Playing Mario Kart 64 with Deep Reinforcement Learning

Young James Yang
The Hong Kong University of Science and Technology
jyyoungaa@connect.ust.hk

1. Introduction

The application of artificial intelligence and machine learning techniques to play video games has emerged as a captivating area of research, showcasing their potential in mastering complex game environments. One widely used machine learning method to play games is through deep reinforcement learning, which combines reinforcement learning and deep learning to train agents to play games. Deep reinforcement learning in video games has made significant progress in recent years, where it has been successfully applied to play games such as Doom [3], various Atari games [2], Starcraft [5] and so on.

This project focuses on applying deep reinforcement learning to play the popular Nintendo 64 (N64) game Mario Kart 64. The aim for this project is to design and train an autonomous game playing agent that is able to navigate and complete a course as fast as possible in the time trial mode. This will be done on the two courses Luigi Raceway and Moo Moo Farm. The project will use real-time screenshot images of the game and data from the game's memory to extract features such as the player's velocity, laps completed, and so on. Then, using the agent's current state, Deep Q-Learning will be applied to determine the action to take.

2. Problem Statement

The problem this project is going to solve is creating a real-time autonomous agent that is able to navigate and complete a course in the time trial mode in Mario Kart 64. In time trial mode, the agent will race on its own and the agent's aim is to try and complete a specified number of laps of the course in the fastest time possible. The agent will be trained and evaluated on the following two courses in Mario Kart 64: Luigi Raceway and Moo Moo Farm.

The project will use real-time screenshot images of the game as well as information from the game's memory as the data set. Using both of these, information such as the laps completed, agent's course progress, kart's velocity, and other game information can be extracted and used in designing and creating a model.

The expected result is for the agent to complete both courses. Using the fastest time for both courses, the results can be evaluated by comparing it to real-human performance as well as results from another study that uses imitation learning to play Mario Kart 64 [1].

3. Technical Approach

This project will specifically aim to use the deep reinforcement learning algorithm Deep Q-Network (DQN) to approach this problem.

3.1. Emulation

To automate the races in Mario Kart 64, the N64 emulator Mupen64Plus will be used which will allow for saving and loading states as well as checking in-game memory for game information. The project will use Farama's Gymnasium framework (fork of OpenAI Gym framework) [4] as an environment wrapper.

3.2. Data Collection and Processing

The model will be run on the courses Luigi Raceway and Moo Moo Farm. They are chosen as they have well-defined walls which mitigates the issue of the agent driving or falling off track, which will help speed up training. The game is ran at a resolution of 640x480 pixels with RGB, where each frame has been down-sampled to 200x66 pixels with RGB. The current neural network consists of 5 CNN layers with 3 fully-connected layers.

3.3. Deep Q-Learning

Q-learning is a reinforcement learning algorithm that enables an agent to learn optimal action-selection policies. The Q-learning algorithm updates the Q-values, denoted by $(Q^*(s, a))$, which represent the expected cumulative rewards for taking action (a) in state (s). The Q-values are iteratively updated using the following equation:

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a')] \quad (1)$$

where (α) is the learning rate, (r) is the immediate reward obtained by the agent, (γ) is the discount factor that determines the importance of future rewards, (s') is the next

state, and (a') is the optimal action in the next state. This update equation allows the agent to learn from its experiences and gradually converge to an optimal policy that maximizes long-term rewards [6].

Deep Q-learning is an extension of Q-learning that leverages deep neural networks to approximate the Q-values, enabling learning in high-dimensional state spaces. The Deep Q-learning algorithm employs a deep neural network with parameters (θ) to approximate the Q-values. The network is trained by minimizing the mean squared error loss between the predicted Q-values and the target Q-values, given by the Bellman equation:

$$L(\theta) = \mathbb{E}s, a, r, s' \left[\left(r + \gamma \max a' Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \quad (2)$$

where (r) is the immediate reward obtained by the agent, (γ) is the discount factor, (s') is the next state, (a') is the optimal action in the next state, and (θ^-) represents the target network parameters. The network is updated using gradient descent to minimize the loss, allowing it to approximate the optimal Q-values and guide the agent's decision-making. Through this process, Deep Q-learning enables agents to learn optimal policies in complex and high-dimensional environments [2].

3.4. Designing State, Action, and Reward

To create a discrete environment for DQN, the car will only take on discrete actions and states. Currently, the plan is to allow for four discrete actions: drive forward, backwards, left, or right. The emulator will use keyboard inputs that are mapped to controller inputs to simulate each action. The car will have the following discrete states which will take on either 0 or 1, where 0 is false and 1 is true:

- Making Progress: Checks if car is making progress in the course. This can be checked in the game memory, where if the value increases from previous value then progress is made.
- Kart is Stuck: If the car's velocity is extremely low for a few frames, then the car is stuck.
- Kart is on Track: If the car's velocity is above a certain threshold, it is moving on the track whereas if the car is off-track or bumping into a obstacle, it will be above the stuck threshold but be below the on-track velocity threshold.

The agent will be rewarded for making progress in the course and completing laps around the course. It will be punished when it falls off the track or gets stuck.

4. Preliminary Results

Current results have performed poorly so far, where many of the runs had the issue where the car did not advance much in the course. This was mainly due to poor state and reward design, where in those trials the only reward given was based on the car's velocity and laps completed. Figure 1 below shows an example run where the car spent a large amount of time driving in circles then going backwards.

Figure 1. Example of run where agent spent most of time driving backwards

Therefore several changes have been made, mainly to the design of the state, action, and rewards, which have been described in this paper. However, they have yet to be tested and further changes and adjustments will be made. For example, more actions may be added or removed to improve performance, such as adding more defined left and right directions (eg. hard left and soft left) or removing the option to reverse and having the car auto-accelerate to simplify the problem.

References

[1] Harrison Ho, Varun Ramesh, and Eduardo Torres Montano. Neuralkart: A real-time mario kart 64 ai, 2017. 1

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. 1, 2

[3] Georgios Papoudakis, Kyriakos C. Chatzidimitriou, and Pericles A. Mitkas. Deep reinforcement learning for doom using unsupervised auxiliary tasks, 2018. 1

[4] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, Mar. 2023. 1

[5] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney

Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017. 1

[6] Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992. 2