Universidad Distrital Francisco José de Caldas

Engineering Faculty

System Sciences Foundations Final Project Report: Multi-Agent RL in a Mixed Tron Light-cycle Environment

Erick Buitrago - 20221020072

Juan Jiménez - 20221020087

Final report for the final project of the
System Sciences Foundations Course

July 11, 2025

Abstract

This is a report on the final project for the Systems Sciences Foundations Course. This report contains the process and analysis taken to a problem in which artificial intelligence agents participate. In this case, the scenario studied is a competitive game called Tron, in which the agents compete against each other in a controlled environment. The way to approach the problem will be by identifying and applying Systemic Analysis, as it is the most complete way to understand how different approaches to solutions may change the behavior of the system. Applying cybernetics concepts to it is also relevant, as it allows a better learning process for agents allowing them to take information from the environment they are part of and how they influence it.
As this is just a draft during the development of the final project, no results are yet provided, it is still speculated that the outcomes for the project will have the agents identifying different strategies to win by making their opponents crash into the different obstacles created during the game. Even though AI can present unexpected results from time to time, further investigation will be performed to get closer to the expected behavior.

Keywords: System Sciences, Multi-Agent Reinforcement Learning, Tron Game, Mixed Cooperative-Competitive Setting

Contents

List of Figures

List of Tables

List of Abbreviations

MARL	Multi-Agent Reinforcement Learning
PPO	Proximal Policy Optimization
MCTS	Monte Carlo Tree Search

Chapter 1

Introduction

- An artificial intelligence multi-agent system is studied within a competitive environment in which reinforcement learning takes a huge part.
- The scope of the project involves the systemic analysis including cybernetics (key concepts for AI development) in a system in which two AI agents are challenged to compete in a Tron-like game, this means agents will have to create strategies to win by making their opponents collapse with borders or light trails (barriers) left by them.
- The objectives of the project are to study the presented system and to develop a solution to train the AI agents to achieve the goal.
- The methodological approach is in first instance to identify system requirements, components and identifying how it essentially works as a system. Then tools to its development will be searched and identify how can it be represented as a mathematical problem to implement it while programming. Rewards and algorithms will be important aspects to take into account as they will define the behavior of the agents. Finally, testing and analyzing results will be essential for future updates.

1.1 Background

This case study was selected due to the highly dynamic and adversarial nature of the environment, where each agent's actions not only affect the shared environment but also influence the actions of each agent. Such interaction offers a concrete opportunity to explore fundamental systems science principles like interdependence, feedback, and non-linearity in real-time decision-making contexts.

Using reinforcement learning, agents aim to maximize cumulative rewards in a non-stationary setting where optimal strategies constantly evolve. This competitive scenario highlights how agents adapt policies, respond to uncertainty, and develop reward-driven behaviors aligned with their final objectives, providing insight into intelligent behavior in complex, multi-agent systems.

1.2 Problem statement

Developing intelligent agents that can operate effectively in fast-paced, competitive environments is a key challenge in the field of multi-agent reinforcement learning (MARL). The Tron light cycle game provides a compelling testbed where agents must continuously adapt their behavior in response to an opponent, while also navigating a dynamic and constantly changing environment.

Conventional methods often struggle to address the intricacies of real-time decision-making and the evolving nature of opponent strategies. This project tackles the problem of creating agents capable of processing complex environmental cues, formulating adaptive strategies, and making rapid decisions—all while contending with an adversary that is also learning and adapting, within a non-stationary and adversarial setting.

1.3 Objectives

General Objective: To design and implement intelligent agents capable of competing in dynamic and adversarial environments using multi-agent reinforcement learning techniques within the context of the Tron light cycle game. **Specific Objectives:**

- To model the Tron game environment as a multi-agent reinforcement learning (MARL) problem by formally defining states, actions, rewards, and game rules.
- To develop autonomous agents using reinforcement learning algorithms such as IQ-Learning, Independent Deep Q-Networks (DQN), or IPPO that can adapt to the opponent's changing behavior.
- To evaluate the performance of the agents in simulated scenarios using metrics such as win rate, survival time, and movement efficiency.

1.4 Solution approach

To address the problem of training two intelligent agents to compete in a dynamic Tron-like environment, a set of tools, libraries, and reinforcement learning algorithms has been selected to support future development and experimentation.

The environment will be developed using PyGame, which allows for precise control over the visual interface, game mechanics, and real-time interactions between agents and the map. For multi-agent simulation, the PettingZoo library will be used, as it provides a standardized API for multi-agent environments. This will be extended using SuperSuit, which offers wrappers for preprocessing observations, stacking frames, and normalizing inputs, making it easier to manage training data and state representations.

