Safer Deep RL with Shallow MCTS: A Case Study in Pommerman

Bilal Kartal*, Pablo Hernandez-Leal*, Chao Gao, and Matthew E. Taylor
Borealis AI
Edmonton, Canada
bilal.kartal,pablo.hernandez,matthew.taylor@borealisai.com

ABSTRACT

Safe reinforcement learning has many variants and it is still an open research problem. Here, we focus on how to use action guidance by means of a non-expert demonstrator to avoid catastrophic events in a domain with sparse, delayed, and deceptive rewards: the recently-proposed multi-agent benchmark of Pommerman. This domain is very challenging for reinforcement learning (RL) — past work has shown that model-free RL algorithms fail to achieve significant learning. In this paper, we shed light into the reasons behind this failure by exemplifying and analyzing the high rate of catastrophic events (i.e., suicides) that happen under random exploration in this domain. While model-free random exploration is typically futile, we propose a new framework where even a non-expert simulated demonstrator, e.g., planning algorithms such as Monte Carlo tree search with small number of rollouts, can be integrated to asynchronous distributed deep reinforcement learning methods. Compared to vanilla deep RL algorithms, our proposed methods both learn faster and converge to better policies on a two-player mini version of the Pommerman game.

KEYWORDS

Monte Carlo tree search; Imitation Learning; Safe Reinforcement Learning; Auxiliary Tasks

1 INTRODUCTION

Deep reinforcement learning (DRL) has enabled better scalability and generalization for challenging high-dimensional domains. DRL has been a very active area of research in recent years [2, 16, 30] with great successes in Atari games [36], Go [46] and very recently, multiagent games (e.g., Starcraft and DOTA 2) [38].

One of the biggest challenges for deep reinforcement learning is sample efficiency [51]. However, once a DRL agent is trained, it can be deployed to act in real-time by only performing an inference through the trained model. On the other hand, planning methods such as Monte Carlo tree search (MCTS) [7] do not have a training phase, but they perform simulation based rollouts assuming access to a simulator to find the best action to take. One major challenge with vanilla MCTS is the scalability to domains with large branching factors and long episodes requiring lots of simulation-based episodes to act, thus, rendering the method inapplicable for applications requiring real-time decision making.

There are several ways to get the best of both DRL and search methods. For example, AlphaGo [45] and Expert Iteration [1] concurrently proposed the idea of combining DRL and MCTS in an imitation learning framework where both components improve each other. These works combine search and neural networks *sequentially* in a loop. First, search is used to generate an expert move dataset, which is used to train a policy network [15]. Second, this network is used to improve expert search quality [1], and this is repeated. However, expert move data collection by vanilla search algorithms can be slow in a sequential framework depending on the simulator efficiency [15].

In this paper, complementary to the aforementioned existing work, we show that it is also possible to blend search with distributed model-free DRL methods such that search and neural network components can be executed *simultaneously* in an *on-policy* fashion. The main focus of this work is to show how we can use relatively weaker demonstrators (e.g., lightweight MCTS with a small number of rollouts or other search based planners [26]) for *safer* model-free RL by coupling the demonstrator and model-free RL interaction through an auxiliary task [20].

Here we focus on the catastrophic events that arise frequently in a recently proposed benchmark for (multi-agent) reinforcement learning: Pommerman [41]. This environment is based on the classic console game *Bomberman*. The Pommerman environment involves 4 bomber agents initially placed at the four corners of a board, see Figure 1, which take simultaneous actions. On the one hand, the only way to make a change in the Pommerman environment (e.g., kill an agent) is by means of bomb placement (and the effects of such an action is only observed when the bomb explodes after 10 time steps). On the other hand, this action could result in the catastrophic event of the agent committing suicide.

In this work we show that suicides happen frequently during learning because of exploration and due to the nature of the environment. Furthermore, the high rate of suicides has a direct effect on the samples needed to learn. We exemplify this in a case for which an *exponential* number of samples are needed to obtain a positive experience. This highlights that performing non-suicidal bomb placement could require complicated, long-term, and accurate planning, which is very hard to learn for model-free reinforcement learning methods.

We consider Asynchronous Advantage Actor-Critic (A3C) [35] as a baseline algorithm and we propose a new framework based on diversifying some of the workers of A3C with MCTS based planners (serving as non-expert demonstrators) by using the parallelized asynchronous training architecture. This has the effect of providing action guidance, which in turns improves the training efficiency measured by higher rewards.

* Equal contribution

Figure 1: A random initial board of Pommerman. Agents have 800 timesteps to blast opponents.

2 RELATED WORK

Our work lies at the intersection of the research areas of safe reinforcement learning, imitation learning, and Monte Carlo tree search based planning. In this section, we will mention some of the existing work in these areas.

2.1 Safe RL

Safe Reinforcement Learning tries to ensure reasonable system performance and/or respect safety constraints during the learning and/or deployment processes [13]. Roughly, there are two ways of doing safe RL: some methods adapt the optimality criterion, while others adapt the exploration mechanism. Since classical approaches to exploration like ϵ-greedy or Boltzmann exploration do not guarantee safety [12, 29], the research area of *safe exploration* deals with the question – how can we build agents that respect the safety constraints not only during normal operation, but also during the initial learning period? [29].

One model-based method proposed to learn the accuracy of the current policy and use another safe policy only in unsafe situations [14]. Lipton et al. [31] proposed learning a risk-assessment model, called *fear* model, which predicts risky events to shape the learning process. Saunders et al. [44] studied if human intervention could prevent catastrophic events; while the approach was successful in some Atari games, the authors argue it would not scale in more complex environments due to the amount of human labour needed.

2.2 Imitation Learning

Domains where rewards are delayed and sparse are difficult exploration RL problems and are particularly difficult when learning *tabula rasa*. Imitation learning can be used to train agents much faster compared to learning from scratch.

Approaches such as DAgger [43] or its extended version [49] formulate imitation learning as a supervised problem where the aim is to match the performance of the demonstrator. However, the performance of agents using these methods is upper-bounded by the demonstrator performance.

