Exploring the Performance of Multiagent-RL Algorithms in Multiagent Space-Invader

Jizhou Wang
University of Montreal
260518646
jizhou.wang@umontreal.ca

Philippe Sayegh
McGill University
260637812
philippe.sayegh@mail.mcgill.ca

Andrew Cook
McGill University
260781415
andrew.cook2@mail.mcgill.ca

Abstract

In this paper, we tried to compare various Multi-Agent Reinforcement Learning (MARL) algorithms, including Q-learning[1], DQN[2], Proximal Policy Optimization (PPO)[3], and Multi-Agent Proximal Policy Optimization (MAPPO)[4], in the multiagent Space-Invader environment. Our quantitative and qualitative results have shown that Q-learning and DQN were able to learn a greedy policy, achieving a higher average return during testing compared to PPO and MAPPO. The policy learned by Q-Learning, DQN, and MAPPO suggests that agents tried to merge come together as this allows for a higher reward when both agents get hit. Further experiments in cooperative environments should be conducted given enough time and resources with a focus on intrinsic vs extrinsic rewards. The source code for this project is available at https://github.com/Jawing/RL-Project.

1 Introduction

Our research focused on a Multi-Agent Reinforcement Learning (MARL) environment. In this environment, each agent has its own policy function and rewards to be maximized. Thus it can be competitive, cooperative, or both. We conducted our experiment using Pettingzoo[5] a gym-like API that supports a variety of multi-agent environments such as the one we used called Space Invaders[6].

All of our algorithms are trained using online reinforcement learning. We used a uniform random policy as our baseline to compare performance with other algorithms. Q-learning with linear function approximation using epsilon-greedy (ϵ-greedy) policy is a model-free, value-based algorithm; it aims to learn the value function of the environment by estimating the agents' expected long-term reward, or Q-value, for each state-action pair.

DQN is a more advanced implementation of a Q-Network algorithm. Deep Q-Networks function similarly to Q-learning, except that a deep neural network replaces the simple dot product to infer the Q-values. It also incorporate experience replay[7]

Our last two algorithms, PPO[3] and MAPPO[4] are actor-critic[8] models that use an objective function combining the policy gradient with an entropy bonus and a clipping mechanism to ensure a "proximal" update. The advantage function is used to weigh the policy gradient by the expected improvement in the value of the policy. Whereas, MAPPO is an extension of PPO for multi-agent RL problems where both agents share the same critic network.

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

We have also experimented with different reward functions. The extrinsic (default) rewards are explicitly defined by the system's designer and provide clear feedback. Intrinsic rewards arise from the agent's own behaviour, such as survival time (step count) which can be more simple to specify.

Overall we've found from the experiments that more extensive training and testing are needed to draw conclusions in regard to the performance of MARL algorithms such as PPO and MAPPO. The preliminary results have shown that using intrinsic reward help to improve the agent's performance. These networks may require longer training time and fine-tuning compared to Q-learning and DQN in order to generate a more optimal policy.

2 Related Work and Background

Reinforcement learning environments are modelled as a Markov Decision Process (MDP) which is a one-agent decision problem. MARL can be seen as a natural evolution of MDP where an environment will be composed of n-MDPs where n is the number of agents[9]. This relation of MARL to single-agent RL allows algorithms developed for single agents to be applied to multi-agent environments. The emergence of MARL as an active area of research has exploded recently due to advancements in single-agent algorithms [10]. MARL's most successful uses have been in games like Go [11] but have potential in many other areas such as autonomous driving or resource management[12].

Previous experiments comparing the performance of MAPPO have been conducted where it often achieves competitive or superior results in both final returns and sample efficiency after conducting ablation studies to identify critical implementation and hyperparameter factors that contribute to MAPPO's performance. The results suggest that PPO-based methods can be a strong candidate for cooperative multi-agent reinforcement learning.[4]

Research has also been conducted on intrinsic reward through social influence[13]. Rewards are given to agents for having causal influence over other agents' actions, assessed through counterfactual reasoning. This leads to enhanced coordination and communication in challenging social environments, improving the learning curves of deep RL agents. The proposed influence rewards can be computed in a decentralized way, enabling agents to learn a model of other agents using deep neural networks.

