Reinforcement Learning in Competitive Game Environments with Unity ML-Agents

Yuvraj Chauhan
IIIT Vadodara - International Campus Diu
202211098@diu.iiitvadodara.ac.in

Srijan Sharma
IIIT Vadodara - International Campus Diu
202211084@diu.iiitvadodara.ac.in

Trijay Patel
IIIT Vadodara - International Campus Diu
202211066@diu.iiitvadodara.ac.in

Suraj Kumar
IIIT Vadodara - International Campus Diu
202211091@diu.iiitvadodara.ac.in

***Abstract*—Reinforcement Learning (RL) has gained prominence as an effective approach for training intelligent agents in dynamic, interactive environments, including video games. This study explores the application of Unity ML-Agents Toolkit in developing competitive agents using Proximal Policy Optimization (PPO). The agents were trained to strategically move, attack, and defend in a two-player competitive environment.**

***Index Terms*—Reinforcement Learning, Unity ML-Agents, Proximal Policy Optimization, Game AI, Competitive Agents, Self-Play.**

I. Introduction

The integration of Artificial Intelligence (AI) into video games has evolved significantly, from pattern-based AI in classic arcade games to advanced neural networks in modern applications. Reinforcement Learning (RL) enables agents to learn optimal actions in environments by maximizing cumulative rewards through trial and error.

This work Utilizes Unity's ML-Agents Toolkit to train agent in a game environment. The primary objective is to develop competitive AI capable of outperforming opponents under various game scenarios.

This work focuses on training agents in a two-player competitive environment using Unity ML-Agents Toolkit. The primary objectives are:

- Developing agents capable of executing strategic behaviors, such as approaching opponents, attacking effectively, and using defensive shields.
- Utilizing RL techniques, particularly Proximal Policy Optimization (PPO), to train agents in a structured and competitive setting.
- Evaluating the performance of trained agents in both familiar and unfamiliar game scenarios.

The study provides insights into the challenges of training competitive agents, the effectiveness of PPO in multi-agent environments, and the importance of reward design and observation space configuration. This work aims to contribute to the broader understanding of RL applications in game AI.

II. Background

A. Unity ML-Agents Toolkit

The Unity ML-Agents Toolkit is a versatile framework designed to bridge the gap between game development and machine learning research. Key components of the toolkit include:

- **Agents**: Game objects representing AI players. Agents perceive their environment through sensors and interact using a defined set of actions.
- **Sensors**: Tools that allow agents to collect observations about the environment. These can include visual inputs, raycasts, or vector-based data.
- **Academy**: The central manager of the training process. It handles environment resets, episode timing, and parameter updates.
- **Behavior Parameters**: Defines how the agent interacts with its environment during training or inference. This includes specifying observation and action spaces, decision-making methods, and policy models.

The toolkit supports various machine learning algorithms, including Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC), making it suitable for a wide range of learning tasks. Additionally, its integration with the Unity engine provides a rich simulation environment for testing and training.

B. Reinforcement Learning and PPO

Reinforcement Learning (RL) is a framework where agents learn to maximize cumulative rewards by interacting with their environment. In RL, an agent observes the state of the environment, takes actions, and receives rewards or penalties based on the outcomes. The agent's goal is to learn a policy that maps states to actions, maximizing long-term rewards.

Proximal Policy Optimization (PPO)[1] is a policy gradient method designed to improve the stability and efficiency of RL training. PPO uses a clipped surrogate objective function to limit the size of policy updates, ensuring that the learning process remains stable.

PPO is widely used in RL applications due to its simplicity, robustness, and ability to handle both discrete and continuous action spaces effectively.

C. Challenges in Competitive RL

Competitive environments add complexity to RL tasks, as agents must learn not only to optimize their actions but also to anticipate and counteract the strategies of opponents. This dynamic interaction creates a non-stationary environment, where the optimal strategy continuously evolves. To address this, techniques such as self-play and adversarial training are often employed, enabling agents to adapt to changing scenarios and learn diverse strategies.

