НАУЧНЫЙ ЖУРНАЛ
НАУКА И МИРОВОЗЗРЕНИЕ

УДК-004.42

DEVELOPING SNAKE AND RACING CAR ENVIRONMENTS FOR REINFORCEMENT LEARNING

Gulov Gurbanberdi
Student of Oguz han Engineering and Technology University of Turkmenistan
Ashgabat, Turkmenistan

Toylyyeva Ogulgozel
Student of Oguz han Engineering and Technology University of Turkmenistan
Ashgabat, Turkmenistan

Hojabalkanova Sapartach
Supervisor: Lecturer of Oguz han Engineering and Technology University of Turkmenistan
Ashgabat, Turkmenistan

Myradov Rahman
Supervisor: Lecturer of Oguz han Engineering and Technology University of Turkmenistan
Ashgabat, Turkmenistan

Abstract
This article explores the development of interactive environments for Reinforcement Learning (RL) using two classic games: Snake and Racing Car. Both games offer distinct challenges and mechanics, making them ideal candidates for testing RL algorithms. The goal of this paper is to describe the process of designing and implementing these environments, integrating state-of-the-art RL techniques, and demonstrating the results of training agents to master these tasks. Key insights into the practical applications of RL in game development are provided, highlighting how RL agents can adapt and optimize strategies within dynamic, rule-based environments.

Keywords: Reinforcement Learning, Snake, Racing Car, AI, Game Development, Environment Design, Machine Learning

1. Introduction

Reinforcement Learning (RL) has gained substantial popularity due to its application in solving complex decision-making tasks. It is often applied in environments that simulate real-world challenges or games, where an agent learns to maximize rewards through interactions. Two such environments, Snake and Racing Car, present different levels of complexity and require diverse approaches for effective agent training.

This paper provides a deep dive into the development of these environments, their integration into RL frameworks, and the results obtained from training agents.

The Snake game, a two-dimensional arcade game, offers a relatively simple and understandable setup, where the agent must control the snake's movements to collect food and avoid obstacles. Despite its simplicity, it has proven to be a useful environment for testing RL algorithms due to its discrete state and action spaces. The Racing Car environment, on the other hand, introduces a higher level of complexity with its continuous action space and the need for precise control over vehicle dynamics. These contrasting environments offer valuable insights into the strengths and limitations of RL techniques when applied to different types of tasks.

In the following sections, we will describe the detailed process of developing both Snake and Racing Car environments. We will also discuss the integration of RL algorithms, the challenges faced, and the learning outcomes achieved by the agents. By doing so, this paper aims to contribute to the growing body of work in RL-based game development and provide practical knowledge for future applications in the field.

2. Snake Environment Design

The Snake game is a classic example of a problem where agents need to learn optimal strategies through reward maximization. The player controls a snake that grows longer as it eats food while avoiding walls and its own tail. The goal is to maximize the snake's length without colliding.

2.1. State Representation

The state space of the game consists of the position of the snake, the food's location, and the direction of movement. The state can be represented as a grid, where each cell denotes a specific part of the snake or an empty space. In more advanced implementations, the state can also be augmented with information such as the distance to the food and the distance to potential obstacles, providing the agent with more context to make better decisions. This setup allows the agent to perform more sophisticated reasoning, improving its ability to avoid self-collisions and efficiently find food.

Moreover, the state representation can be further enhanced by including a dynamic map of the snake's movements. For example, the environment can track the growth of the snake and update its state accordingly, which is essential for adjusting strategies as the snake becomes longer. This information helps the agent avoid getting trapped in corners or running into its own body as it expands. The complexity of the state space can thus be controlled to match the agent's capabilities, ensuring that the learning process remains manageable.

To simplify the agent's task, the state can also be discretized. In this case, the entire grid is divided into smaller sections, each representing a specific state of the environment. This discretization can make it easier for the agent to understand the world and reduce computational complexity.

2.2. Action Space

The action space consists of four discrete actions: moving up, down, left, and right. The agent’s objective is to learn which action maximizes the length of the snake while avoiding self-collisions. This simple action space allows the agent to focus on decision-making without being overwhelmed by too many choices, making it suitable for Q-learning algorithms. The challenge, however, lies in the fact that the agent must learn to adapt its strategy based on the ever-changing environment, as the snake's position and food placement change with each move.

As the agent learns, the action space may evolve to incorporate more nuanced controls, such as allowing diagonal movement or other specialized actions. This would add another layer of complexity to the problem, requiring the agent to adapt even more quickly and optimize its strategy under different conditions. Additionally, the introduction of new actions could lead to a more efficient exploration of the state space, helping the agent find food faster and avoid collisions more effectively.

