Multi-Agent Strategies for Pommerman

Dhruv Shah [*1] **Nihal Singh** [*2] **Chinmay Talegaonkar** [*1]

Abstract

Despite the advances in reinforcement learning in a wide variety of applications, the domain of multi-agent systems remains vastly unexplored. In this work, we discuss strategies for approaching multi-agent reinforcement learning problems by the tool of Pommerman, an online multi-agent research testbed. We present and evaluate algorithms for gameplay involving independent and coordinated agents and discuss practical concerns.

1. Introduction

Reinforcement learning (RL) has recently been succesfully applied to solve challenging problems, from game playing (Mnih et al., 2015) to robotics (Matas et al., 2018). Most of the successes of RL, however, have been in single agent domains, where either the environment is stationary or modeling/predicting the behaviour of other actors in the environment is largely unnecessary.

However, there are a number of important applications that involve interaction between multiple agents, where emergent behavior and complexity arise from agents co-evolving together. For example, multi-robot exploration (Matignon et al., 2012), multiplayer games (Peng et al., 2017) etc. all benefit from a multi-agent perspective of the system. Multi-agent modeling has also shown tremendous potential in efficient routing in communication networks and sensor networks. Additionally, the recent demonstrations of multi-agent self-play in *AlphaGo* (Silver et al., 2016) has paved way for a new training paradigm. Successfully scaling RL to environments with multiple agents is crucial to building AI systems that can productively interact with each other.

Unfortunately, traditional reinforcement learning approaches such as Q-learning or vanilla policy gradient are poorly suited to multi-agent environments. A multi-agent system faces various challenges which makes it difficult to predict the outcome of a particular plan and thus complicates finding good plans: (i) *outcome uncertainty*, due to non-stationarity of the environment from the perspective of any individual agent in a way that is not explainable by changes in the agent's own policy (Oliehoek & Amato, 2016); (ii) *state uncertainty*, due to partial observability, corrupted/faulty sensor measurements etc. which further complicates the planning and control problem, often leading to *perceptual aliasing*; (iii) *huge joint action space*, for the multi-agent Markov game (Littman, 1994) thus formed. All routines are exponential in the size of action space; (iv) *credit assignment* to individual agents, especially when the environment offers sparse, common rewards to the system (Minsky, 1961). These present learning stability challenges and prevent the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning. Policy gradient methods, on the other hand, usually exhibit very high variance when coordination of multiple agents is required. Model-based policy optimization can be used to learn optimal policies via back-propagation, but this requires a (differentiable) model of world dynamics and assumptions about the interactions between agents.

While a plethora of methods employing clever learning techniques have been proposed recently, most of them are extremely data and compute hungry. Over shorter training periods, such an end-to-end learning method also fails to use domain knowledge and tries to learn strategies by exploring. Towards this direction, we explore algorithms for RL in environments with multiple agents with a focus on minimizing the amount of time and compute resources required for training by using domain knowledge and supervision. We use the challenging yet interpretable gaming environment of Pommerman (Resnick et al., 2018), which comprises a four-way *battle royale* gameplay, for deploying and testing our strategies. We split our methods into two broad directions: (i) individual gameplay, where the goal of the agent is to be the last one remaining and no cooperation/competition is involved, and (ii) 2x2 team-based gameplay, where teams of two agents each compete to be the last team standing.

The remainder of the report is organized as follows: in section 2 we present preliminaries to analyze a multi-agent system and discuss related work. Section 3 describes the problem setup and choice of features. Section 4 presents methods, evaluation and discussion of strategies for the indi-

[*]Equal contribution [1]Department of Electrical Engineering, Indian Institute of Technology Bombay, India [2]Department of Civil Engineering, Indian Institute of Technology Bombay, India. Correspondence to: <{dhruv.shah, nihal111, chinmay0301}@iitb.ac.in>.

vidual four-way gameplay, and section 5 presents methods and discussion of strategies for 2x2 teamplay. Finally, we conclude with a discussion of implications, challenges and future directions of work in section 6.