To implement and train reinforcement learning agents, the project will rely on MARLlib, a multi-agent reinforcement learning library built on PyTorch, which supports various algorithms and facilitates experimentation with competitive setups. Although TensorFlow and TF-Agents are considered viable alternatives, PyTorch has been chosen for its flexibility and compatibility with MARLlib.

In terms of algorithms, the planned progression includes:

IQ-Learning, as a starting point for understanding Q-value estimation and basic policy imitation in a simplified setting.

Iterative Deep Q-Networks (IDQN), to integrate deep learning and enable agents to approximate Q-values from high-dimensional input spaces like image-based observations.

Independent Proximal Policy Optimization (IPPO), to optimize each agent's policy independently, allowing for stable learning in competitive scenarios.

Self-Play as a training technique, where agents learn by competing against past versions of themselves or each other, enabling continuous adaptation and strategic depth.

The agents will rely on convolutional neural networks (CNNs) to process spatial information from the observation tensor, which is designed to capture features such as positions, directions, light trails, and environmental boundaries.

Together, these tools and methods form a robust foundation for implementing and testing reinforcement learning strategies in a multi-agent competitive game, supporting future experimentation and refinement as the code is developed.

Chapter 2

Literature Review

The development of intelligent agents for competitive environments, such as the Tron light cycle game, has been explored through various approaches, ranging from heuristic-based methods to advanced reinforcement learning techniques. This section reviews key contributions that inform the current project. The different works presented in this chapter were split into different subsections, by taking into account similarities in their solutions.

2.1 Voronoi Graph/Heuristics

2.1.1 Andy Sloane – Google AI Challenge 2010

In the 2010 Google AI Challenge, Andy Sloane (a1k0n) developed a winning bot for the Tron game. In his post-mortem, he describes the process taken to achieve the victory for his bot in the most games played.

The methodology used consisted of partitioning the game grid into regions, based on proximity to each player with the "Voronoi Heuristic" method. This allowed the bot to estimate control over space. Then, identifying articulation points int the grid, causing the number of disconnected regions increase, avoiding traps. Finally, he used a minimax search to evaluate future game states efficiently.

This approach demonstrated the effectiveness of combining spatial analysis with strategic search in real-time decision making scenarios. This approach to the solutions develops very well with some initial conditions, e.g., the size of the map. There are still other works that compare their performance with this bot. [1].

2.1.2 Kang – Endgame Detection with Heuristics in MCTS

The paper by Kang, investigates if an agent based in MCTS develops better if evaluation funcions are added between the simulation phase as a play-out.

The model introduces two specific heuristics, one based on Voronoi Diagrams and the other is the "Tree of Chambers" used inside de MCTS. It also include upgrades like the MCTS-Solver, which helps to detect the ending positions of the game.

The results he arrives, are that the incorporation of evaluation functions of the MCTS can upgrade its efficiency by identifying the ending positions and make better decisions on an end-game. [2]

2.2 Deep Q-Learning / Neural Networks

2.2.1 Thomas Jacquemin – Deep q-learning for tron

There is another work by Thomas Jacquemin (To-jak) present on his github repository. It shows a starting approach to solve the problem by using Deep Q-Learning.

The tools used in this work are PyTorch to use the neural networks it provides. Here, rewards are adjusted to compensate survivability and winning the game, while losing will grant them a penalty.

It is a pretty simple representation of the game, having the minimum amount of players needed, as most of the approaches, and having a controlled map with a small size. It still demonstrates the potential of deep RL in learning from raw state representations [3].

2.3 Modular Reinforcement Learning (MRL)

2.3.1 Jeon et al. – Modular reinforcement learning for playing the game of tron

Another work, is a paper for the IEEE done by Jeon et al. This paper shows a different approach to the problem which implies the use of Modular Reinforcement Learning (MRL).

The methodology used is to decompose the Tron game into stationary and non-stationary phases. Then, separate models are trained for each phase. The first phase is always non-stationary, as it implies that the opponent can have control over the area an agent can "possess". Once they are divided into two separate grids, the other algorithm is used to fill the available area.

The conclusion they arrive is that the RL approach is very viable as it can suit well with problems that can be divided. It is still a different solution to the one presented in the report, as it opposes the idea of the contest and turns the problem into a "single-agent" problem once the area is divided completely. [4].

2.4 Opponent Modeling + Q-Learning

2.4.1 Knegt et al. - Opponent modeling in the game of tron using reinforcement learning

The work by Knegt et al. published on ICAART proposes a reinforcement learning agent that improves its performance by learning to model its opponent's behavior.