Lagoudakis et al. [25] proposed a classification-based RL method using Monte-Carlo rollouts for each action to construct a training dataset to improve the policy iteratively. Recent works such as Expert Iteration [1] extend imitation learning to the RL setting where the demonstrator is also continuously improved during training. There has been a growing body of work on imitation learning where human or simulated demonstrators' data is used to speed up policy learning in RL [9, 11, 17, 37, 48].

Hester et al. [17] used demonstrator data by combining the supervised learning loss with the Q-learning loss within the DQN algorithm to pre-train and showed that their method achieves good results on Atari games by using a few minutes of game-play data. Cruz et al. [11] employed human demonstrators to pre-train their neural network in a supervised learning fashion to improve feature learning so that the RL method with the pre-trained network can focus more on policy learning, which resulted in reducing training times for Atari games. Kim et al. [23] proposed a learning from demonstration approach where limited demonstrator data is used to impose constraints on the policy iteration phase and they theoretically prove bounds on the Bellman error.

In some domains (e.g., robotics) the tasks can be too difficult or time consuming for humans to provide full demonstrations. Instead, humans can provide *feedback* [9, 32] on alternative agent trajectories that RL can use to speed up learning. Along this direction, Christiano et al. [9] proposed a method that constructs a reward function based on data containing human feedback with agent trajectories and showed that a small amount of non-expert human feedback suffices to learn complex agent behaviours.

2.3 Combining Search, DRL, and Imitation Learning

AlphaGo [45] defeated the strongest human Go player in the world on a full-size board. It uses imitation learning by pretraining RL's policy network from human expert games with supervised learning [27]. Then, its policy and value networks keep improving by self-play games via DRL. Finally, an MCTS search skeleton is employed where a policy network narrows down move selection (i.e., effectively reducing the branching factor) and a value network helps with leaf evaluation (i.e., reducing the number of costly rollouts to estimate state-value of leaf nodes). AlphaGo Zero [46] dominated AlphaGo even though it started to learn *tabula rasa*. AlphaGo Zero still employed the skeleton of MCTS algorithm, but it employed the value network for leaf node evaluation without any rollouts. Our work can be seen as complementary to these Expert Iteration based methods and differs in multiple aspects: (i) our framework specifically aims to enable on-policy model-free RL methods to explore *safely* in hard-exploration domains where negative rewarding terminal states are ubiquitous, in contrast to off-policy Expert-Iteration based methods which use intensive search to fully learn a policy; (ii) our framework is general in that other demonstrators (human or other simulated sources) can easily be integrated to provide action guidance by using the proposed auxiliary loss refinement; and (iii) our framework aims to use the demonstrator with a small lookahead (i.e., shallow) search to filter out actions leading to immediate negative terminal states so that model-free RL can imitate those safer actions to learn to safely explore.

3 PRELIMINARIES

3.1 Reinforcement Learning

We start with the standard reinforcement learning setting of an agent interacting in an environment over a discrete number of steps. At time t the agent in state s_t takes an action a_t and receives a reward r_t. The discounted return is defined as $R_{t:\infty} = \sum_{t=1}^{\infty} \gamma^t r_t$. The state-value function,

$$V^{\pi}(s) = \mathbb{E}[R_{t:\infty} | s_t = s; \pi]$$

is the expected return from state s following a policy $\pi(a|s)$ and the action-value function is the expected return following policy π after taking action a from state s:

$$Q^{\pi}(s; a) = \mathbb{E}[R_{t:\infty} | s_t = s; a_t = a; \pi]:$$

The A3C method, as an actor-critic algorithm, has a policy network (actor) and a value network (critic) where the actor is parameterized by $\pi(a|s;\theta)$ and the critic is parameterized by $V(s;\theta_v)$, which are updated as follows:

$$\triangle\theta = \nabla_{\theta} \log \pi(a_t|s_t;\theta) A(s_t; a_t; \theta_v);$$

$$\triangle\theta_v = A(s_t; a_t; \theta_v) \nabla_{\theta_v} V(s_t)$$

where,

$$A(s_t; a_t; \theta_v) = \sum_{k}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) - V(s_t)$$

with $A(s;a) = Q(s;a) - V(s)$ representing the *advantage* function. The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. It is common to use one softmax output for the policy $\pi(a_t|s_t;\theta)$ head and one linear output for the value function $V(s_t;\theta_v)$ head, with all non-output layers shared.

The loss function for A3C is composed of two terms: policy loss (actor), $\mathcal{L}_{\pi}$, and value loss (critic), $\mathcal{L}_V$. An entropy loss for the policy, $H(\pi)$, is also commonly added which helps to improve exploration by discouraging premature convergence to suboptimal deterministic policies [35]. Thus, the loss function is given by:

$$\mathcal{L}_{\text{A3C}} = \lambda_V \mathcal{L}_V + \lambda_{\pi} \mathcal{L}_{\pi} - \lambda_H \mathbb{E}_{s\sim\pi}[H(\pi(s;\cdot;\theta))]$$

with $\lambda_V = 0{:}5$, $\lambda_{\pi} = 1{:}0$, and $\lambda_H = 0{:}01$, being standard weighting terms on the individual loss components.

UNREAL [20] proposed unsupervised *auxiliary tasks* (e.g., reward prediction) to speed up learning, which require no additional feedback from the environment. In contrast to A3C, UNREAL uses an experience replay buffer that is sampled with more priority given to positively rewarded interactions to improve the critic network. The UNREAL framework optimizes a single combined loss function $\mathcal{L}_{\text{UNREAL}} \approx \mathcal{L}_{A3C} + \lambda_{AT}\mathcal{L}_{AT}$, that combines the A3C loss, $\mathcal{L}_{A3C}$, together with an auxiliary task loss $\mathcal{L}_{AT}$, where λ_{AT} is a weight term.

