Praktikum aus Künstlicher Intelligenz
Lab Report of Group AC

Birger Clausen,[1] **Christoph Dickmanns,**[1] **Lukas Schäfer,**[1] **Paul Schneidler,**[1] **Xiangyu Jia**[1]
[1]TU Darmstadt
birger.clausen@stud.tu-darmstadt.de, christoph.dickmanns@stud.tu-darmstadt.de, lukas.schaefer.64@stud.tu-darmstadt.de, paul.schneidler@stud.tu-darmstadt.de, xiangyu.jia@stud.tu-darmstadt.de

Abstract

We deal with the implementation of a machine learning approach in the context of the game Pommerman. We describe the implementation of a variation of the so called Actor-Critic approach, the Advantage-Actor-Critic approach (A2C) in the Pommerman team environment with a partially observable state space and communication possibilities. Our communication protocol relies on building an estimated board with positions of the teammate and enemies for the unobservable part. Additionally we deal with the needing of an action filter to prevent unintended behaviour of the agents, such as killing themselves, furthermore we address the problem of finding appropriate rewards and training the agents properly.

1 Introduction

Pommerman (Resnick et al. 2018) is based on the famous game Bomberman and enables the possibilty to easily make use of machine learning techniques. The game consists of a board with a discrete size of 11x11 fields. On this board four players, so called agents, compete against each other by trying to eliminate the opponents by placing bombs. Each of those agents starts in one of the four corners and needs to find a way towards the opponents. Walkable corridors are initially blocked by destroyable wooden walls, hence these walls need to be destroyed by placing bombs as well. Sometimes a wooden wall can reveal one of the three powerups shown in table 1. An exemplary board at the beginning of a game is shown in figure 1.

Table 1: Possible Powerups

Extra Bomb	Increases amount of bombs by one
Increase Range	Increases the explosion size by one
Can Kick	Enables the ability to kick bombs in a corridor

All possible actions for an agent are up, down, left and right, as well as doing nothing or placing a bomb. These movements are encoded by numbers from 0 to 5. There are multiple game modes available in Pommerman. The one we use for this work is the *Radio Competition Mode*. Here the agents team up in a group of two and can only observe a 6x6 part around them while the rest of the board is covered by fog. Additionally in each step an agent can sent a message over a communication channel to its team mate. Our implementation of that is discussed further in section 3. The goal for an AI in this scenario is to find the best possible action for a given state and win the game by eliminating all opponents. For our work we decided to rely on the so called Actor-Critic-Approach or more precisely the Advantage-Actor-Critic-Approach (A2C) which is explained in more detail in section 3.

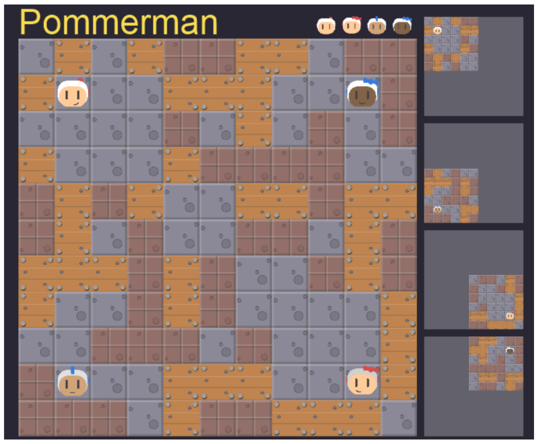

Figure 1: The 11x11 board at the beginning of a game. The partially observed part of each agent can be seen on the right.

Structure of this work In section 2 a brief overview of the related work to our implementation is given. Next in section 3 the basics of the Actor-Critic algorithm, the communication part as well as the importance of an action filter are described. In section 4 follows the description of the neural network architecture, we explain in detail how the state representation is built and how the communicated information is added to it, also the meaning of the messages is presented and reward shaping is addressed. Finally the results are evaluated in section 5 and a conclusion as well as possible future work is given in section 6.