2. Preliminaries

2.1. Q-Learning

Q-Learning (Watkins & Dayan, 1992) has been central to the task of control for a Markov Decision Process $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$. The objective here is to learn the *action value function* $Q^\pi(s, a)$ for policy π by minimizing the expected loss $\mathcal{L}(\theta)$.

$$\mathcal{L}(\theta) = \mathbb{E}_\pi\Big[(Q_\theta(s,a) - \underbrace{(r + \gamma \max_{a'} Q_{\theta'}(s', a'))}_{\text{target value}})^2\Big] \quad (1)$$

Q-Learning in its vanilla form tends to be highly unstable especially in the form where Deep Neural Networks (DNNs) are used for function approximation. Deep Q-Learning (Mnih et al., 2015) was proposed to overcome this problem by introducing the notion of target network whose parameters are kept constant for a finite number of training steps and is used to generate the values for y. The other technique used to stabilize the performance and overcome the problem of catastrophic forgetting is to use an experience replay buffer (Lange et al., 2012) from which transitions are sampled at random.

2.2. Policy Gradient Methods

Policy Gradient (PG) methods differ from Q-Learning and other value function estimation methods in the sense that they explicitly learn a stochastic policy π_θ. Choosing the maximization objective $J(\theta)$ to be the expected long-term reward over a trajectory τ induced by π_θ, we get the gradient as

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\big[\nabla \log \pi_\theta(\tau) r(\tau)\big] \quad (2)$$

The gradient comes out to be independent of the environment dynamics and the ergodic distribution. This means we can now just run Monte-Carlo simulations and approximate the gradient to find the best parameters and the computer gradient is an unbiased estimator of the true gradient. Replacing $r(\tau)$ by the discounted returns gives us the REINFORCE algorithm (Williams, 1992).

2.3. PG & High Variance

The objective used in PG methods leads to very high variance models. Any erratic trajectories which produce unusual rewards would cause an unexpected change in the resulting distribution. To mitigate this problem, an idea that helps reduce the variance is to instead maximize an objective which keeps track of the relative reward difference, leading to an algorithm called REINFORCE with Baseline. The gradient thus becomes

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla\Big(\sum_{t=1}^{T} \log \pi_\theta(s_t|a_t)\Big)(G_t - b)\Big] \quad (3)$$

where b is the introduced baseline. Choosing the right baseline is a task in its own right, and various methods resolve this using a parametrized estimate of the value function $V^\eta(s)$ as the baseline, commonly known as the *critic* – such a model is called an Actor-Critic (AC) model. The difference of the objective and the baseline is also known as the *advantage estimate*.

The problem of high-variance is exacerbated in multi-agent settings since an agent's reward usually depends on the actions of several other agents. Trust region policy optimization methods (Schulman et al., 2015) maximize an objective function similar to simple PG subject to a constraint on the size of policy update. However, TRPO is relatively complicated and not compatible with architectures that include parameter sharing. Proximal policy optimization family of algorithms (Schulman et al., 2017) attain the data efficiency and performance of TRPO while using more efficient optimization and generalize to a wider range of applications. In order to stay close to the current policy at each step, PPO optimizes a clipped objective as follows

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\big[\min(r_t(\theta)\hat{A}(\theta), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon))\big] \quad (4)$$

where $r_t(\theta)$ is the ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, and ϵ is the clipping hyperparameter.

2.4. Multi-Agent AC

As discussed in section 1, the action space in a multi-agent system grows exponentially with the number of agents. This necessitates the need of a decentralized policy which only depend on the local observations of the agents (Kapoor, 2018). This also helps account for complications owing to partial observability. *At training time*, such a system of decentralized agents (actors) can be augmented with a centralized critic providing indirect observations in a laboratory setting. The theoretical foundations governing this framework are covered in detail in (Oliehoek & Amato, 2016). Two recent approaches of interest in this direction use (i) counterfactual baselines for an agent a which allows the centralized critic to reason about counterfactuals in which only a's actions change (Foerster et al., 2018), and (ii) infers policies of other agents in an AC framework and is able to learn policies involving complex multi-agent coordination (Lowe et al., 2017).