3.2 Monte Carlo Tree Search

Monte Carlo Tree Search is a best-first search algorithm that gained traction after its breakthrough performance in Go [10]. Other than for game playing agents [6, 47] and playtesting agents [4, 18, 54], MCTS has been employed for a variety of domains such as robotics [52], continuous control tasks for animation [40], and procedural puzzle generation [22]. One recent paper [50] provided an excellent unification of MCTS and RL.

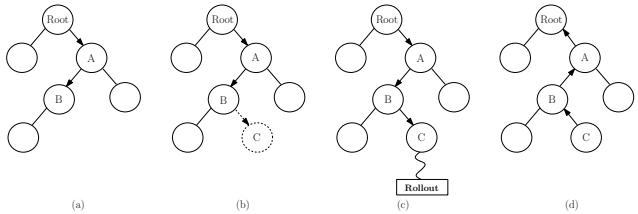


Figure 2: Overview of Monte Carlo Tree Search: (a) *Selection*: UCB is used recursively until a node with an unexplored action is selected. Assume that nodes A and B are selected. (b) *Expansion*: Node C is added to the tree. (c) *Random Rollout*: A sequence of random actions is taken from node C to complete the partial game. (d) *Back-propagation*: After the rollout terminates, the game is evaluated and the score is back-propagated from node C to the root.

In MCTS a search tree is generated where each node in the tree represents a complete state of the domain and each link represents one possible action from the set of valid actions in the current state, leading to a child node representing the resulting state after applying that action. The root of the tree is the initial state (for example, the initial configuration of the Pommerman board including the agent location). MCTS proceeds in four phases of: selection, expansion, rollout, and back-propagation (see Figure 2). The MCTS algorithm proceeds by repeatedly adding one node at a time to the current tree. Given that actions from the root to the expanded node is unlikely to terminate an episode, e.g., a Pommerman game can take up to 800 timesteps, MCTS uses random actions, a.k.a. *rollouts*, to estimate state-action values. After rollout phase, the total collected rewards during the episode is back-propagated through all existing nodes in the tree updating their empirical state-action value estimates.

Exploration vs. Exploitation Dilemma. Choosing which child node to expand (i.e., choosing which action to take) becomes an exploration/exploitation problem. We want to primarily choose actions that have good scores, but we also need to explore other possible actions in case the observed empirical average scores do not represent the true reward mean of that action. This exploration/exploitation dilemma has been well studied in other areas. Upper Confidence Bounds (UCB) [3] is a selection algorithm that seeks to balance the exploration/exploitation dilemma. Using UCB with MCTS is also referred to as Upper Confidence bounds applied to Trees (UCT). Applied to our framework, each parent node s chooses its child s' with the largest $UCB(s;a)$ value according to Eqn. 1.

$$UCB(s;a) = Q(s;a) + C\sqrt{\frac{\ln n(s)}{n(s')}} \tag{1}$$

Here, $n(s)$ denotes number of visits to the node s and $n(s')$ denotes the number of visits to s', i.e., the resulting child node when taking action a from node s. The value of C determines the rate of exploration, where smaller C implies less exploration. Kocsis and Szepesvari [24] showed that $C = \sqrt{2}$ is necessary for asymptotic

convergence, however it can be tuned depending on the domain. In this work, we employed the default value of $\sqrt{2}$ for the exploration constant.

4 SAFE RL WITH MCTS ACTION GUIDANCE

In this section, firstly, we present our framework PI-A3C (Planner Imitation - A3C) that extends A3C with a lightweight search based demonstrator through an auxiliary task based loss refinement as depicted in Figure 3. Secondly, we present the Pommerman domain in detail to analyze and exemplify the exploration complexity of that domain for pure model-free RL methods.

4.1 Approach Overview

We propose a framework that can use planners, or other sources of demonstrators, along with asynchronous DRL methods to accelerate learning. Even though our framework can be generalized to a variety of planners and distributed DRL methods, we showcase our contribution using MCTS and A3C. In particular, our use of MCTS follows the approach of Kocsis and Szepesvari [24] employing UCB to balance exploration versus exploitation during planning. During rollouts, we simulate all agents as random agents as in default unbiased MCTS and we perform limited-depth rollouts to reduce action-selection time.

The motivation for combining MCTS and asynchronous DRL methods stems from the need to improve training time efficiency even if the planner or environment simulator is very slow. In this work, we assume the demonstrator and actor-critic networks are decoupled, i.e., a vanilla UCT planner is used as a black-box that takes an observation and returns an action resembling UCTtoClassification method [15], which uses MCTS to generate an expert move dataset and trains a NN in a supervised fashion to imitate MCTS. However, they used a high-number of rollouts (10K) per action selection to construct an expert dataset. In contrast, we show how vanilla MCTS with a small number of rollouts[1] (≈ 100) can still be employed in an *on-policy* fashion to improve training efficiency for actor-critic RL in challenging domains with abundant easily reachable terminal states with negative rewards.

Within A3C's asynchronous distributed architecture, all the CPU workers perform agent-environment interaction with their neural network policy networks, see Figure 3(a). In our new framework, PI-A3C (Planner Imitation with A3C), we assign $k \geq 1$ CPU workers (we present experimental results with different k values in Section 5.2) to perform MCTS based planning for agent-environment interaction based on the agent's observations, while also keeping track of what its neural network would perform for those observations, see Figure 3(b). In this fashion, we both learn to imitate the MCTS planner and to optimize the policy. The main motivation for PI-A3C framework is to increase the number of agent-environment interactions with positive rewards for hard-exploration RL problems to improve training efficiency.