2 Related Work

Concerning the implementation of the Advantage-Actor-Critc we had to rely primarily on our own work regarding the addaptation to the Pommerman environment because no related work for that specific problem had already been done in depth. Instead, we relied on other use cases such as the one described in (Suran, A. 2020), which addresses the Cart-Pole problem from OpenAI Gym. A promising approach for the state representation has been mentioned in (Meisheri et al. 2020), additionally they address a possibility for the communication part in the partially observable environment, as the communicated information is used to build up a 'believe state'. That is an estimated board state, which can additionally be fed into the neural network. Our network architecture is loosely based on the work in (Walther, K. 2019) and the *Skynet-Agent* described in (Gao et al. 2019) but a little more simplified. For reward shaping and especially assigning intermediate rewards, we refer to the approach described by Shelke et al. in (Shelke et al. 2021).

3 Approach

As already mentioned, our approach relies on a machine learning algorithm called Advantage-Actor-Critic. In the following section the necessary basics for understanding this approach are described.

In general an *Actor-Critic* algorithm is a policy based reinforcement learning technique. That means an agent is learning a specific policy, which it follows to maximize its reward throughout the game. The policy, which is also called *actor*, is made of a probability distribution which gives the probabilities of each possible action that can be taken, given the current state. In addition to that a value function is needed as well. This value function is referred to as the *critic*. As the name suggests it criticizes the chosen action from the policy in a given state, whether it was good or bad (n e) and hence allows for adjusting the probabilities for the given state/action pair. The update rule is given with:

$$\Delta J(\pi) = E_T\left[\sum_{t=0}^{T} \nabla log\pi(a_t, s_t)(Q(s_t, a_t) - V(s_t))\right] \quad (1)$$

It describes the so called policy gradient. The result of this equation is estimated by a neural network which outputs the values of the policy $\pi(a_t, s_t)$ and the critic $V(s_t)$. The Q-value, which other than V, also depends on the action taken at step t can be approximated with the following equation:

$$Q(s_t, a_t) \approx R(s_t, a_t) + \gamma \cdot V(s_{t+1}) \quad (2)$$

This has the benefit that no extra neural network for approximating the Q-Value is needed. The discount is performed with a discount factor γ. This is done because a game in Pommerman has a potentially large amount of steps, hence a discount factor allows for rewarding earlier actions more than later ones. The latter part of equation 1 is some sort of temporal difference known from other reinforcement algorithms. In this case it is called the *advantage*: $A = Q(s_t, a_t) - V(s_t)$. In other words that means that V estimates the predicted rewards the agent will achieve following the given policy. Hence the difference between Q and V switches signs when the expected reward is larger or less than the actual achieved one. This allows for increasing or lowering the probability of choosing an action given a specific state/action pair and therefore maximizing the reward. The critic is simply updated by calculating the mean-squared-error between the actual reward and the expected reward. The data flow between the individual components is shown in figure 2.

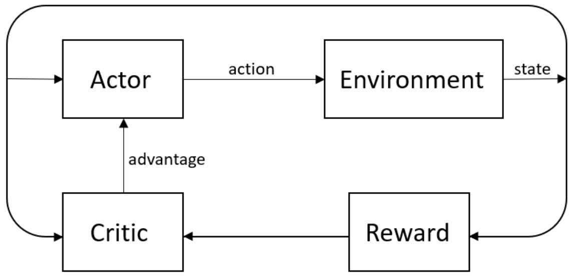

Figure 2: The dependencies between all involved parts of the A2C approach (a o). The Actor outputs an action based on the current state of the Environment. This state is updated based on the action. Then the reward is calculated and the actor updated based on the advantage between the Critic value and the reward.

