Cooperative Multi-Agent Reinforcement Learning in 2-Dimensional Bullet-Hell Games

Thomas Bird

Rensselaer Polytechnic Institute
tommycbird@gmail.com

Abstract

In this paper, I present a unique application of reinforcement learning for training cooperative multi-agent characters in a fast-paced, 2D, bullet hell game, using the Unity ML platform. My approach focuses on developing a single shared brain that emphasizes both cooperation and coordination. The results showcase the effectiveness of this approach in enabling the agents to defeat a boss character with varying behaviors and power-ups. This research contributes to the growing field of cooperative multi-agent reinforcement learning and highlights its potential for developing advanced AI in complex gaming environments.

Introduction

The field of reinforcement learning has seen significant advancements in recent years, with applications ranging from robotics to natural language processing. In this paper, I focus on applying reinforcement learning to train cooperative multi-agent characters within a 2D game inspired by the MMORPG Realm of the Mad God. The primary goal of the project is to create a team of agents capable of cooperating effectively to defeat a challenging boss character.

There are a number of approaches one can take to a problem like this. The biggest decision for this was whether to give the Agent's individual minds or to give them a collective brain. This decision will be detailed later.

Technology

The software I used is mostly all in Unity. I used Unity 2D to create the environment and all of the scripting. The reinforcement learning itself is power by Unity's ML-Agents library. Specifically, ml-agents v0.29.0, with PyTorch 1.13.1+cu117[1]. I also utilized TensorFlow for data visualization. For what it counts, I drew the pixel art in Piskel.

Motivation and Background

The motivation for this project stems from the increasing interest in cooperative multi-agent systems, particularly in gaming contexts. In many modern games, players must work together as a team to overcome challenges, necessitating the development of AI agents that can cooperate and adapt to various situations. Furthermore, emulating a realistic gameplay experience, where players communicate and strategize in real-time, serves as an additional motivation for the project.

Realm of the Mad God (henceforth ROTMG) is a game I had played growing up and is the inspiration for this project. It is very fast-paced and involves dodging hundreds of projectiles continuously; hence the name of the genre, bullet hell. This made it a very interesting choice to apply reinforcement learning to, as the agents would have to learn to avoid a plethora of on-screen hazards.

ROTMG gameplay is typically centered around boss fights. You either strive to defeat over-world bosses, or enter dungeons where you'll encounter a boss in the final chamber. To replicate this, I programmed a boss fight of my own, as well as three playable agent characters inspired by the game.

The environment is an island, where the three agents are confined to. The boss of this arena is a cyclops character dubbed "Polyphemus." He boasts rudimentary AI, with some set behaviors and attacks which will be detailed later.

These are the three agents:

- **Wizard**: A high attack, high range, but low health character. Effectively a glass cannon.
- **Priest**: A low attack, high range, and medium health character. A well-balanced support.
- **Warrior**: A high attack, low range, and high health character. The tank of the group.

Each of the agents are able to move in all directions, shoot in all directions, and activate their unique special moves on a cool-down just like ROTMG. The wizard can fire a high damage projectile, the warrior can buff nearby allies by temporarily boosting their fire-rate, and the priest can heal nearby allies.

With this mix of traits, the agents should hopefully cooperate to overcome Polyphemus.

Related Work

Previous work in cooperative multi-agent reinforcement learning has explored various aspects, such as learning algorithms, exploration strategies, and communication mechanisms. Examples include research on the StarCraft II envi-

[1]A version that allows you to train with GPU.

ronment (Vinyals et al. 2019) and OpenAI's work on Dota 2 (OpenAI 2019). While these projects demonstrate the potential of reinforcement learning in creating advanced AI for gaming, there remains room for further exploration in 2D gaming contexts and distinct cooperation strategies. Here are some examples:

- Cooperative Deep Reinforcement Learning in Parameterized Action Space (CDRLPAS) (Papoudakis et al. 2019): This work introduces a novel cooperative deep reinforcement learning framework in parameterized action space for multi-agent problems. The proposed framework could be adapted to the 2-dimensional bullet-hell games context, like Realm of the Mad God.
- Learning to Communicate in Multi-Agent Reinforcement Learning (Foerster et al. 2016): This paper presents a method that allows reinforcement learning agents to learn communication protocols while training, enabling improved cooperation between agents. This approach could be particularly useful in a fast-paced 2D game environment where agents must communicate effectively.
- Multi-Agent Reinforcement Learning for 2D Platform Games (Moura and Campos 2019): This work explores the application of multi-agent reinforcement learning to 2D platform games, particularly the Mario AI Benchmark. While this focuses on platformers, it demonstrates the potential for using multi-agent reinforcement learning in 2D gaming contexts.
- Cooperative Heterogeneous Deep Reinforcement Learning (Yang, Zhang, and Wu 2019): This research proposes a cooperative heterogeneous deep reinforcement learning framework, where agents with different roles collaborate to achieve a common goal. This framework can be adapted to address the challenges in 2-dimensional bullet-hell games like Realm of the Mad God, where players with different classes must work together.

These examples highlight the potential of applying cooperative multi-agent reinforcement learning techniques to 2-dimensional bullet-hell games. By building upon these works, my project aims to demonstrate the effectiveness of these approaches in the context of Realm of the Mad God.

Approach

To train the three agent characters (wizard, warrior, and priest), I employ a single shared "hive-mind" brain, which enables them to cooperate more effectively than individual brains. This design choice is inspired by the original game, where players communicate and coordinate their strategies in real-time. Using the Unity ML platform, I implement reinforcement learning to enable the agents to adapt and learn strategies for defeating the boss character, Polyphemus.

The Boss AI

The boss was created with rudimentary AI, such that he would have set and predictable behavior (just like in ROTMG). This behavior can mostly be gleaned from the provided pseudocode in 1.

In this algorithm, the basic AI is outlined.

Algorithm 1: Polyphemus Behavior

Update Runs every frame

```
1:  if !phase2 AND at half hp then
2:      firerate *= 1.5
3:  end if
4:  if !crying then
5:      // SLASH BEHAVIOR
6:      Agent a = closest agent
7:      move position toward Agent a
8:      if Agent a is in range then
9:          fire slash attack with Agent a as target
10:     end if
11:     if HP reaches 75%, 50%, or 25% then
12:         crying = true
13:         rewardAgents(50) //for making him cry
14:     end if
15: else
16:     // CRYING BEHAVIOR
17:     if not at the center then
18:         move to the center
19:     else
20:         fire tears //alternates angle after each call
21:     end if
22:     if 15 seconds have elapsed since start of cry then
23:         crying = false
24:     end if
25: end if
```

Lines 4-14 are Polyphemus' actions when in his default, "Chase and Slash" behavior. He simply locates the nearest agent, moves toward it, and (if he is in range) he will fire the slash attack at their direction.

Lines 15-24 are Polyphemus' actions when in his "Crying" behavior. Three times throughout the boss fight, he will walk to the middle of the arena and burst out tears radially. They will also alternate slightly in angle so as to cover the most area as possible.

Lines 1-3, lines 11-14, and lines 22-24 are simply checks to see whether Polyphemus needs to change his behavior.

The code is as simple as it is so that the boss's move set and attacks can be varied, but ultimately still predictable for the agents and players alike.

The Agents

Here I will outline the agents[2] and the details for observing, rewarding, and selecting actions.

Actions There are a total of 12 continuous actions, as well as 6 discrete actions in the action space.

As each agent can travel or shoot in any direction, two continuous actions compose each of their movement vectors and two more compose their aiming direction vector. That makes four continuous for each of the three agents, or 12 total.