Comparing how different algorithms perform in the same environment is an essential way to test their performance and choices of hyper-parameters. Other comparisons of algorithms in a MARL environment include testing a new method of clustering agents into roles against well-known algorithms in a Starcraft II environment [14], or by creating benchmarks for algorithms in a hide and seek environment [15]. The space invaders environment has been used for algorithm comparison in both its single agent form [16]. However, it is an intriguing environment for MARL research due to its simplistic nature with a reward structure that is both competitive and cooperative[17].

2.1 Space Invaders Environment from PettingZoo

PettingZoo is a Python library made to build and test MARL algorithms. It contains a panoply of environments with their own agents, rules, states, rewards, etc. The environments it offers are divided into collections, one of which being *Atari*, a collection of environments based on (or copied from) games of the eponymous video game system from the 1970s and 1980s.

The environment for the experiment is the PettingZoo version of the game "Space Invaders," in which two agents control a spaceship each and aim to shoot as many alien ships as possible. The agents can take six possible actions (up, down, left, right, fire, stop) and receive rewards depending on the aliens they hit. The game runs at a resolution of 210x160 pixels, and the agents can take action at every frame when run in parallel. The agents have a shared pool of three lives, and the game continues until the pool is exhausted. The environment provides rewards for hitting enemy ships with points ranging from 5-100, as well as betraying friendly ships granting 200 points to the survivor, leaving the opportunity for interesting strategies to be learned.

2.2 Q-Learning with LFA and DQN with ϵ-greedy

The Q-learning algorithm we used works as follows: the agent selects an action using an ϵ-greedy policy, exploring the state space randomly or greedily choosing the action with the highest Q-value. A feature vector describes states and actions from the state-action space, which can be a vector of floats or a one-hot encoding of state variables; a situation-dependent design choice. Each agent has a weight vector, and to estimate the Q-value, the agent takes the dot product of the weight vector with the feature vector. Ideally, the weights will be such that the dot product is a good estimate of any state-action pair's Q-value. We update the weights through a learning process.

After an action is taken, the agent uses the dot product method again to estimate the Q-value of each state-action pair from the new state. It updates its weight vector using an update rule based on the Bellman equation, where δ is the temporal difference error, s is the original state, a is the action taken, s' is the new state, r is the reward obtained, a' is the action with the highest Q-value in state s', and $f(s, a)$ is the feature vector for the (s, a) pair. The learning rate and discount factor are denoted by α and γ, respectively:

$$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$$

$$w \leftarrow \alpha \delta f(s, a)$$

DQN approximates the Q-learning policy function with a deep convolutional neural network[18]. It is able to better handle high-dimensional spaces and stabilize learning with experience replay.

Experience replay involves storing the agent's experiences (s_t, a_t, r_t, s_{t+1}) in a buffer $\mathcal{D}$, where s_t is the state at time t, a_t is the action taken at time t, r_t is the reward obtained at time t, and s_{t+1} is the next state observed at time $t + 1$. During training, the agent samples a minibatch of experiences (s_i, a_i, r_i, s_{i+1}) from the buffer and computes the temporal difference (TD) error shown below:

$$\delta_i = r_i + \gamma \max_{a'} Q^\pi(s_{i+1}, a'; \theta^-) - Q^\pi(s_i, a_i; \theta)$$

where θ^- are the parameters of a separate target network used to compute the target Q-value. This network is updated periodically to match the parameters of the main network θ. The Q-function is then updated by minimizing the mean squared error between the predicted Q-value and the target Q-value:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \Big(Q^\pi(s_i, a_i; \theta) - (r_i + \gamma \max_{a'} Q^\pi(s_{i+1}, a'; \theta^-)) \Big)^2$$

where N is the batch size. The parameters θ are updated using gradient descent.