2.5. Imitation Learning

While deep RL systems have achieved excellent performance in several difficult decision-making problems, they tend to be data hungry. In fact, their performance during learning can be extremely poor. This may be acceptable for a simulator, but it severely limits the applicability of deep RL to many real-world tasks, where the agent must learn in the real environment. *Imitation learning* algorithms concern with matching the performance in simulation. A popular algorithm, DAGGER (Ross et al., 2011), iteratively produces new policies based on polling the expert policy outside its original state space, showing that this leads to no regret over validation data in the online learning sense and requires the expert to be available during training to provide additional feedback to the agent. The early successes of AlphaGo (Silver et al., 2016) rely on pre-training from an expert human database, before interacting with the real task. Deep Q-Learning from Demonstrations (DQfD) (Hester et al., 2017) is a recent method that leverages small sets of demonstration data to accelerate the learning process, by combining temporal difference updates with supervised classification of the demonstrators action.

3. Setup

Pommerman[1] is a testbed for multi-agent reinforcement learning, based on the popular game Bomberman. Every game is played on a 11×11 game with 4 agents, each spawned at the corners of the grid. The grid can be free, or have rigid or wooden walls in each block. Rigid walls are indestructible and an agent can neither pass through those walls nor blast them through a bomb. The wooden walls can broken by a bomb, and may release a fixed set of power-ups. Until the wooden walls are broken, they cannot be traversed through. More details on the environment can be found in appendix A.

There are 2 game settings in which the game can be played, namely *Free-For-All* and *Teamplay*. In FFA or *Battle Royale*, the 4 agents play against each other with the aim of longest survival, i.e. the agent who survives till the end wins and gets a reward of 1. The other agents get a reward of -1. The Teamplay variant involves the players playing in teams of 2 without communicating with each other. Both the players get a reward of 1 if their team wins, -1 if they lose or 0 otherwise.

Feature Engineering

For an agent in the Pommerman environment, the state space consists of its resources, state of the grid and location of bombs, agents and power-ups. Owing to the effects of actions by multiple agents, the environment is largely non-Markovian. We use a finite visual field of depth to form a reduced perceptual state of the system (Tan, 1993). Note that while this induces partial observability, it helps ameliorate non-Markov nature of the whole environment and explicitly helps an agent to focus on its surroundings for taking the next action. For the purpose of experiments, we use a visual field of depth 2, i.e., a 5×5 reduced perceptual state (see figure 1). Table 1 lists our choice of features and corresponding dimensions for a $p \times p$ perceptual state. Note that this state space always has the agent at its centre, and positions are relative to the agent.

[1]Read more at https://www.pommerman.com/

Figure 1. Reduced perceptual state of an agent with a visual field of depth 2

Feature	Dimension
Enemy positions	$p \times p$
Rigid wall blocks	$p \times p$
Wooden wall blocks	$p \times p$
Passage blocks	$p \times p$
Bomb life	$p \times p$
Active flame blocks	$p \times p$
Bombs	$p \times p$
Power-ups	$p \times p$
Ammunition count	1
Bomb strength	1
Kick	1
Total	$(p \times p \times 8) + 3$

Table 1. Choice of features for a Pommerman agent with a perceptual state of size $p \times p$

4. FFA Gameplay

The simplest version of the game consists of four independent agents competing in *Battle Royale* – last agent standing

