Project Proposal

Reinforcement Learning in Mario Kart Wii

## Background

* ~~What is already out there~~
  + ~~Tango AI YouTube video/s~~
    - ~~Used self-made checkpoint system~~
  + ~~Jack Boynton Presentation~~
    - ~~Found optimal trajectories/path around the track~~
    - ~~More suited to more realistic racing games/sims~~
      * ~~Or real life? Cross-Track error applied to self-driving cars~~
        + ~~Would need near-perfect knowledge of road boundaries/lanes etc~~
* How is mine different
  + ~~None show a human playing against the AI, I will do this at different stages and see the progress~~
  + ~~Looking like I may use gecko codes and get data from screen dump, others did not do this – will hopefully allow ai to train more quickly~~

Due to the popularity of Mario Kart Wii, there are a few similar projects to this already out there. Firstly TangoAI 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 in his videos, but he does outline his reward function. The main driver of this is a checkpoint system that he designed, that uses the on-screen mini-map. 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 punishing 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 screen data and data from the games memory to represent the state. He also used cross-track error, which calculates how far off course the agent is.

To differentiate myself from these approaches, I have decided to have more 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

## Project Aims/Objectives

* Aims
  + Train an AI model to play mariokart 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
  + Designing RL Approach

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

**Environment**

Mariokart Wii is a complex game with lots of things 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 outcome 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. The track that I will train on will be SNES Mario Circuit 3. I feel this hits the sweet spot in terms of difficulty as there is only one straight section, but also has enough turns to make 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 Dolphin emulator allows for different types of emulated controllers to be used (Wiimote, Nunchuck, GameCube Controller). I have chosen to go with 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)
3. Joystick angle -> Steer (0-14 with 0 = hard left, 7 = neutral, 14 = hard right)
4. Dpad 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 increases the top speed of a vehicle in a straight line, and a miniturbo (a result of drifting) 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 instead of turning because this speed is maintained through a drift but decreases slightly during steering.

* + - ~~Complete reading on RL topics/papers~~
    - ~~Define:~~
      * ~~Environment~~
        + ~~State Space~~
        + ~~Action Space~~
      * Reward Function
      * RL Algorithm
        + DQN
      * Model Architecture
        + Main screen area

Downsample, greyscale

* + - * + Cheat code information
  + Training the AI
    - Use CNNs to transform framedumps from the game into inputs to a neural network
    - Train this neural network using a reward function that has many parameters that will change over time. I’m not expecting the learning to be very efficient straight away, but I will tune this reward function and its weights throughout the learning process as it encounters different difficulties
    - Through the emulator there are ‘cheats’ that show you important values on screen, as I have been unsuccessful in getting a memory viewer to work, I could crop these areas of the framedump and use text recognition to get the values and then feed these as inputs to NN
      * Will need to do more reading on this
        + Image manipulation in python
        + Text recognition possibly?
  + Play against the AI
    - I will save the state of the neural network at different stages of the learning process and be able to reiterate its inputs into the emulator
      * May have to be non-simultaneous, but simultaneous would be a lot better
    - Dolphin savestates may not be able to be used, controller inputs are looking most likely
  + Show major breakthroughs
    - Again storing controller input, I want to set flags that save the neural network when it completes first turn, lap, 3lap, beats easy difficulty cpu, beats staff ghost, beats exert staff ghost

## Plans to Achieve Objectives

**Reading**

* Already done
  + Textbook on Reinforcememnt learning with Python
    - Very informative on concepts within RL and gives examples with code
  + 3-D hand posing with CNN
    - Proposes feature extraction using CNN, along with an RL module for path optimisation
    - Goes from 3-d point cloud to joint/finger estimation
  + Hierarchical RL for self-driving decision-making
    - Proposes multi layer RL architecture
      * High-level manouevre selection
      * Low-level motion control
        + Could adopt manouevre selection idea and pre-define inputs instead of motion control learning
* To Do
  + Reinforcement Learning textbook from Library
  + RL Games Conference

**Problem Formulation**

* Reward Function
  + This will depend mostly on what information I’m able to retrieve from the emulator
  + At the moment will learn on previous 4 frames to give awareness of movement
    - Maximise: Speed, Checkpoint, Lap completion %
      * (Checkpoint ~= Lap Completion)
    - Bonus: Perform MT, Perform wheelie (maintain > 87kph speed for x seconds)
    - Penalty: Speed lower than Flame Runner base speed
    - Terminate: Hit object/offroad (speed < 50)

**Emulator Interaction/Configuration**

* Gym Environment
  + Open source but not updated/maintained
  + Contains local paths from creator
  + Could modify gym to get only what I need
    - Visualisations and memory viewer not needed
    - Framedumps are needed
      * Add functionality to get key values from on-screen ‘cheat’ codes

\*\* Is a gym env 100% required? If so would it be possible to make my own? If not then I will have to use a different game, In which case I would search for the gym environment first and then select a game that has one already.

* Controller compatibility
  + Gym env uses unix Pipes, but my attempts at doing so in the past have failed
  + If they don’t work then I will have to modify the controller class to use PyAutoGUI, along with the dolphin config files

\*\* I will require computing power from UoB for the learning process, based on other works, around 60 hours will be sufficient to train

**Programming**

* I would like to start the learning process before the start of the Christmas holidays and leave the time after for documentation, data analysis, tuning, and possibly considering different Reward Functions
* In terms of the environment, I will give myself until 13th November to get that working, at which point I will change which game I do the learning process for, based on what gyms are out there already. In this case I will have to change my approach depending on the style of games that are available. If possible I would still like to do a racing game as some aspects will carry over
* If the mkwii gym environment is successful then I aim to have the learning process fully functional by the winter break, In the event that I am unsuccessful in this, I will spend time over the holidays to catch up, so that I can start it when I come back

06/11/23 – COMPLETE PRORMPOSAL!!!

1. Research
2. Problem Formulation
3. Formalisation
4. Emulator Interaction/Configuration
5. Programming
6. Learning Process
7. Data Analysis