2.3 PPO, MAPPO

PPO and MAPPO are the main algorithms used in our experiments with deep convolutional neural networks[18] as function approximators. They can handle both continuous and discrete action spaces. They are both actor-critic models that use a policy gradient method which optimizes the policy directly with an entropy bonus as well as a clipping mechanism to ensure a "proximal" policy for stable learning. It also uses an advantage function to weigh the policy gradient by the expected improvement in the value of the policy. The objective function in PPO is given by the following:

$$\mathcal{L}^{AC}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t|s_t) A_t^\pi \right] - \beta \mathbb{E}_t[H(\pi_\theta(s_t))]$$

Where A_t^π is the advantage function, defined as the difference between the estimated value $V^\pi(s_t)$ and the expected cumulative reward $Q^\pi(s_t, a_t)$ of taking action a_t in state s_t. And $H(\pi_\theta(s_t))$ is the entropy bonus, which encourages exploration by penalizing policies that are too deterministic. The hyperparameter β controls the strength of the entropy bonus.

MAPPO extends PPO for multi-agent scenarios, incorporating decentralized learning using independent agent policy networks and a shared critic network. These algorithms may require more hyperparameter tuning and training to achieve optimality.

3 Methodology

3.1 Implementation and Setup

Overall the experiment is set up using the multi-agent space invaders environment from pettingzoo[6]. It is set with $moving_shields = True$ and all other settings as $False$. Other settings could potentially increase difficulty. For training, we ran 1000 episodes (100 for Q-learning) with $gamma = 0.999$ as the discount factor. For testing, we ran 100 episodes to record the returns and the steps of each episode. Algorithms using function approximations shared the same convolutional neural network parameters to map the input frame array onto a hidden layer of 1280 features. This input observation array is first normalized into grayscale from RGB and resized to a 64x64 array with 4 frame stacks.

The baseline used for our experiment was a uniform random policy. By comparing the performance of other algorithms to the uniform random policy baseline, we can determine whether the algorithms are learning to cooperate or compete effectively with other agents and whether they are able to improve upon the performance of a simple random policy, quantitative and qualitatively.

3.1.1 Q-learning and DQN

Q-learning implementation requires several design choices and hyperparameters, as mentioned in Section 2.2. In the Space Invaders environment, the state information is a 3D RGB array flattened into a 100,800-dimensional vector, making it challenging for the algorithm to learn useful representations. To include the action, we concatenated an array of 16,800 entries to the feature vector, where each entry is defined as $action \times 51$. This extra layer represents the action taken, with the intensity levels indicating the action choice. We chose 16,800 entries to make the representation relevant to the dot product without giving it equal importance to the frames themselves.

We had issues with weights going to infinity, leading to NaN values. To fix this, we capped the weight update increments and made the learning rate decay with episodes. We experimented with a grayscale state representation to reduce the vector size. Hyperparameter choices were also experimented with, including action representation size and increment value cap. This feature vector design is suboptimal, but we chose it due to time constraints and to focus on other advanced algorithms.

DQN was chosen as it is an evolution from Q-learning. We wanted to see the effect of replacing the Q-learning linear function approximator with a neural network. Each agent has its own DQN agent consisting of two neural networks, an action-value network, and a target network. The agent uses experience replay to store past experiences in a replay buffer. Using this strategy over traditional sampling prevents older experiences from being overwritten and allows the agent to learn from a single experience multiple times. The agent then randomly samples from the replay buffer and uses those past experiences for training.

3.1.2 PPO, MAPPO, and Reward functions

For PPO and MAPPO, we would like to experiment and understand the effect of multi-agent training in a centralized vs decentralized setting. Hyperparameters between these networks are kept the same such as batch size, learning coefficients, etc. MAPPO has a shared critic network whereas PPO does not. We have also trained the networks with different reward functions. Extrinsic reward is defined as the default environment rewards output and intrinsic reward is defined as +1 for each step count in a given episode.

The PPO models were trained and tested with each agent having its own policy and value network, with one model based on the extrinsic reward given by the environment and the other based on intrinsic reward.

