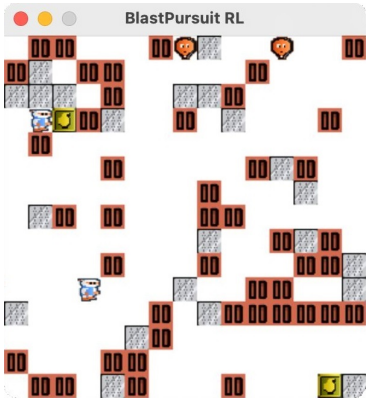
Project Documentation

BlastPursuit

Guido Laudenzi and Sami Osman
Course Code: 91267 - Multi-Agent Systems

February 11, 2024



Abstract

In this documentation, we present *BlastPursuit*, a project that integrates the classic elements of the Bomberman game from Nintendo with advanced multi-agent system concepts, developed as part of our coursework in Multi-Agent Systems. Our project combines artificial intelligence with strategic gameplay, set within a grid-based environment. It features intelligent agents, divided in Bombers, and Killers, equipped with reinforcement learning algorithms, enabling them to make autonomous decisions and learn from their interactions. The objective of *BlastPursuit* is to provide a platform that not only offers an engaging gaming experience but also serves as an educational tool, demonstrating the application of AI in dynamic, multi-agent settings, providing interesting insights of emerging behaviour only obtained with reinforcement learning. This documentation details the development process, technical aspects, and the challenges we faced in bringing this innovative project to fruition. At the end, we will show the results achieved applying a single parameter-sharing model that controls all the agents.

1 Introduction

In our project *BlastPursuit*, we have ventured to synergize the thrilling essence of the classic Bomberman game (Nintendo, 1983), augmented with different kind of agents, with the complexities of artificial intelligence, particularly focusing on multi-agent systems.
The heart of *BlastPursuit* lies in its grid-based world, where autonomous agents – Bombers and Killers – exhibit emerging strategic behaviors driven by reinforcement learning. These agents engage in a dynamic interplay of navigation, decision-making, and tactical maneuvers within an environment filled with both indestructible and destructible walls. This setup creates a rich tapestry of strategic possibilities and challenges, encouraging players to think critically and adapt their strategies.
In *BlastPursuit*, we leveraged Python and Pygame for robust and visually engaging game development, complemented by the integration of a OpenAI's *gym* [1] (now, Farama Foundation's *gymnasium* [2]) environment for AI training. Advanced algorithms like CNNs for observations' feaures extraction, PPO (Proximal Policy Optimization) [3], SAC (Soft Actor-Critic) [4], and DQN (Deep Q-Network) [5] from the *Stable-Baselines3* library [6] were employed to effectively facilitate and enhance learning of AI agents within the game.
In this documentation, we delve into the technical intricacies of developing *BlastPursuit*, detailing the AI algorithms, the programming challenges we encountered, and the solutions we implemented. Our goal is to showcase the potential of multi-agent systems in creating deeply engaging and intelligent gaming experiences and to offer a comprehensive resource for those interested in the practical application of AI in gaming. Through *BlastPursuit*, we aim to bridge the gap between theoretical AI concepts and their real-world applications, offering both an entertaining game and an educational insight into the world of artificial intelligence.

2 Background and Motivation

2.1 Description of the Original Bomberman Game

The original Bomberman game from Nintendo, a classic in the realm of video games, has captivated players for decades with its simple yet engaging gameplay. In Bomberman, players navigate a character through a maze-like grid, placing bombs to destroy walls and opponents while avoiding being caught in bomb explosions themselves. The game is renowned for its strategic depth, requiring players to think ahead and plan their moves carefully. Bomberman's enduring popularity lies in its blend of easy-to-understand mechanics and challenging gameplay, making it an ideal candidate for exploring more complex gaming concepts.