Note that the planner-based worker still has its own neural network with actor and policy heads, but action selection is performed by the planner while its policy head is used for loss computation. In particular, the MCTS planner based worker augments its loss function with the auxiliary task of *Planner Imitation*[2]. The auxiliary loss is defined as $\mathcal{L}_{PI} = -\frac{1}{N}\sum_{i}^{N} a_0^i \log(\hat{a}_0^i)$, which is the supervised cross entropy loss between a_0^i and $\hat{a}_0^i$, representing the one-hot encoded action the planner used and the action the actor (with policy head) would take for the same observation respectively during an episode of length N. The demonstrator worker's loss after the addition of *Planner Imitation* is defined by

$$\mathcal{L}_{\text{PI-A3C}} = \mathcal{L}_{A3C} + \lambda_{PI}\mathcal{L}_{PI}$$

where $\lambda_{PI} = 1$ is a weight term (which was not tuned). In PI-A3C the rest of the workers (not demonstrators) are left unchanged, still using the policy head for action selection with the unchanged loss function. By formulating the *Planner Imitation* loss as an auxiliary loss, the objective of the resulting framework becomes a multi-task learning problem [8] where the agent aims to learn to both maximize the reward and imitate the planner.

4.2 Pommerman

In Pommerman, each agent can execute one of 6 actions at every timestep: move in any of four directions, stay put, or place a bomb. Each cell on the board can be a passage, a rigid wall, or wood. The maps are generated randomly, albeit there is always a guaranteed path[3] between any two agents. Whenever an agent places a bomb it explodes after 10 timesteps, producing flames that have a lifetime of 2 timesteps. Flames destroy wood and kill any agents within their blast radius. When wood is destroyed either a passage or a power-up is revealed. Power-ups can be of three types: increase the blast radius of bombs, increase the number of bombs the agent can place, or give the ability to kick bombs. A single game is finished when an agent dies or when reaching 800 timesteps.

Pommerman is a challenging benchmark for multi-agent learning and model-free reinforcement learning, due to the following characteristics:

Multiagent component: the agent needs to best respond to any type of opponent, but agents' behaviours also change based on the collected power-ups, i.e., extra ammo, bomb blast radius, and bomb kick ability.

Delayed action effects: the only way to make a change to the environment (e.g., kill an agent) is by means of bomb placement, but the effect of such an action is only observed when the bombs explodes after 10 time steps.

Sparse and deceptive rewards: the former refers to the fact that the only non-zero reward is obtained at the end of an episode. The latter refers to the fact that quite often a winning reward is due to the opponents' involuntary *suicide*, which makes reinforcing an agent's action based on such a reward *deceptive*.

For these reasons, we consider this game challenging for many standard RL algorithms and a local optimum is commonly learned, i.e., not placing bombs [41].

Some other recent works also used Pommerman as a test-bed, for example, Zhou et al. [53] proposed a hybrid method combining

[1] In Pommerman 10K rollouts would take hours due to very slow simulator and long horizon [34].

[2] Both Guo et al. [15] and Anthony et al. [1] used MCTS moves as a learning target, referred to as *Chosen Action Target*. Our *Planner Imitation* loss is similar except in this work, we employed cross-entropy loss in contrast to a KL divergence based one.

[3] Although this path can be initially blocked by wood, thus, needing clearance by bombs.

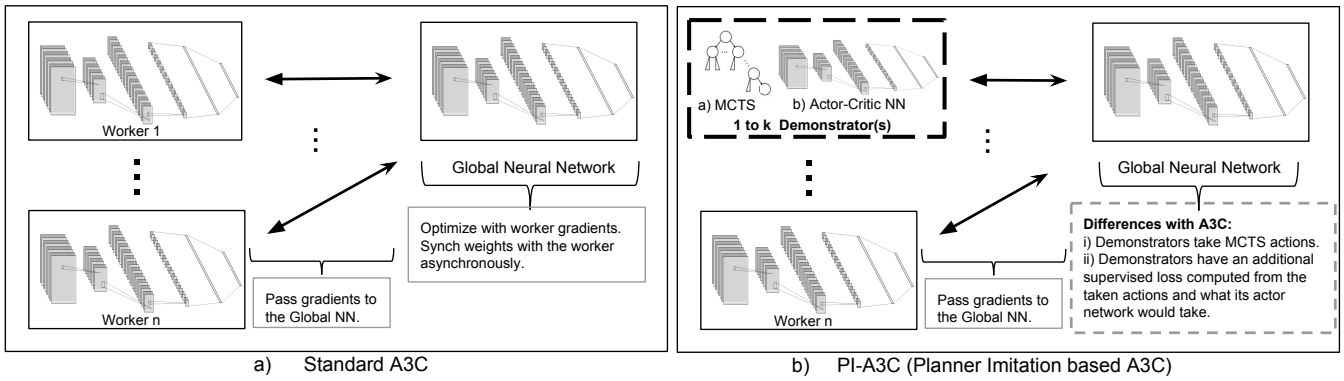


Figure 3: a) In the A3C framework, each worker independently interacts with the environment and computes gradients. Then, each worker *asynchronously* passes the gradients to the global neural network which updates parameters and synchronizes with the respective worker. b) In our proposed framework, Planner Imitation based A3C (PI-A3C), $k \geq 1$ CPU workers are assigned as MCTS based demonstrators taking MCTS actions, while keeping track of what action its actor network would take. The demonstrator workers have an additional auxiliary supervised loss. PI-A3C enables the network to simultaneously optimize the policy and learn to imitate the MCTS.

rule-based heuristics with depth-limited search. Resnick et al. [42] proposed a framework that uses a single demonstration to generate a training curriculum for sparse reward RL problems (assuming episodes can be started from arbitrary states). Lastly, relevance graphs, which represent the relationship between agents and environment objects [33], are another approach to deal with the complex Pommerman game.

4.3 Catastrophic events in Pommerman: Suicides

Before presenting the experimental results of our approach we motivate the need for safe exploration by providing two examples and a short analysis of the catastrophic events that occur in Pommerman.

It has been noted that in Pommerman the action of bomb placing is highly correlated to losing [41]. This is presumably a major impediment for achieving good results using model-free reinforcement learning. Here, we provide an analysis of the suicide problem that delays or even prevents to learn the *bombing skill* when an agent follows a random exploration policy.

In Pommerman an agent can only be killed when it intersects with an exploding bomb's flames, then we say a *suicide* event happens if the death of the agent is caused by a previously placed own bomb. For the ease of exposition we consider the following simplified scenario: (i) the agent has `ammo=1` and has just placed a bomb (i.e., the agent now sits on the bomb). (ii) For the next time steps until the bomb explodes the agent has 5 actions available at every time step (move in 4 directions or do nothing); and (iii) all other items on the board are static.

