Workshop No.1

Buitrago, Erick
Student ID: 20221020072

Jiménez, Juan
Student ID: 20221020087

Universidad Distrital Francisco José de Caldas

April 2025

Contents

Abstract

This work explores the final project of the Systems Sciences Foundations course as a system showing the respective System Analysis to it as well as the process that will be useful for it to be successful. The document contains specific parts of the system as well as some concepts of RL and its algorithms that will be used during its development, starting by using IQ-learning until it reaches to IPPO with Self-Play.

Keywords: System Analysis, System Architecture, Reinforcement Learning, Neural Networks, Multi-Agent

The present document aims to present the System Analysis of the project Tron. As well it is presented the trajectory of the development of the project and shows the tools that will fit for its achievement.The main objective of this project is to successfully train two AI agents and have them compete against each other in a controlled environment.

1. SYSTEM REQUIREMENTS

1.1 Functional Specifications

Sensors: The sensors will consist of a 3D matrix representing the state of the map, including the position of light trails, walls, free spaces and the position and direction of the rival player. This information will be used for processing and decision making.

Actuators: The actuators will correspond to the input keys used to control the agent's movement and to activate or deactivate the light trail.

Rewards: The reward system will use two variables: `reward_player1` and `reward_player2`. Each will be increased by 0.3 units for every second the agent survives. An additional 10 units will be granted if the rival agent crashes into a light trail or a wall. In contrast, 10 units will be subtracted if the agent crashes. This setup encourages survival, promotes competitive behavior, and penalizes collisions.

1.2 Use Cases

The agent and the environment will interact in such a way that both agents will obtain the map information for each frame game, those information will be awarded in a 3D matrix format, where each cell contains the position of the objects that can be interacted with, and each level or dimension will have a distinct object that has the following representation:

- **W**: Walls
- **P1**: Player 1
- **P2**: Player 2
- **TL**: Light trail
- **dir_p1_x**: X component of Player 1's direction
- **dir_p1_y**: Y component of Player 1's direction
- **dir_p2_x**: X component of Player 2's direction

- **dir_p2_y**: Y component of Player 2's direction

With this information, agents will be able to identify dangerous areas and safe paths. This structured representation will help them detect obstacles and make decisions to avoid collisions.

Once the matrix observation is processed, the agent selects any action to generate an output, which is a player direction, such as turn left, turn right or keep going forward, searching for the objective of survival and trying to survive and make the opponent crash.

These objective is incentivizes with the rewards, making a learning for each agent about those actions, consequences, and strategies in the environment.

2. HIGH-LEVEL ARCHITECTURE

2.1 Component Diagram

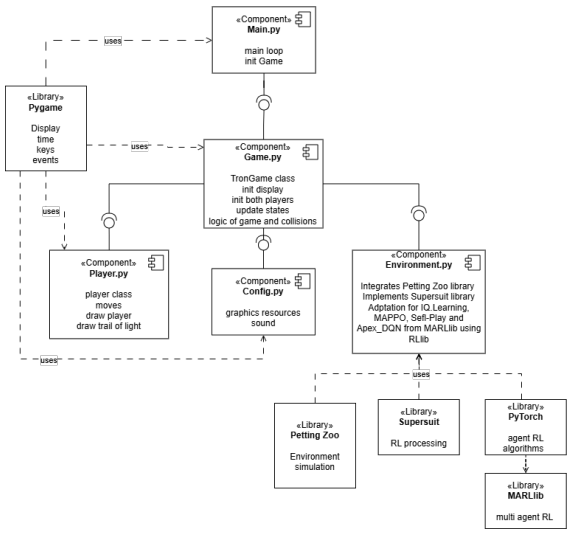

Figure 1: High-level component diagram of the system

2.2 Feedback Loops

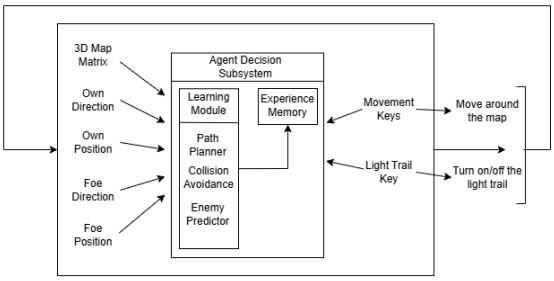

Figure 2: Feedback Loop

3. PRELIMINARY IMPLEMENTATION OUTLINE

3.1 Potential Frameworks and Libraries

3.1.1 Pygame

This framework will be useful for the visualization aspect of the game as well as the game development.