Similarly, MAPPO models were trained and tested with a model based on intrinsic reward and a model based on extrinsic reward. We've also trained a MAPPO model on extrinsic reward with the critic network incorporating the output of both actor policy network features as input to its layer, alongside the hidden features of the state. This should give more information to help each agent decide on its own action given the observation it sees from the critic network.

3.2 Training and Evaluation Metrics

During training, agents from different methods learn to maximize the reward signal using their relative optimization algorithm and loss function based on the predicted and target values.

Evaluation metrics used for our experiments are mainly focused on the average cumulative return, and average cumulative discounted return of both agents combined, measured across multiple episodes. We compared the mean and standard deviation of the returns over all episodes and plotted them in a bar graph in comparison with different algorithms to see the difference in performance. We have also recorded the step count to termination per episode for each of the experiments.

4 Results

4.1 Q-Learning with LFA and Epsilon Greedy

The Q-learning results were not successful. The large feature vector used in the Q-learning algorithm had each entry representing a pixel's color intensity or lightness, which did not take into account the two-dimensionality of the state observation image. This feature vector design was inappropriate for the task of inferring things from an image. A convolutional neural network, which takes into account the 2D relationship between pixels, is much more appropriate. In our case, the feature vector design used in the Q-learning algorithm resulted in a single-layer feed-forward MLP with no bias term, which was not promising for the task of image inference.

Hence, the training performance is mostly all over the place. Let's first take a look at an example single run training of a pair of agents. Note that in all the graphs about Q-learning, the rewards of both agents are summed at each step. We can see in Fig. 1(a) that while the rewards seem to trend up slightly, the variance is large and the training didn't really converge. The pattern continues with the run without training in Fig. 1(b). We did record an episode of these trained agents playing the game, and it seems that the agents learned some recurrent strategies. In the recorded episode, the right-hand side ship repeatedly went to the left-hand side and stayed in near-perfect overlap with it. This strategy yields relatively high score gains due to the 200-point reward when the other ship dies; a similar strategy was observed with DQN. Shooting-wise however, not much strategy was qualitatively observed in Q-learning. However, this trend seems to not be a common occurrence, no matter what hyperparameters were used. Usually, the curves were more along the lines of Fig. 2.

4.2 Deep Q Network

DQN learned to exploit the competitive aspect of the game to cooperatively maximize the total reward. When one agent is shot by an alien and loses a life the other agent gains 200 points, this is tied for the highest reward in the game. However, the agents learned that if both of them lose a life at the same time they both receive 200 points. This is shown in Fig. 6 So with three lives, the possible total reward without ever shooting an alien doubled from 600 to 1200. So instead of learning the best way to shoot down the aliens, the agents learned the best way to get shot by an alien together.

The DQN agents by learning this double death strategy performed well with an average score of 1170 in the test runs. Doing so in an average time of 1728 steps. The use of experience replay in training the model helped reinforce the model that the double death strategy was the best option so when testing the agents were focused on overlapping each other.

4.3 PPO and MAPPO

Most of the models trained with PPO and MAPPO had a similar reward curve as shown in Fig. 3. These models struggled to learn a policy that had a significant impact on maximizing the reward function. This could be caused by the complexity of the models, initialized with multiple independent agents. More training and hyperparameter optimizations may be required to train the models to visualize a better trend.

Quantitatively, all the PPO and MAPPO have a similar mean cumulative return compared to the uniform baseline. It seems that using the intrinsic reward function with either PPO or MAPPO to train the agents slightly improved the mean cumulative return over all the episodes in testing. Similarly, incorporating the output of both actor features as input to the critic network in MAPPOA (shown as mMAPPOA in Fig. 4(a) also slightly improved the returns compared to the default MAPPO agents.

Qualitatively based on our sampled video outputs of the agent testing runs, PPO agents seem to have learned to move toward one side. More interestingly, MAPPO agents seem to have learned to move towards each other as well as trying to match the movement of the other agent, by moving together. This can be advantageous as the hit of both agents at the same time can give each agent 200 points.

4.4 Comparisons

All the comparison results are shown in Fig. 4.

