**Using ML Agents in Unity**

Shape

Description automatically generated

In this project, I have used ML agents in Unity to train a kart to drive around a racetrack on its own. We will see how this was done, the results at different instances and even certain different routes that the agent decided to take.

**What are ML Agents?**

ML-Agents is a framework for training intelligent agents using Machine Learning (ML) techniques, developed by Unity Technologies. It allows developers to train autonomous characters or agents in virtual environments, using techniques from reinforcement learning, imitation learning, and other ML approaches.

ML-Agents is designed to be flexible and modular, so developers can easily customize the learning algorithms and environments to fit their needs. It is implemented in Python and can be integrated with the Unity game engine, allowing developers to build interactive environments and simulations for training and evaluating intelligent agents.

There are 3 important components when it comes to understanding ML Agents in Unity:

1. Learning Component:

This is the unity scene that acts as the environment for the agent to be trained in

1. Python API

This contains the RL algorithm, in this case PPO (Proximal Policy Optimization), which is used to train the agent in the environment and test it

1. External Communicator

This is the communicator between the API and the learning environment

Diagram

Description automatically generated

**Why PPO?**

In this project, we have trained a kart agent to drive around a racetrack using PPO. When making its decision between if the kart should turn left or right, even though these are two discrete actions, the action space is actually continuous as the amount of turning to the left or right also matters. And it would also improve the efficiency of the learning process. Hence using PPO seemed like the best option.

The general form of the PPO loss function is:

Loss = Clip(r \* A, 1 - epsilon, 1 + epsilon) \* A

Where:

* Loss is the PPO loss function.
* Clip(x, min, max) is a function that clips the value of x to the range [min, max].
* r is the importance sampling ratio, which is a measure of the relative probability of the current policy compared to the previous policy.
* A is the advantage function, which is a measure of the value of an action compared to the average value across all actions.
* epsilon is a hyperparameter that controls the amount by which the policy can change from one iteration to the next.

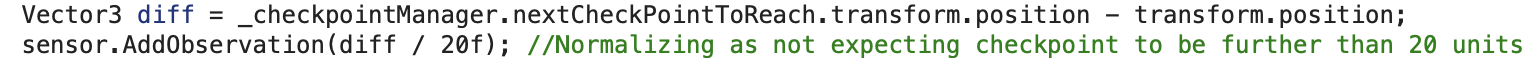
The PPO loss function is designed to balance the trade-off between exploration (trying new actions to improve the policy) and exploitation (using the current policy to maximize reward). By clipping the importance sampling ratio, the PPO loss function limits the amount that the policy can change from one iteration to the next, which helps to stabilize the learning process and prevent the policy from becoming too erratic.

**Initial Setup**

The great thing about using unity is that you can train several models of the agent simultaneously to speed up the process. I made use of this and trained many of the kart agents together. While they are able to detect walls and checkpoints, they were set up so as to not detect each other.

The rewards have been set up in a checkpoint system. There are many checkpoints around the racetrack. The goal of the kart is to always reach the next checkpoint, no rewards are awarded for going back to the previous checkpoint.

Calculate distance between next checkpoint and the kart:



Reward system:

Graphical user interface, text, application

Description automatically generated

The karts also have 2 ray perception sensors, one to observe its surroundings and the other as a look ahead sensor mainly to detect the next checkpoints

