CSCI 527 : Applied Machine Learning for Games

Engineering Design Document

Jet Racing Bot

Fall 2020

Zaiyan Alam
Hanieh Arabzadeh
Hunny Vankawala
Raunaq Polwal

Table of Contents

1. Revision History

Date	Version	Description
09/15/2020	<1.1>	Initial document layout design & setup
09/30/2020	<1.2>	Update regarding the methodology to be incorporated
10/12/2020	<1.3>	Experimental results added along with mid-term updates
10/29/2020	<1.4>	Model optimized implementing Curriculum Learning
11/10/2020	<1.5>	Incorporated LSTM into the model increasing performance
11/20/2020	<1.6>	Final updates regarding the end submission

2. Objective

Our main aim is to design and develop a 3D racing game in which Human player vs multiple ML Agents scenario can be incorporated . The autonomous jet racing bot is trained as an opponent to a Human player through different world scenarios driven by a physics engine. We have published our game through a WebGL build so that it can provide a fun and challenging game-playing experience to the users.

Advent of Deep Learning techniques in the contemporary academia and industry highly motivated us to come up with this challenging game. It works at the intersection of Deep Reinforcement Learning methods like PPO and the real world physics which is exhibited within the game play. The same is further applied on generalized race tracks with copious amounts of obstacles for the ML agent to dodge. Thus, building a ML agent which displays such adaptability and flexibility given the complexity of tracks, highly motivated us to work on this game.

3. Background & Overview

3.a About

Our Jet-Bot game requests various aircrafts to rival each other in order to dominate the race. So as to make the game all the more testing and included we have advanced the aircrafts with abilities to upgrade their capacities in the event that they effectively evade certain impediments or fly through all checkpoints with no crashes. Aside from traditional flying activities, we have given the Boost capacity to the aircrafts so as to fly 'brilliant' not simply effectively. Coordinating the prize based learning not just makes the connection with the aircraft more instinctive yet in addition assists aircrafts with settling on advanced choices while exploring. Our Race-path and 3D environment is made out of specific structures on the green territory, which the ML operator must abstain from crashing into. There are various checkpoints which the ML operator must go through so as to dominate the race.

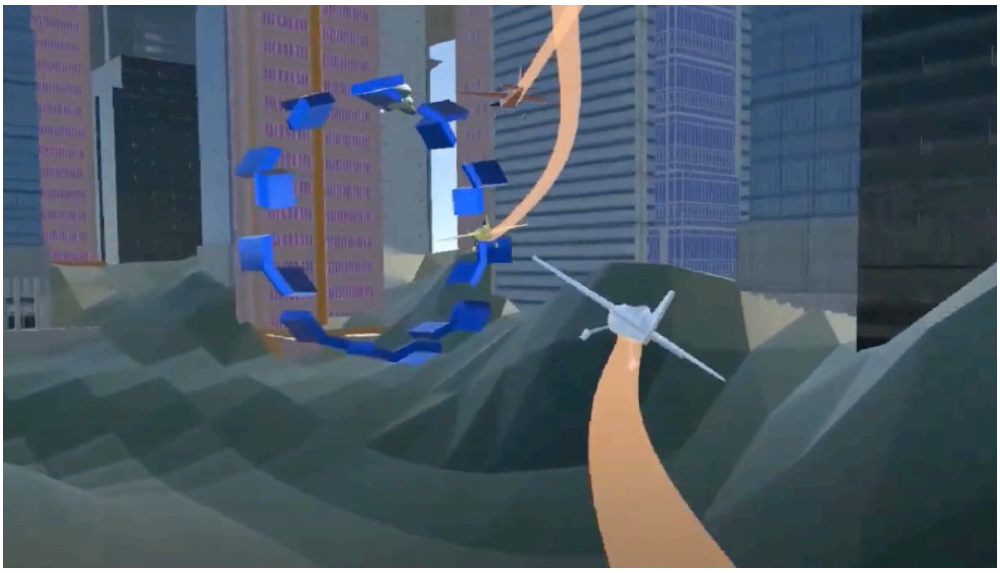

Figure.1 Game Overview

3.b Software Design

In order to live up to the expectations of Reinforcement Learning, many different environments are needed with varied complexity. This is where Unity comes into the picture. With built in toolkits like ML-Agents, it directly plugs itself into the engine and makes the task of building different environments easier. It complements the learning aspect and involves the same in the environment creation.