2.2 Augmented Version with Multi-Agent Systems

Our motivation to create an augmented version of Bomberman with multi-agent systems stems from a desire to explore the intersection of traditional gaming with modern AI technology. Multi-agent systems represent a significant area in artificial intelligence, where multiple autonomous agents interact within a shared environment. By incorporating these systems into the Bomberman framework, we aim to elevate the game's strategic complexity and introduce elements of AI-driven decision-making and learning.
This augmented version, *BlastPursuit*, is designed to serve as a platform for studying and demonstrating the principles of AI and multi-agent systems in a practical and engaging context. The game's environment, populated with non-standard intelligent agents that learn and adapt their strategies through reinforcement learning, offers a rich ground for exploring AI behaviors, such as autonomous navigation, tactical planning, and agent interactions. By integrating multi-agent systems into Bomberman, we not only enhance the gaming experience but also contribute to the broader field of AI research and education, providing valuable insights into the capabilities and challenges of AI in dynamic, interactive settings.

2.3 Reinforcement Learning (RL)

Reinforcement learning (RL) is a machine learning paradigm where agents learn to make decisions by interacting with an environment. In the context of BlastPursuit, RL can be employed to train agents to navigate the game intelligently. Q-learning, a fundamental algorithm in RL, plays an exemplar role. At its core, Q-learning involves estimating the quality of different actions in a given state, represented by a Q-value. These Q-values are updated iteratively as the agent explores the environment, aiming

to maximize its cumulative reward over time. The Q-learning function computes the optimal policy by balancing exploration and exploitation [7]:

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

- $Q(s,a)$: the Q-value, which is the expected cumulative reward of taking action a in state s.
- α: the learning rate, a constant between 0 and 1 that determines the extent to which new information overrides old one.
- r: the reward obtained after taking action a in state$s$.
- γ: the discount factor, a constant between 0 and 1 that discounts the importance of future rewards.
- $\max_{a'} Q(s',a')$: the maximum Q-value over all possible actions a' in the next state s'. This term accounts for the estimated maximum future cumulative reward.

In the multi-agent setting, the interaction between agents introduces an additional layer of complexity. Cooperative and competitive scenarios represent two primary types of multi-agent settings. In cooperative settings, agents work collaboratively towards a shared goal, promoting teamwork and coordination. Contrastingly, competitive settings involve agents with conflicting objectives, encouraging strategic decision-making and competition among them. Designing effective RL models for *BlastPursuit* involves considering the dynamics of these multi-agent settings, as the agents must adapt their strategies based on the behavior of both teammates and opponents.
RL has shown outstading results in different fields such as videogames [8], autonomous driving [9] or robot control [10]. As such, it is increasingly recognized as a powerful and versatile paradigm for training intelligent systems to learn complex and dynamic environments, demonstrating its potential for addressing a wide range of real-world challenges across various domains.

2.4 Related work

Several studies have explored multi-agent deep reinforcement learning (MARL) in game environments, often focusing on competitive settings.
In *Pommerman* [11], authors introduce a benchmark based on the classic Bomberman game, featuring scenarios with cooperative elements. The agents in Pommerman must learn to collaborate towards shared goals while navigating around the grid-world.
Another research [12] introduces the notion of Long Short-Term Memory (LSTM) and various state representations for training a PPO algorithm. However, the environment still consists of only one type of agent.
We will implement a unique approach to reinforcement learning, where a single model is tasked with controlling a set of agents belonging to two different teams, each with its own distinct goals.

3 Game Description

The game revolves around Bomber agents navigating through a grid interspersed with indestructible and destructible walls. The Bombers' main goal is to strategically place bombs to clear paths, destroy walls, and eliminate Killer agents, known as Killermen. Each bomb is set with a countdown timer and a defined blast radius, impacting nearby cells when detonated. Simultaneously, the Killermen move with the aim of surviving and disrupting the Bombers' strategies. These Killermen add an element of unpredictability and challenge, as they can potentially lead Bombers into traps or force them into dangerous situations. The Bombers must skillfully avoid these explosions and the Killermen's chasings to survive. The game concludes either when all Killermen or all Bombermen are eliminated, emphasizing the tactical interplay between the two types of agents.