Their method uses Q-learning with a multilayer perceptron and introduces small "vision grids" to simplify the input space. They also train the network to predict the opponent's next move and use Monte Carlo rollouts to improve decision-making.

They conclude that modeling the opponent greatly enhances performance, especially when combined with simplified input representations. The approach proves effective against both random and semi-deterministic opponents, achieving near-perfect win rates.[5].

2.4.2 David Churchill - Tron Competition

In this video, from the professor David Churchill, a contest between different AI agents, each with a different algorithm, are tested against each other. The goal is to show the strategies and efficiency in real time.

Contestants use RL techniques, specially driven by Q-Learning and MCTS. Neural networks are used as well to analyze the board in an efficient way and predict behavior of the player and the opponent.

The results from the contest show that the solutions that combine the model of the opponent and the planification such as MCTS and foe prediction, tend to dominate simpler methods. This reinforces the idea that integrating various techniques in a competitive environment shows potential. [6].

2.5 Monte Carlo Tree Search and UCT

2.5.1 Samothrakis et al. - A uct agent for tron: Initial investigations

This paper explores the use of Upper Confidence bounds applied to Trees (UCT), a variant of MCTS, as an agent for the game Tron. Here, Samothrakis et al. aim to assess whether UCT is a viable and competitive strategy for real-time adversarial environments like Tron, which involve spatial reasoning and collision avoidance.

The UCT agent is evaluated against a series of baseline agents, including random and greedy strategies. The game is treated as a two-player zero-sum game, and the agent performs forward simulations within a time budget per move, using a simple roll-out policy. To fit the real-time constraints of Tron, the algorithm limits the number of play-outs and explores the impact of varying the exploration constant.

Their results show that the UCT agent significantly outperforms baseline strategies, especially in open spaces where deeper simulations help avoid traps. However, in tightly constrained areas or near the endgame, the simplistic roll-out policy weakens performance. The study concludes that UCT is promising for Tron, but further enhancements, such as, better heuristics or adaptive roll-outs, are needed for more consistent dominance. [7]

2.5.2 Perick et al. - Comparison of different selection strategies in MCTS for the game of tron

This paper investigates how different selection policies within MCTS perform in the real-time, simultaneous-move context of Tron, where agents must decide under strict time constraints (100 ms per move). The core question is whether classic bandit-based strategies, like UCB1, remain effective in such dynamic scenarios.

The authors implement and compare twelve policies, six deterministic and six stochastic within MCTS tailored to simultaneous move settings. They adapt the UCT framework to account for joint player moves and measure performance across numerous simulated games on a 20×20 grid.

Results clearly show that deterministic policies, particularly UCB1-Tuned, outperform stochastic alternatives in Tron's real-time environment, with UCB1-Tuned ranking highest and UCB-Minimal also performing notably well. The study highlights that deterministic bandit strategies are better suited for rapid, simultaneous decision-making in complex adversarial games like Tron. [8]

2.6 Genetic Algorithms/Neuroevolution

2.6.1 Tristan Fogt - Evolving Neural Networks for Tron

This other video by Tristan Fogt (Its-Triggy), show how can a neural network evolves to play this classic game. It show the capability of this method to learn new survival strategies without direct supervision.

The way it is developed, is by using genetic algorithms to adjust weight on a neural network which receives the map configuration as an entry. This is repeated in multiple generations, showing how AI upgrades its performance avoiding collisions and anticipating movements.

The evolutionary approach shows that it can be a promising way to train agents for discrete games. Even if it shows a poorer job compared to others, it is still a good way to show different approaches for the same problem. [9]

Chapter 3

Methodology

This section details the methodology used to model the system and how its simulation works. It also describes the training for intelligent agents in a competitive Tron-inspired environment. The explanation is divided into system specifications, cybernetic diagrams, observation model, reward functions, mathematical modeling, an explanation for the algorithm implemented.

3.1 Scope

This project focuses on modeling and implementing a multi-agent reinforcement learning environment inspired by the classic Tron light-cycle game. The scope of the work includes the development simulation system with light trail mechanics, integration of partial observability through cone-shaped vision, and the design of a reward structure that can recreate a Tron-like game.

The environment is fully compatible with modern reinforcement learning frameworks such as PettingZoo and Gymnasium, allowing interaction between multiple agents and the game board. It also supports training algorithms including Proximal Policy Optimization (PPO) through RLlib, with the goal of achieving competitive and convergent agent behavior.

While the environment is optimized for autonomous agent competition, the current scope does not include a human interaction, i.e., only agents can play between themselves.

3.2 System Specifications

Before entering into the agent solution, the system must be described first.