[2]When I refer to "the agents" I am referring to the single brain that they share.

trainer_type	ppo	batch_size	128
buffer_size	4096	learning_rate	0.0002
beta	0.01	epsilon	0.2
lambd	0.95	num_epoch	10
hidden_units	512	num_layers	2
gamma_extrinsic	0.98	strength_extrinsic	1.0
max_steps	8000000	time_horizon	64
summary_freq	50000		

Table 1: Hyperparameters used for training with PPO

As for discrete actions, each agent can either be shooting, not be shooting, be using their special, or not be using their special. With two discrete actions each, that makes 6 total.

Observations The observation space totals to 60. They are as follows:

- 1: the current time
- 15: the agents [5 for each agent]
- 8: the boss
- 36 for the projectiles [4 for each projectile]

Rewards and Punishments Algorithms like the one shown in Algorithm

Hyperparameters

The hyperparameters I used to accomplish good training results can be seen in 1. I only ever altered the gamma from 0.99 to 0.98, and I altered the max_steps from 5 million to 8 million. From there I never again saw a need to change these settings.

Results

The results of the training process are presented using several TensorFlow graphs, which illustrate the agents' progress in learning to cooperate and defeat Polyphemus. The agents demonstrate an ability to effectively utilize their unique abilities and coordinate strategies to overcome the boss character's varying behaviors and power-ups.

Trial and Error

Throughout the course of training there were several hiccups and mistakes made, and I have detailed them all in the following runs. Each run represents a different training model, with different parameters. Below I detail each of their unique characteristics and the changes that they required.

Runs 1-2 These runs are preliminary runs to test the base incentive values. This resulted in a pretty simple strategy wherein the wizard ran in circles to distract the boss while the priest and warrior sat in the corner buffing one another. Initially, to incentivize team-play, there was a reward given for buffing or healing an ally[3] so the two agents with support abilities (the warrior and priest) farmed this reward while the wizard dealt with the boss. This is obviously not intended behavior, so I decided to slightly tweak some rewards and remove this flat buff/heal reward to make it more dynamic.

[3]Regardless of whether anyone needed it.

Run 3 This run was particularly interesting as it led to a strategy wherein the wizard and warrior agents immediately walked into the boss to die, and the priest then kited the boss in circles. It would occasionally take damage intentionally and then heal itself. This is because the reward for healing didn't outweigh the punishment for taking damage, so it incentivized hurting oneself so as to heal it back and gain more net reward. So I then made it so the priest got the reward based on the percentage of health healed, and was punished for taking damage equally plus a flat -0.1f punishment.

It was initially my intention to allow the agents to observe every projectile of the bosses (as there can be around 100 on screen during the cry phase). This led to terribly long learning times, so I decided to cut it. Now they can only observe 9 projectiles max (instead of 120), and these observations get padded during the cry phase. This may be perplexing, as one may ask how will they know where the tear projectiles are during the cry phase? This is because those projectiles will always fall in the same places and at the same angles. In place of the 120 projectile observations, I gave them knowledge of when the boss is crying, and at which point he begins to cry. They can then infer where the tears are at that time based on the agents' positions relative to the center of the arena.

Run 4 This run also had an interesting strategy. To incentivize hitting the boss I had initially given out a flat reward of 0.1f for every projectile that made contact as well as a percentage of the damage that was done. This then implied that there was potential for a larger total reward if you killed the boss with more projectiles. For instance, say the boss had 100 health, agent A dealt 5 damage per projectile, and agent B dealt 20 damage per projectile. If agent A attacked the boss alone, it would take 20 projectiles and give a flat 2.0f reward on top of the dynamic damage-based reward, whereas agent B would only get 0.5f total as it would kill the boss in 5 hits. In the scope of this demo, this led to a behavior wherein the priest would be the sole attacker, and the wizard and warrior would simply not attack. I then removed this flat reward.

Run 5 During this run the agents would make it to through the first cry phase and occasionally the second. While it was the furthest they had ever made it, their strategy was lacking. It seemed that the wizard was the only one doing the work, and the other two agents would simply die immediately, allowing the wizard to solo the boss. I then realized I had a choice to make so that I could inspire the agent's to live longer and fight together. Thus, I chose to make the base action punishment a function of the number of dead agents, and increase the agent death punishment.