3.1 Agents Description

- **Bomber Agents:** Bomber agents possess the capability to move in four directions within the grid and strategically plant bombs. Their primary objectives include navigating through a maze of obstacles, avoiding potential threats, and employing bombs to forge clear paths and target Killer agents. The Bombers' role is crucial in determining the flow of the game, as they must balance offensive actions with defensive maneuvers to survive and succeed.

- **Killer Agents:** Known as Killermen, these agents exhibit random movement patterns across the grid, contributing to the dynamic nature of the game. Their main goals are to endure the challenging environment and actively seek opportunities to undermine the efforts of Bomber agents. The presence of Killermen introduces a significant element of unpredictability and complexity, challenging the Bombers to adapt their strategies in response to the evolving game scenarios.

3.2 Grid World and Environment Setup

The *BlastPursuit* game unfolds within a grid environment, designed to offer a diverse and interactive playing field. This grid includes various cell types, each distinctly represented by unique images, aiding players in distinguishing between different game elements. The environment setup is as follows:

- **Empty Cells:** These cells represent the open space within the grid, offering freedom of movement for all agents. They are crucial for navigating the environment and positioning for bomb placements or evasions.
- **Indestructible Walls:** Serving as permanent barriers, these walls create a strategic layout within the grid. They are strategically placed to shape the gameplay, compelling agents to navigate around them and find alternate routes. Their presence adds complexity to the game, forcing agents to adapt to the constraints of the environment.
- **Destructible Walls:** Unlike indestructible ones, these barriers can be destroyed through bomb explosions. Destructible walls add a dynamic aspect to the game, as their removal alters the grid layout, opens new pathways, and provides opportunities for agents to reach previously inaccessible areas. This feature encourages strategic bomb placement and can be important in gaining an advantage over opponents.
- **Agents – Bombers and Killers:** The grid is populated with two types of agents: Bombers and Killers (Killermen). Bombers are tasked with clearing paths and targeting Killermen, while Killermen aim to survive and impede the Bombers' progress, trying to chase and catch them. The interaction between these two types of agents forms the core of the game's strategic depth.
- **Bombs:** Placed by Bomber agents, bombs are a critical element of the game. Each bomb is set with a specific countdown timer and blast radius, determining the area of impact upon detonation. The use of bombs is a key tactical tool for Bombers, enabling them to clear destructible walls, and attempt to eliminate Killermen. The strategic placement and timing of bombs are essential, as they can significantly influence the game's outcome. Bombs also pose a threat to their owners, requiring careful consideration of their placement to avoid self-harm.

4 Technical Details

In our development of *BlastPursuit*, we have implemented two interactive 7x7 and 11x11 grid environments, where autonomous agent strategically interact. The game's foundation is built upon the *Gymnasium* toolkit, which facilitates the development and comparison of AI agents using reinforcement learning. It allows easy representation and debugging of the main fundamental characteristics of agents:

- **State** (or observation) refers to the current situation or configuration of the environment that the learning agent is interacting with.
- **Actions** represent the set of possible decisions or moves that an agent can make in a given state, influencing the subsequent state the system transitions into.
- **Rewards** are numerical values assigned to the agent based on the consequences of its actions in a particular state, serving as feedback to reinforce or discourage certain behaviors and guide the learning process.

PettingZoo framework [13] allows for efficient extension to multi-agent environments. Algorithmic implementation is instead facilitated by *Stable-baselines3*. We have chosen to make a comparison of different algorithms as PPO, A2C and DQN; the first one being the most popular approach, the latter the most historical. *Supersuit*'s microwrappers [14] allow the connection between the environment and the algorithmic setup.

At initialization of the game, two teams composed of 3 agents each, randomly spawn on a position x, y of the grid. With them, 7% of the map is occupied by indestructible walls; 12% with destructible. A safety check is applied to avoid agents ending up surrounded by walls.
For the visual representation of the game, we developed a custom function to draw the grid and its elements, using Pygame. This function ensures that the current state of the game is accurately and dynamically represented, providing players and experimenters with a clear and engaging visual interface.