3.2.1 System Description

This system shows a discrete time, in which players interact in a mixed way, this means the players must cooperate in case they are from the same team, and they are competing against the other team.

The game environment is defined on a two-dimensional grid of size $M \times N$, where $M = 35$ and $N = 21$ represent the width and height of the full map, respectively. The set of valid positions (or playable cells) is denoted by $\mathcal{P} \subseteq \mathbb{Z}^2$, excluding the outermost border. Thus, the effective playable area is 33×19 cells.

The system is pretty dynamic, as it involves 4 players playing at the exact same time and as they move they modify the map placing obstacles. The game faces a stability and it is when everything is returned to its initial conditions, this is because the whole game is stochastic, meaning there is much chaos during its execution. Another stability is having both teams an approximated equally amount of wins or ending the process in draw.

3.2.2 Reinforcement Learning Requirements

Sensors: The sensor is a vision-based model, which is able to gather information such as obstacles, opponent's positions as well as light trails and the time they take to disappear. This information will be used for processing and decision making. (A further explanation is given in the "Cybernetic Feedback Integration" section)

Actuators: The actuators correspond to the input keys used to control the agent's movement and to activate or deactivate the light trail, i.e., to leave the barrier behind or not.

- **Agent Perception:** Each agent must perceive relevant environmental information, this information includes map limits, positions of a seen agent, and the presence of light trails, to make informed decisions.
- **Agent Actions:** Agents should be able to move in four discrete directions (up, down, left, right).
- **Collision Detection:** The system must detect collisions between agents and walls or light trails, determining game outcomes such as crashes.
- **Reward System:** Implement a reward mechanism that incentivizes survival, aggressive play, and penalizes collisions.
- **Real-Time Updates:** The environment should update at a fixed frame rate (e.g., 60 FPS), ensuring timely state changes and responsive agent actions.
- **Training Environment Support:** Provide interfaces for reinforcement learning frameworks to interact with the environment, enabling observation retrieval and action submission.

To evaluate the performance of the agents on different contexts there are provided 4 maps, which will be chosen randomly. Those maps are provided in the Figure 3.1.

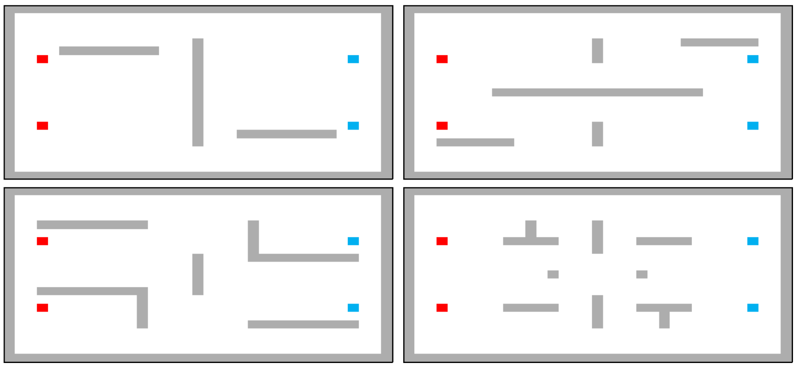

Figure 3.1: Four different maps.

3.3 Feedback Loops

The system dynamics are governed by feedback loops that regulate agent behavior and environment interaction in the Tron-inspired game. Each agent operates within a closed-loop control system, where sensors, decision-making modules, and actuators continuously interact.

Sensors provide each agent with a visualization of the map, the observation of the agents is further explained. The information gathered is encoded in the observation tensor and forms the basis for perception.

The decision-making subsystem processes sensor inputs and the agent's experience memory to select the next action. This subsystem incorporates learning mechanisms that adjust policies based on received rewards and observed outcomes.

Actuators execute the selected actions, which is the direction that is going to be taken movement command. These actions update the environment by moving the agent and modifying the map through barrier placement.

