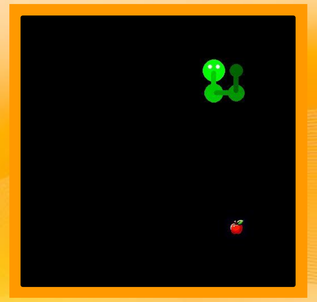
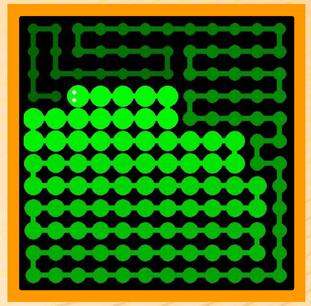


AI Mastery in the Snake Game: First Step to a Self-Driving Future



Introduction

- Driving a car is one of the most dangerous things people do every day, as over 6 million car crashes occur in the US every year. The high rate of accidents highlights the urgent need for **safer driving practices.**
- One promising solution for road safety is the development of **self-driving cars**, a recent technology using complex **artificial intelligence (AI)**. Equipped with sensors and AI algorithms, self-driving cars can make quick, safe decisions on the road, offering the possibility of accident-free roads.
- This project dives into a simplified model of a self-driving car — the classic Snake game. By navigating the game's challenges, the project demonstrates the process of **evaluating and responding to dynamic environments**, similar to the **decision-making** required for real-world driving.
- Inspired by the real machine learning models used in self-driving cars, this project utilizes **deep reinforcement learning (DRL)** and **convolutional neural networks (CNNs)** to solve the complex decision-making tasks in the Snake game. This is the **exact same type of AI** that is used in self-driving cars, but small enough to run on a single home computer.

Background

DRL — Deep Reinforcement Learning is a type of machine learning that is primarily based on trial and error. This type of machine learning helps the AI model optimize itself by giving it reward functions to aim for, and those reward functions are all directed toward a goal.

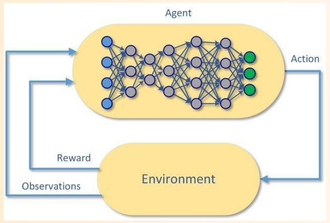


Figure 1. Structural diagram of Deep Reinforcement Learning

From IEEE Access: "Investigation on Works and Military Applications of Artificial Intelligence"

CNN — A Convolutional Neural Network is a type of neural network that processes images. It has many different layers and non-linearities, mainly used to simplify, learn the main features of, and classify images. CNNs process visual information from the game, enabling the system to interpret and react to its surroundings, much like a self-driving car would in the real world.

Snake Game — The classic game of Snake is quite simple. The player controls a snake that can turn in the cardinal directions (but not backwards), and tries to eat as many apples as it can without crashing into a wall or its own body. Each time an apple is eaten, it spawns in a new random unoccupied spot. If there is no more room for apples to spawn, the game is won. In this project, the board is 12x12, allowing the snake to reach a max size of 144.

Engineering Goals

1. **Develop** a deep reinforcement learning model that can excel at the Snake game.
2. **Optimize** the learning algorithm for efficiency and performance in the game.
3. **Learn** how to use reinforcement learning for AI decision making, as applied to algorithms for self-driving cars.

Procedure

1. Learn how to use **Stable Baselines 3**, which is a deep reinforcement learning package in **Python** that is built on top of the **OpenAI Gym** package.
2. **Code an interface** between OpenAI Gym and my custom Snake game that allows the trained AI to play the Snake game.
3. Program reward functions and train a neural network to play the Snake game by using the **MaskablePPO** class in Stable Baselines 3.
4. Save the trained network and **test the results** by having the AI play the Snake game, recording the average total reward that the AI achieves.
5. Review the AI's gameplay and attempt to **improve the model** based on the mistakes made.
6. Add **statistics** for test runs and tweak the necessary values until a completely optimized model has been produced.

Challenges and Process

1. The first model I used was the CNN with 3 reward functions — getting food, staying alive, and moving towards food. The CNN is effective because it can see several steps into the future and therefore knows how to find open paths to the food. But eventually, it would get stuck or turn the wrong way.
2. The most important change to improve the model was the **Non-trap reward** function — this function made sure that the AI's snake head always has direct access to most of the remaining empty tiles on the board, or the model would be penalized. This simple change was **incredibly effective and eventually brought the AI's success rate from 0% to over 90%** after properly tuning the reward function.

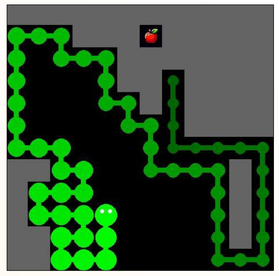


Figure 2. Illustration of accessible vs non-accessible area by snake head. The black area is accessible, the gray area is non-accessible.

3. Definition of the non-trap award function.

$$R = W * (P_a - 0.5) * P_s$$

R: Reward

W: Weight Factor

P_a: Percentage of Accessible Area

P_s: Percentage of Snake Growth

Figure 3. Non-trap award function.

4. The most tedious process was fine-tuning the parameters. After making a new reward function that was very successful, I needed to optimize it such that it was the perfect value between risky and cautious, so I tried more than 50 different models. Instead of training models from scratch, I could also load and refine models for maximum optimization.

Results

1. To monitor the training process, **I used a helpful tool called TensorBoard**. TensorBoard allowed me to see the progress of the trainings in graphs. In this project, the length and reward of each model over the training episodes were recorded.
2. TensorBoard also let me select the models to display on the graph, allowing me to **see how the improvement of different models compared to each other**. I could check the data from each model to see what needed to be changed. Using TensorBoard, I was able to see the immense difference between my first CNN model all the way to my refined Non-trap reward designs.

