Project Proposal

Reinforcement Learning in Mario Kart Wii

## Background

Due to the popularity of Mario Kart Wii, there are a few similar projects to this one already out there. Firstly, Tango AI on YouTube has a series dedicated to an [AI learning Mario Kart Wii](https://www.youtube.com/playlist?list=PL5CVAogrehQHKKIgJmF2t3p16Aygy0Rxc). He doesn’t go into too much detail on the exact process, but he does outline his reward function. The main factor of this is a checkpoint system that he designed, that uses the on-screen minimap. This, combined with the on-screen lap counter, gives a complete representation of how much of the race the agent has completed. In later videos he combined this approach with determining the speed and rewarding the agent for maintaining a high speed and penalizing it for going below it.

[JackWBoynton](https://github.com/JackWBoynton) on GitHub presented a different approach to the problem using deep RL. Instead of optimising the controller inputs required to play the game, he focused on optimizing the trajectories of the agent, aiming for the lowest time-trial time. He created a gym environment which used on-screen data combined with data from the games memory to represent the state. He also used cross-track error, which is a measure of how far off-course the agent is, as a reward.

To differentiate myself from these approaches, I have decided to have more of a focus on interaction with the AI through playing against it. At various points in the learning process, I will save the state of the learning so that it is possible for a human to play against it. This will be a set of controller inputs and associated timings. This can be very useful as a demonstration of the learning process and any potential roadblocks and how these were circumvented.

Additionally, through the use of cheat codes supported by the emulator, it is possible to get lots of useful information about the state of the game printed to the screen. These values will be used to represent the state, along with the main area of the screen.

A screenshot of a computer

Description automatically generated

Figure 1 – A screenshot of information provided by the cheat code

## Project Aims/Objectives

**Aims**

* Train an AI model to play Mario kart Wii using reinforcement learning.
* Design an appropriate RL approach to train it and evaluate this approach.
* Play against this ai at different levels in its development.
* To show the major breakthroughs and demonstrate the learning process.

**Objectives**

I have a good idea of the approach that I will take, which will be refined as my research into this field continues.

**Environment**

I will be using the open-source unofficial GameCube and Wii Emulator called [Dolphin](https://dolphin-emu.org/). It is by far the most widely used and recommended within the community. It has many features built in to support this kind of project, such as input recording/replaying, save state support and framedumping.

Mario kart Wii is a complex game with lots of concepts for the model to learn, so I will simplify the environment from the chaotic races that make up the core game-mode. Firstly, the AI will not have access to Items, these are rather complex to understand as even good items can lead to a bad outcomes if used incorrectly. Secondly the AI will train on the track by itself. Having no opponents means the AI can learn how the game works without the additional clutter of other racers. The character/kart combo that will be used is Funky Kong on the Flame Runner. This combo is the most widely used in online play due to its high top speed and tight drifting capabilities. This will allow me to compare the achieved lap time to that of an experienced human player. The track that I will train on will be SNES Mario Circuit 3. I feel this hits the sweet spot in terms of difficulty and making the learning process interesting.

The state space will consist of features extracted from screenshots of every frame, split into 2 sections. Firstly, we will have the main area of the screen, which contains the character and the track. To get necessary information about the direction of the track and the kart, I will downsample and greyscale the image and then feed it into a CNN for feature extraction. The main rationale behind this is that CNNs excel at recognizing spatial relationships, and the relationship between the direction of the kart and the upcoming direction of the track is exactly what I need to make sure the kart stays on the track. Additionally, there will be a section on the left-hand side of the screen that contains information about the state of the game, obtained through cheat codes. Not all of this is relevant, but I will be able to crop what I need. As this information will be text in an image, I will use optical character recognition from Tesseract to extract this into text, and then numbers. The information that I will use will be:

1. XZ Vel: Current horizontal Velocity (kph)
2. Race%: Current race completion (0-3)
3. SSMT/MT/SMT: MT stands for Mini-Turbo, which is a speed boost you get after drifting for ~1s.

The action space will include all controller inputs that have a direct effect on driving the vehicle, for example the pause button will be disabled along with ‘Home’. The emulator allows for different types of emulated controllers to be used (Wiimote, Nunchuck, GameCube Controller). I have chosen the GameCube controller for one main reason, it replaces shaking the Wiimote with a simple button press. This input results in the character performing a wheelie, temporarily increasing its top speed in a straight line. Additionally, the use of a joystick instead of rotating the Wiimote like a wheel, allows for more fine-tuned turning inputs which will result in more accurate driving and therefore a faster lap. Some actions are controlled by multiple buttons on the GameCube controller, for simplicity only one of each will be used. The list of inputs and resulting actions are as follows:

1. Button A -> Accelerate.
2. Button B -> Drift (when held with direction)

-> Hop (when tapped)

-> Brake (when held with no direction).