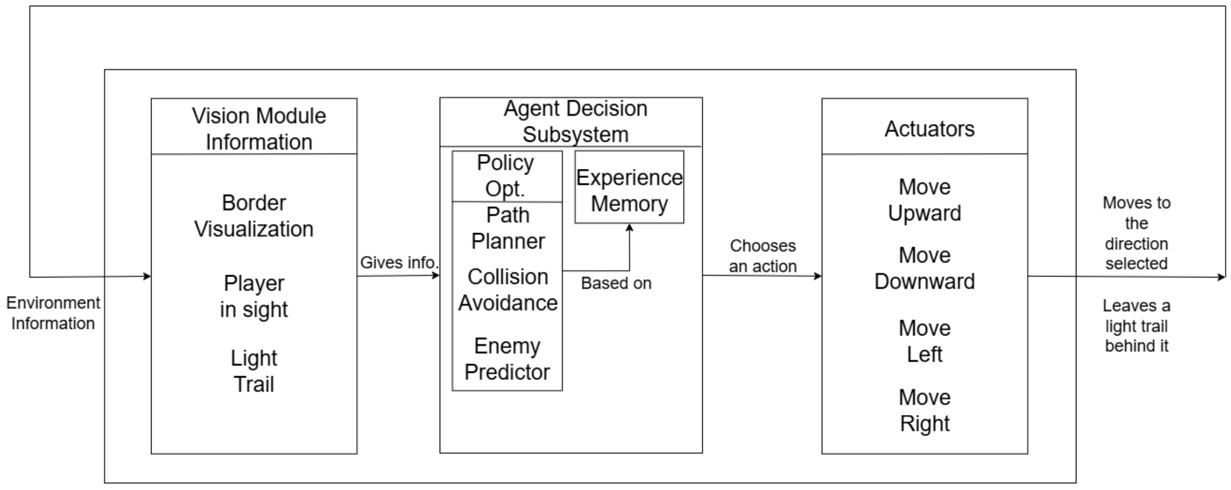

Figure 3.2: Cybernetics feedback loop diagram in the Tron environment.

As the agents act, the environment changes, closing the loop by providing updated sensory inputs. This continuous cycle of sensing, acting, and learning forms the feedback loop essential for agent adaptation, strategic planning, and the emergence of complex behavior.

3.4 Observational Model

In the redesigned environment, each agent no longer receives a full tensor representation of the entire game grid. Instead, the observation model is now based on a dynamic and directional field of vision, mimicking a realistic perception system similar to a cone-shaped flashlight.

At each frame, the environment generates a 3-channel observation tensor representing the entire grid. These channels include borders, player positions and light trails:

- Channel 0: Grid borders.
- Channel 1: Positions of players 1 through 4.
- Channel 2: Light trails.

Each agent has a vision cone defined as a triangular field projecting from its current position and orientation. This cone is parameterized by a field-of-view angle, number of rays, and a maximum distance. The cone simulates partial visibility with occlusion: rays are interrupted when they encounter walls or other players, preventing the agent from seeing beyond them. Light trails, however, are treated as transparent and do not block vision.

To handle this, the environment first generates a simplified obstacle matrix derived from the full observation tensor, encoding walls, trails, and players. Then, a ray-casting algorithm is used to produce a set of visible coordinates for each agent based on its direction and obstacles in its path. These visible coordinates are used to extract the corresponding vectors from the observation tensor, forming a list of visible cell observations.

Since the number of visible cells varies depending on the agent's orientation and environment layout, each agent's observation is padded with zeros to form a fixed-size matrix of shape $(38, 3)$, ensuring compatibility with neural network input requirements.

This cone-based observational model enables more realistic and decentralized perception, supports learning under partial observability, and allows agents to generalize better across different map layouts and scenarios.