3.1.2 Petting Zoo

This library was chosen to simulate the environment to train the agents. We decided to use it instead of other ones like Gymnasium due to the multi-agent training present in the problem.

3.1.3 SuperSuit

This is will work as an "upgrade" to Petting Zoo as it allows the use of wrappers to increase the effectiveness of the environment of Petting Zoo.

3.1.4 NumPy

It will be useful to the data manipulation, as well with the develop of the Q-tables and states for the IQ-Learning Algorithm.

3.1.5 Matplotlib

This will fit for the visualization of metrics, e.g. rewards and agents politics.

3.1.6 MARLlib

This will allow the implementation of algorithms as IDQN and IPPO

3.1.7 PyTorch

As MARLlib uses PyTorch on its backend, this will be useful to personalize algorithms, in this case by adding the Self-Play technique.

3.2 Reinforcement Learning Algorithm's Timeline

The following are the Reinforcement Learning Algorithm's that will be applied at some point of the project development. This is due to their complexity and to identify how each one changes the behaviour of the agents with fastest learning or advanced tactics.

3.2.1 IQ-Learning

This algorithm was chosen because its a easy way to introduce into the RL, as it focuses directly on learning Q functions based on data and expert's politics.

As said by Garg et al.(2021) in their arXiv preprint: "we introduce a method for dynamics-aware imitation learning which avoids adversarial training by learning a single Q-function, implicitly representing both reward and policy".

3.2.2 IDQN

We plan to continue with IDQN, as this algorithm is a great approach to the neural networks, and it allows to use deep learning to scale the solution and adapt to a multiagent environment.

The algorithm IDQN, as Munguia, Tan and Ji (2021) stated in their arXiv preprint "IDQN performs better because our $\iota(s)$ function avoids impractical actions such as crashing against a wall or an enemy. This behavior may take more than 2,000 episodes for DQN to learn".

3.2.3 IPPO

Then, the IPPO algorithm fits perfectly with the objective of the project, as this algorithm allows each agent to optimize their politics independently, which is perfect for a multi-agent scenery with competition.

In their arXiv preprint paper, Schroeder et al. (2020) says about the algorithm that "despite its various theoretical shortcomings, Independent PPO (IPPO), a form of independent learning in

which each agent simply estimates its local value function, can perform just as well as or better than state-of-the-art joint learning approaches on the popular multi-agent benchmark suite SMAC with little hyperparameter tuning".

3.2.4 IPPO + SelfPlay

Finally, it is planned to complement the previous algorithm with Self-Play, which is not an algorithm but a complementary technique used to train the agents with previous versions of themselves or the other agents, simulating a dynamic environment.

This technique has proven to be very powerful, as Snyder (2017) stated "AlphaGo Zero started with just knowledge of the rules and learned from the success of a million random moves it made against itself [...] This demonstrates a decades-old idea called reinforcement learning".

4. REFERENCES

Garg, D., Chakraborty, S., Cundy, C., Song, J., Geist, M., & Ermon, S.(2021).*IQ-Learn: Inverse soft-Q Learning for Imitation*.arXiv.https://arxiv.org/abs/2106.12142

Munguia, F., Tan, A., & Ji, Z.(2023).*Deep Reinforcement Learning with Explicit Context Representation*.arXiv.https://arxiv.org/abs/2310.09924

Schroeder, C., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P., Sun, M., & Whiteson, Shimon.(2020).*Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?*.arXiv.https://arxiv.

Snyder, A.(2017, October 18).*New AlphaGo learns without help from humans*.Axios.https://www.axios.com/2017/ alphago-ai-learns-without-help-from-humans-1513306264