To prevent the agents from eliminating themselves in the beginning because of random bomb placements we added an action filter based on the work in (o a). This allows for directing the agents towards a better policy right from the beginning. The filter manually checks the surrounding of an agent and intervenes when an action chosen based on the policy would lead to death, e.g. by stepping into a bomb's explosion range, additionally this filter allows for observing the past two states to be able to recognize kicked bombs. Without using this action filter the agents start to stand still after a while which can be interpreted as a strategy of doing nothing and instead hoping for the enemies to kill themselves.

As for the communication part the agents have the ability to sent an integer tuple with each entry being a value between 1 and 8 to their teammate in each step. This leads to 64 possible permutations which can encode different meanings. For our approach, we used 50 out of those 64 possibilities to encode potential positions of enemies in the quadrants of the board. Additionally the own position of the agent sending the message in the upper or lower half of the board is given as well. The meanings of those messages are given in section 4.

4 Technical Details

In the following part details about the neural network architecture as well as the reward shaping problem and commu-

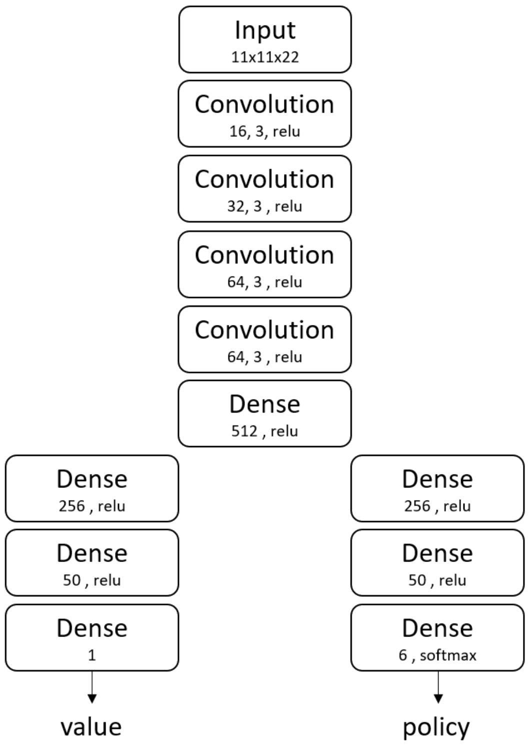

Figure 3: Architecture of the neural network. The first few layers are shared between actor and critic and use convolutions with 16, 32 and 64 filters with a 3x3 window each and a 'relu' activation function. Then the network splits into two parts for the two outputs with fully connected dense layers on each path. For the policy output a 'softmax' activation function is used to receive probabilities for each action.

nication encodings are presented.

4.1 Neural Network Architecture

Our neural network architecture is shown in figure 3. We decided to use a single neural network for approximating both, the actor and the critic instead of using two separate networks. This is done because of two main reasons. First it is easier and more efficient to train a single neural network instead of two. Second it is shown that the first few layers of two separated networks tend to represent the same features of the input and as such, the logical consequence is to simply bring them together. A convolutional neural network is used because it is assumed that it allows to recognize patterns in board states similar to the use case of object detection in images.In addition, to let the output of softmax function avoid being 0 or 1, we clipped the output policy values which will be approximated to 0 or 1 (Walther, K. 2019).

As shown in figure 3 the input consists of a 11x11x22 state representation. The 22 channels consist of the information shown in table 2. Wherever possible a one-hot or binary encoding is used for a channel to prevent the assumption of linear dependencies between each channel. We decided not to use a centered state representation around each agent. This might have been beneficial to reduce the otherwise huge state space but as already mentioned we build a believe state from the messages to encode possible positions in the foggy area of partners and enemies. A centered state would be obstructive in this case. The approximated positions are filled in the respective channels 12-15 as described in table 2. This is kind of vague because if an agent vanishes in the fog, his possible positions are constrained by the distance he could have traveled in the passing time steps since his disappearance. We assume for now that every position in the quadrant is a valid one.