When comparing all algorithms together all of our implementations performed at least as well as the random baseline. Q-learning and DQN are two outliers with both achieving significantly higher rewards than the rest. This difference is most likely caused by the learned double-death strategy with faster shots upon reaching the edge.

When comparing the intrinsic vs extrinsic reward for PPO and MAPPO, intrinsic is the clear winner on average having a total reward that is 50 points higher than when based on the extrinsic reward.

When comparing step counts most are around the same being 125 ± 25 more than our baseline. There are two exceptions to this with the first being intrinsic MAPPO having 300 more steps than our baseline and Q-Learning which has 1050 more steps than our baseline. MAPPO having more step count makes sense as it has learned to maximize the step count as its intrinsic reward. Q-learning has an even larger step count as it has on average been able to surpass the first round of the game and was able to more consistently progress to the second round compared to the other algorithms.

The discounted reward shows the DQN may have learned a policy in which both agents choose to get hit faster (higher earlier reward) compared to Q-learning which focuses more on surviving but also maintaining its cumulative reward.

5 Conclusions

We had some non-algorithm-specific limitations in our work. The main one was the lack of sufficient computing resources for efficient experimentation. Another one was the fact that our code base was split throughout multiple notebooks, and thus our environment setups were slightly different.

For Q-learning, a complete redesign of the feature vector would be required for the algorithm to have any chance to perform reasonably well. Here are some ideas. The simplest would be to reduce the frame size in order to increase the impact of individual pixels, especially as the dot product does not lend itself well to learning representations of 2D states and 2D relationships between pixels. Another idea would be to explore ways to reduce the state description size into a few features. One such way would be to somehow process the RGB image to extract information from it such as ship positions based on e.g. detecting clusters of pixels of a certain colour. This would effectively circumvent the issue of learning 2D representations from an image.

DQN was run on Google Colab which came with limitations on GPU ram and idle time. This limited the size and complexity of the neural networks as the training time had to be within the limit otherwise all data would be lost. The limitation of the ram reduced how big the replay buffer could be as the past experience tuples were stored in memory. Would be interesting to test a more complex network with a larger replay buffer and train for longer to see how the results would compare to the current implementation. Fig. 5 shows that training size can have a big impact on the reward. Future work could compare this implementation to different versions of DQN such as Double or Dueling DQN.

As mentioned above for PPO and MAPPO, further experiments on hyperparameter and neural network architecture can be further tested to train the models. More experiments on MAPPOA (where the critic network takes into account the actions of each agent) with intrinsic reward can be conducted. Different methods of preprocessing of the environment state input such as different numbers of frames and input sizes can also be experimented with.

Exploration in terms of offline-RL, learning from expert input data like human input using the above algorithms can also be an interesting way to pre-train and initialize their online training.

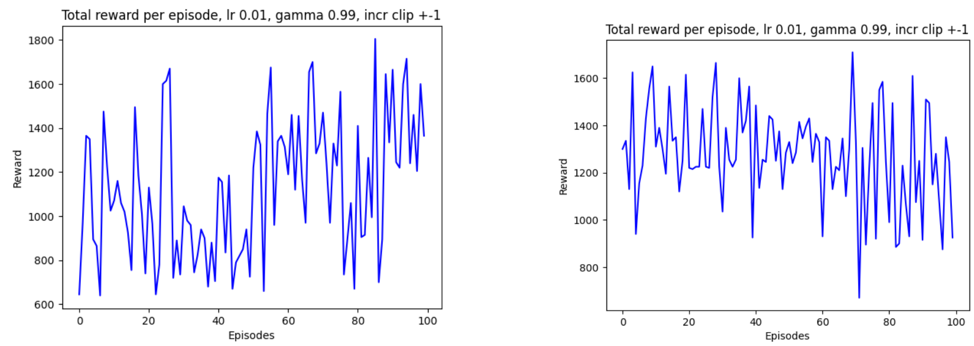
Overall more experiments need to be conducted with each algorithm to verify the hypothesis of the results with a higher degree of confidence.

