Multi-Agent RL in a Competitive Tron Light-Cycle Environment

Erick, Buitrago
Systems Engineering Student
Universidad Distrital Francisco José de Caldas
Email: esbuitragop@udistrital.edu.co

Juan, Jiménez
Systems Engineering Student
Universidad Distrital Francisco José de Caldas
Email: juajimenezp@udistrital.edu.co

This paper was written as part of the final project for the System Sciences Foundations course.

***Abstract*—Reinforcement learning has shown promising results in enabling agents to make decisions in dynamic and adversarial environments. This paper presents the development of a competitive Tron-like game in which two AI agents are trained using multi-agent reinforcement learning techniques, incorporating a custom observation tensor and a reward structure. The trained agents learn to avoid threats such as walls, trails, and opponents, though sustained exploration requires careful tuning of epsilon parameters to prevent early convergence.**

***Index Terms*—System Sciences, Multi-Agent Reinforcement Learning, Tron Game, Mixed Cooperative-Competitive Setting**

I. Introduction

To explore and analyze the foundational concepts of system sciences, we developed a project involving the training of artificial intelligence (AI) agents within a mixed cooperative-competitive environment. The chosen game is Tron, a game in which two teams take part, each with at least one member. The present proposal involves four agents, split into two teams. Players control a motorcycle that leaves behind a light trail acting as a barrier. The objective is to force the opponent to crash, either into the map's limits or into a light trail.

The main challenges arise from the highly dynamic nature of the system, which is constantly changing and where each action has repercussions not only on the environment but also on the agents themselves. The system lacks a stable equilibrium until the very end of the game when all conditions reset. Another way it can achieve its equilibrium is when both teams have an equal number of wins.

Many solutions have been proposed over the years. For instance, as shown in [1], algorithms have been used to estimate and maximize the available space for each agent. This often results in a passive playstyle focused on limiting the opponent's movement. In another example, [2] suggests using sensors to help agents detect proximity to obstacles, rewarding area coverage rather than victory. Our system seeks to prioritize competitive interactions and winning strategies.

II. Methods and Materials

The proposed solution involves developing four competitive autonomous agents that must learn to survive and defeat their opponent. The environment is built using PyGame, enabling detailed control over graphics, movement, collisions, and events. Each agent leaves behind a light trail, which acts as a lethal obstacle, making the game highly competitive.

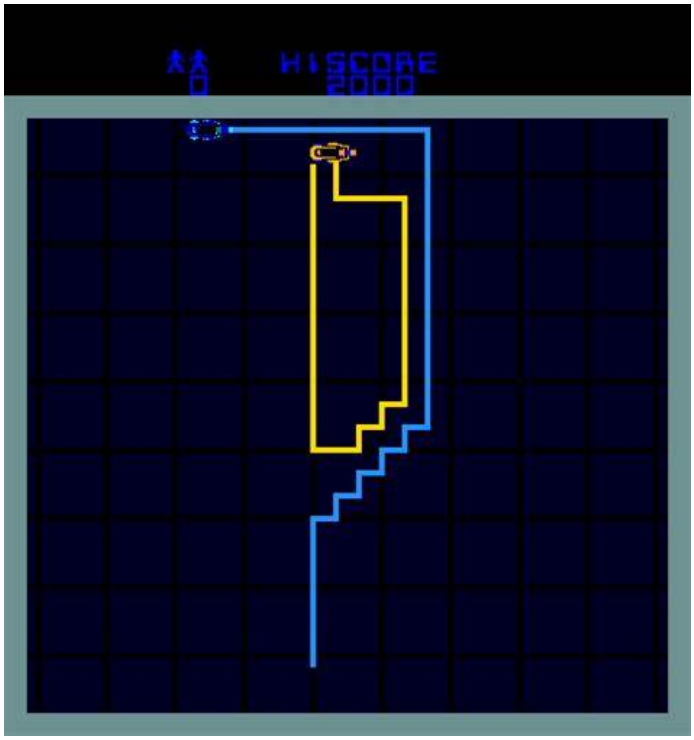

Fig. 1. Game board with both agents and their light trails.

To enable agents to perceive and interact with their environment, a dynamic and directional cone of vision was implemented instead of a fixed full-map tensor. This cone simulates realistic visual perception by including only tiles within a forward-facing triangle. Occlusions are respected: when an obstacle is encountered, vision is blocked beyond that point, though light trails do not cause occlusion (as they are made of light).

Every frame, the system builds a 3-channel observation tensor representing the game map, including elements like detecting a player, borders, and trail segments. The agent's cone of vision (based on raycasting within a field of view) extracts the visible portions of this tensor, which is then padded to a fixed shape of (38, 3) for compatibility with neural networks.

Layer	Description	Purpose
0	Map borders	Detect and avoid collisions
1	Player positions	Locate agents on the map
2	Light trails	Detect trail segments to avoid or exploit

TABLE I
Observation tensor structure filtered by cone of vision before padding.