Table 2: Channel encodings

Channels	Meaning
0-2	passsages, rigid walls, wooden walls
3-5	bombs, flames, fog
6-8	power-ups
9-11	bomb life, bomb blast strength, bomb moving direction
12-15	position of agents
16-18	alive encoding of other agents
19-21	ammo and abilities

4.2 Communication

The radio communication gives information about the enemies that are currently in the field of view, of the agent sending the message. As there are only 64 possible messages that can be sent, the position of the enemies can not be communicated precisely with concrete coordinates. Instead we encode the respective quadrant in the message. The quadrants are numbered in mathematical fashion counterclockwise, starting from the top right. Additionally the own position of the agent sending the message is given whether being in the upper or lower half of the board. In the following table 3 all used message encodings are given in pairs of numbers from one to eight. As for the meaning: the first entry of a cell denotes the position of the first enemy, while the second entry denotes the position of the second enemy. The third entry specifies the position of the agent sending the message. In this case 'q1' to 'q4' gives the respective quadrant while 'u' and 'l' have the meaning of upper and lower. A 'n' at any position means that no information about the respective agent is present.

Table 3: Communication encodings

	1	2	3	4	5	6	7	8
1	q1,q1,u	q1,q1,l	q2,q1,u	q2,q1,l	q3,q1,u	q3,q1,l	q4,q1,u	q4,q1,l
2	q1,q2,u	q1,q2,l	q1,q3,u	q1,q3,l	q1,q4,u	q1,q4,l	q2,q2,u	q2,q2,l
3	q2,q3,u	q2,q3,l	q2,q4,u	q2,q4,l	q3,q2,u	q3,q2,l	q3,q3,u	q3,q3,l
4	q3,q4,u	q3,q4,l	q4,q2,u	q4,q2,l	q4,q3,u	q4,q3,l	q4,q4,u	q4,q4,l
5	q1,n,u	q1,n,l	q2,n,u	q2,n,l	q3,n,u	q3,n,l	q4,n,u	q4,n,l
6	n,q1,u	n,q1,l	n,q2,u	n,q2,l	n,q3,u	n,q3,l	n,q4,u	n,q4,l
7	n,n,u	n,n,l						
8								

4.3 Reward Shaping

Initially we tried to train the network only with the default final rewards given by:

- win: +1
- lose: -1

With this approach, the reward for the agent depends only on the outcome of the game. This however, did not allow us to directly influence the behavior of our agent. Thus, it is possible that he learns a strategy that is not suitable for good performance (e.g., camping). This problem may be exacerbated in the case of multi-agents, that we face (Castellini et al. 2020), (Shelke et al. 2021). So to improve the performance of the agents in multiple points, intermediate rewards are added. First of all we introduce rewards for exploring unvisited cells. We assume that this will help encourage the agent for more exploration than exploitation. We also set up the reward for getting closer to the enemies, as our action filter is set aggressively, this would help the agent kill enemies.In addition, we give a small reward for destroying wooden walls, which should help with exploration and should encourage the agents to search for power-ups. Picking up those power-ups gives a reward, too. To let the agents learn a basic way of team-play there are coupled rewards for both agents which are related to eliminating an enemy and eliminating the partner (Chen et al. 2020). The intermediate rewards are shown in table 4.

In the end, we also give a final reward for winning or losing a game:

- win: +1000
- lose: -1000

We upscaled the final rewards, since in our point of view the result of the games is the most important part. This is combined with the intermediate rewards after each round.

Table 4: Rewards

+2	getting closer to enemies
+5	exploring an unexplored cell
+2.5	destroying wooden walls
+5	picking up power-ups
+50	kill an enemy
-50	kill a partner

5 Evaluation

5.1 Tournaments Results

1st Tournament: disqualified (Due to a problem with Docker)
2nd Tournament: 2nd place
3rd Tournament: penultimate place