Developed by Unity Technologies, Unity ML-Agents is a new plugin for the game engine that helps the agents to train themselves or pre-developed or customized environments in one of the best game engines across the world.

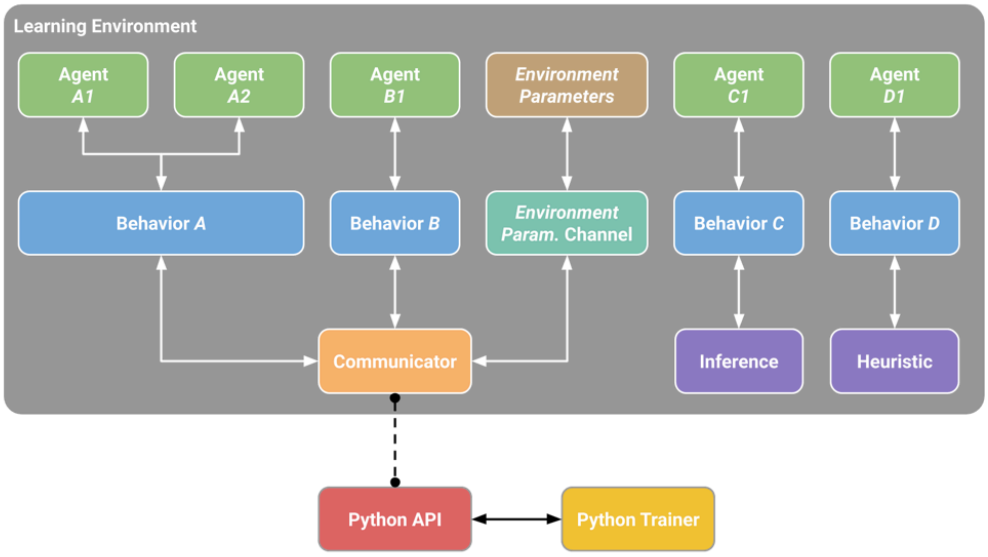

Figure.2 Unity ML Agents Design

Learning Environment —

This encompasses the Main Scene and the environment elements. It deals at the game level, so all of the Unity Components are considered in this particular component.

- Agent : The entertainer of the scene. He's him that we will prepare by enhancing his arrangement (that will mention to us what move to make at each state) called Brain
- Academy : This component coordinates specialists and their dynamic cycle. Think about this Academy as a maestro that handles the solicitations from the python API.

Python API —

It contains the RL calculations, (for example, PPO and SAC). We utilize this API to dispatch preparing, to test, and so on It speaks with the Learning climate through the outer communicator. *Tensorflow* connects through this API with the unity environment.

4. Prior Research Work

Myriads of different works had been performed in the domain of Deep reinforcement Learning. All of these laid a perfect framework for our execution, but their roots varied as most of them were aimed to perform course completion without navigating the obstacles. Discrete environment based work portrayed the capabilities of flying over the track with no response from environment factors like terrain effects and other obstacles that may arrive.

We perused through multiple research papers capturing the essence of PPO in context of Bot Modeling for a game, but all of them utilized the discrete environment algorithms which was designed to mitigate the problem at hand, but was limited by the fact that it used discrete action space.

The availability of Reinforcement Learning algorithms for training of agents in a dynamic gaming environment is rampant [4,5], with examples like *Deep Q-Network (DQN)* [7], *State Action Reward State Action (SARSA)* [6], *Soft Actor Critic (SAC)* [3], *Proximal Policy Optimization (PPO)* [10] and many more. The aforementioned algorithms exhibit several strong performance traits and weaker ones also. However, our game environment is a challenging one with continuous observations rather than discrete observations. Thus, SARSA and DQN fall short in this scenario as both are more suitable for discrete environments rather than continuous environments.

Henceforth, we need Policy Gradient methods as they are extremely good at handling continuous environment observations and will improve the performance of our game. previous work that had been done is heavily based on the single track mechanism. It holds true in its context, but performs poorly if deployed upon byzantine tracks with convoluted maps. The majority of the implementation was executed using agents performing concise actions at particular instances of time as directed by the action scripts.