Example 1. We take as example the board configuration depicted in Figure 1 and show the probability of ending in a certain position for $t = 9$ (which is the time step that the bomb will explode) for the 4 agents, see Figure 4. This is a typical starting board in Pommerman, where every agent stays in its corner; they are disconnected with each other by randomly generated wood and walls. From this figure

	■	■	.08	■	■	■	■	■	■	■
■	★.01	★.18	.13	■	■	■	.15	★.23	★.01	■
■	★.18	.20	■	■	■	■	.08	■	★.14	.12
.08	.13	■		■	■	■	.04	■	.12	.11
■	■	■	■			■	■	■	■	■
■	■	■	■			■	■		■	■
■	■	■	■	■	■		■	■	■	■
■	.15	.08	.04	■	■	■	.04	.05	.05	.04
■	★.23	■	■	■		■	.05	.06	★.06	■
■	★.01	★.14	.12	■	■	■	.05	★.06	★.06	★.17
■	■	.12	.11	■	■	■	.04	■	★.17	.17

Figure 4: Probabilities of being at each cell for $t = 9$ (after a bomb explodes) for each agent. ? indicates the cell is covered by flames and ■ represents an obstacle, which can be wood or a rigid cell. For each agent, the probabilities of suicide are $\approx 0{:}4$. Survival probability after placing b bombs is computed by $(1 - 0{:}4)^b$, showing that it quickly approaches 0 (similar the probability of getting b heads in b consecutive coin flips), showing high exploration difficulty.

we can see that for each of the four agents, even when their configuration is different, their probabilities of ending up with suicide are $\approx 40\%$ (0:39; 0:38; 0:46; 0:38, respectively, in counter clockwise order starting from upper-left corner) — these are calculated by summing up the positions where there is a flame, represented by ?.

Indeed, the problem of suicide stems from acting randomly without considering constraints of the environment. In extreme cases, in each time step, the agent may have only one survival action, which means the safe path is always unique as t increases while the total number grows exponentially. We illustrate this by the following corridor example.

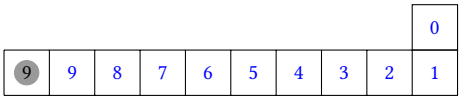


Figure 5: The corridor scenario: the agent places a bomb with `strength=10` on the leftmost cell. For each passage cell, the marked value means the minimum number of steps it is required to safely evade from impact of the bomb. After placing the bomb, in the next step the bomb has life of 9, thus in the remaining 9 steps, the agent must always take the unique correct action to survive.

Example 2. Figure 5 shows a worst-case example: the agent is in a corridor formed by wood at two sides and places a bomb. If using random exploration the chance of ending up in suicide is extremely high since among the 5^9 "paths," i.e., action trajectories — *only one* of them is safe. In order to survive, it must precisely follow the right action at each time step.

Note that in cases like the corridor scenario even if the agent is modestly model-aware, i.e., it may only look one step-ahead, the number of possible action trajectories is still exponential, while the survival path remains unique. This also implies that for such sub-problems in Pommerman, acquiring one positive behaviour example requires *exponential* number of samples.

5 EXPERIMENTS AND RESULTS

In this section, we present the experimental setup and results against different opponents in a simplified version of the Pommerman game. We run ablation experiments on the contribution of *Planner Imitation* to compare against the standard A3C method for: (i) single demonstrator with different expertise levels, (ii) different number of demonstrators with the same expertise level, and (iii) using rollout biasing within MCTS in contrast to uniform random rollout policy.

All the training curves are obtained from 3 runs with different random seeds. From these 3 separate runs, we compute average learning performance (depicted with bold color curves) and standard deviations (depicted with shaded color curves). At every training episode, the board is randomized ,i.e., the locations of wood and rigid cells are shuffled.

5.1 Setup

Because of all the complexities mentioned in this domain we simplified it by considering a game with only two agents and a reduced board size of 8×8, see Figure 6. Note that we still randomize *the location of* walls, wood, power-ups, and the initial position of the agents for every episode. We considered two types of opponents in our experiments:

- *Static* opponents: the opponent waits in the initial position and always executes the 'stay put' action. This opponent provides the easiest configuration (excluding suicidal opponents). It is a baseline opponent to show how challenging the game is for model-free RL.
- *Rule-based* opponents: this is the benchmark agent within the simulator. It collects power-ups and places bombs when it is near an opponent. It is skilled in avoiding blasts from bombs. It uses Dijkstra's algorithm on each time-step, resulting in longer training times.

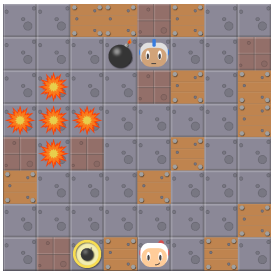


Figure 6: An example of the 8×8 Mini-Pommerman board, randomly generated by the simulator. Agents' initial positions are randomized among four corners at each episode.

Details regarding neural network architecture and implementation are provided in the appendix.