5 Implementation and RL Algorithms

5.1 Algorithmic settings

The RL algorithm selected for the comparison were kept with the standard hyperparameters provided by the *Stable-baselines3* library. In all of the settings, agents were trained for about 10M steps (steps here referring to the set of selected action for all the agents). An evaluation episode happens every $15.000 \cdot N$, where N is the overall number of agents; if the reward exceeds the previous saved one, the model policy is kept for later evaluation.

5.2 Observation state

The state is defined as a set of observation for each agent in game. Each observation is a matrix O_i, $i = 1, ..., N$ with N number of agents, with size $S \times S \times 2$, where S is the size of the chosen grid-world. The first layer of the matrix includes information about the various elements displaced in the map (where walls, agents and bombs are located). These elements are encoded with an integer ranging from 0 to 5; the matrix will be normalized before being fed to the neural approximator. The second layer encodes the position of the agent i considered, with a 1 in its position, 0 otherwise. This last layer is important to give an indicator to each agent about where they are located in the grid.
Each observation is enriched with the previous 3 time-steps matrix, therefore, for each agent the overall size of the matrix O_i is $(S \cdot 4) \times S \times 2$.
The set of observations is fed into the chosen neural approximator to obtain information useful for the Q-value maximization.

5.3 Actions

The action set varies depending on the agent under consideration. Both Bombermen and Killermen have the freedom to move in four directions (up, down, left, right). However, Bombermen possesses the additional ability to place bombs at their current location.
We have chosen not to include the option of remaining stationary to enhance the dynamism of the environment.

5.4 Rewards

The reward function holds huge importance within a RL algorithm, particularly in scenarios involving multi-agent control. Crafting appropriate rewards for each agent has proven challenging, primarily due to the competitive nature of the setting. The overarching objective involves optimizing the mean reward across all agents.
Our model operates on the premise of parameter sharing among agents. Consequently, an improperly designed reward function could introduce bias, unfairly favoring certain types of agents over others.
For both kind of agents, the reward function is given by R_B and R_W:

$$R_B = R_t + R_k + R_w + R_b + R_d + R_{win} + R_e + R_n + R_r$$

$$R_K = R_t + R_k + R_b + R_d + R_{win} + R_r$$

- R_t is the time-out reward; if the number of steps is over 50, the reward is -10, 0 otherwise.
- R_k is the kill reward; if an agent has killed an opponent, each element of the squad receive a reward of 10, 0 otherwise.
- R_w is the reward for Bombermen destroying a wall; it is 0.2, 0 otherwise.

- R_b is the bad-move reward; if an agent decides to move against a wall, it is -0.5, 0 otherwise
- R_d is the death reward; -20 if an agent dies, 0 otherwise.
- R_{win} is the win reward: each agent in the winning team receives 50, 0 otherwise. In this case, Killermen receive the reward only if Bombermen were not eliminated using their own bombs.
- R_e is a Bombermen's small negative reward to encourage exploration (if an agent did not kill, did not destroy a wall or placed a bomb); -0.1 if so, 0 otherwise.
- R_n is a Bombermen's small negative reward: if a bomb exploded and did not eliminate an opponent or did not destroy a wall, it is -0.1, 0 otherwise.
- R_r is the radius reward: if an agent is within a bomb radius, it receives -0.2, 0 otherwise.

5.5 Neural Approximation

The neural approximator's architecture consists of two straightforward 2D Convolutional Layers with a 3×3 kernel and ReLU activation, mapping the third channel of the images to 16 and 32 feature maps, respectively. Subsequently, the resulting matrix is flattened into a vector, followed by a linear layer that condenses the information into another vector with a length of 64.
Regarding the model selection, in approaches like PPO and A2C, the network's scheleton is commonly shared between the actor and the critic. Here, the actor selects actions based on the current state and outputs one of the actions, while the critic evaluates these actions, providing a real-valued number as output. Conversely, in the DQN network, the head directly outputs the chosen action.
In all the chosen models, we maintain alignment with the standard hyperparameters provided by the framework for both the neural approximator and the RL algorithm.