A further extension of the action space could include a feedback loop where the agent receives additional actions based on its previous performance. For example, the agent could be given an option to "boost" or "slow down" under specific conditions, such as when it is trapped or when it has gathered a significant amount of food. These added actions would introduce more strategic thinking, requiring the agent to plan its movements carefully and adjust its behavior dynamically.

2.3. Reward Function

The reward function assigns positive rewards when the snake eats food and negative rewards for collisions with the wall or the snake’s body. This setup encourages the agent to learn safe navigation while continuously seeking food. The simplicity of this reward structure allows the agent to focus on mastering basic survival techniques before it can develop more complex strategies for maximizing its length.

In addition to the basic reward structure, more complex reward functions could be introduced to further guide the agent’s behavior. For instance, the agent could be rewarded for maintaining a certain level of distance from the walls or its body, encouraging it to avoid dangerous situations. Similarly, bonuses could be given for long survival times, motivating the agent to prioritize safety over aggressive food collection. These enhancements allow the agent to develop a more balanced approach, learning both efficient exploration and risk management.

Another potential modification to the reward function could be the introduction of negative rewards for wasting time or making repetitive moves. This would incentivize the agent to actively search for food and avoid aimless movement. As the agent’s performance improves, the reward function can be fine-tuned to focus more on long-term survival rather than short-term gains, fostering the development of a more sophisticated decision-making process.

3. Racing Car Environment Design

The Racing Car environment involves a car navigating a race track, where the goal is to complete the lap in the shortest time possible. The environment's complexity lies in the agent's need to handle velocity, steering, and road boundaries.

3.1. State Representation

The state space includes the car’s position, velocity, heading, and distance to the track boundaries. A continuous space representation provides more detailed information to the agent, enabling finer control over decision-making. The position and velocity components are critical for understanding the car's movement, while the heading angle allows the agent to plan the optimal trajectory along the track. Additionally, the distance to the track boundaries serves as a crucial feature for preventing off-road driving and collisions with barriers.

In advanced setups, the state representation can be enriched with visual inputs from the environment. This could include pixel-based images or more abstract representations, such as top-down views of the track. By incorporating these visual cues, the agent can better understand its surroundings and make more informed decisions. This approach is particularly useful in complex environments where the agent needs to process large amounts of information quickly and accurately.

Furthermore, the state space can be adapted to include information about the car’s engine performance, such as throttle and brake usage. This additional layer of detail would help the agent optimize its acceleration and deceleration strategies, ensuring smoother navigation of the track. By taking into account these various factors, the agent can develop more advanced techniques for controlling its speed and direction, ultimately improving lap times.

3.2. Action Space

The action space involves continuous steering and acceleration/deceleration inputs. These actions are essential for the agent to adapt to the dynamic environment and optimize its racing strategy. The continuous nature of the action space presents a greater challenge compared to discrete environments, requiring the agent to learn fine-grained control over its movements. This complexity makes the Racing Car environment particularly suitable for algorithms like Proximal Policy Optimization (PPO), which can handle continuous spaces effectively.

The inclusion of both steering and throttle actions gives the agent the ability to make more nuanced decisions based on its current state. For example, the agent can learn to brake when approaching sharp corners or accelerate on straights to maximize speed. However, this freedom of action also introduces potential risks, such as overshooting corners or losing control due to excessive speed. The challenge lies in balancing acceleration with precise steering to maintain control while optimizing lap times.

In future enhancements, the action space could be expanded to include additional vehicle parameters, such as suspension adjustments or tire pressure changes. These additional inputs would allow the agent to fine-tune its handling characteristics, simulating a more realistic racing environment. Such expansions would require more advanced RL algorithms capable of processing a higher-dimensional action space, pushing the boundaries of what can be achieved in simulated racing environments.

3.3. Reward Function

The reward function in this case is continuous, with positive rewards for speed and lap completion. Negative rewards are assigned for off-track behavior or collisions with barriers, encouraging the agent to balance speed with stability. To prevent the agent from simply driving recklessly, the reward function must carefully weigh the importance of maintaining control versus pushing for faster lap times. A well-designed reward function ensures that the agent learns to optimize its actions for both speed and safety, which is crucial in a racing environment.

One approach to refining the reward function is to introduce intermediate goals, such as passing checkpoints or maintaining a certain speed for a specific period. These intermediate rewards can motivate the agent to improve its overall racing strategy, rewarding consistent performance rather than occasional bursts of speed. This setup also allows the agent to experiment with different tactics, such as optimizing cornering techniques or managing fuel consumption.