5. Methodology

The proposed methodology in the paper uses Deep reinforcement learning algorithm in the subdomain of Policy Gradient methods along with a Deep Neural Net i.e Policy Network to train the aircraft agent. Our ***PPO-2*** model has LSTM and Curriculum Learning as additions so as to achieve better performance. There are certain actions assigned to the aircraft agent like Rotate left or right, Move left or right and Glide up or down. The aircraft agent's objective involves going through each checkpoint without colliding with the obstacles (like buildings and checkpoints) or the environment (the green terrain).

5.a Training Environment development

We used **Blender 2.8** to design the each environment component i.e Game object such as Buildings, Checkpoints, Terrain, Boundaries etc. We incorporated the *RigidBody* features with each one of the component in **Unity 3D** so as to make the environment like real world scenario with appropriate physics engine embedded in it. The whole approach makes the environment development more closer to the real world situation which helps our agent to learn better. From the figure below we can see our sample environment where the Race path is shown along with checkpoints and obstacles.

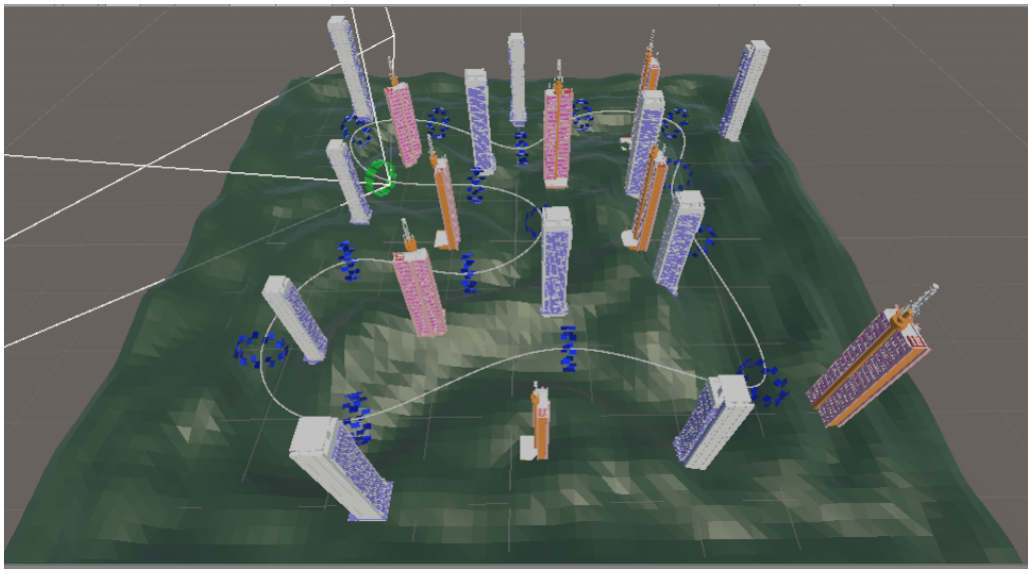

Figure.3 Training Environment

5.b Testing Environment development

We developed a separate Testing environment, which was basically an advanced version of the training environment, more complex and loaded with much more obstacles like bridges, towers etc. These new obstacles act as "**unseen**" components in the environment and provide a good measure of observing the agents behavior in a partially observed environment with new components. We used this environment for final play-testing the game and carrying out the main competing race between Humans vs ML agents.

