Workshop No.2

Buitrago, Erick
Student ID: 20221020072

Jiménez, Juan
Student ID: 20221020087

Universidad Distrital Francisco José de Caldas

May 2025

Contents

Introduction

This document focuses on the development of a system dynamics model for a two-agent game, inspired by Tron, focusing on agent behavior and interaction within a simulated environment. The initial model defines agent states, actions, environment dynamics, and agent observation. A basic reward system is implemented, providing small rewards for survival and significant rewards and penalties for winning or losing, respectively. Further refinement of the feedback loop is proposed to enhance agent intelligence, stability, and convergence.

Keywords: System Dynamics, Agent-Based Modeling, .

1. SYSTEM DYNAMICS ANALYSIS

1.1 Mathematical/Simulation Model

In this section we will find various equations that are based on how agents develop and interact in their environment, including their movement, observation of their environment, and reward depending on time. The equations are focused more on the parameters they have giving some time-dependent factors, rather than a specific function.

For some clarity to the time-dependent factors the unit they will be working on is seconds.

1.1.1 Agent's State

The state of an agent consists on its current position and the direction where it's going. It is defined as a vector, where x_t represents the agent's coordinates on the x axis, and y_t its coordinates on the axis on time t. The parameters d_{x_t} and d_{y_t} represent the two components of the agent's directions, the values can be seen better in the Agent's Actions subsection described in the actions taken.

To resume the current state and current direction in following equations it will be used P_t and d_t respectively.

$$P_t = [x_t, y_t]$$

$$d_t = [d_{x_t}, d_{y_t}]$$

The velocity of the agents will be the magnitude of it and its represented as v. The position of the agents is calculated as the following equation:

$$P_{t+1} = P_t + (d_t * v)$$

So the state of the agent would be seen as the next vector.

$$S_t = [P_t, d_t]$$

1.1.2 Agent's Actions

The actions the agent can perform are their movement through the space and switching their light trails on or off (creating barriers). In the following equations it is described how its direction changes based on the action taken.A_t is a variable that represents the action of the agent.

$$A_t = (m_t, l_t)$$

$m_t \in \{U, L, D, R\}$ is the movement action, which could be to move rather up, left, down, or right (U, L, D, R, respectively). Each action is seen just like the d_t vector.

$l_t \in \{0, 1\}$ is the light trail state, where 1 means the trail is ON and 0 means it is OFF.

The equation that represents the value of the next direction makes it equal as the action the agent takes (m_t). This is due to the fact that the action taken represents a vector. Each action is defined as the following vectors:

$$U = [0, -1]$$
$$L = [-1, 0]$$
$$D = [0, 1]$$
$$R = [1, 0]$$

The next direction would be:

$$d_{t+1} = m_t$$

The light trail state is updated as:

$$\ell_{i+1} = l_i$$

This allows the agent to independently decide both its movement direction and whether to leave a trail.

1.1.3 Environment Dynamism

This equations show how the environment changes. The changes the environment suffer, are due to the movement of the agents updating their position on the field and also leaving behind a light trail creating obstacles to avoid.

As the game starts, each agent position will be updated based on their direction (it is important to say that opposite directions to the current cannot be taken immediately, e.g., if the agent is moving to the right it won't be able to take left instantly), this is represented by the equations described in the agent's state on the position.

For the light trail there are two important concepts. The light trail is limited to a certain number, which will represent the maximum "barriers" the agent can used. Once it reaches this amount, the barriers will be disposed like a queue, i.e., the first barrier added will de-spawn to create the new barrier.

$$L_i = L_{i-1} \cup \{x_t, y_t\}$$

1.1.4 Agent Observation

The agent observes its environment through an observation matrix composed of 9 basic layers. The first layer contains information about the borders. The second layer represents the position of agent 1, and the third layer represents the position of agent 2. The fourth layer contains the light trail positions of agent 1, while the fifth layer contains the light trail positions of agent 2. The sixth and seventh layers represent the direction of agent 1 (one for the x-direction and one for the y-direction). Similarly, the eighth and ninth layers represent the direction of agent 2.

"In agent-based modeling (ABM), a system is modeled as a collection of autonomous decision-making entities called agents. Each agent individually assesses its situation and makes decisions on the basis of a set of rules"(Bonabeau, 2002, par. 2).

1.1.5 Reward

The rewards the agents have consist of a macro-reward, to complete the main goal, and other sub-goals to achieve it. The main objective of the agents would be to win the game by having an aggressive play style, so the reward they would get by eliminating the opponent with their own light trail would be much more significant, they will get $+100$ by doing it. They would get a lower reward if the opponent crashes on its own with a border or their own trail, they will get $+50$.