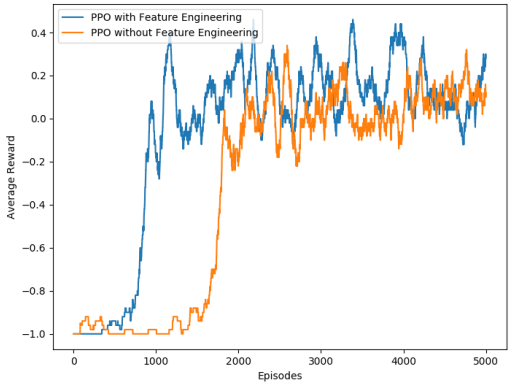

Figure 2. Moving average of reward with episodes while training

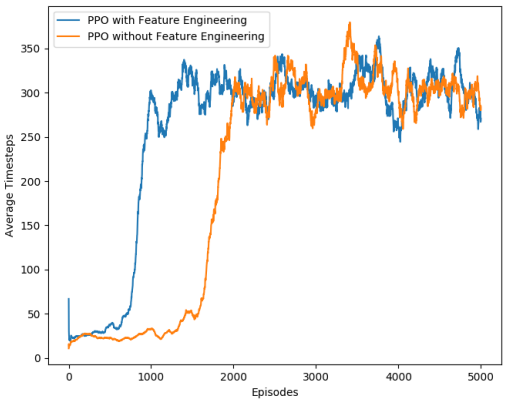

Figure 3. Moving average of timesteps per episode with episodes while training

wins. A rule-based "simple planner" is made available with the Pommerman environment, as a baseline agent. We investigated the performance of the agent after training it using 2 different approaches – PPO and DHqD (see section 2.3). Dependence on choice of feature space and reduced perceptual state was also considered while evaluating the agents.

4.1. PPO Agent

Proximal policy optimization algorithms are well suited for policy optimization on large state/action spaces, like that of Pommerman. We consider a PPO agent using the naive feature set given by the observations of the gaming environment and a dense policy network (for architecture, see figure 4) to learn suitable policies for the FFA gameplay. To improve the performance of the agent, we impose domain knowledge in form of the features discussed in section 3. The agents were trained by playing against a series of random and rule-based opponents. Training curves of the agents with and without feature modifications in terms of average reward and lifespan (timesteps per episode) are shown in figures 2 & 3, respectively.

Evaluation & Discussion

We notice that despite using an improved feature set, the PPO agent seldom manages to beat a single rule-based agent. Exhaustive results with multiple rule-based agents (see tables 2 & 3) highlight the gap in performance as the difficulty increases. Analyzing the gameplay reveals an interesting insight – the agents learn that a bomb is dangerous (and can lead to suicide) and hence deter from using it. The reader is encouraged to view the supplementary video for better understanding (Authors, 2018). *Therefore, the agent does not learn how to win; it simply learns to not die!*

Algorithm	W1	W2	W3
PPO without Feat. Mod	47.5	32	12
PPO with Feat. Mod	48.5	22.5	17.5
DQfD with Feat. Mod	51.6	24.2	20.8
DQfD (small net)	50.1	23.1	19.2
DQfD (deep net)	52.3	25.1	22.6

Table 2. Comparing the winning percentages of the algorithms for the 3 configurations, i.e. against differnt number of simple agents (1,2,3 corresponding to W1, W2 and W3)

Algorithm	T1	T2	T3
PPO without Feat. Mod	282.4	257.6	226.7
PPO with Feat. Mod	256.4	212.2	192.2
DQfD with Feat. Mod	250.5	243.2	228.9
DQfD (small net)	261.5	221.7	200.8
DQfD (deep net)	274.2	265.4	240.1

Table 3. Comparing the average episode duration of the algorithms for the 3 configurations, i.e. against differnt number of simple agents (1,2,3 corresponding to T1, T2 and Th3)