Further research can also be conducted with self-supervised learning using contrastive representations to predict and align with an agent's future moves and state, resulting in better strategies for exploiting/cooperating with other agents as described in[19] or aligning with one's own policy more efficiently as well as improving the agent's generalization capabilities shown in[20]. MARL in relation to self-supervised contrastive representation learning techniques, given enough data can train more robust and adaptive agents. The impact of these advancements can potentially help develop safe, cooperative and aligned AI systems.

6 Contributions

Everyone contributed to the write-up and the presentation of the project relative to each part described below. Jizhou worked on the project and environment setup, implementation, and experimentation of PPO, and MAPPO algorithms with different network structures and reward functions. Philippe worked on the setup of the environment in Google Colab, and the implementation and experimentation of the Q-learning algorithm. Andrew worked on the implementation and experimentation of the DQN algorithm.

Appendix



(a) 100 episode training of agents

(b) 100 episode ϵ-greedy run of trained agents

Figure 1: Q-learning with LFA and ϵ-greedy performance over training and running. Rewards of both agents were summed at each step.

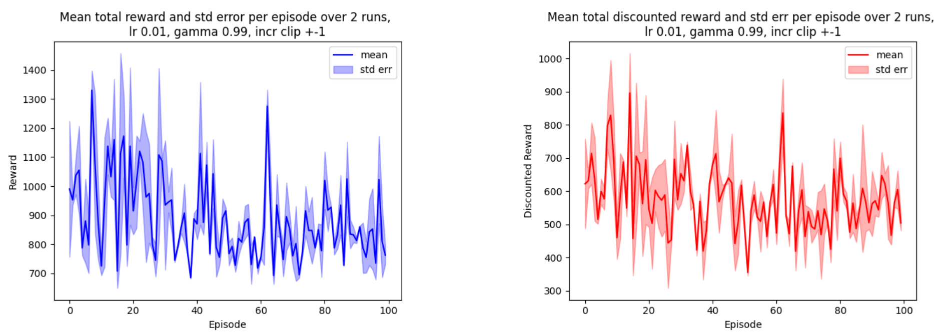


Figure 2: Q-learning with LFA and ϵ-greedy mean total rewards and discounted total rewards over two independent runs of training, with standard error over runs. Rewards of both agents were summed at each step.

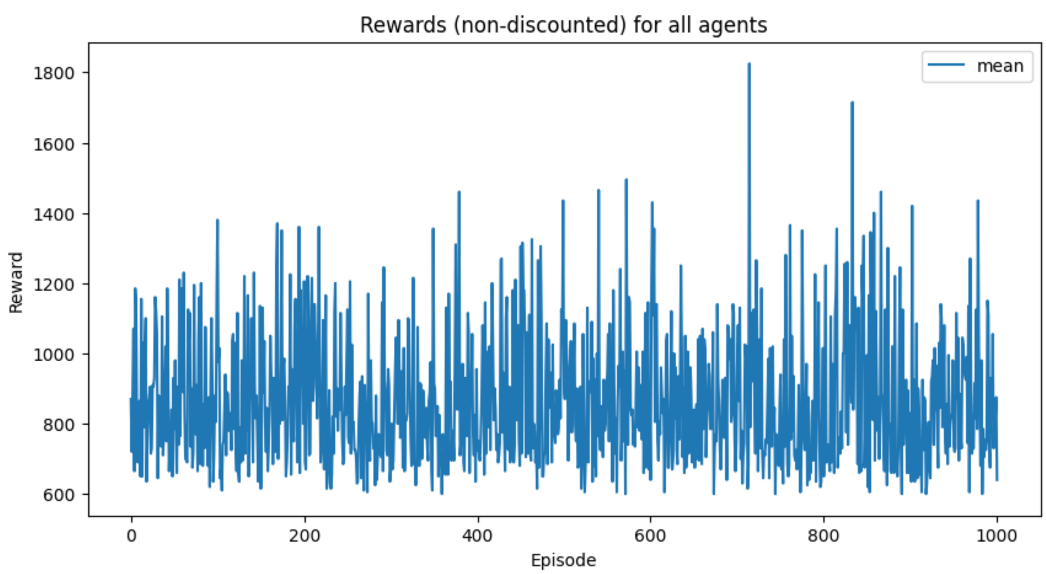
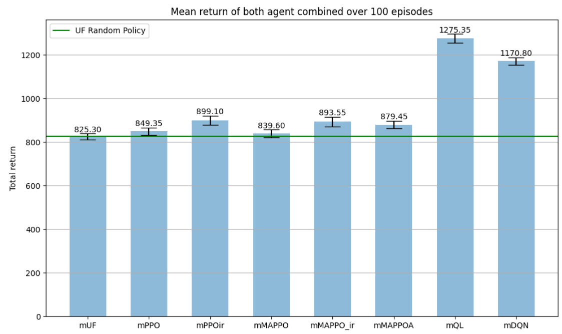
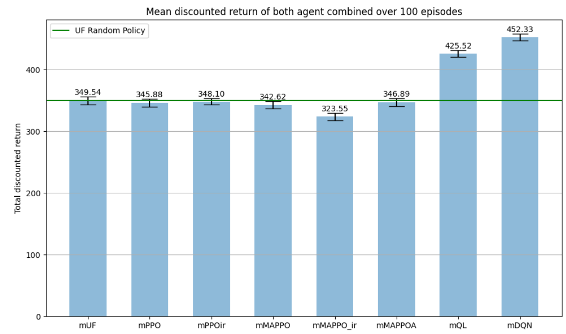