A. System Design Decisions

The transition to a vision-cone-based approach was motivated by several factors:

- **Realism:** Agents must explore and react with limited perception.
- **Strategic Depth:** Occlusion-aware mechanics promote reasoning.
- **Performance:** Reduced visible tiles improve efficiency.

The environment is built using the PettingZoo Parallel API. For agent training, RLlib is used as the reinforcement learning framework. The selected algorithm is Proximal Policy Optimization (PPO), configured with two shared policies and centralized observation inputs to approximate a Multi-Agent PPO (MAPPO) setup. Agents rely on convolutional neural networks (CNNs) to process the padded observation tensor and generate action outputs.

B. Reward Functions

The reward system has been redesigned to promote aggressive, team-oriented, and strategic behavior among agents. It combines incentives for eliminating opponents with penalties for self-destructive actions. The current reward structure is as follows:

- **Elimination Reward (Opponent):** An agent receives **+100** for eliminating an opponent team member.
- **Penalty for Dying:** Agents receive **-20** if they crash into any obstacle (wall, trail, etc.).
- **Step Penalty:** Each timestep incurs a small reward of **+0.1** to avoid suicide.
- **Victory Reward:** The winning team receives **+200**.

This structure encourages offensive coordination and punishes inefficient or reckless behavior.

This structure follows the principles of Kill and Goal Reward decomposition, where high-impact events (e.g., defeating an opponent) are rewarded significantly more than incremental survival. This approach has been shown to accelerate the learning of offensive strategies in multi-agent environments, particularly when rewards are split across different behavioral objectives such as “kill”, “goal achieved”, and “survived” [3].

C. Framework Selection: TensorFlow vs. PyTorch

As an alternative implementation, TensorFlow was considered because of its native integration with TensorBoard, a tool that allows real-time visualization of training metrics, reward evolution, entropy, loss functions, and configuration comparisons. This visualization is particularly useful in dynamic environments such as this one.

PyTorch was ultimately chosen for this project due to its compatibility with MARLlib and RLlib, ease of debugging, and flexibility for experimentation. Its dynamic computation graph and imperative programming style make it a preferred framework in reinforcement learning research and rapid prototyping scenarios [4]. In contrast, TensorFlow remains a robust alternative, particularly in production environments where deployment tools and detailed training visualization (e.g., through TensorBoard) are crucial [5].

III. Results

The agents successfully learn to avoid common threats such as other players, walls, and light trails. This behavior can be observed during testing with the trained model, where agents demonstrate strategic movement to survive and navigate the environment efficiently.

At some point, exploration tends to stop, leading to repetitive and deterministic behavior. However, this issue can be addressed by adjusting the epsilon-related parameters in the algorithm to encourage continued exploration and prevent early overfitting.

As it can be seen on the graphic, the rewards for both teams vary on each iteration reaching to a equilibrium. This is due to the lack of exploration, but as well as the learning from the enemy adapting to it.

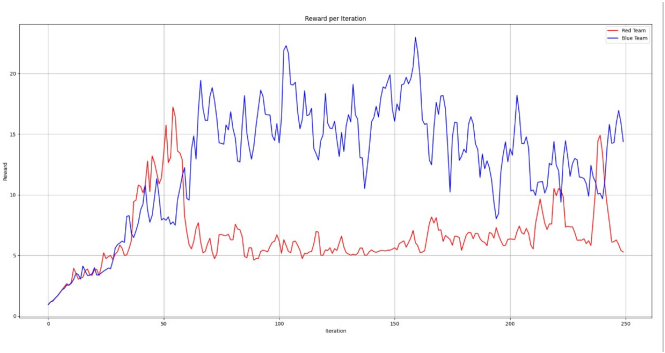

Fig. 2. Comparison among red and blue team rewards.

IV. Conclusions

This work establishes a foundation for a competitive multi-agent reinforcement learning environment inspired by the Tron game. With a custom observation system, a refined reward function, and dynamic interaction mechanics, the system fosters aggressive and strategic behavior.

Though training is pending, the environment’s modular design and PyTorch-based setup enable easy experimentation. Future work includes implementing training loops, benchmarking, and analyzing agent strategies.

References

[1] A. Sloane, “Google ai postmortem,” Available at: https://www.a1k0n.net/2010/03/04/google-ai-postmortem.html, 2010, a1k0n.net, Mar. 4, 2010.

[2] T. Vogt, “Tron_evolution,” Available at: https://github.com/Its-Triggy/TRON$_evolution, 2025, gitHubrepository, Accessed : May15, 2025.

[3] S. Zhang, H. Zhao, and Y. Bachrach, “Multi-dimensional distributional dqn,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 869–881.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1912.01703

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," *arXiv preprint arXiv:1605.08695*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1605.08695