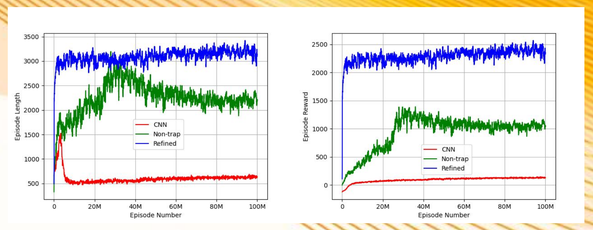


Figure 4. The episode length (left) and episode reward (right) in the training process.

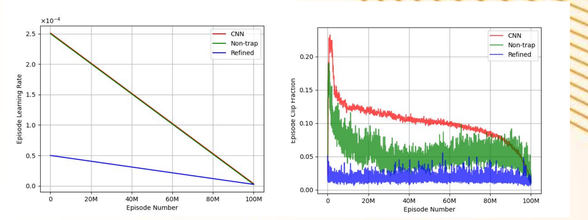


Figure 5. The learning rate schedule (left) and episode clip fraction (right) in the training process.

3. **After models are trained, tests are run using each model.** I exported the result data for each model, including average reward, score, and max score count.
4. **I implemented a histogram function** (Figure 6) that recorded stats from test runs and turned them into an easy-to-read bar graph.

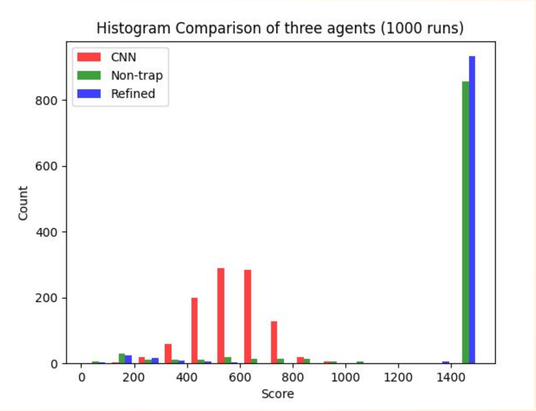


Figure 6. The histogram comparison of three different agents.

Red — The starting CNN model with a 0% finish rate and average score of 470.

Green — The model implemented with Non-trap reward with a 78.5% finish rate and average score of 1260.

Blue — Start with the green model, re-trained with a different learning rate schedule and greater weight factor for Non-trap reward, this model achieved a 93.3% finish rate and average score of 1340.

All images and graphics were created by the researcher unless otherwise noted

Conclusions

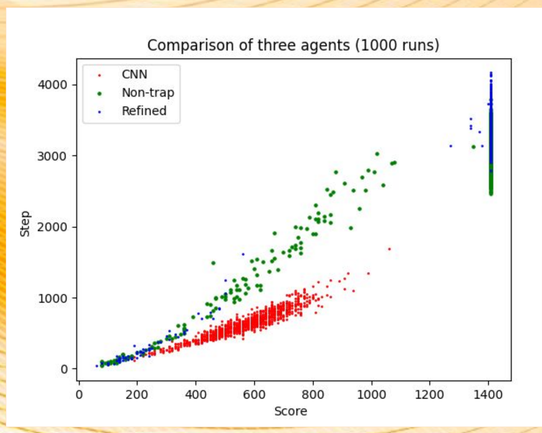


Figure 7. Comparison of three agents, each runs 1000 tests.

1. These graphs show the trend in the behavior of the models shown. Models like the CNN are inconsistent and often are limited to a certain score, meanwhile the Non-trap models are **very consistent after a certain threshold**.
2. This project highlights the significance of **analyzing the effects of various parameters** during training and **identifying the optimal approach** that leads to a snowball of positive results. With a decent model and a great reward function, the challenge of the Snake game transitioned from being seemingly unbeatable to easily manageable.

Applications

1. Companies like Tesla currently use advanced decision-making models to create self-driving cars, which could save millions of lives. **The AI for the Snake game models the AI in self-driving cars** using **deep reinforcement learning (DRL)**.
2. In the Snake game, the AI learns to **navigate the grid** to eat apples without crashing. This fundamental DRL approach is similar to how self-driving car algorithms learn to **navigate roads, avoid obstacles, and reach destinations safely.**
3. The Snake game requires the AI to make decisions with **incomplete information** and **anticipate future scenarios**. Similarly, self-driving cars must **predict the actions of other road users** and make decisions that minimize risk.
4. By working with deep reinforcement learning in the Snake game, AI researchers like me can **learn skills for AI decision making, model optimization, and image processing with just a single home computer**, turning the problem of self-driving cars from impossible to easily accessible.
5. AI researchers should use **clever reward functions** to improve performance, such as how the Non-trap function **greatly impacted the snake model with only a few lines of code**. This technique also **reduces energy usage** because we can train models for far less time — this is essential because machine learning already uses more energy than the country of Ireland, and it will keep on rising rapidly in the future.

References

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017). Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560.

Rafat, T., & Islam, M. R. (2021). A memory efficient deep reinforcement learning approach for snake game autonomous agents. arXiv preprint arXiv:2301.11977

Rana, A. (2018). Introduction: Reinforcement Learning with OpenAI Gym. Towards Data Science.

Guszejnov, D. (2022). How to Train an AI to Play Any Game: Using Reinforcement Learning Techniques. Towards Data Science.