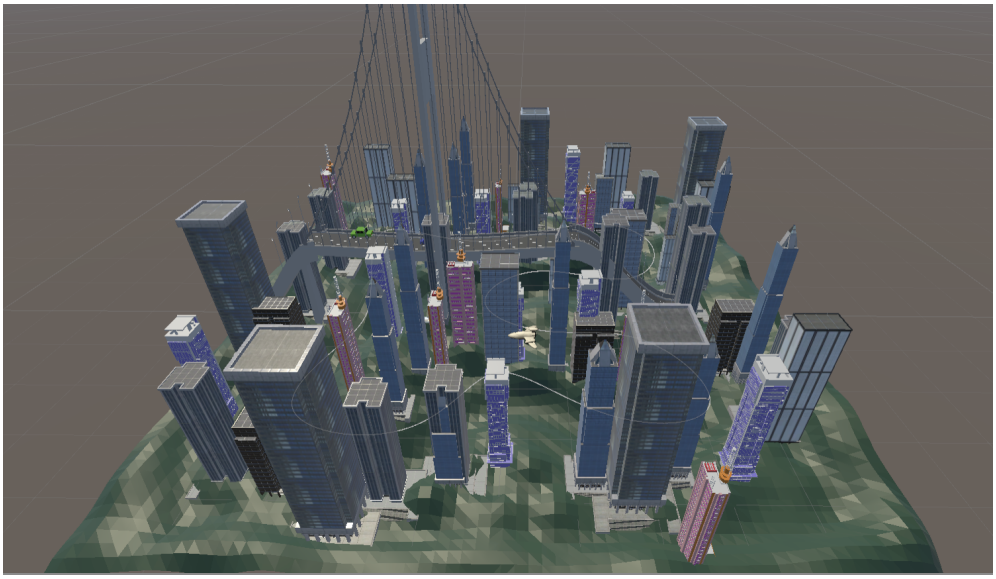

Figure.4 Testing Environment

5.c ML Agent Development

Designing the ML agent was a bit tricky task as we had to decide a lot on the dynamic behavior of the agent along with its core functionality related to movement. The challenge arises mainly because this agent performs actions in continuous environment, thus, every aspect needs to be carefully thought through.
We used **Blender 2.8** once again to design our agent and imported it in **Unity 3D** so as to develop its dynamic behavior. From the following figure we can see that **Ray-casts** are integrated in the agent so as to detect objects in it's path. They turn Red when they detect an obstacle, else they remain white.

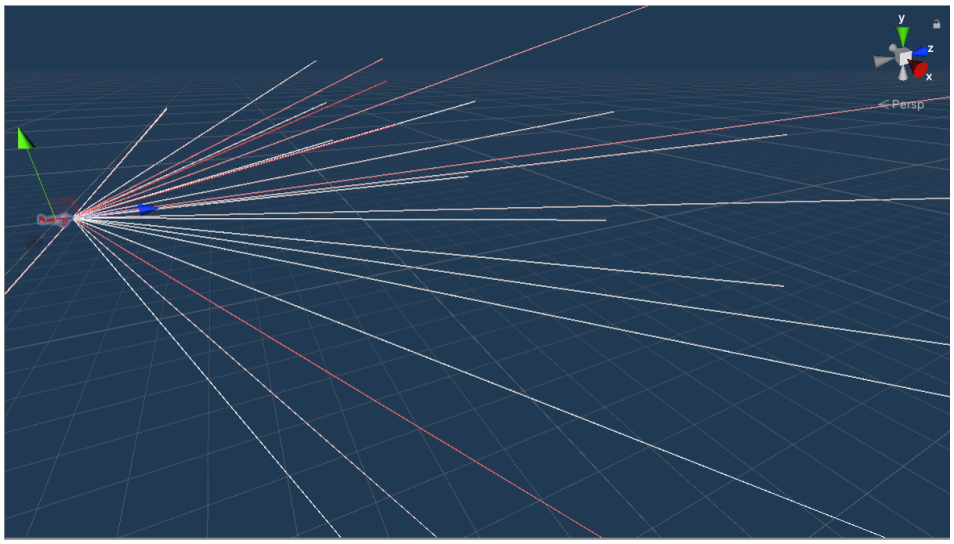

Figure.5 ML Agent

5.d Proximal Policy Optimization (PPO)

PPO algorithm is a policy gradient method which gives high performance regarding continuous action space and environment. The PPO algorithm uses Policy network, a Deep Neural Network (DNN, as shown in the figure) in order to optimize the objective function by running a number of epochs of Gradient descent over mini batches in order to give optimal Policy as output. Another great advantage of PPO is that it avoids large Policy updates by using a ratio that shows the difference between the Old vs New policy, thus improving performance and training experience.
The idea of policy gradient Surrogate Objective function was by taking gradient ascent steps, we make our agent understand to take actions that lead to rewards and avoid penalty. However, there are *2 issues* with this implementation. If we kept the step size too small, the training took too long, on the other hand, if we increased the step size, we faced too much variability while training.