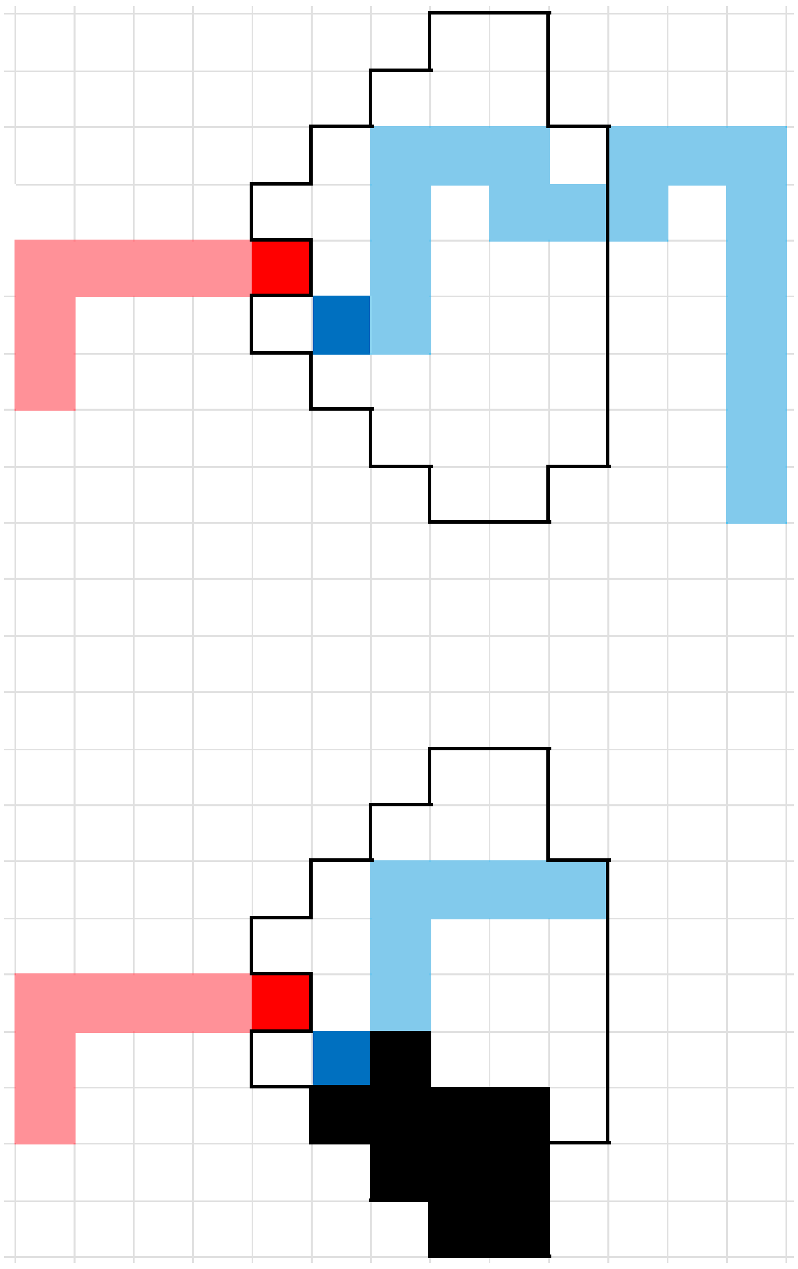

Figure 3.3: Figure 3.2: Illustration of the agent's triangular cone of vision. Vision is limited and blocked by obstacles.

3.5 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 or inefficient actions. The current reward structure is as follows:

- **Elimination Reward (Opponent):** An agent receives a significant positive reward of $+100$ for successfully eliminating a member of the opposing team using their own light trail.
- **Penalty for Dying:** Any agent that crashes—whether into a wall, their own trail, or an enemy's trail—receives a penalty of -20, encouraging caution and spatial awareness.
- **Step Reward:** For every step taken, alive agents will get a positive reward of $+0.1$. It was changed back to a reward rather than a penalty as agents would commit suicide to avoid losing its reward, which avoided them exploring.
- **Victory Reward:** At the end of the game, the team on the winning receives a $+200$ reward, representing a major strategic achievement.

This reward structure aligns with the competitive objectives of the game, incentivizing agents to act both offensively and cooperatively while penalizing inefficient or destructive behavior. Future extensions may include rewards for trapping opponents (i.e., limiting their available movement space) or penalties for erratic maneuvers such as unnecessary sharp turns.

3.6 Mathematical Model

At each discrete time step $t \in \mathbb{N}$, the system state is represented as:

$$\mathcal{S}_t = \left\{P_t^i, d_t^i, L_t^i, B\right\}$$

Where:

- $P_t^i = [x_t^i, y_t^i] \in \mathcal{P}$ is the position of agent i at time t.
- $d_t^i = [d_{x,t}^i, d_{y,t}^i] \in \{-1, 0, 1\}^2$ is the direction vector of agent i at time t.
- $L_t^i = \{(x_k, y_k, \tau_k)\}_{k=1}^{n_i}$ is the set of light trail positions left by agent i, each with remaining lifetime τ_k (measured in seconds).
- B represents the static borders of the map.

The evolution of the system is governed by the transition function:

$$\mathcal{S}_{t+1} = T(\mathcal{S}_t, A_t^i)$$

Where $A_t^i = (m_t^i)$ denotes the action of agent i, consisting of: $m_t^i \in \{[0, -1], [0, 1], [-1, 0], [1, 0]\}$ for movement (Up, Down, Left, Right).

The updated position is calculated as:

$$P_{t+1}^i = P_t^i + m_t^i$$

and the updated direction is:

$$d^i_{t+1} = m^i_t$$

The agent leaves a new trail segment at P^i_t with an initial duration $\tau = 10$ seconds. Trails decay over time and are removed once their lifetime is over.

Collisions with walls, other agents, or light trails result in terminal states for the affected agents. Each agent's reward r^i_t is computed based on the following:

- $+200$ to all agents that belong to the winner team.
- $+100$ if agent i eliminates an opponent using its trail.
- $+0.1$ per step survived.
- -20 for dying.