III. Methodology

A. Environment Design

The training environment was developed in Unity, designed to simulate a competitive two-player scenario. Key features include:

- **Arena Setup**: A bounded area where agents compete by attacking each other and avoiding damage.
- **Game Mechanics**: Each agent has a health bar, and actions such as attacking or defending affect the opponent's health and the game outcome.
- **Dynamic Elements**: Environmental features, such as obstacles or terrain variations, encourage strategic positioning and movement.

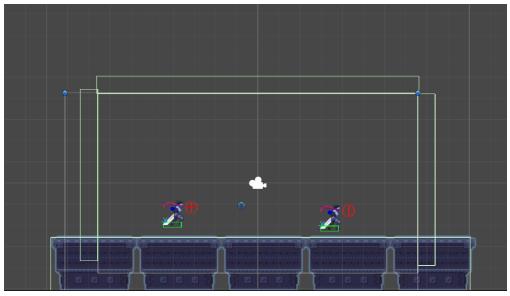

Fig. 1. Game enviroment

B. Observation Space

The observation space defines the information available to agents for decision-making. In this setup, agents observe:

- Their position, velocity, and health status.
- The opponent's position, velocity, and shield status.
- Environmental data, such as proximity to the ground and attack range.

The observation space is structured as a numerical vector, providing a concise representation of the environment while retaining all essential information.

C. Action Space

The action space, refer fig 2, defines the set of possible actions an agent can take at each decision step. Agents have the ability to choose between a total of 6 discrete actions to perform. The actions are divided across 4 separate branches with 3 actions in first branch and in other branches, 2 actions each. Agents are only allowed to choose one action from each branch at a time during decision periods. These actions enable a combination of offensive and defensive strategies, fostering dynamic interactions between agents.

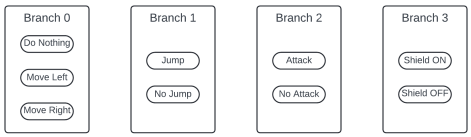

Fig. 2. Action Space

D. Reward System

Agents can receive rewards during training in order for the trainer to optimize its behavior policy. They can be both positive and negative, resulting in either penalizing or reinforcing a certain behavior. In this manor, rewards are accumulated with each step by either incrementation or decrementation.

Action/Event	**Reward/Penalty**
Approaching Enemy with Low Health	+0.1 (if distance decreases)
Moving Away from Enemy	-0.2 (if distance exceeds 5 units)
Successful Attack	+1.5
Attack Blocked by Enemy Shield	-0.5
Missed Attack	-0.2
Taking Damage	-1.0
Enemy Defeated	+2.0; End Episode
Player Defeated	-2.0; End Episode
Shield Activation	No reward
Shield Absorbs Damage	+1.0
Shield Deactivation (Timeout)	No reward
Episode Timeout with Higher Health	+1.0 (for winner)
Episode Timeout with Tied Health	+0.5
Episode Timeout with Lower Health	-1.0 (for loser)

TABLE I
Reward Table for Agent

E. Reasoning for the Reward System

The reward system is designed to promote optimal agent behavior through positive reinforcement for effective actions and penalties for poor decisions. Below is the reasoning for each category:

Positive Rewards:

- **Approaching Enemy with Low Health** $(+0.1)$**:** Encourages aggression against vulnerable opponents to maximize winning chances.
- **Successful Attack** $(+1.5)$**:** Rewards precise execution of attacks, crucial for defeating the opponent.
- **Enemy Defeated** $(+2.0)$**:** Reflects the primary goal of the episode.

- **Shield Absorbs Damage ($+1.0$):** Reinforces effective defensive timing and strategy.
- **Episode Timeout with Higher Health ($+1.0$):** Rewards resource management and survivability.
- **Episode Timeout with Tied Health ($+0.5$):** Acknowledges partial success, incentivizing improvement.