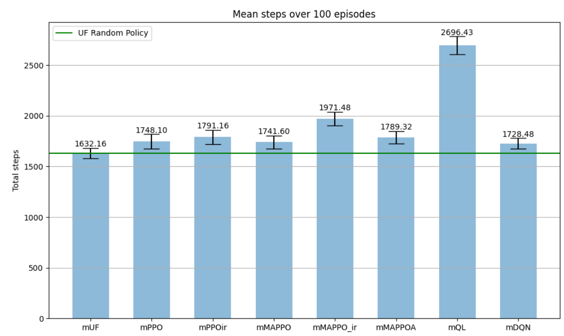
Figure 3: PPO total reward learning curve



(a) Mean reward over 100 episodes



(b) Mean discounted reward over 100 episodes



(c) Mean steps per episode over 100 episodes

Figure 4: Comparision of the mean rewards, discounted rewards, and step count of all our MARL algorithms after 100 episodes of testing.

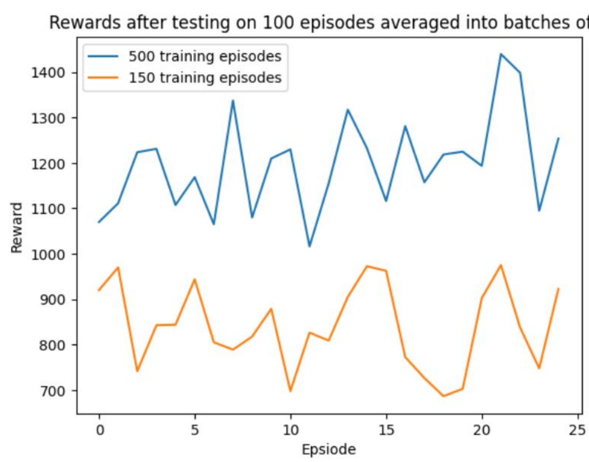


Figure 5: Effect that the training size has on the DQN rewards when tested on 100 episodes, where rewards are averaged in 4 episode batches in order to reduce noise.