The full system evolves according to both internal dynamics (agent movement and trail decay) and external inputs (agent actions). This formulation sets the foundation for modeling, training, and analyzing agent behavior using reinforcement learning techniques

3.7 Tools and Frameworks Used

The system architecture integrates several specialized tools to support environment simulation, agent training, and performance analysis:

- **PyGame (2.5.2):** Provides the graphical interface and game mechanics simulation, enabling visualization of the game board, agent movement, and light trails.
- **PettingZoo (1.23.1):** A multi-agent reinforcement learning environment library facilitating agent-environment interactions under a unified API.
- **Ray[RLlib](2.9.3):** Provides the implementation for the PPO algorithm.
- **PyTorch(2.7.1):** The backend deep learning framework used for building neural networks and training reinforcement learning agents.
- **NumPy(1.26.4):** Used for optimal structures.
- **Pandas(2.2.2):** Used for data manipulation.
- **Matplotlib(3.8.4):** Used for visualization of training metrics, i.e., rewards.

3.8 Comparison of Reinforcement Learning Algorithms

Several reinforcement learning algorithms are planned for implementation and evaluation, each offering different advantages for the multi-agent Tron scenario:

Algorithm	**Description**	**Strengths and Use Cases**
IQ-Learning	A simple approach that learns Q-values by mimicking expert behavior. Often used for baseline testing or educational purposes.	Useful for initial experimentation and understanding basic Q-learning mechanics.
Independent Deep Q-Network (IDQN)	Enhances Q-learning with deep neural networks to handle high-dimensional states. Each agent learns independently.	Works well in competitive or partially observable settings; decouples agent learning.
Proximal Policy Optimization (PPO)	A stable policy-gradient algorithm that maximizes a clipped objective to limit destructive updates [10].	Serves as the foundation for both IPPO and MAPPO implementations. Widely supported in frameworks like RLlib.
Independent Proximal Policy Optimization (IPPO)	A decentralized version of PPO where each agent trains its own policy independently [11].	Directly supported in RLlib; well-suited for competitive environments with independent learners.
Multi-Agent Proximal Policy Optimization (MAPPO)	An extension of PPO using centralized critics and shared policies across agents [12].	Although not natively available in RLlib, it can be implemented via shared-policy PPO setups and centralized observation inputs.
Self-Play	A training technique where agents compete against past versions of themselves or others.	Enables continual learning and dynamic adaptation to evolving opponents.

Table 3.1: Comparison of reinforcement learning algorithms relevant to the Tron multi-agent environment.

3.9 Limitations

Despite achieving a functional and well-structured environment for training competitive agents, the development process encountered several limitations, particularly related to library compatibility and tool integration.

Initially, the project intended to use MARLlib as the primary training framework. However, this library is tightly coupled with older versions of Gym (specifically gym==0.21.0), which presented significant compatibility issues when attempting to run on a modern Linux-based virtual machine. These conflicts affected both the installation of dependencies and the correct execution of environments with customized observation spaces.

As a result, the MARLlib-based pipeline had to be discontinued, and the project pivoted to using RLlib,a more flexible and well-maintained alternative. By adapting the custom PettingZoo Paralallel API environment to RLlib's specifications, including the integration of the observation tensor and action spaces, it was possible to successfully initialize training runs using the IPPO algorithm.

Although this adaptation proved effective, it required additional effort to understand RLlib's multi-agent configuration model and to ensure that the custom environment was correctly wrapped and registered. These technical adjustments delayed experimentation and limited early testing opportunities.

Furthermore, no graphical user interface was developed for human interaction with the game, as the focus remained on autonomous agent training. Likewise, the project did not explore imitation learning, meta-learning, or other higher-level reinforcement learning strategies, which are reserved for future work.

These limitations, while presenting technical and temporal constraints, did not hinder the core objectives of the project. On the contrary, they prompted adaptive solutions that enriched the understanding of multi-agent system integration and policy training pipelines. The insights gained during these challenges laid the groundwork for the experimental phase, where agent performance, learning dynamics, and environment behavior can now be rigorously tested and analyzed.

Chapter 4

Results

The agents 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.