In the beginning, our agent lost against the rulebased agent. In these test runs, our agent still had an unoptimized network and was barely trained. As we made improvements to the agent, the first tournament took place. In this tournament, our agent was disqualified because the Docker image did not work on the tournament environment. We were not able to reproduce this bug and therefore we used a lot of time to fix this. The agent had shown a very passive behavior (moving only in its starting corner) in our running environment during the time of the first tournament, and also it kept forgetting to place bombs and placing bombs before it without any strategy or sense. Between the first and second tournament we tried to adjust the action filter, we tried to restrict the agent further in the hope that he would not forget how to place bombs. We discarded these adjustments because it doesn't allow him to learn to place bombs ahead of time. In addition, normal rewards(plus points for a win/minus points for a loss) were added. Also, a communication protocol was added to the agent, which allowed for a meaningful exchange of information with his teammate. With these enhancements, the agent took 2nd place in the 2nd tournament, losing only to the rule-based agent. For the third tournament we tried to improve the network and change the action filter. The rewards were enhanced, the agent now got reward for exploring. Since there were again unexpected bugs with Docker we could only train the agent very little for the third tournament. In addition, we only saved the weights. With these changes we reached the second last place in the third tournament.

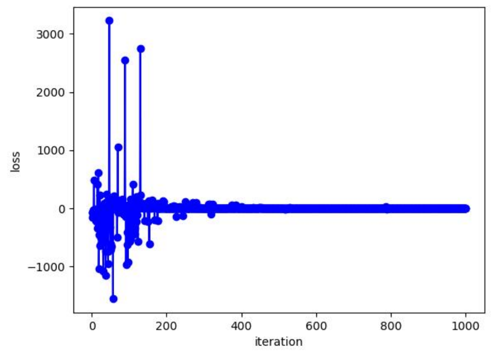

Figure 4: The actor_loss without action filter.

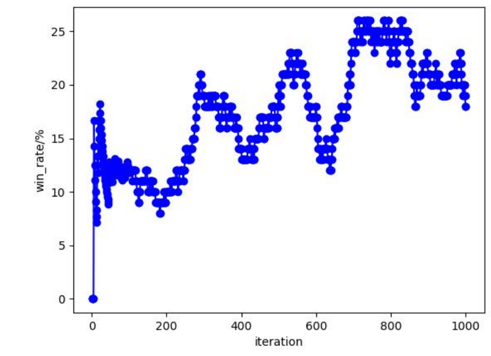

Figure 5: The win_rate without action filter during the first 1000 iterations.

5.2 Training without Action Filter

First, here is the training result of our agents without utilization of an action filter. We set all the actions (0-5) as valid actions. It can be seen from figure 4 that the losses of the actor decreases and the win_rate (Percentage of wins over the last 100 games) has increased overall as shown in figure 5. But actually after 1000 iterations, the agent is very passive and never moves again. So the win_rate of around 25% appears to be equivalent to the suicide rate of the simple agent. The critic loss has some strange behaviour as shown in figure 6, as it is pretty low in the mean but has huge spikes. It is not clear where this behaviour results from.

5.3 Training with Action Filter

By attaching the action filter to the agent, we found that the agents can sometimes win within around 50 steps. The reason for that is always the suicide of the opponents. So we filtered out the iteration with fewer than 50 steps and only trained the model with valid iterations. The win_rate against simple agent was significantly increased to around 70% as shown in figure 7 . From figure 8 and 9, it can be seen that the losses of actor and critic are very low from the beginning. It also shows that there is no actual learning of the agent.

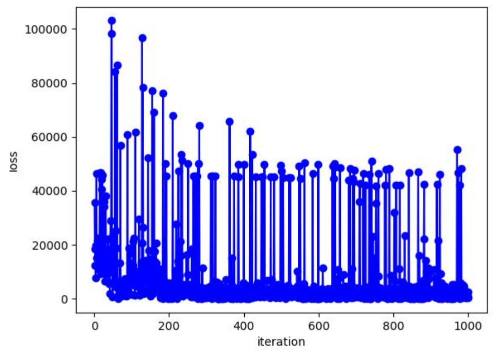

Figure 6: The critic_loss without action filter.