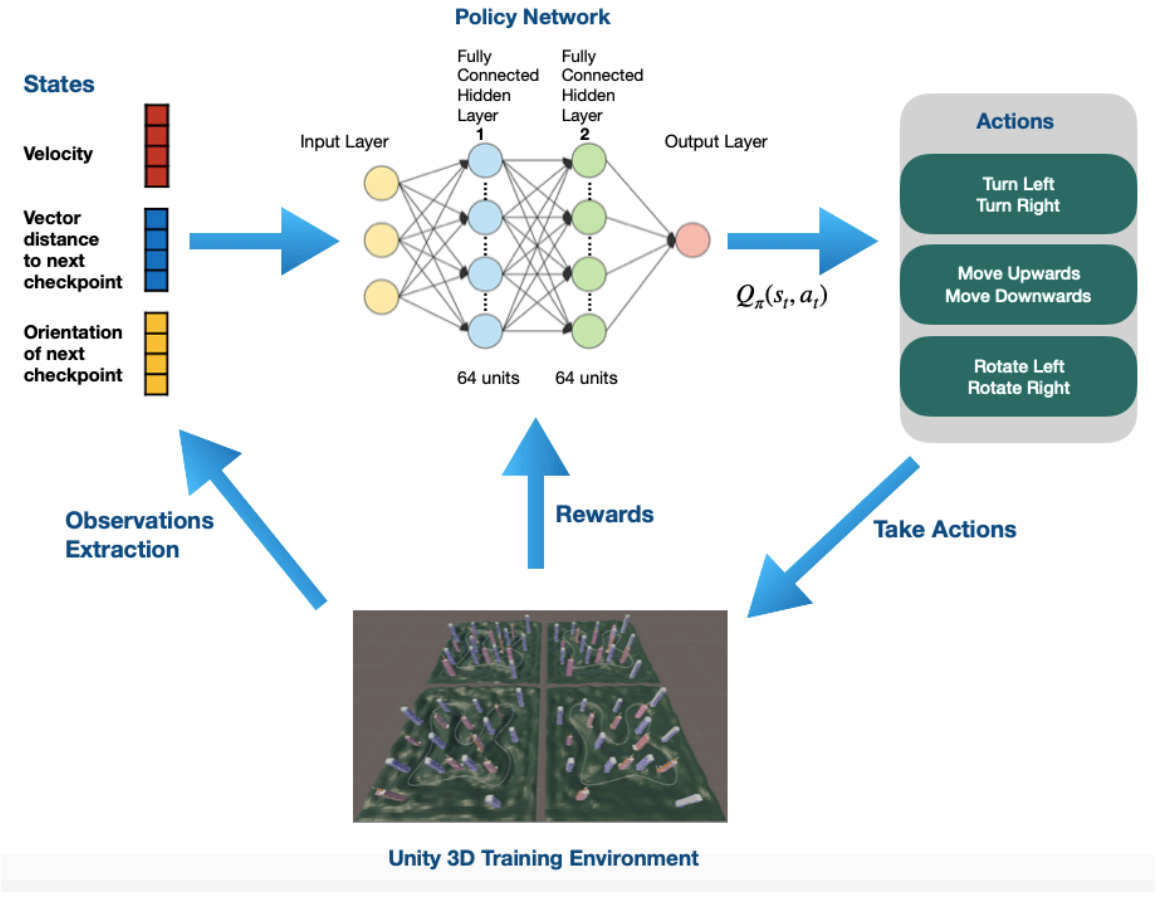

Figure.6 PPO Model Architecture

5.e Proximal Policy Optimization-2 (PPO-2)

PPO algorithm is an already advanced Deep Reinforcement Learning algorithm with significant performance. However, we observed it was not competitive enough to give a difficult and challenging game experience to the Human player. Thus, we incorporated two more methodologies in our baseline PPO model — **LSTM** and **Curriculum Learning**. Using these methodologies made the game more challenging, thereby giving our PPO model a significant performance boost and as a result we got the PPO-2 ML agent.

In comparison to the baseline PPO model, which exhibits Medium level of gaming agent, PPO-2 provided a great challenging Hard level of experience to the human player. In the following figure we can see the PPO-2 model architecture with LSTM layer acting as a feedback loop to the Policy Network.