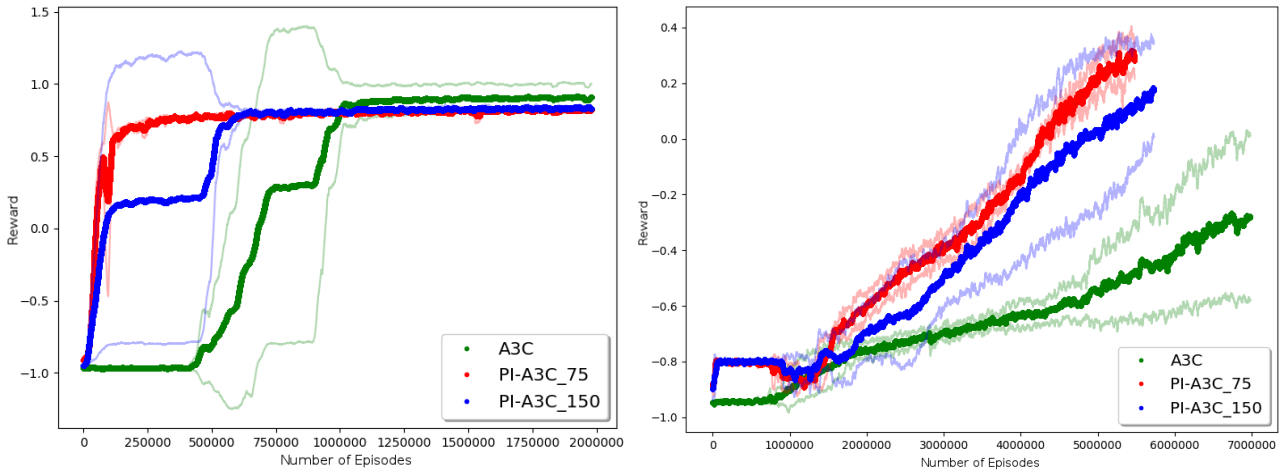
5.2 Results

We conducted two sets of experiments learning against *Static* and *Rule-based* opponents. We compare our proposed methods with respect to standard A3C with learning curves in terms of converged policies and time-efficiency. All approaches were trained using 24 CPU cores. For a fair comparison all training curves for PI-A3C variants are obtained by using only the neural network policy network based workers, excluding rewards obtained by the demonstrator worker to accurately observe the performance of model-free RL.

On action guidance, quantity versus quality. Within our framework, we can vary the expertise level of MCTS by simply changing the number of rollouts games per move for the demonstrator worker. We experimented with 75 and 150 rollouts. Given finite training time, higher rollouts imply deeper search and better moves, however, it also implies that number of guided actions by the demonstrators will be fewer in quantity, reducing the number of asynchronous updates to the global neural network. As Figure 7 shows against both opponents the relatively weaker demonstrator (75 rollouts) enabled faster learning than the one with 150 rollouts. We hypothesize that the faster demonstrator (MCTS with only 75 rollouts) makes more updates to the global neural network, warming up other purely model-free workers for *safer* exploration much earlier in contrast to the slower demonstrator. This is reasonable as the model-free RL workers constitute all but one of the CPU workers in these experiments, therefore the earlier the model-free workers can start safer exploration, the better the learning progress is likely to be.[4]

On the trade-off of using multiple demonstrators. Our proposed method is built on top of an asynchronous distributed framework that uses several CPU workers: $k \geq 1$ act as a demonstrator and the rest of them explore in model-free fashion. We conducted one experiment to better understand how increasing the number of

[4]Even though we used MCTS with fixed number of rollouts, this could be set dynamically, for example, by exploiting the reward sparsity or variance specific to the problem domain, e.g., using higher number of rollouts when close to bombs or opponents.



(a) Learning against a Static Agent

(b) Learning against a Rule-based Agent

Figure 7: Both figures are obtained from 3 training runs showing the learning performance with mean (bold lines) and standard deviation (shaded lines). 24 CPU workers are used for all experiments, and all PI-A3C variants use only 1 Demonstrator worker. a) Against Static agent, all variants have been trained for 12 hours. The PI-A3C framework using MCTS demonstrator with 75 and 150 rollouts learns significantly faster compared to the standard A3C. b) Against Rule-based opponent, all variants have been trained for 3 days. Against this more skilled opponent, PI-A3C provides significant speed up in learning performance, and finds better best response policies. For both (a) and (b), increasing the expertise level of MCTS through doubling the number of rollouts (from 75 to 150) does not yield improvement, and even can hurt performance. Our hypothesis is that slower planning decreases the number of demonstrator actions too much for the model-free RL workers to learn to imitate for safe exploration.

demonstrators, each of which provides additional *Planner Imitation* losses asynchronously, affects the learning performance. The trade-off is that more demonstrators imply fewer model-free workers to optimize the main task, but also a higher number of actions to imitate. We present the results in Figure 8 where we experimented 3 and 6 demonstrators, with identical resources and with 150 rollouts each. Results show that increasing to 3 improves the performance while extending to 6 demonstrators does not provide any marginal improvement. We can also observe that 3 demonstrator version using 150 rollouts presented in Figure 8 has a relatively similar performance with the 1 demonstrator version using 75 rollouts depicted in Figure 7(b), which is aligned with our hypothesis that providing more demonstrator guidance early during learning is more valuable than fewer higher quality demonstrations.

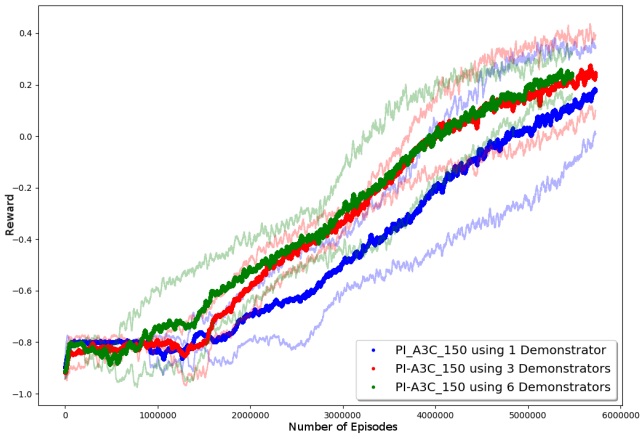


Figure 8: Learning against Rule-based opponent. Employing different number ($n = 1; 3; 6$) of Demonstrators within the asynchronous distributed framework. Increasing from 1 to 3 the number of demonstrators also improved the results, however, there is almost no variation from 3 to 6 demonstrators. They all outperform standard A3C, see Figure 7(b).