To achieve the main goal there are other sub-goals, e.g., surviving, it was taken before as the main objective but it would not achieve the wanted competition among the agents, so it will be a little reward $+0.1$ that the agent will get for each step taken without dying.

As one must compensate the good behavior on the agent, it must also be penalized the unwanted one, the agent will have -20 by dying with any obstacle present.

1.2 Phase Portrait

The game will start with agent 1 on the left side and agent 2 on the right side. The first move of both agents will be toward the other side of the board. Then, they will decide how to move. They can fall into walls or any light trail. The agents cannot turn back in one move, for example, if an agent is moving to the right, it cannot turn left with a single input. All of this can be seen on the Figure 1 below.

"The AI agent learns optimal decision-making by interacting with the game environment, analyzing states, and receiving rewards"(Mohammed, Geeth, Kumar, Sagar, Sankar & Karwa, 2024, par.1).

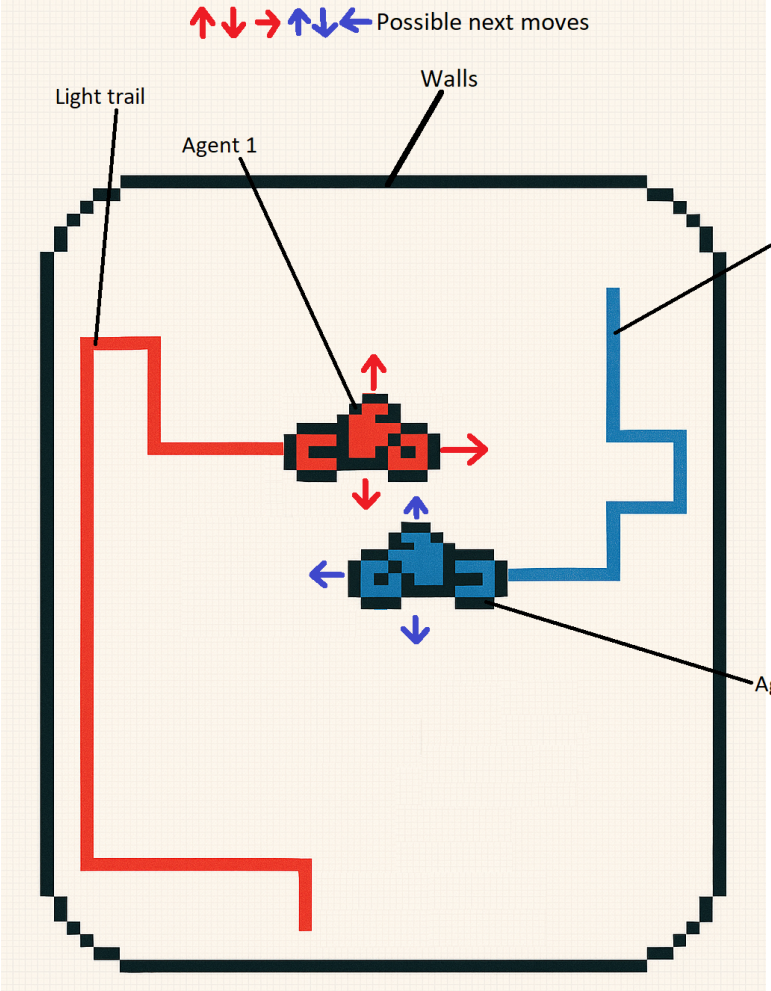

Figure 1: Example of Tron game board with agents and light trails

2. FEEDBACK LOOP REFINEMENT

2.1 Enhanced Control Mechanisms

To enable an intelligent, stable, and convergent behavior, we may add more forms of rewards to incentivize some behaviors and avoid others.

We can add a reward for attacking the enemy, to encourage aggressive behavior. The agent will be rewarded when the enemy has few free spaces to move, that is, when the agent is cornering the opponent. This reward will complement the reward for the enemy's death.

Also new reward may be to promote fluid movement, avoiding clumsy or cyclic behavior. It will compare the player's previous and current direction if it is the same, the agent will be rewarded with a light reward, but if the agent makes a 180° turn, it will be penalized lightly (this way, if

it is to avoid a wall, it will prefer to do a 180 degree turn rather than dying, as it has a bigger penalization).