Run 6 Eureka! They've finally conquered the boss for the first time. I would be celebrating, however, their tactics simply did not exhibit any cooperation. As the boss character chases whoever's nearest, his movement becomes less predictable when more agents are on screen. In other words, it reduces the variability in the game when you take some of the agent characters out of the picture. At the beginning of each run for this model, the warrior would immediately

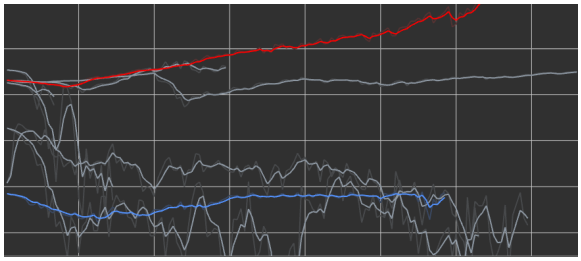

Figure 1: Cumulative Agent Reward

.

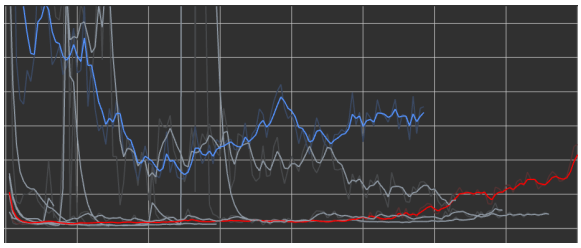

Figure 2: Agent Value Loss

.

be sacrificed. In an attempt to fix this behavior I made it so episodes end on the death of a single agent.

Run 7-8 Not only did the previous change lead to no victories, it led to hardly any progress in the fight. Training took 24-hours, and while better results could have come up with more training time, there was no indication that they were progressing. After some reflection, I also realized that ending the episode on a single agents death was against the nature of the game, so I had to inspire them to win in another way. To figure this out I asked myself, why is the warrior the first to be sacrificed.

I then realized that it was due to the nature of accuracy. In my model, I punished agents 0.1f for shooting a projectile, but rewarded 0.1f back to them upon hitting the boss. Therefore, you were punished for missing. This behavior was never learned, and my method didn't seem to work. Furthermore, since the warrior buffs himself the most and has the shortest range, he frequently has a boosted fire rate and the least accuracy of the agents. This means he was racking up tons and tons of inaccuracy punishments, so I removed this system.

Run 9: The Final Run After over 150 hours of training, and over 70 million steps of training across all models, I have arrived at decent results. The boss is killed (in about 1 in every 5 runs), the priest heals hurt allies, and the warrior buffs the wizard and priest. The tactics are not particularly efficient, but they are safe, strategic, and effective.

TensorFlow Graphs

The results and progress of each of these runs can be seen in the TensorFlow graphs. 1 displays the Cumulative Reward over time for each environment, and 2 shows the Value Loss.

Some important or at least notable trends would be in run 6 and 9, as those two saw the most success. I've highlighted them in the figure to make them stand out.

Run 6 (red) can be seen to have a very steady incline in terms of reward, and a loss that stays low but slowly climbs up toward the end. This is indicative that the predicted value estimates are increasingly diverging from the actual value estimates. In other words, the model is becoming less accurate in predicting the expected cumulative rewards from a given state-action pair over time. This is a curious trend coming from one of the most successful runs, and I believe it is due to a trade off in exploration and exploitation tradeoff. That is, as the model becomes better at exploiting its current knowledge of the environment, it may explore less frequently.

Run 9 (blue) has a very steady, seemingly flat reward graph. This is no illusion. Despite the model's success, it actually didn't have much change in its reward. The model itself made it further and further into the fight, but ultimately there was little variation in his mean reward. As for its losses, they were heavily varied, but generally decreased and tapered out toward the end. This suggests that the model has reached a stable performance level, where it has learned to consistently tackle the task at hand but may not be fully optimized for achieving the highest possible rewards or exploring more efficient strategies. The model was successful, but perhaps not at the same caliber that Run 6 was.