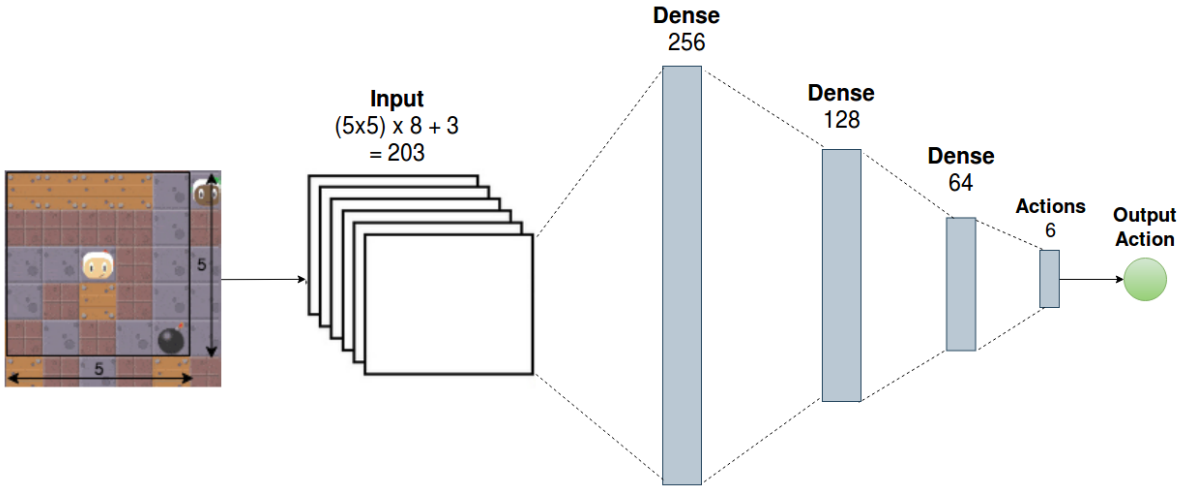

Figure 4. Network architecture of the policy network used for PPO agent. 203 features are derived from the perceptual state of teh agent as per table 1 and passed through 3 dense layers consisting of $256, 128, 64$ neurons each.

4.2. Learning from Demonstrations

To allow usage of domain knowledge and carefully train the agent to not learn undesirable policies, we tried using a supervised, imitation learning approach to the learning problem, like the use of a human expert in AlphaGo. Since the simple agent is quite adept at playing the game, one to way to improve the performance of the learning agent is training it to behave like the simple agent first and then seek subsequent improvements. We generated multiple episodes of 4 simple agents competing against each other, and then used these episodes to pre-train a Deep Q-Learning from Demonstrations (DQfD) agent for 1000 such demonstrations. The feature modifications and the perceptual state were preserved.

After pre-training the DQfD agent and training it in the actual environment, we observe significant improvements over the PPO agent. The video clearly demonstrates the improvements. The DQfD agent explores the board efficiently while learning to plant bombs and evading them. It also learns to kick the bombs and trap other agents into a corner and bombing them. Comparative results in terms of win rate and lifespan of the agents with modifications in network architecture and feature set can be found in tables 2 & 3. The reader is directed to view the supplementary video for further results (Authors, 2018).

5. Towards Teamplay [2]

The team version of the game comprises of two teams of two agents each combatting each other. Partial observability is implemented as a restricted field of depth and communication between agents may be allowed. For the discussion in this section, we do not consider partial observability and communication.

5.1. Non-cooperative Agents

A baseline approach to solving teamplay would require that the agents do not target each other as enemies. In this regard, we consider an approach where both agents are trained and deployed independently, but are invisible to each other. That is, each agent only sees two enemies in the grid. A simple implementation in Pommerman shows that while this ameliorates the risk of attack within a team, no strategy can be learned and a need for a more evolved approach arises.

5.2. Cooperative *Super Agent*

An interesting way to train a pair of agents is to treat them as a cohesive *super agent*. Such an agent models the environment with the same state space of an individual agent and a joint action space given as $\mathcal{A}_s = \mathcal{A}_1 \times \mathcal{A}_2$. Since most RL libraries support only single agent training (see below), this approach is naturally suited for a direct implementation with an optimized library like TensorForce.

[2]This section comprises of ideas encountered during the later half of the project. Since this was not our main focus, we only have preliminary implementations and no strong results to compare. We present a brief summary of our approaches here.