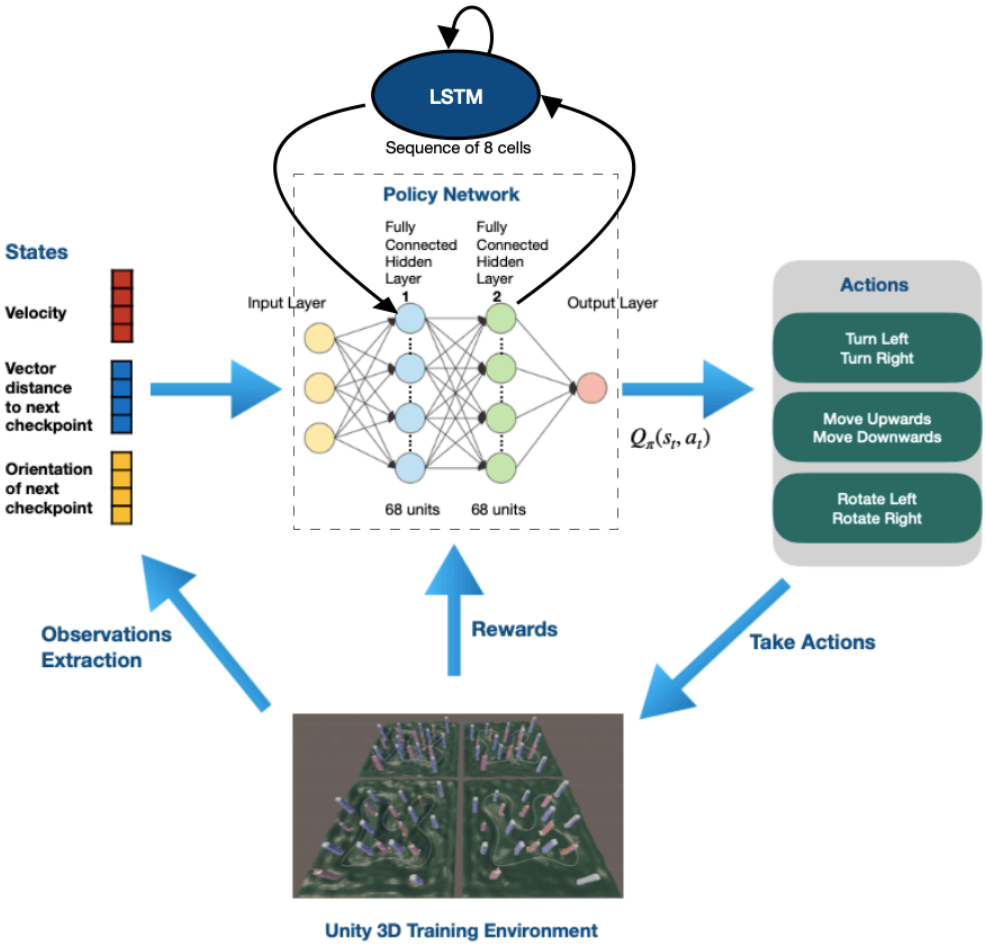

Figure.7 PPO-2 Model Architecture

5.f Long Short Term Memory Network

Reinforcement Learning is a Markov decision process which is memoryless and this proves to be an obstacle for our PPO agent which will step back and forth or circles. In order to overcome this obstacle, we included a feedback loop of LSTM recurrent connection between the 2 hidden layers of the Policy network in our PPO-2 model. An LSTM is a gated recurrent neural network capable of learning longer-term dependencies (sequences) making it ideal to provide a memory layer for the agent. LSTM networks are essentially made up memory blocks called cells which are responsible for remembering and memory manipulations that update the hidden state (memory).

As mentioned before LSTMs are gated so the cells can store or delete information by opening or closing the gates and that is how LSTMs are able to read, write, delete information from the memory. In our proposed PPO-2 model we have incorporated a **sequence of 8 cells**, which helps the agent remember past 8 steps of experiences in the decision making process and provide optimal Policy for a certain action. Now our agent must remember how to maneuver when a checkpoint arrives and it remembers what is the optimal way to make 'smart' turns which helps our agent to take the lead. Since LSTMs are recurrent, back-propagating the output errors helps the network to learn over time steps. The following figure shows a sample LSTM sequence of cells.