1. Joystick angle -> Steer (0-14 with 0 = hard left, 7 = neutral, 14 = hard right).
2. D-pad Up -> Wheelie.

**Reward Function**

I want to reward the agent for driving around the track and doing it quickly. For driving around the track the Race% will be used as it gives the most detail on how much of the race has been completed. To encourage driving quickly, I will use the XZ Vel value. This is all that is needed as the track being used is completely flat. As I mentioned earlier, performing a wheelie temporarily increases the top speed of a vehicle in a straight line, and a miniturbo (a result of holding a drift for ~1s) also does so. The most optimal way to drive around the track is to wheelie for all the straight sections and drift, performing a miniturbo, around all the corners. To encourage this behaviour, I will give a small bonus reward every time the agent performs one of these actions. Additionally, hopping repeatedly greatly decreases the speed, so to discourage this I will give a penalty if the speed goes below the vehicles normal top speed. This will also help in encouraging drifting around a corner instead of just turning because this speed is maintained through a drift but decreases slightly during steering.

**RL Algorithm**

Due to the large search space, I will be using a Deep Q-Network for the model. This uses a neural network as an approximation of the Q function to give an estimated value for taking an action in a given state.

**Play against the AI**

To enable people to play against the AI I will use Dolphin’s in-built input recording and replaying feature. I will record the AI’s inputs and replay them so that it is possible for a human to play against it. Unfortunately, Dolphin does not currently support this simultaneously so a simple lap time comparison will be used. This feature will also be useful for documenting the learning process as I can record the inputs after certain amounts of learning and review its progress.

## Plans to Achieve Objectives

**Reading**

Self-driving cars also need systems to drive effectively, J. Duan et al [1] proposed a hierarchical RL approach, where a manoeuvre policy was used, giving a manoeuvre based on the environment, which was then fed into a sub-policy detailing what brake/accelerator and steering inputs are needed. This resulted in a ~25% reduction in the driving time when compared to normal RL, meaning the car was driving faster. I considered taking this approach, specifying a set of inputs such as sharp turn left/right, slight turn left/right and straight on, but decided against it as I would prefer to see the AI learn these actions by itself, driven by a well-formulated reward function.

Many papers from the AAAI Conference propose RL strategies for games. Hansin Ahuja et al. [2] propose a graph-aware RL approach to win games of Diplomacy. The data was first classified, then a reward estimation was trained, with the winner of the game having the highest reward. They found it difficult to judge the true value of singular actions, as only that action was taken in each state. In a different approach, D. Choe et al. [3] suggested a combination of user and card style modelling with DNN to mimic a user’s playing style. They found it useful for increasing the life span of various online games. This could be applied to Mario kart Wii and provide a higher difficulty of CPU characters to play against, if someone were to take this approach, it would be interesting to see how it compares to mine.

## Hardware/Software Resources

I will require computing power from the University of Birmingham for the learning process, based on other works, around 80 hours will be sufficient to train the AI to a suitable skill level.

## Datasets

I will not require any datasets, as I am using an unsupervised learning approach.

## Project Plan

**Gantt Chart**

A screenshot of a computer

Description automatically generated

1. Reading – Central and General reading/research attaining to my project.
2. Emulator Setup – Configuring emulator to allow programmatic interaction.
3. Emulator Interaction – Setting up real-time frame-dumps and controller inputs programmatically.
4. Problem Design – Considering what I have available from the emulator, design my approach.
5. Problem Formulation – Formalise my problem in python.
6. Learning Process – Using hardware resources from the school, run the RL process.
7. Testing/Tuning – Monitor the learning and check its progress, potentially adjust the formulation accordingly.
8. Data Analysis – Show results from the learning.
9. Evaluation – Reflect on how successful my approach was, using the data collected.

## References

[1] Duan, J. *et al.* (2020) *Hierarchical reinforcement learning for self-driving decision-making without reliance on labeled driving data*, *arXiv.org*. Available at: https://arxiv.org/abs/2001.09816 (Accessed: 04 November 2023).

[2] Ahuja, H., Ng, L.H.X. and Jaidka, K. (2022) *Using graph-aware reinforcement learning to identify winning strategies in diplomacy games (student abstract)*, *arXiv.org*. Available at: https://arxiv.org/abs/2112.15331 (Accessed: 04 November 2023).

[3] Daegeun Choe, Y.J. (2023) *UCSM-DNN: User and card style modeling with Deep Neural Networks for personalized game AI*, *AAAI*. Available at: https://aaai.org/papers/13158-ucsm-dnn-user-and-card-style-modeling-with-deep-neural-networks-for-personalized-game-ai/ (Accessed: 04 November 2023).