentralized version of the solution

Initialize values:

$Q_M^i(s,a_1..a_n) \leftarrow 1$

$Q_N^i(s,a_1..a_n) \leftarrow 0$

$V(s,x) \leftarrow 1$ Where x is the agent

while Not end of all iterations **do**

Select actions with ϵ -greedy policy:

with ϵ probability, return an action uniformly at random,

otherwise, return an action with probability π

Run environment, get rewards $r_i..r_n$ as result

for all agents x **do**

$$Q_N^x(s,x,a_1..a_N) \leftarrow (1-\alpha)\cdot Q_N(s,x,a_1..a_N) + \alpha\cdot\left(r_i + \beta\cdot \mathrm{Nash}Q^x(s')\right)$$

where Nash$Q$ is defined in Eq. (10) $Q_M^x(s,x,a,o) \leftarrow (1-\alpha)\cdot Q_M(s,x,a,o) + \alpha\cdot\left(r_i + \gamma\cdot V(s',x)\right)$

Update $\pi(s,x) \leftarrow (1-\delta)\cdot \pi_N(s,x) + \delta\cdot \pi_M(s,x)$

Update $V(s,x)$

Where $\pi_M(s,x)$ and $V(s,x)$ are calculated by linear programming

end for

end while

Algorithm 2 Linear programming for the Minimax-Q algorithm

Maximize x_0 such that

$$\begin{aligned}
&x_0 - Q(s,a_1,o_1) - \ldots - Q(s,a_n,o_1) \le 0\\
&\vdots\\
&x_0 - Q(s,a_1,o_m) - \ldots - Q(s,a_n,o_m) \le 0\\
&x_1 + \ldots + x_n = 1\\
&0 \le x_1 \le 1\\
&\vdots\\
&0 \le x_n \le 1
\end{aligned}$$

$V(s) \leftarrow x_0 \quad \pi \leftarrow x_1..n_n$

To ensure convergence, β and γ were selected as 0.1, meanwhile α was selected as 0.5. These values were selected via empirical observation. It is important to note that convergence was achieved when a collision only provided a penalty and did not terminate the iteration, probably due to the fact that without it, there were a very small amount of samples where the agents did not collide with each other while reaching the enemy correctly.

Rewards had to be also carefully picked, because if the reward for successfully reaching the enemy was significantly higher than the penalty for each timestep (to make sure that the agents are trying to reach the enemy as fast as they can), the algorithm did not converge. Eventually, the reward for completing the scenario was set to zero and only penalties were given to the agents for the timesteps as well as a big penalty on collision to ensure future collision avoidance.

It is important to take the problem group of exploration and exploitation into consideration. When the agent is following the optimal policy, it is called exploitation, meanwhile differing from the optimal policy is called exploration. Exploration is required to have the agent visit all the states and not getting stuck in a local maximum. To evade the problem of getting stuck at a local maximum, an ϵ -greedy strategy was used. In this case, the agent "exploits", follows the optimal policy by $1-\epsilon$ chance. or "explores", selects its action from a uniform random distribution with chance.

As shown on Fig. 2, in the centralized task, the agent was able to decrease the number of steps taken until finding the enemy agent. In the decentralized task the number of steps did not reduce significantly during training due to the required extra steps for collision evasion, but the number of collisions was reduced by learning (by at least 50 %).

Fig. 3 and Fig. 4 show the results of the testing for fleeing and random opponents, respectively. These steps are the last four steps of two distinct games. It can be seen on the aforementioned figures that the agents are trying to "exploit" the fleeing agent by having one agent chasing it up until the wall (in the upper left corner), paying attention to the policy of the fleeing enemy, meanwhile when the agent is random, the agents are more agressive at catching and are utilizing another approach by having more agents chasing the opponent near each other.

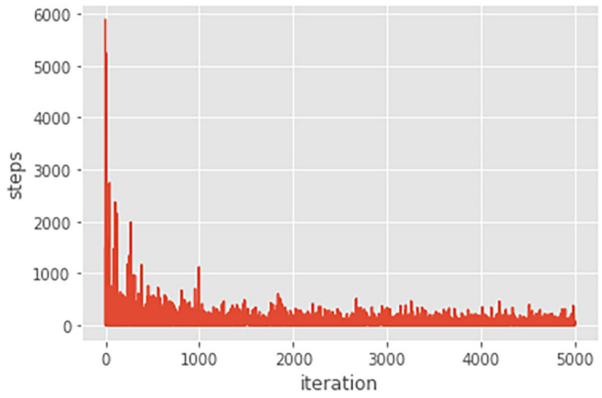

Fig. 2 Number of steps taken to find the enemy per iteration.

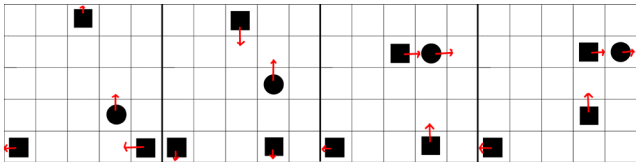

Fig. 3 An example of catching the opponent with fleeing policy.