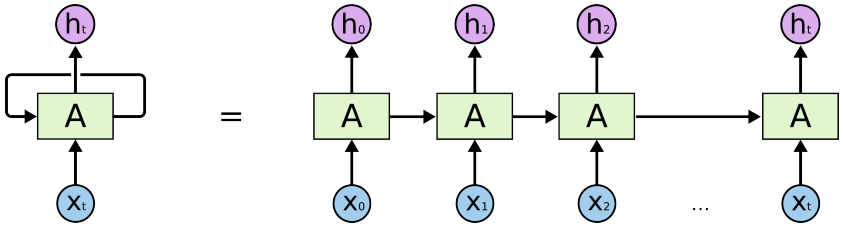

Figure.8 LSTM sequence of cells

5.g Curriculum Learning

To train our agent we have used adaptive curriculum learning known as “incremental curriculum learning” wherein each lesson needs to be trained for a certain number of iterations. However, we cannot put a finger on how many iterations that will be hence incremental curriculum learning allows the user to adapt the number of iterations for each lesson to optimize training. So incremental curriculum learning trains the network for a respecified iterations giving the user the flexibility to stop the learning early or train for more iterations if need be. Many metrics can be used to determine the epochs for each lesson which includes but not limited to loss, entropy or rewards. After analyzing all the metrics, we found that total cumulative reward generated the best model for navigation with our incremental curriculum learning.

In our model, when we reach a reward of 3 we move onto the next lesson. For example Lesson 2 starts when we have reached a total cumulative reward of 3 and we decrease the diameter from 55m to 50m in lesson 2. We repeat this process for **4 lessons**, until our agent is properly able to maneuver the checkpoint of diameter 40cm. After about 500,000 iterations, our PPO was able to learn a lesson by achieving a cumulative reward gain of +3 and then move onto the next lesson. In the following figure, we can see the decrement in the diameter of the checkpoints as the lesson moves forward from 1 to 4.

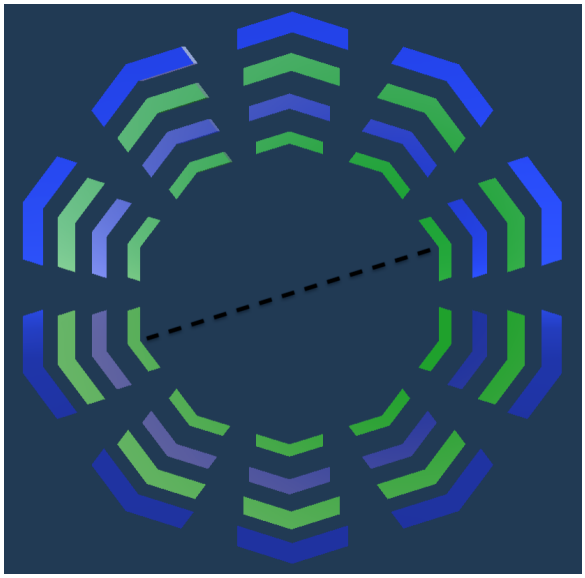

Figure.9 Checkpoint Diameter sizes through each curriculum

5.h States

In our gaming environment, a particular state is defined through a collection of observations :

- Velocity
- Vector distance to the next checkpoint
- Orientation of the next checkpoint.

Since the observations are made over a **3D** environment, a total of **9 observations** are made. These observations are saved into a vector format of 9 float values, at every **5 timesteps** (equivalent to **0.1 seconds**) and then fed into the neural network. **Raycasts** (over **300m** long) are also used in order to make observations regarding the objects around the ml agent which helps in object detection and collision avoidance.

5.i Actions

There are mainly **3** actions used by the agent with positive and negative values of each : **Rotate** left or right, **Move** left or right and **Glide** up or down. Thus the action sequence vector consists of **6 values** corresponding to each direction. The agent is also provided with the ability to use **Boost** in a certain direction with higher speed. If the agent collides with either of the buildings or terrain or goes out of boundary it explodes and re-spawns on the last visited checkpoint.

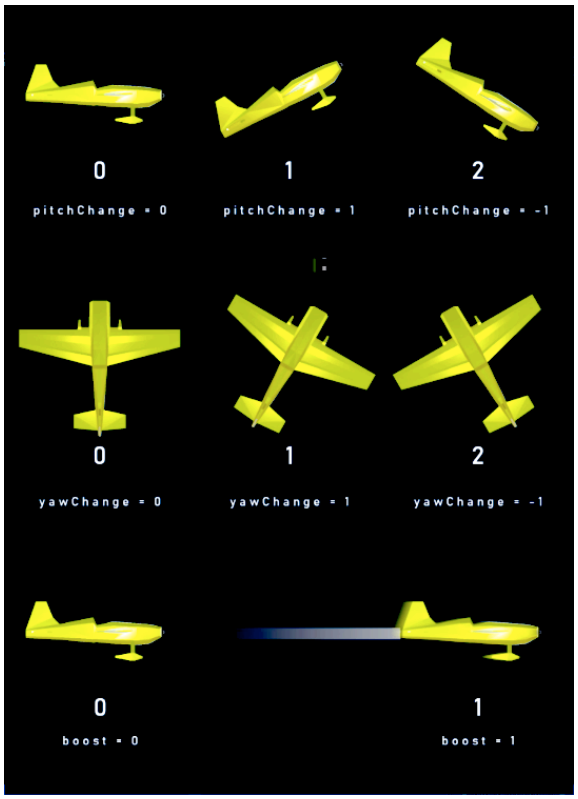

Figure.10 Agent Actions

5.j Rewards

The following reward system is used for training the agent :