6 Results

The experiments were conducted using teams composed of 3 agents each, employing different algorithms such as DQN, PPO, and A2C across varying grid-world sizes (7×7 and 11×11).
The training phase of our multi-agent deep reinforcement learning (DRL) model for the Bombermen environment proceeded smoothly, demonstrating promising results in terms of convergence and performance. The model effectively learned the dynamics of the multi-agent environment, with Bombermen placing bombs to eliminate Killermen.
During the convergence, in all the models considered (PPO, A2C and DQN), the mean reward approximally reaches between -10 and 0 (see Appendix A). Perfect convergence would have been given if a greater reward would be obtained, that is when a team achieves a win against the other.
However, upon transitioning to the evaluation phase, unexpected behaviors surfaced, notably Bombermen exhibiting suicidal tendencies during the episodes. This behavior diverged from the expected learned strategies and impacted the overall performance of the model during evaluation scenarios.
As each agent is trained with a competitive objective, where their aim is to win while minimizing rewards for the opposing team, Bombermen exhibit a preference for self-elimination as a strategy to reduce rewards for their opponents.
Upon closer examination, it became evident that the discrepancy in the outcomes could be attributed to a potential misalignment in the reward function. The inherent complexity of the Bombermen environment, coupled with the enhanced interactions between the two kinds of agents, Bombermen and Killermen, may have contributed to the difficulty in accurately shaping the reward function, leading to suboptimal decision-making.

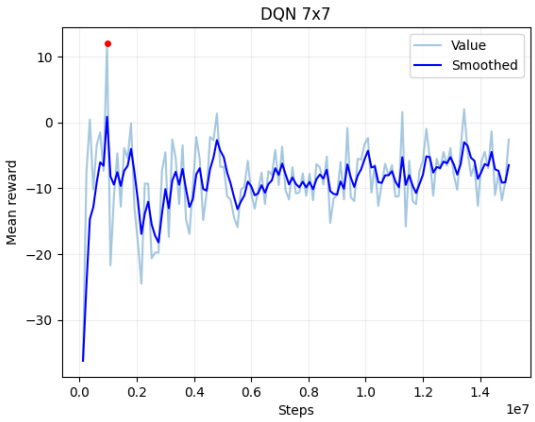
7 Conclusion

In this project, we developed a multi-agent Deep Reinforcement Learning model for BlastPursuit, a variant of Nintendo's Bomberman game, which comprehends different kind of agents, with different goals. It showcased promising training results with agents effectively learning how to maximize the reward function to compete within the multi-agent setting. However, challenges emerged during the

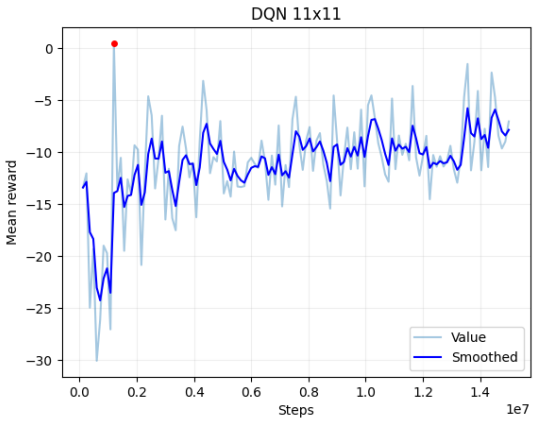
evaluation phase, particularly with Bombermen displaying unexpected behavior by prematurely terminating episodes through self-elimination.
Moving forward, addressing these challenges necessitates refining the reward function to better align with desired behaviors while mitigating unintended consequences. Additionally, exploring alternative training strategies, such as adversarial training, in which each team is independently trained using different models, holds potential for enhancing the model's adaptability and robustness across diverse scenarios.
By pursuing these avenues of research, we aim to further advance the capabilities of DRL in complex, competitive environments, ultimately facilitating the development of more resilient and effective intelligent agents.