Demonstrator biasing with policy network. Uniform random rollouts employed by vanilla MCTS to estimate state-action values provide an unbiased estimate, however due to the high variance, it requires many rollouts. One way to improve search efficiency has been through different biasing strategies rather than using uniform rollout policy, such as prioritizing actions globally based on their evaluation scores [21], using heuristically computed move urgency values [5], or concurrently learning a rollout policy RL [19]. In a similar vein with these methods, we let the MCTS based demonstrator to use the policy network during the rollout phase. We name this refined version as PI-A3C-NN (Planner Imitation - A3C with Neural Network). Our results suggest that employing a biased rollout policy provides improvement in the average learning performance, however it has higher variance as depicted in Figure 9.

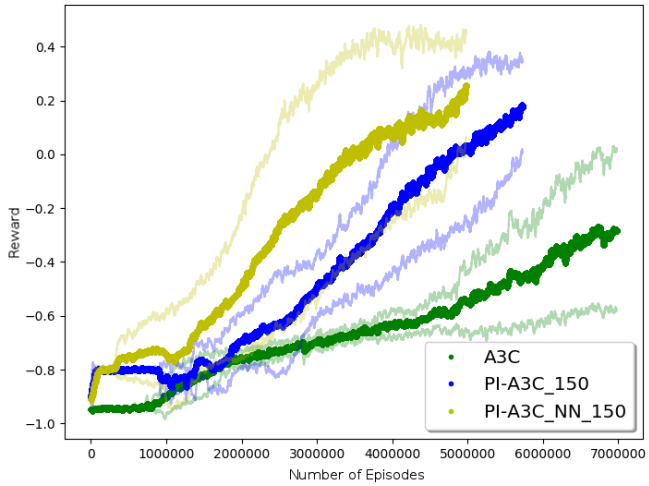


Figure 9: Learning against Rule-based opponent. Employing policy network during MCTS rollout phase within the demonstrator provides improvement in learning speed, but it has higher variance compared to employing the standard uniform random rollouts.

6 DISCUSSION

In Pommerman, the main challenge for model-free RL is the high probability of suicide while exploring (yielding a negative reward), which occurs due to the delayed bomb explosion. However, the agent cannot succeed without learning how to stay safe after bomb placement. The methods and ideas proposed in this paper address this hard-exploration challenge. The main idea of our proposed framework is to use MCTS as a shallow demonstrator (small number of rollouts). This yielded fewer training episodes with catastrophic suicides as imitating MCTS provided improvement for a safer exploration. Concurrent work has proposed the use of pessimistic scenarios to constraint real-time tree search [39]. Other recent work [28] employed MCTS as a high-level planner, which is fed a set of low-level offline learned DRL policies and refines them for safer execution within a simulated autonomous driving domain.

There are several directions to extend our work. One direction is to investigate how to formulate the problem so that *ad-hoc* invocation of an MCTS based simulator can be employed. Currently, MCTS provides a safe action continuously, but action guidance can be employed only when the model-free RL agent indeed needs one, e.g., in Pommerman whenever a bomb is about to go off or near enemies where enemy engagement requires strategic actions.

All of our work presented in the paper is on-policy for better comparison to the standard A3C method — we maintain no experience replay buffer. This means that MCTS actions are used only once to update neural network and thrown away. In contrast, UNREAL uses a buffer and gives higher priority to samples with positive rewards. We could take a similar approach to save demonstrator's experiences to a buffer and sample based on the rewards.

7 CONCLUSIONS

Safe reinforcement learning has many variants and it is still an open research problem. In this work, we present a framework that uses a non-expert simulated demonstrator within a distributed asynchronous deep RL method to succeed in hard-exploration domains. Our experiments use the recently proposed Pommerman domain, a very challenging benchmark for pure model-free RL methods as the rewards are sparse, delayed, and deceptive. We provide examples of these issues showing that RL agents fail mainly because the main skill to be learned is highly correlated with negative rewards due to the high-probability of catastrophic events. In our framework, model-free workers learn to safely explore and acquire this skill by imitating a shallow search based non-expert demonstrator. We performed different experiments varying the quality and the number of demonstrators. The results shows that our proposed method shows significant improvement in learning efficiency across different opponents.

APPENDIX

Neural Network Architecture: For all methods described in the paper, we use a deep neural network with 4 convolutional layers, each of which has 32 filters and 3×3 kernels, with stride and padding of 1, followed with 1 dense layer with 128 hidden units, followed with 2-heads for actor and critic (where the actor output corresponds to probabilities of 6 actions, and the critic output corresponds to state-value estimate). Neural network architectures were not tuned.

NN State Representation: Similar to [41], we maintain 28 feature maps that are constructed from the agent observation. These channels maintain location of walls, wood, power-ups, agents, bombs, and flames. Agents have different properties such as bomb kick, bomb blast radius, and number of bombs. We maintain 3 feature maps for these abilities per agent, in total 12 is used to support up to 4 agents. We also maintain a feature map for the remaining lifetime of flames. All the feature channels can be readily extracted from agent observation except the opponents' properties and the flames' remaining lifetime, which can be tracked efficiently by comparing sequential observations for fully-observable scenarios.

Hyperparameter Tuning: We did not perform a through hyperparameter tuning due to long training times. We used a $\gamma = 0{:}999$ for discount factor. For A3C, the default weight parameters are employed, i.e., 1 for actor loss, 0:5 for value loss, and 0:01 for entropy loss. For the *Planner Imitation* task, $\lambda_{PI} = 1$ is used for the MCTS worker, and $\lambda_{PI} = 0$ for the rest of workers. We employed the Adam optimizer with a learning rate of 0:0001. We found that for the Adam optimizer, $\epsilon = 1 \times 10^{-5}$ provides a more stable learning curve (less catastrophic forgetting) than its default value of 1×10^{-8}. We used a weight decay of 1×10^{-5} within the Adam optimizer for L2 regularization.

REFERENCES

[1] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*. 5360–5370.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[4] Igor Borovikov, Yunqi Zhao, Ahmad Beirami, Jesse Harder, John Kolen, James Pestrak, Jervis Pinto, Reza Pourabolghasem, Harold Chaput, Mohsen Sardari, et al. 2019. Winning IsnâĂŹt Everything: Training Agents to Playtest Modern Games. (2019).