Training such an agent is tricky, since writing a rule-based/heuristic agent for demonstration (as was the case with FFA, see section 4.2) is significantly more complicated than a single agent. To address this, we use a DQfD agent pre-trained on demonstrations by a team of simple agents (rule-based, single player) to learn basic heuristics. This is followed by Q-learning from deployment, in order to better learn strategies and team-specific ideas. Progress is slow on testing these ideas, owing to time constraints and limited resources (in terms of libraries and compute) for multi-agent learning, but we hope to continue exploring this direction in the future.

Remarks on Multi-Agent Training with Tensorforce

Like most RL libraries, TensorForce doesn't support training multiple agents using the same runner object[3]. A possible way to train multiple agents with such a restriction would be to train them in round-robin fashion, with batch updates, i.e. one agent is being trained for a few episodes while the others are held fixed and so on. While this is not the most accurate way to train a multi-agent systems, the heuristic enables ease of training using optimized libraries for testing our hypotheses.

6. Conclusion

In this work, we discuss various strategies for solving multi-agent reinforcement learning problems using the Pommerman testbed. For single agent FFA gameplay, we evaluate a policy gradient agent based on PPO for finding suitable strategies. However, we notice that despite tuning the network architeture and feature set, the designed agent is unable to perform significantly better than a rule-based agent. Further scrutiny revealed that the agent settles for a policy of not using any bombs and evading them – learning to not die, rather than to win. To combat this and impose domain knowledge, we use imitation learning-based DQfD and use a rule-based agent to pretrain our agent from demonstrations. We see that this agent performs significantly better than the PPO agent and can explore the map, plant bombs, strategize etc. Further, we also discussed strategies for teamplay and possible implementations using an optimized off-the-shelf library like TensorForce.

Despite clever tuning and heuristics, our best agent only wins the toughest challenge $\sim 25\%$ of the times. This calls for more careful scrutiny of the problem and better approaches to combine domain expertise with fast learning. The problem of training multi-agent systems and devising optimal policies for the joint state/action spaces is another possible path to explore.

[3] https://github.com/reinforceio/tensorforce/issues/371

References

Authors. Demonstration video (supplementary material), 2018. URL https://prieuredesion.github.io/pommerman.

Foerster, JN, Farquhar, G, Afouras, T, Nardelli, N, and Whiteson, SA. Counterfactual multiagent policy gradients. pp. 2974–2982. AAAI Press, 2018.

Hester, Todd, Vecerik, Matej, Pietquin, Olivier, Lanctot, Marc, Schaul, Tom, Piot, Bilal, Sendonaris, Andrew, Dulac-Arnold, Gabriel, Osband, Ian, Agapiou, John, Leibo, Joel Z., and Gruslys, Audrunas. Learning from demonstrations for real world reinforcement learning. *CoRR*, 2017.

Kapoor, S. Multi-Agent Reinforcement Learning: A Report on Challenges and Approaches. *ArXiv e-prints*, July 2018.

Lange, Sascha, Gabel, Thomas, and Riedmiller, Martin. *Batch Reinforcement Learning*, pp. 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Littman, Michael L. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, 1994.

Lowe, Ryan, WU, YI, Tamar, Aviv, Harb, Jean, Pieter Abbeel, OpenAI, and Mordatch, Igor. Multi-agent actor-critic for mixed cooperative-competitive environments. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6379–6390. 2017.

Matas, Jan, James, Stephen, and Davison, Andrew J. Sim-to-real reinforcement learning for deformable object manipulation. In *CoRL*, 2018.

Matignon, Laetitia, Jeanpierre, Laurent, and Mouaddib, Abdel-Illah. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. 2012.

Minsky, M. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, Jan 1961. ISSN 0096-8390.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

Oliehoek, Frans A. and Amato, Christopher. *A Concise Introduction to Decentralized POMDPs*. 1st edition, 2016.