Appendices

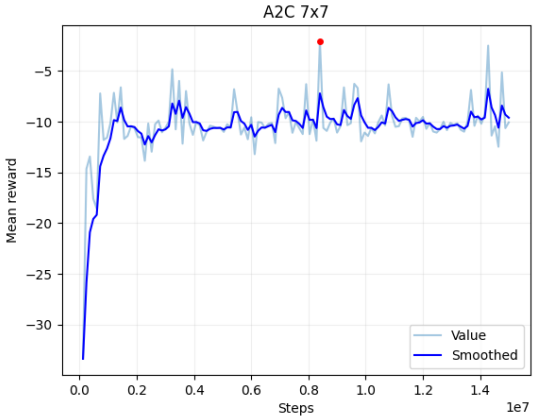
A Training results



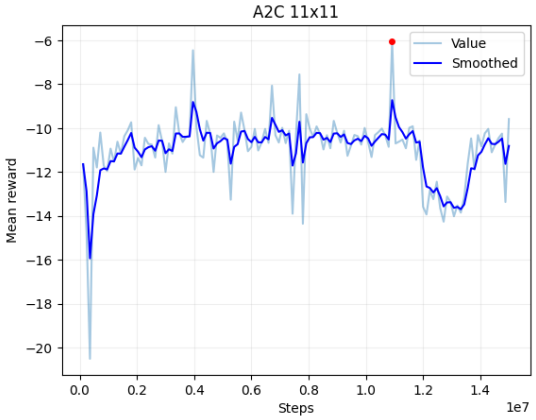
(a) DQN algorithm with 7×7 grid



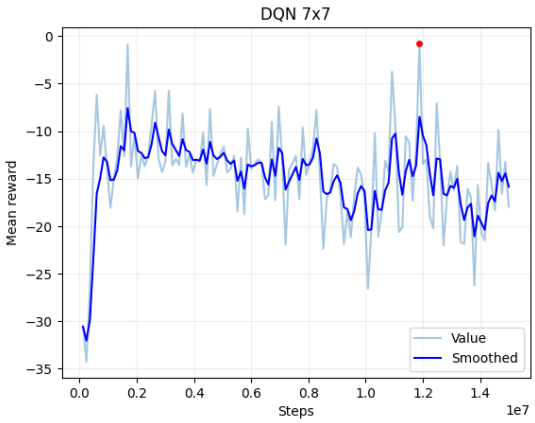
(b) DQN algorithm with 11×11 grid



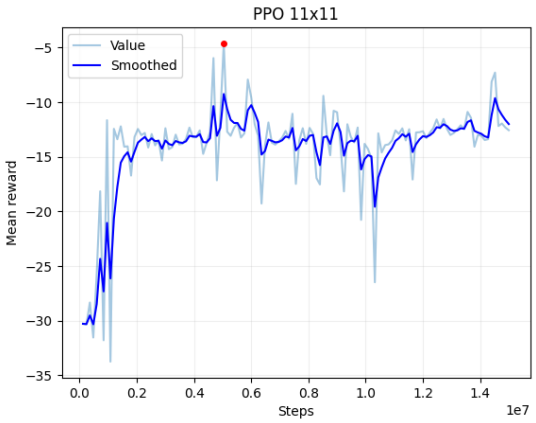
(c) A2C algorithm with 7×7 grid



(d) A2C algorithm with 11×11 grid



(e) PPO algorithm with 7×7 grid



(f) PPO algorithm with 11×11 grid

Figure 1: Mean reward for every possible combination of grid size and algorithms: it is worth noticing how all of them reach convergence, showing promising results thanks to the chosen state encoding.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[2] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[7] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. A Bradford book, MIT Press, 1998.

[8] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[9] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[10] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, IEEE, 2019.

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

[12] Goulart Faria Motta França, A. Paes, and E. Clua, *Learning How to Play Bomberman with Deep Reinforcement and Imitation Learning*, pp. 121–133. 11 2019.

[13] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15032–15043, 2021.

[14] J. K. Terry, B. Black, and A. Hari, "Supersuit: Simple microwrappers for reinforcement learning environments," *arXiv preprint arXiv:2008.08932*, 2020.