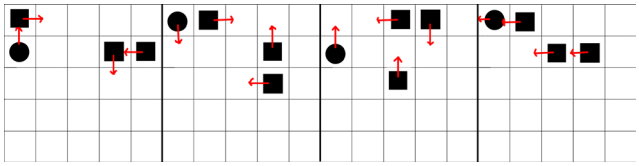

Fig. 4 An example of catching the opponent when its polciy is random.

4 Conclusion

The learned model of the scenario was able to chase the adversary correctly, with only small chance of collision with other friendly agents.

To speed up the training process, a variable learning rate can be used, for example with the "Win or Learn Fast" (WoLF) method [5]. This method varies the learning rate based on heuristics that decide whether the agent is performing well in a specific situation, and if it performs well, the learning rate is smaller. In situations where the agent is performing relatively badly, the learning rate is taken higher to ditch the actual policy and learn a newer one.

The problem can be extended to greater scenes as well as a real-world scenario by the utilization of function approximation, for example, the utilization of neural networks. Deep neural networks, however, tend to diverge for multi-agent scenarios, thus it should be stabilized to be able to learn the scenario properly.

Acknowledgment

The research reported in this paper was supported by the Higher Education Excellence Program in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKP-MI/SC).

References

[1] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments", [cs.LG], arXiv:1706.02275, Cornell University, Ithaca, NY, USA. [online] Available at: https://arxiv.org/abs/1706.02275 [Accessed: 02 March 2020]

[2] Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., Russell, S. "Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient", Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), pp. 4213–4220, 2019. https://doi.org/10.1609/aaai.v33i01.33014213

[3] Littman, M. L. "Markov games as a framework for multi-agent reinforcement learning", In: Cohen, W. W., Hirsh, H. (eds.) Machine Learning Proceedings 1994, Morgan Kaufmann, San Francisco, CA, USA, 1994, pp. 157–163. https://doi.org/10.1016/B978-1-55860-335-6.50027-1

[4] Hu, J., Wellman, M. P. "Nash Q-Learning for General-Sum Stochastic Games", Journal of Machine Learning Research, 4, pp. 1039–1069, 2003. https://doi.org/10.1162/1532443041827880

[5] Bowling, M., Veloso, M. "Multiagent learning using a variable learning rate", Artificial Intelligence, 136(2), pp. 215–250, 2002. https://doi.org/10.1016/S0004-3702(02)00121-2

[6] Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C, M., Torr, P. H. S., Foerster, J., Whiteson, S. "The StarCraft Multi-Agent Challenge", [cs.LG], arXiv:1902.04043, Cornell University, Ithaca, NY, USA, 2019. [online] Available at: https://arxiv.org/abs/1902.04043 [Accessed: 02 March 2020]

[7] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., Tsing, R. "StarCraft II: A New Challenge for Reinforcement Learning", [cs.LG], arXiv:1708.04782, Cornell University, Ithaca, NY, USA, 2017. [online] Available at: https://arxiv.org/abs/1708.04782 [Accessed: 02 March 2020]

[8] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S. "Counterfactual Multi-Agent Policy Gradients", [cs.AI], arXiv:1705.08926, Cornell University, Ithaca, NY, USA, 2017. [online] Available at: https://arxiv.org/abs/1705.08926 [Accessed: 02 March 2020]

[9] Peng, P., Wen, Y., Yang, Y., Yuan, Q., Tang, Z., Long, H., Wang, J. "Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games", [cs.AI], arXiv:1703.10069, Cornell University, Ithaca, NY, USA, 2017. [online] Available at: https://arxiv.org/abs/1703.10069 [Accessed: 02 March 2020]

[10] Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., Whiteson, S. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning", [cs.LG], arXiv:1803.11485, Cornell University, Ithaca, NY, USA, 2018. [online] Available at: https://arxiv.org/abs/1803.11485 [Accessed: 02 March 2020]

[11] Foerster, J., Nardelli, N., Farquhar, G., Torr, P. H. S., Kohli, P., Whiteson, S. "Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning", [cs.AI], arXiv:1702.08887, Cornell University, Ithaca, NY, USA, 2017. [online] Available at: https://arxiv.org/abs/1702.08887 [Accessed: 02 March 2020]

[12] Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., Bowling, M. "The Hanabi Challenge: A new frontier for AI research", Artificial Intelligence, 280, Article Number: 103216, 2020. https://doi.org/10.1016/j.artint.2019.103216

[13] Foerster, J. N., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M., Bowling, M. "Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning", [cs.MA], arXiv:1811.01458, Cornell University, Ithaca, NY, USA, 2019. [online] Available at: https://arxiv.org/abs/1811.01458 [Accessed: 02 March 2020]

[14] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Garcia Castañeda, A., Beattie, C., Rabinovitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., Graepel, T. "Human-level performance in 3D multiplayer games with population-based deep reinforcement learning", Science, 364(6443), pp. 859–865, 2019. https://doi.org/10.1126/science.aau6249