[5] Bruno Bouzy. 2005. Associating domain-dependent knowledge and Monte Carlo approaches within a Go program. *Information Sciences* 175, 4 (2005), 247–257.

[6] Ivan Bravi, Diego Perez-Liebana, Simon M Lucas, and Jialin Liu. 2018. Shallow decision-making analysis in General Video Game Playing. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8.

[7] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[8] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[9] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*. 4299–4307.

[10] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, H. Jaap van den Herik, Paolo Ciancarini, and Jeroen Donkers (Eds.). Vol. 4630 of LNCS. Springer, 72–83.

[11] Gabriel V Cruz Jr, Yunshu Du, and Matthew E Taylor. 2017. Pre-training Neural Networks with Human Demonstrations for Deep Reinforcement Learning. *arXiv preprint arXiv:1709.04083* (2017).

[12] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2019. Go-Explore: a New Approach for Hard-Exploration Problems. *arXiv preprint arXiv:1901.10995* (2019).

[13] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015).

[14] Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. 2016. Safe policy improvement by minimizing robust baseline regret. In *NIPS*. 2298–2306.

[15] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. 2014. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in neural information processing systems*. 3338–3346.

[16] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2018. Is multiagent deep reinforcement learning the answer or the question? A brief survey. *arXiv preprint arXiv:1810.05587* (2018).

[17] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. 2017. Deep Q-learning from Demonstrations. *arXiv preprint arXiv:1704.03732* (2017).

[18] Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. 2018. Automated playtesting with procedural personas through MCTS with evolved heuristics. *arXiv preprint arXiv:1802.06881* (2018).

[19] Ercüment Ilhan and A Şima Etaner-Uyar. 2017. Monte Carlo tree search with temporal-difference learning for general video game playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 317–324.

[20] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397* (2016).

[21] Bilal Kartal, John Koenig, and Stephen J Guy. 2014. User-driven narrative variation in large story domains using monte carlo tree search. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 69–76.

[22] Bilal Kartal, Nick Sohre, and Stephen J Guy. 2016. Data Driven Sokoban Puzzle Generation with Monte Carlo Tree Search. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[23] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. 2013. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*. 2859–2867.

[24] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. 282–293.

[25] Michail G Lagoudakis and Ronald Parr. 2003. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 424–431.

[26] Steven M LaValle. 2006. *Planning algorithms*. Cambridge university press.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[28] Jaeyoung Lee, Aravind Balakrishnan, Ashish Gaurav, Krzysztof Czarnecki, and Sean Sedwards. 2019. WiseMove: A Framework for Safe Deep Reinforcement Learning for Autonomous Driving. *arXiv preprint arXiv:1902.04118* (2019).

[29] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. 2017. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883* (2017).

[30] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[31] Zachary C Lipton, Kamyar Azizzadenesheli, Abhishek Kumar, Lihong Li, Jianfeng Gao, and Li Deng. 2016. Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear. *arXiv preprint arXiv:1611.01211* (2016).

[32] Robert Tyler Loftin, James MacGlashan, Bei Peng, Matthew E Taylor, Michael L Littman, Jeff Huang, and David L Roberts. 2014. A Strategy-Aware Technique for Learning Behaviors from Discrete Human Feedback.

[33] Aleksandra Malysheva, Tegg Taekyong Sung, Chae-Bong Sohn, Daniel Kudenko, and Aleksei Shpilman. 2018. Deep Multi-Agent Reinforcement Learning with Relevance Graphs. *arXiv preprint arXiv:1811.12557* (2018).

[34] Tambet Matiisen. 2018. Pommerman baselines. https://github.com/tambetm/pommerman-baselines. (2018).

[35] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[37] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6292–6299.

[38] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/. (2018). [Online; accessed 15-October-2018].

[39] Takayuki Osogami and Toshihiro Takahashi. 2019. Real-time tree search with pessimistic scenarios. *arXiv preprint arXiv:1902.10870* (2019).

[40] Joose Julius Rajamäki and Perttu Hämäläinen. 2018. Continuous control monte carlo tree search informed by multiple experts. *IEEE transactions on visualization and computer graphics* (2018).

[41] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. 2018. Pommerman: A Multi-Agent Playground. *arXiv preprint arXiv:1809.07124* (2018).

[42] Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alex Peysakhovich, Kyunghyun Cho, and Joan Bruna. 2019. Backplay:" Man muss immer umkehren". *AAAI-19 Workshop on Reinforcement Learning in Games* (2019).

[43] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. 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*. 627–635.

[44] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. 2018. Trial without error: Towards safe reinforcement learning via human intervention. In *AAMAS*. 2067–2069.

[45] David Silver, Aja Huang, Chris J Maddison, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[46] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[47] Nathan R Sturtevant. 2015. Monte Carlo Tree Search and Related Algorithms for Games. *Game AI Pro 2: Collected Wisdom of Game AI Professionals* (2015), 265.

[48] Kaushik Subramanian, Charles L Isbell Jr, and Andrea L Thomaz. 2016. Exploration from demonstration for interactive reinforcement learning. In *International Conference on Autonomous Agents & Multiagent Systems*. 447–456.

[49] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. 2017. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030* (2017).

[50] Tom Vodopivec, Spyridon Samothrakis, and Branko Ster. 2017. On Monte Carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research* 60 (2017), 881–936.

[51] Yang Yu. 2018. Towards Sample Efficient Reinforcement Learning.. In *IJCAI*. 5739–5743.

[52] Mabel M Zhang, Nikolay Atanasov, and Kostas Daniilidis. 2017. Active end-effector pose selection for tactile object recognition through monte carlo tree search. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. 3258–3265.

[53] Hongwei Zhou, Yichen Gong, Luvneesh Mugrai, Ahmed Khalifa, Andy Nealen, and Julian Togelius. 2018. A hybrid search agent in pommerman. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM, 46.

[54] Alexander Zook, Brent Harrison, and Mark O Riedl. 2015. Monte-carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th Conference on the Foundations of Digital Games*.