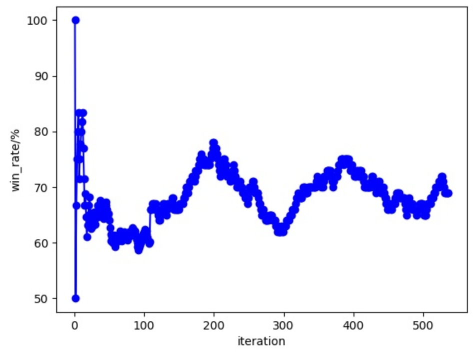

Figure 7: The win_rate with action filter and small rewards.

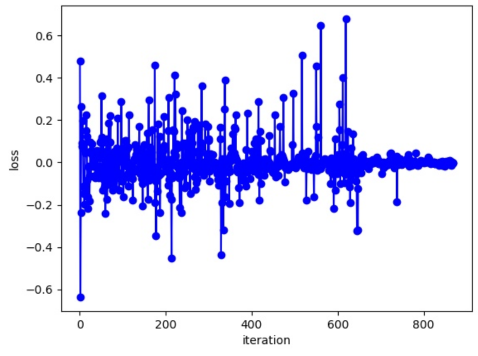

Figure 8: The actor_loss with action filter and small rewards.

5.4 Training with Scaled Rewards

As our loss value is always very small, we tried to scale the rewards up 100 times (the scaled rewards are described in section 4.3). The win_rate in figure 10 is similar to the results with small rewards. The loss results are shown in figure 11 and figure 12. We can conclude that this scaling of the rewards has increased the critic_loss 10000 times and does not have much influence on the actor_loss. Which makes sense,

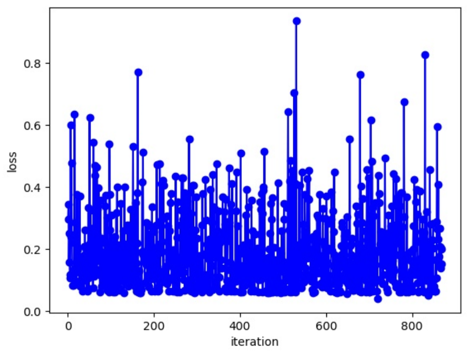

Figure 9: The critic_loss with action filter and small rewards.

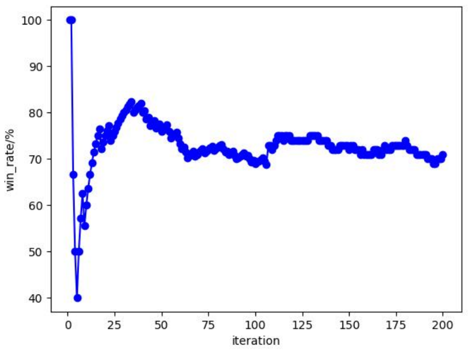

Figure 10: The win_rate with action filter and scaled-up rewards.

hence the critic value describes the expected future reward of the agent. The rate of valid actions chosen by the agent shown in figure 13. This rate is always around 60%, which means that the intervention rate of the action filter is at 40% and hence the agent relies heavily on the filter.

Overall, the performance of our agent is not perfect. The win_rate has reached 70% but it is more likely due to the action filter. The value of our actor_loss is easy to get close to zero(normally only after 100 iterations), so it's hard to optimize through training.

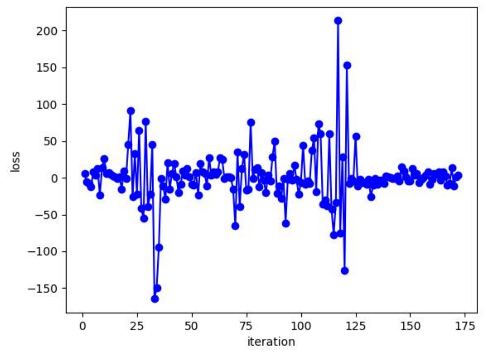

Figure 11: The actor_loss with action filter and scaled-up rewards.