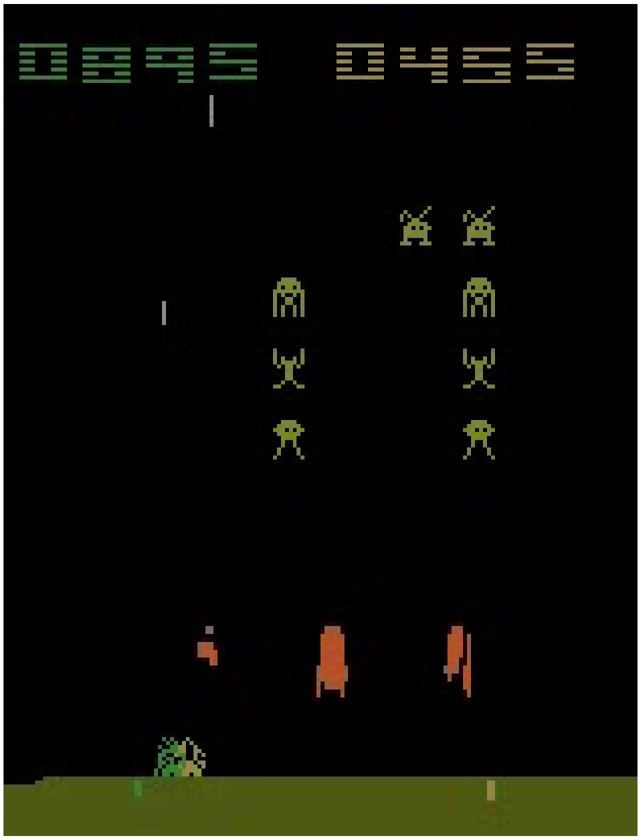


Figure 6: Double hit reward mechanic

References

[1] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[2] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[3] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[4] Chao Yu et al. *The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games*. 2022. arXiv: 2103.01955 [cs.LG].

[5] Justin K. Terry et al. "PettingZoo: Gym for Multi-Agent Reinforcement Learning". In: *CoRR* abs/2009.14471 (2020). arXiv: 2009.14471. URL: https://arxiv.org/abs/2009.14471.

[6] Farama Foundation. *Space Invaders PettingZoo Environment Documentation*. 2022. URL: https://pettingzoo.farama.org/environments/atari/space_invaders/.

[7] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG].

[8] Shengyi Huang et al. *A2C is a special case of PPO*. 2022. arXiv: 2205.09123 [cs.LG].

[9] Junling Hu and Michael P. Wellman. "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm". In: *International Conference on Machine Learning*. 1998.

[10] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms". In: *Handbook of Reinforcement Learning and Control*. Ed. by Kyriakos G. Vamvoudakis et al. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-60990-0. DOI: 10.1007/978-3-030-60990-0_12. URL: https://doi.org/10.1007/978-3-030-60990-0_12.

[11] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362 (Dec. 2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[12] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172. DOI: 10.1109/TSMCC.2007.913919.

[13] Natasha Jaques et al. "Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 3040–3049. URL: https://proceedings.mlr.press/v97/jaques19a.html.

[14] Tonghan Wang et al. "RODE: Learning Roles to Decompose Multi-Agent Tasks". In: *CoRR* abs/2010.01523 (2020). arXiv: 2010.01523. URL: https://arxiv.org/abs/2010.01523.

[15] Chao Yu et al. *Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms*. 2021. URL: https://openreview.net/forum?id=t5lNr0Lw84H.

[16] raghmura. *Space invaders challenge: A reinforcement learning competition*. Apr. 2020. URL: https://wandb.ai/raghmura/qualcomm/reports/Space-Invaders-challenge-a-Reinforcement-Learning-competition--Vmlldzo5MzEzMg.

[17] Ken Ming Lee, Sriram Ganapathi Subramanian, and Mark Crowley. "Investigation of independent reinforcement learning algorithms in multi-agent environments". In: *Frontiers in Artificial Intelligence* 5 (2022). ISSN: 2624-8212. DOI: 10.3389/frai.2022.805823. URL: https://www.frontiersin.org/articles/10.3389/frai.2022.805823.

[18] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[19] DiJia Su et al. "Competitive Multi-Agent Reinforcement Learning with Self-Supervised Representation". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 4098–4102. DOI: 10.1109/ICASSP43922.2022.9747378.

[20] Benjamin Eysenbach et al. *Contrastive Learning as Goal-Conditioned Reinforcement Learning*. 2023. arXiv: 2206.07568 [cs.LG].