Negative Rewards (Penalties):

- **Moving Away from Enemy (-0.2):** Discourages passive play and loss of pressure.
- **Attack Blocked by Enemy Shield (-0.5):** Penalizes ineffective attacks and resource waste.
- **Missed Attack (-0.2):** Encourages accuracy without overly discouraging aggression.
- **Taking Damage (-1.0):** Motivates avoidance of harm and better defense.
- **Episode Timeout with Lower Health (-1.0):** Encourages improvement and prioritizing health.

Neutral Actions:

- **Shield Activation (No Reward):** Prevents overuse of basic mechanics.
- **Shield Deactivation (Timeout, No Reward):** Reflects expected system behavior.

This reward system balances aggression, defense, and strategic decisions, guiding the agent to achieve the desired objectives effectively.

IV. Hyperparameters

Hyperparameters are critical for the efficiency and stability of training. Key parameters include:

- **Batch Size**: The number of training samples processed in each iteration. Larger batch sizes improve stability but increase computational requirements.
- **Learning Rate**: Controls the step size for policy updates. A smaller learning rate ensures gradual learning, while a larger rate accelerates convergence but risks instability.
- **Discount Factor**: Determines how much future rewards are valued compared to immediate rewards, shaping the agent's strategy for long-term planning.
- **Entropy Coefficient**: Encourages exploratory actions, preventing agents from converging prematurely on suboptimal policies.
- **Clipping Parameter**: Limits the magnitude of policy updates to maintain stability during training.

These parameters were tuned iteratively to balance training efficiency and performance.

V. Training

The training process uses a self-play method where two agents compete in a two-player environment. To improve learning, the agents switch teams periodically. This helps them gain experience in both offensive and defensive roles, making their behavior more adaptable and effective.

A. Self-Play Training Method

Self-play is a way for agents to improve by competing against themselves or other agents. The main features of this method are:

- **Simultaneous Training**: Both agents are trained at the same time, using the same type of neural network but with separate strategies.
- **Learning Through Competition**: Agents start with random strategies and improve by learning from their matches.
- **Dynamic Environment**: The competitive environment forces agents to adapt as their opponents' strategies evolve.

B. Team Switching for Better Learning

A key part of the training is switching teams regularly, which works as follows:

- **Switching Roles**: Agents take turns being Player 1 and Player 2, so they experience both offense and defense.
- **Balanced Experience**: Switching prevents agents from focusing only on one type of role, helping them develop broader skills.
- **Learning Diverse Strategies**: By playing both sides, agents learn to predict and counter their opponent's moves, making them more versatile.

VI. Results

This section presents the performance of three distinct training configurations evaluated using the *Environment Cumulative Reward* metric. The results are visualized through graphs where the *gray line* represents Configuration 1 (Speed-Oriented Training), the *pink/magenta line* represents Configuration 2 (Robust Learning), and the *red line* represents Configuration 3 (Exploration-Focused Training).

- **Configuration 1: Speed-Oriented Training (Gray Line)**
  - **Objective:** Focused on quick iteration to test agent behaviors, reward structures, and environment setup.
- **Configuration 2: Robust Learning for Complex Strategies (Pink/Magenta Line)**
  - **Objective:** Designed for environments requiring intricate strategies and long-term planning, with a focus on stability and precision.
- **Configuration 3: Exploration-Focused Training (Red Line)**
  - **Objective:** Encourages extensive exploration in environments with sparse or deceptive rewards.

A. Cumulative Reward

The cumulative reward achieved during training highlights the following:

- **Gray Line:** Configuration 1 serves as a baseline with moderate learning rates and standard network architecture. It shows a steady but slower growth in cumulative reward over time.

- **Pink Line:** Configuration 2 emphasizes exploration with a lower learning rate and broader network settings. It achieves a significantly faster and sustained increase in cumulative reward, outperforming the others.
- **Red Line:** Configuration 3 focuses on exploitation through higher learning rates. It shows an initial rapid increase but plateaus, indicating limited long-term improvements.

The pink line (Configuration 2) demonstrates the best performance in terms of cumulative reward growth, showcasing the effectiveness of balanced exploration and learning.