Another newly introduced granular reward is a small positive reinforcement for survival. The agent will receive $+0.1$ for each step it takes without crashing. This encourages the agent to avoid unnecessary risks and maintain movement, while still keeping the ultimate goal of defeating the opponent as its primary goal.

2.2 Stability and Convergence

The system while running is a pretty dynamic system, as it modifies each time the agents move across the map and leave a trail (barriers) as they move. The stability of the system cannot be seen as the agents are "playing" but instead once the game is over having a winner team.

The stability of the system consists basically in the restart of the map going back to the initial conditions. This would take the agents back to their starting position as well as remove the barriers left in the past game.

Each agent would get a stable behavior when the game have ended or have just barely started, as its in this point that system does not have enough changes or chaos surrounding it.

2.3 Feedback Loops

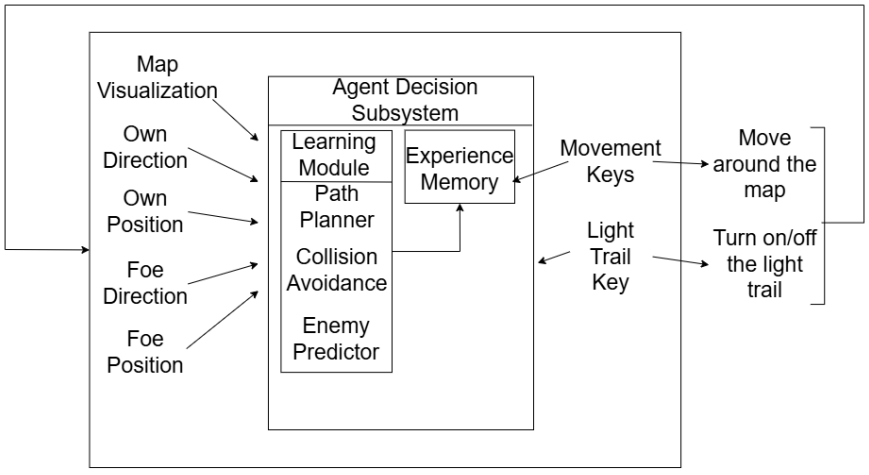

Figure 2: Cybernetics feedback loops

The feedback loops present on the system can be seen as its shown in figure 2. The sensors that each agent has are the visualization they have of the map (consisting on the map limits and the light trails left), and the own state and its foe's state as well. All of them present on the observability matrix each agent have. This will give information to let the agent make a decision.

The decision it makes can be seen as a sub system, with a learning module that also gathers information of its experience memory.

Finally, the actuators the agents have are the movement keys and the key that allows them to leave a light trail on the map. This way they can place barriers and move through the map which will update all of the information present on the observability matrix creating the loop for the agent.

3. ITERATIVE DESIGN OUTLINE

3.1 Potential Frameworks and Libraries

It was identified that there is a need to implement a more refined reward system in the project to improve the agent's behavior and decision-making, enhance overall performance, and accelerate the learning process. This system should not only penalize defeat and reward victory, but also encourage strategic and intelligent behavior through more granular feedback.

In addition, alternative neural network frameworks can be considered, such as TensorFlow. TensorFlow provides a complete environment for building and training learning models, including those used in reinforcement learning. It also supports TF-Agents, a library specifically designed for reinforcement learning within the TensorFlow ecosystem. This library includes modular components for constructing agents, policies, replay buffers, and environments, similar to what is offered by MARLlib or Stable-Baselines3.

One major advantage of using TensorFlow in this type of project is its seamless integration with TensorBoard. With TensorBoard, the training process can be visualized in real-time, allowing users to monitor reward evolution, entropy, policy loss, and compare different agent configurations. This visualization capability is key to supporting iterative design and analysis, as it enables developers to diagnose training issues and explore alternative models more effectively.

Although PyTorch was selected for this project due to its flexibility and compatibility with MARLlib, TensorFlow remains a powerful and viable alternative, especially in scenarios where native integration with TensorBoard is a high priority.

4. REFERENCES

Bonabeau, E. (2002, May 14).Agent-based modeling: Methods and techniques for simulating human systems.*Proceedings of the National Academy of Sciences, 99*.(Suppl. 3), 7280-7287. 1 Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC128598/

Mohammed, T. K., Geeth, M. S., Kumar, K. S., Sagar, N. S., Sankar, S. S., & Karwa, M. (2024).

AI Agent For Multiplayer Games Using Reinforcement Learning. *International Journal of Creative Research Thoughts (IJCRT), 12(5)*, 4496-4506. Retrieved from https://www.ijcrt.org/papers/IJCRT2411646.pdf