Peng, Peng, Yuan, Quan, Wen, Ying, Yang, Yaodong, Tang, Zhenkun, Long, Haitao, and Wang, Jun. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *CoRR*, abs/1703.10069, 2017.

Resnick, Cinjon, Eldridge, Wes, Ha, David, Britz, Denny, Foerster, Jakob, Togelius, Julian, Cho, Kyunghyun, and Bruna, Joan. Pommerman: A multi-agent playground, 2018.

Ross, Stephane, Gordon, Geoffrey, and Bagnell, Drew. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I., and Abbeel, Pieter. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Silver, David, Huang, Aja, Maddison, Christopher J., Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, and Hassabis, Demis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

Tan, Ming. Multi-Agent Reinforcement Learning : Independent vs . Cooperative Agents. In *International Conference on Machine Learning (ICML)*, 1993.

Watkins, Christopher J. C. H. and Dayan, Peter. Q-learning. In *Machine Learning*, pp. 279–292, 1992.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

A. Pommerman – Environment Description

Pommerman is a testbed for multi-agent reinforcement learning, based on the popular game Bomberman. Every game is played on a 11×11 game with 4 agents, each spawned at the corners of the grid. The grid can be free, or have rigid or wooden walls in each block. Rigid walls are indestructible and an agent can neither pass through those walls nor blast them through a bomb. The wooden walls can broken by a bomb, and may release a fixed set of power-ups. Until the wooden walls are broken, they cannot be traversed through.

The game begins with the agent having one bomb. Every time it deploys a bomb, the ammo decreases by 1. After the explosion of the bomb, the ammo count again increases by one. The agent also has a blast strength, with a default value of 3. Every bomb laid by the agent is set to that agent's blast strength, which is how far in the vertical and horizontal directions that bomb will effect. Every bomb has a life of 10 steps. When the agents move 10 steps after planting the bomb, it explodes and destroys any wooden walls, agents, power-ups or other bombs in its range (blast strength).

There are 2 game settings in which the game can be played, namely *Free-For-All* and *Teamplay*. In FFA or *Battle Royale*, the 4 agents play against each other with the aim of longest survival, i.e. the agent who survives till the end wins and gets a reward of 1. The other agents get a reward of -1. The Teamplay variant involves the players playing in teams of 2 without communicating with each other. Both the players get a reward of 1 if their team wins, -1 if they lose or 0 otherwise.

The game ending depends on the variant of the game being played. In the FFA variant, the game ends when only one agent remains, or both the last remaining agents die and the game ends in a tie. In teamplay, the game ends when both the players of a team die. Ties can happen when the game does not end before the max steps or if both teams' last agents are destroyed on the same turn.

A.1. Actions and Observations

At any instant of the game, these are the six actions from which the agent can choose:

1. **Stop:** This action is a pass
2. **Up:** Move up if possible, else stay at the same position
3. **Down:** Move down if possible, else stay at the same position
4. **Left:** Move left if possible, else stay at the same position
5. **Right:** Move right if possible, else stay at the same position
6. **Bomb:** Lay a bomb

After every action it takes, the agent receives the following dictionary of observations:

- **Board:** 121 Integers. The flattened board. All squares outside of the agent's purview will be covered with the fog value (5).

- **Position:** 2 Integers, each in [0, 10]. The agent's (x, y) position in the grid.
- **Ammo:** 1 Integer. The agent's current ammo.
- **Blast Strength:** 1 Int. The agent's current blast strength.
- **Can Kick:** 1 Integer, 0 or 1. Whether the agent can kick or not.
- **Teammate:** 1 Integer in [-1, 3]. Which agent is this agent's teammate. For FFA, this will be -1.
- **Enemies:** 3 Integers, each in [-1, 3]. Which agents are this agent's enemies. If this is a team competition, the last Int will be -1 to reflect that there are only two enemies.
- **Bombs:** List of Integers. The bombs in the agent's purview, specified by (X, Y, Blast Strength).