Discussion and Future Work

This project showcases the potential of cooperative multi-agent reinforcement learning in developing advanced AI for complex gaming environments, particularly in 2D contexts. Future work may explore alternative cooperation strategies, refinements to the "hive-mind" brain approach, or extensions to other gaming genres. Additionally, incorporating more advanced communication mechanisms among agents could further improve the AI's ability to adapt and strategize during gameplay.

My aim was to make an environment that best mimicked multiple players who are constantly communicating and working toward a common goal. I am not completely satisfied in this regard, and am eager to try new approaches to this problem in the future. Bullet hell games are interesting as their are many many variables to take into consideration given the nature of the genre. It is difficult then to make a perfect model which can account for all of the variables and still manage to produce consistent, positive results.

Conclusion

In this paper, I explored the application of reinforcement learning to train cooperative multi-agent characters within a 2D game inspired by the MMORPG Realm of the Mad God. The primary goal of the project was to create a team of agents capable of cooperating effectively to defeat a challenging boss character. Using a single shared "hive-mind" brain, I trained the agents to coordinate their unique abilities and strategies to overcome the boss character, Polyphemus.

Through multiple iterations of the training process, I experienced various challenges and discovered interesting strategies developed by the agents. These trials and errors allowed me to refine the rewards, punishments, and hyperparameters to ultimately achieve a satisfactory level of cooperation and performance.

Although the agents' tactics were not always efficient, they demonstrated safe, strategic, and effective cooperation. The agents were able to defeat the boss in approximately 1 in every 5 runs, with the respective agents abiding by their class-specific roles[4]. My project contributes to the growing body of research in cooperative multi-agent reinforcement learning, particularly within the context of 2D gaming environments. It also highlights the potential of using reinforcement learning to create advanced AI for gaming, opening avenues for further exploration and improvement in the field.

Acknowledgments

I would like to express my appreciation to Professor Mei Si for her exceptional teaching and invaluable guidance throughout the course of this project. Her expertise and dedication have been inspiring.

I would also like to extend my gratitude to DECA Games, the team behind Realm of the Mad God, for creating an engaging and challenging environment that provided a rich foundation for this research. Their continued support and commitment to the gaming community have been instrumental in the success of this work. It would be interesting to see DECA incorporate ML like this into the game some day.

Furthermore, I would like to acknowledge the developers of Unity's ML-Agents toolkit for their significant contributions to the field of machine learning and game development. Their open-source framework has made it possible for researchers like myself to explore innovative applications of artificial intelligence in gaming contexts.

Lastly, to USD student Daniel Daugbjerg, I want to acknowledge the art he contributed to the project. He made all of the boss and agent sprites.

References

Grandmaster level in StarCraft II using multi-agent reinforcement learning Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

OpenAI Five OpenAI. 2019. OpenAI Five. https://openai.com/projects/five/.

Cooperative Deep Reinforcement Learning in Parameterized Action Space Papoudakis, G.; Christianos, F.; Schäfer, L.; and Albrecht, S. V. 2019. Cooperative Deep Reinforcement Learning in Parameterized Action Space. In *Advances in Neural Information Processing Systems*, 12335–12345.

[4]That is, wizard for damage, priest for support, and warrior as a mix.

Learning to communicate with deep multi-agent reinforcement learning Foerster, J.; Assael, I. A.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2137–2145.

Multi-Agent Reinforcement Learning for 2D Platform Games Moura, T. S.; and Campos, L. C. 2019. Multi-Agent Reinforcement Learning for 2D Platform Games. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Cooperative Heterogeneous Deep Reinforcement Learning Yang, Z.; Zhang, Y.; and Wu, Y. 2019. Cooperative Heterogeneous Deep Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 4708–4714. AAAI Press.