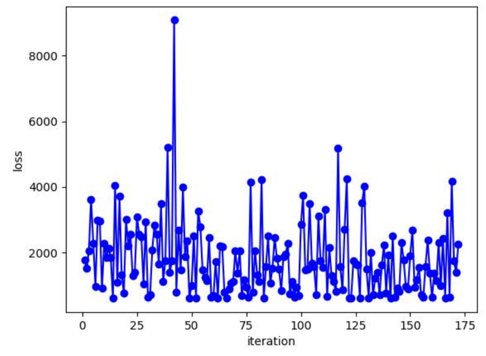

Figure 12: The critic_loss with action filter and scaled-up rewards.

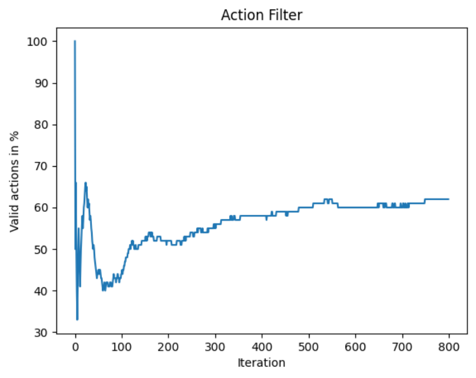

Figure 13: The rate of intervention from the action filter.

6 Conclusion

The goal of this work was to implement two agents in the partially observable Pommerman environment with radio communication. Because for most of us this was the first bigger project in the context of AI, our expectation was to beat the simple agent provided in pommerman consistently and not really to focus on beating the other groups. The final result is that we somehow managed to be competetive with the simple agent, but we assume this is mainly due to interventions of the action filter, that has a intervention rate of greater 40% and the simple agent not working that great with the partially observable state space. The problem which is yet to be solved, is that our agents do not really learn over time. The calculated loss to update the neural network is small from the beginning and oscillates around zero which is definitely an issue. As shown in other works the Actor-Critic approach should be one of the best performing ones in this context and with more fine tuning our agents should be able to perform much better. One approach for improving would be to adjust the state representation. As mentioned in section 4 we have a pretty vague way of approximating the expected position of agents communicated over the radio. One idea is to give possibilities of enemy positions that update at each time step and get less likely the more time elapses from the last time the respective agent has been seen.

References

Castellini, J.; Devlin, S.; Oliehoek, F. A.; and Savani, R. 2020. Difference Rewards Policy Gradients URL http://arxiv.org/abs/2012.11258.

Chen, B.; Qiu, J.; Raj, S. K.; Shi, S.; and Cheng, W. 2020. Pommerman Fight: A Detailed Analysis of Reinforcement Algorithms URL https://csci599-s2020-pommerman.github.io/pdf/paper_final.pdf.

Do, V. Q.; and Koo, I. 2018. Learning frameworks for cooperative spectrum sensing and energy-efficient data protection in cognitive radio networks. *Applied Sciences* 8(5): 722.

Gao, C.; Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2019. Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition. In *4th Multidisciplinary Conference on Reinforcement Learning and Decision Making*.

Meisheri, H.; and Khadilkar, H. 2020. Sample Efficient Training in Multi-Agent Adversarial Games with Limited Teammate Communication. *arXiv preprint arXiv:2011.00424* .

Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124* .

Shelke, O.; Meisheri, H.; and Khadilkar, H. 2021. School of hard knocks: Curriculum analysis for Pommerman with a fixed computational budget. *arXiv preprint arXiv:2102.11762* .

Suran, A. 2020. Actor-Critic With TensorFlow 2.x. URL https://towardsdatascience.com/actor-critic-with-tensorflow-2-x-part-2of-2-b8ceb7e059db.

Tensorflow. 2022. Playing CartPole with the Actor-Critic Method. URL https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic.

Walther, K. 2019. Pommerman: Multi-Agent Learning Using Actor-Critic Methods. *Technical University of Denmark* .