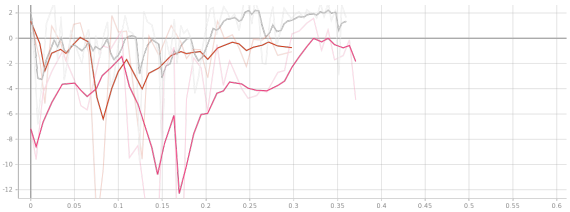

Fig. 3. Cumulative reward in the environment.

B. Policy Loss

The policy loss during training offers insights into how well each configuration optimizes its policy over time:

- **Gray Line:** Configuration 1 shows stable policy loss reduction but converges slowly, reflecting a conservative training setup.
- **Pink Line:** Configuration 2 again stands out with consistent and smooth policy loss reduction, achieving lower loss values earlier in the training process.
- **Red Line:** Configuration 3 initially reduces policy loss rapidly but struggles with stability, resulting in oscillations and a higher final loss.

The pink line (Configuration 2) achieves the lowest and most stable policy loss, reinforcing its superior training dynamics.

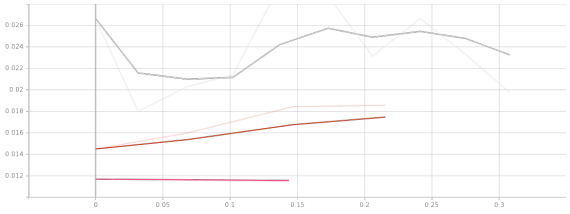

Fig. 4. Policy loss.

C. Value Loss

The value loss, which reflects the model's error in estimating the value function, shows the following trends:

- **Gray Line:** Configuration 1 exhibits a gradual decline in value loss, mirroring its slower but stable training progress.
- **Pink Line:** Configuration 2 achieves the most rapid reduction in value loss, maintaining stability and lower values throughout training.
- **Red Line:** Configuration 3, while initially reducing value loss quickly, shows fluctuations and higher final loss values compared to Configuration 2.

The pink line (Configuration 2) again leads in terms of stability and final loss, indicating a more accurate estimation of the value function during training.

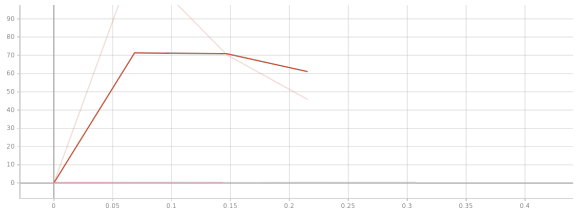

Fig. 5. Value loss.

D. Discussion

The comprehensive evaluation across cumulative reward, policy loss, and value loss reveals Configuration 2 (pink) as the most effective approach, achieving the highest rewards, the lowest policy loss, and the most accurate value function estimation. Configuration 1 (gray) provides a reliable baseline with slower but steady improvements, while Configuration 3 (red) demonstrates aggressive initial learning but suffers from instability and weaker long-term performance.

VII. Conclusion

This study demonstrates the potential of RL for training agents in competitive game environments. While PPO and Unity ML-Agents provided a strong foundation, further refinements are needed to enhance robustness and generalization. These findings contribute to the growing field of game AI and RL applications.

References

[1] J. Schulman, Y. Wu, J. Donahue, P. Dhariwal, and A. Ashfordin, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347.

[2] P. Andersson, *Future-proofing video game agents with reinforced learning and unity ml-agents*, 2021.

[3] G. Martinez-Arellano, R. Cant, and D. Woods, "Creating ai characters for fighting games using genetic programming," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, pp. 1–1, Dec. 2016. DOI: 10.1109/TCIAIG.2016.2642158.

[4] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, *Creating pro-level ai for real-time fighting game with deep reinforcement learning*, Apr. 2019. DOI: 10.48550/arXiv.1904.03821.

[5] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 12, no. 1, pp. 1–26, 2020. DOI: https://doi.org/10.48550/arXiv.1912.10944.