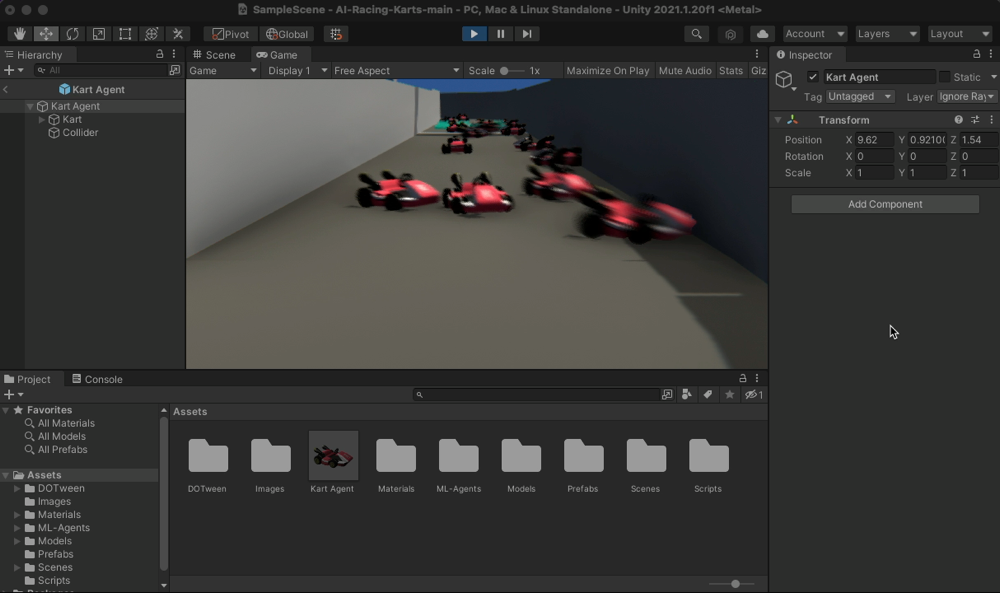
Ray Perception sensors:

A picture containing diagram

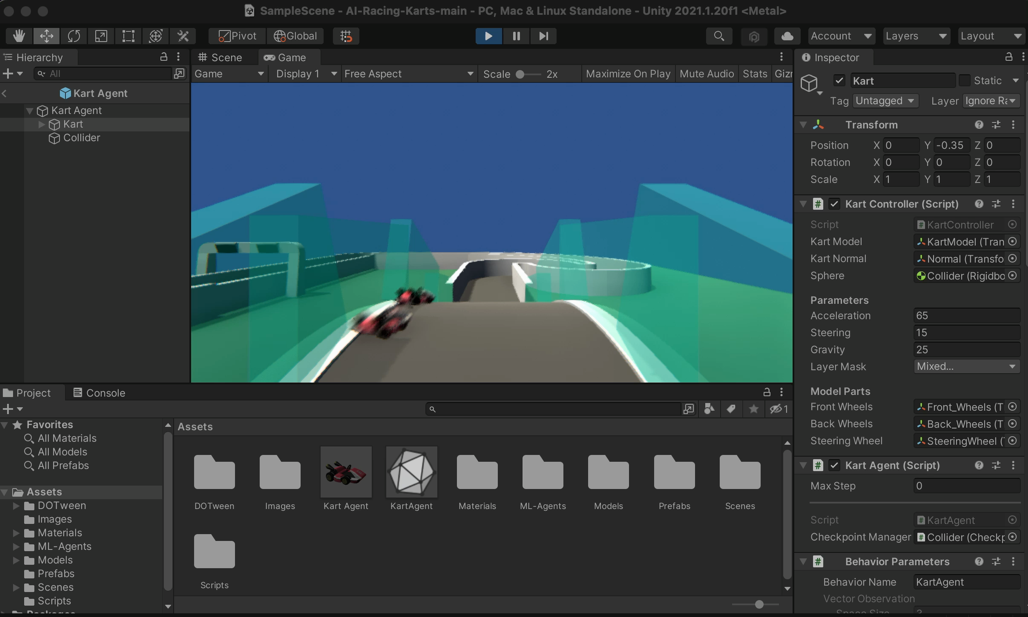
Description automatically generated

**Result videos**

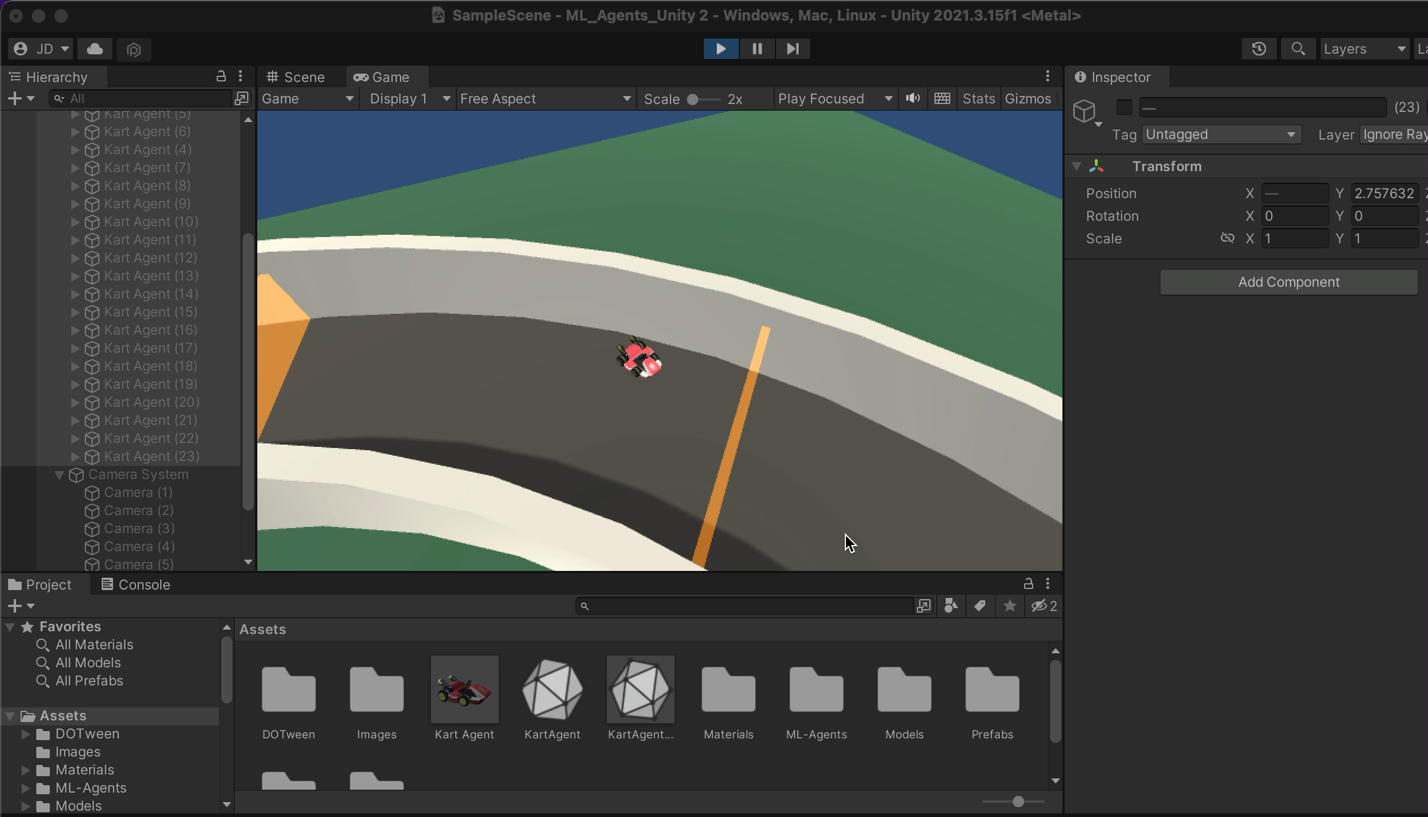
Below we can see how the kart agents initially started training



During one of the trainings, the karts even decided to run the track in reverse

****

After a 36-hour training model, we finally got a pretty good looking model. We can see in the video below that the kart uses the walls to break its speed when making sharp turns. To avoid this in the future I can set up negative rewards for every time the kart hits against the wall and see how it’ll train itself then. However, it seems to run the length of the racetrack in very good time.

****

**Text

Description automatically generated**

**References:**

1. <https://towardsdatascience.com/an-introduction-to-unity-ml-agents-6238452fcf4c>
2. <https://www.youtube.com/watch?v=i0Vt7l3XrIU&t=1s>
3. Kart - <https://github.com/mixandjam/MarioKart-Drift>
4. <https://www.kenney.nl/assets/racing-kit>
5. <https://adimen.si.ehu.es/~rigau/teaching/EHU/TAIA/Curs2021-2022/Homework1/G9_report.pdf>
6. <https://unity.com/products/machine-learning-agents>
7. <https://github.com/Unity-Technologies/ml-agents>