Comparison of Two Decision-Making Approaches in Tron Battle: A Simple Rule-Based Algorithm and Reinforcement Learning (Q-Learning)

Hiva Abolhadizadeh
Professor: Dr. Eftekhari
Artificial Intelligence Course

February 6, 2025

Contents

1 Introduction

1.1 Brief Description of Tron Battle

Tron Battle is a classic arcade-style game in which players control motorcyclists that leave a trail of light behind them. The goal is to make other players crash into a light wall or an obstacle, eliminating them from the game. Despite its simple environment, the game requires fast and precise decision-making, making it a suitable challenge for artificial intelligence algorithms.

1.2 Importance of Intelligent Decision-Making in This Game

Due to its high-speed gameplay and constrained environment, Tron Battle demands intelligent decision-making strategies. Traditional rule-based approaches may fail in more complex situations. On the other hand, reinforcement learning algorithms such as Q-Learning allow the agent to learn and improve its decisions over time. Comparing these two approaches helps us understand their strengths and weaknesses and develop better solutions for similar games.

2 First Approach: Rule-Based Algorithm

2.1 General Structure

In this approach, a simple rule-based algorithm determines the player's movement based on the current game grid and player position. The game grid is represented as a 30×20 two-dimensional array, where each cell represents a position on the game board. Occupied cells are marked as True to indicate that movement is not allowed.

The algorithm follows these steps: - The game grid is initialized with False values. - Player positions and trails are updated in the grid. - A function evaluates valid moves to determine the next movement direction.

2.2 Decision-Making Based on Valid Moves

The decision-making process follows these steps: 1. Possible directions (UP, DOWN, LEFT, RIGHT) are defined, each with respective coordinate changes (dx, dy). 2. The algorithm calculates new coordinates (nx, ny) based on each move. 3. The function is_valid_move checks if the new position is within bounds and unoccupied. 4. If a valid move is found, the algorithm selects it. 5. If no valid moves exist (i.e., near defeat), the default move UP is chosen.

2.3 Strengths and Weaknesses of This Approach

- **Strengths:**
  - Simple and fast execution with minimal computational resources.

  - Suitable for basic environments with ample free space.
  - Easy to implement without advanced AI knowledge.

- **Weaknesses:**
  - Cannot predict future consequences of current moves.
  - Struggles in complex environments with many obstacles.
  - No learning mechanism to improve over time.
  - Does not consider opponents' positions.

3 Second Approach: Reinforcement Learning with Q-Learning

3.1 Introduction to Q-Learning and Its Application in Decision-Making

Reinforcement Learning (RL) allows an agent to interact with an environment and learn an optimal policy through trial and error. Q-Learning is an off-policy algorithm that updates Q-Values iteratively using the following formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_a Q(s',a) - Q(s,a)\big]$$

where: - $Q(s,a)$: Current Q-Value of state s and action a. - α: Learning rate. - r: Reward for the action taken. - γ: Discount factor for future rewards. - s': New state after taking action a.

3.2 Updating the Q-Table and Using the Epsilon-Greedy Policy

The Epsilon-Greedy policy is used to balance exploration and exploitation: - With probability ϵ, the agent selects a random action (exploration). - With probability $1-\epsilon$, it selects the action with the highest Q-Value (exploitation).

- **Updating the Q-Table:**
  - The agent identifies the current state.
  - It selects an action using the Epsilon-Greedy policy.
  - Rewards are assigned based on action validity.
  - The Q-Value is updated using the Q-Learning formula.
  - The agent moves to the next state and repeats the process.

4 Comparison Criteria

4.1 Intelligence and Decision-Making Complexity

When comparing decision-making algorithms, intelligence refers to the model's ability to make optimal decisions based on data and the environment. A simple algorithm, due to its reliance on predefined rules, makes fast but relatively basic decisions. In contrast, Q-Learning, by utilizing reinforcement learning, can optimize its behavior based on past experiences and perform better in more complex scenarios.

4.2 Adaptability to Complex Environments

One of the main advantages of Q-Learning is its adaptability to dynamic and complex environments. This algorithm continuously evaluates Q values, allowing it to respond to environmental changes and find the optimal path. In contrast, simpler algorithms generally struggle in complex environments and require manual adjustments to adapt to new conditions.

4.3 Computational Resources and Learning Time

While a simple algorithm is highly efficient in terms of computational resources and execution time, Q-Learning consumes more resources due to the need for frequent calculations to update the Q table and search for optimal values. This can be a drawback in applications with time or resource constraints.

4.4 Performance Against Simple and Complex Competitors

A simple algorithm performs well against basic competitors because it makes quick decisions based on clear rules. However, when facing more complex opponents that require dynamic strategies, its performance declines. Q-Learning, due to its ability to adapt and learn from the environment, demonstrates significantly better performance against complex competitors.

5 Performance Comparison

5.1 Analysis of Advantages and Disadvantages of Each Method

A simple algorithm offers many advantages due to its ease of implementation and fast execution. It is a suitable choice for problems with low environmental complexity, where predefined rules work effectively. However, its adaptability is limited in dynamic and unpredictable environments.

In contrast, Q-Learning provides a dynamic solution capable of learning and improving its performance. However, its long learning time and the need for fine-tuning hyperparameters such as alpha and gamma are considered drawbacks.

5.2 Suitable Scenarios for Each Approach

A simple algorithm is the best choice for problems involving a limited number of states and options, with low environmental complexity. On the other hand, Q-Learning performs better in environments with multiple states, dynamic changes, and complex interactions. For example, in a complex game that requires learning from an opponent's behavior, Q-Learning is preferable.

5.3 Examples and Experimental Results

For instance, in a simulated environment where an agent must navigate, a simple algorithm performs well in a small network with static obstacles. However, in a network with dynamic and changing obstacles, Q-Learning successfully learns from its interactions, finds optimal paths, and demonstrates superior performance. Comparisons in such environments indicate that Q-Learning reduces the time required to reach the goal and improves the success rate.

6 Conclusion

6.1 Summary of the Comparison Between the Two Methods

Overall, a simple algorithm is suitable for low-complexity environments due to its ease of implementation and execution speed. On the other hand, Q-Learning is a better choice for dynamic and complex environments due to its learning and adaptability capabilities. The choice between these two methods depends on the problem's requirements and computational constraints.

6.2 Recommendations for Choosing the Right Approach in Different Scenarios

For simple and static problems, it is recommended to use a simple algorithm. In contrast, for dynamic problems and complex environments, Q-Learning is the preferred approach. In projects where computational resources are limited but learning is required, a hybrid approach combining both methods can be employed.

7 Future Work Suggestions

7.1 Improving the Simple Algorithm

One way to improve the simple algorithm is to introduce adaptive capabilities. For example, designing rules that dynamically adjust based on environmental changes can enhance performance. Additionally, integrating this algorithm with search techniques like A* can increase its efficiency in more complex environments.

7.2 Combining Q-Learning with Advanced Techniques (such as Deep Q-Learning)

To enhance the performance of Q-Learning, combining it with neural networks in the form of Deep Q-Learning is suggested. This method can learn from multiple environmental states and offers better scalability. Additionally, using distributed learning techniques can reduce training time and improve efficiency. Implementing these techniques could lead to the development of an algorithm capable of decision-making in highly complex environments.