As the agent becomes more skilled, the reward function can be adjusted to introduce new challenges, such as incorporating weather conditions or different track surfaces. These changes would add variability to the environment, forcing the agent to adapt to changing conditions while still striving for the best performance. This dynamic reward structure ensures that the agent continues to learn and evolve as the racing environment becomes more complex.

4. Implementing Reinforcement Learning

Both Snake and Racing Car environments were integrated with popular RL frameworks such as OpenAI Gym and TensorFlow. These platforms provide pre-built functionalities for simulating environments and training agents using algorithms like Q-learning and Proximal Policy Optimization (PPO).

4.1. Q-learning for Snake

For the Snake game, a tabular Q-learning algorithm was applied. The agent learns by exploring the environment, choosing actions that maximize cumulative rewards over time. Despite the simple design, the agent was able to gradually improve its performance by avoiding obstacles and consuming food efficiently. The agent’s learning was driven by the principle of temporal difference, where it updated its value estimates based on the rewards received from the environment.

The simplicity of Q-learning made it a perfect fit for the discrete nature of the Snake environment. However, as the state space grows larger with longer snakes and more complex movements, the algorithm's efficiency can decrease. One solution is to use function approximation techniques, such as deep Q-networks (DQN), to generalize the learning process across larger state spaces. This approach would allow the agent to handle more complex scenarios, such as navigating tighter spaces as the snake grows.

Despite the straightforward nature of the task, the Q-learning agent faced challenges in balancing exploration and exploitation. Initially, the agent preferred exploring random actions, which led to suboptimal performance. However, as the agent gained more experience, it began to exploit its knowledge by choosing actions that maximized its reward. The balance between these two behaviors was essential for the agent to master the Snake game.

4.2. PPO for Racing Car

For the Racing Car environment, Proximal Policy Optimization (PPO) was utilized due to its efficiency in continuous action spaces. The agent learned to navigate the race track by optimizing its control over the car's speed and direction, aiming to minimize lap times. PPO is known for its stability and effectiveness in complex, high-dimensional action spaces, making it ideal for this environment.

One key advantage of PPO over traditional RL methods is its ability to handle large, continuous action spaces. This was particularly beneficial for the Racing Car environment, where precise control over the car's steering and throttle is crucial. The agent was able to learn more nuanced driving strategies, such as adjusting its throttle at different points on the track to maintain an optimal speed.

Despite the success of PPO in the Racing Car environment, the agent faced challenges when dealing with extreme driving conditions, such as sharp turns or sudden changes in the track layout. The agent's performance was heavily influenced by the quality of the reward function, which had to carefully balance the importance of speed and control. Fine-tuning the reward structure played a crucial role in helping the agent optimize its driving skills.

5. Results and Analysis

The RL agents demonstrated significant improvements over time. In the Snake game, the agent quickly learned to avoid collisions while maximizing its length. The learning curve showed steady progress as the agent adapted its strategy to avoid dangerous situations and improve its food collection efficiency.

In the Racing Car environment, the agent showed increased lap times as it learned to balance acceleration with steering. Initially, the agent struggled with maintaining consistent lap times, frequently making errors due to poor throttle control. However, with continued training, the agent began to recognize patterns in the track and adjust its behavior accordingly.

Both agents benefited from the application of appropriate reward functions and fine-tuned exploration strategies. The Snake agent was able to refine its movements, learning how to optimize its trajectory to consume food while avoiding collisions. The Racing Car agent, on the other hand, was able to improve its lap times by focusing on more efficient navigation and optimal throttle control. These results demonstrate the potential of RL in developing intelligent agents capable of mastering complex, dynamic environments.

6. Conclusion

This paper highlights the importance of designing effective environments for RL agents to develop and refine decision-making strategies. Both Snake and Racing Car environments present unique challenges that can be adapted for various RL techniques. By providing a detailed overview of the implementation process and results, this work contributes to a broader understanding of how RL can be applied to game development and agent training.

The success of the RL agents in mastering both the Snake and Racing Car games demonstrates the versatility of RL algorithms in tackling a wide range of tasks. Although both environments vary in complexity, they share the common goal of allowing agents to learn through experience and maximize cumulative rewards. Future work will involve further refinements to these environments, as well as exploring additional RL algorithms that can handle more complex state and action spaces.

In conclusion, this paper offers valuable insights into the practical applications of RL in game development. By developing environments that challenge agents to optimize their strategies, this work provides a foundation for future research and applications in the field of artificial intelligence. The ability to train agents in these environments opens up new possibilities for developing intelligent systems that can adapt to dynamic, real-world challenges.

7. References

1. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*.
2. Schulman, J., et al. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.
3. Bellman, R. (1957). Dynamic Programming. *Princeton University Press*.
4. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. *MIT Press*.