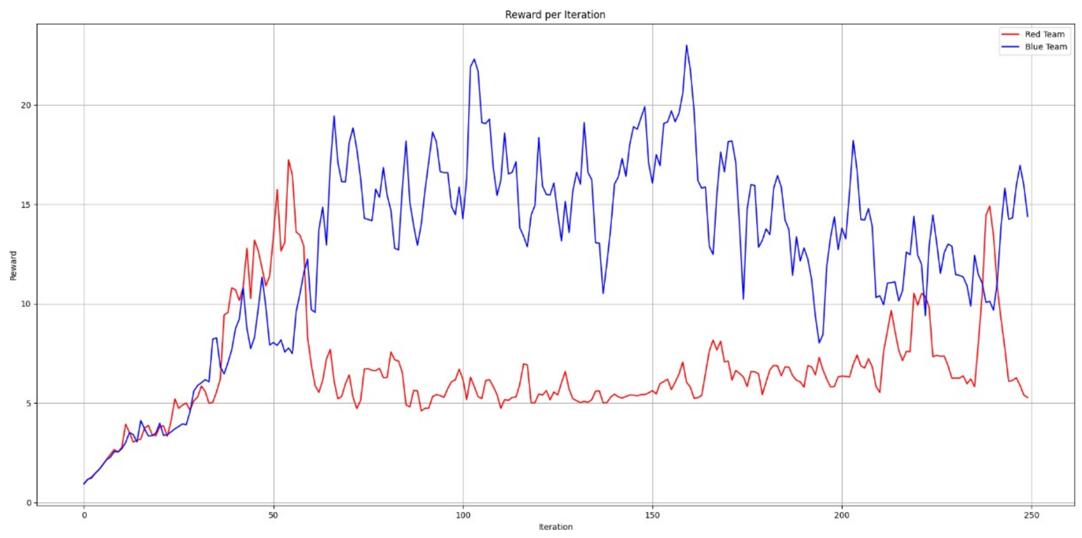

Figure 4.1: Comparison among red and blue team rewards.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This project focused on developing a reinforcement learning framework for the Tron game using the Proximal Policy Optimization (PPO) algorithm in a multi-agent competitive environment. Specifically, we simulated a MAPPO-style setup by training two distinct policies—one for each team—enabling agents to cooperate within teams while competing against the opposing side.

The results demonstrate that PPO can effectively handle the coordination required in team-based settings, particularly when agents share rewards and observations. However, challenges such as overfitting to static map layouts and the emergence of deterministic strategies highlighted the need for sustained exploration and greater environmental diversity.

In conclusion, the solution approach to Tron provides promising insights into cooperative-competitive reinforcement learning. Future work could involve scaling to more agents, introducing dynamic maps, and improving opponent modeling to foster more generalizable strategies.

5.2 Future work

Future work will focus on testing different variations:

- The light trail switch was disposed as it didn't cause many differences, perhaps trying it on agents that are partially trained might change the behavior.
- Adding an unseen map to see how they develop on other conditions.
- Implementing and comparing reinforcement learning algorithms such as IQ-Learning, IDQN, and IPPO with Self-Play.

This tests might give a better idea on how they might be able to face some stochastic behavior to the system rather than agents behaviors.

References

[1] A. Sloane, "Google ai challenge post-mortem," https://www.a1k0n.net/2010/03/04/google-ai-postmortem.html, 2010, accessed: 2025-05-15.

[2] L. Kang, "Endgame detection in tron," B.Sc. thesis, Maastricht University, June 2012, investigates MCTS enhancements using evaluation heuristics.

[3] T. Jacquemin, "Deep q-learning for tron," https://github.com/To-jak/Deep-Q-Learning_TRON, 2019, accessed: 2025-05-15.

[4] M. Jeon, J. Lee, and S.-K. Ko, "Modular reinforcement learning for playing the game of tron," *IEEE Access*, vol. 10, pp. 63 394–63 402, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3175299

[5] S. J. L. Knegt, M. M. Drugan, and M. A. Wiering, "Opponent modelling in the game of tron using reinforcement learning," in *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART)*, 2018, pp. 29–40. [Online]. Available: https://doi.org/10.5220/0006536300290040

[6] D. Churchill, "Comp4303 - video game ai - lecture 19 - tron ai tournament," https://www.youtube.com/watch?v=dsWB9AM0bEY, 2023, accessed: 2025-05-15.

[7] S. Samothrakis, D. Robles, and S. M. Lucas, "A uct agent for tron: Initial investigations," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 365–371.

[8] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "Comparison of different selection strategies in monte-carlo tree search for the game of tron," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 242–249.

[9] I. Triggy, "Ai learns to play tron," https://www.youtube.com/watch?v=uZ9CcdBHp-I, 2018, video, accessed July 2025.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1707.06347

[11] D. Li, Y. Wen, W. Yang, C. Zhang, and J. Wang, "Marllib: A unified library for multi-agent reinforcement learning," GitHub Repository: https://github.com/Replicable-MARL/MARLlib, 2022, configuration for IPPO available via PPO with separate policies.

[12] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The the surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2103.01955