(i) Small positive reward for passing through a checkpoint -

$$R(Checkpoint) = 0.5$$

(ii) Negative reward regarding collision with solid objects -

$$R(Collision) = -1$$

(iii) Small negative reward if the agent stays still and takes no action -

$$R(Stationary) = -1 / (Max\_step\_value) \quad [Max\_step\_value = 5000]$$

(iv) Negative reward if next checkpoint not reached within 300 timesteps -

$$R(Checkpoint\_timeout) = -0.5$$

Therefore, total cumulative reward -

$$R(Cumulative) = R(Checkpoint) + R(Collision) + R(Stationary) + R(Checkpoint\_timeout)$$

5.k Training

We used certain hyperparameters from Unity-ML agents config file in order to tune the PPO model performance and achieve better results. These hyperparameters were chosen to provide a balance between the performance and model complexity so as to avoid poor performance in case the model becomes too complex. We observed the training results on various hyperparameter values regarding the following ones :

batch_size = 512
num_hidden_units = 128
num_hidden_layers = 2
*buffer_size = 4*batch_size*
num_epochs = 6
sequence_length = 8
memory_size = 128
num_lessons = 4
minimum_episode_length = 100

From the following figure we can see the Training scene on Unity 3D, which has 4 simultaneously deployed environments. Each environment has 4 ML agents training on the designed race path and making observations in the learning process. The Ray-casts being used by the agent can also be seen clearly as they are navigating through the environment.

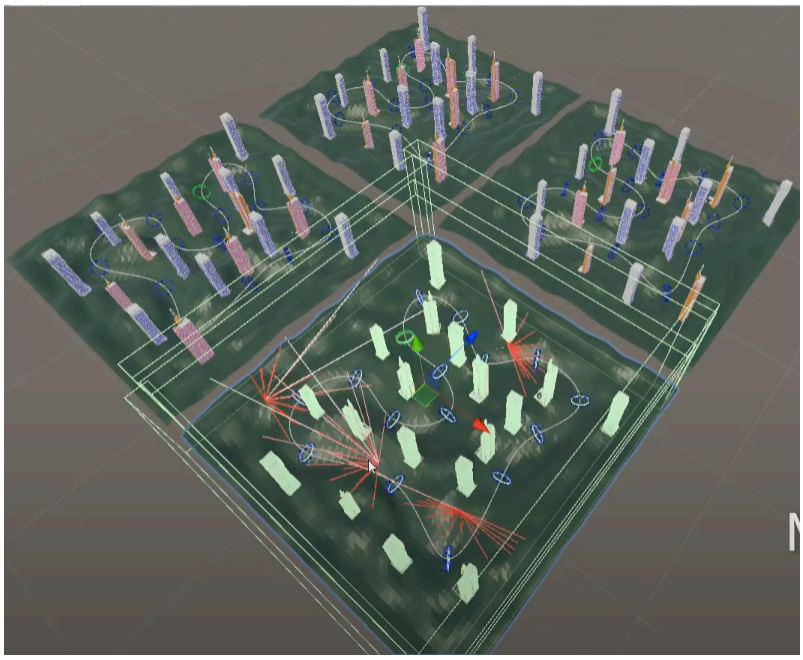

Figure.11 Training Scene

5.k Testing

We used the developed Testing environment, as mentioned earlier in order to compete the ML agents against the Human player. We performed multiple play-tests during which we observed the behavior of the ML agents competing against the human player. From the following figure we can observe the results of 10 races. It's evident that **PPO-2** wins the maximum number of races proving to be a great challenge against the Human Player. Also, PPO model gives okay performance coming third or second most of the times. We also used Simple RL model in order to give a base reference point to other ML - agents, and we can also see that it's performance is extremely poor coming fourth in all the races.

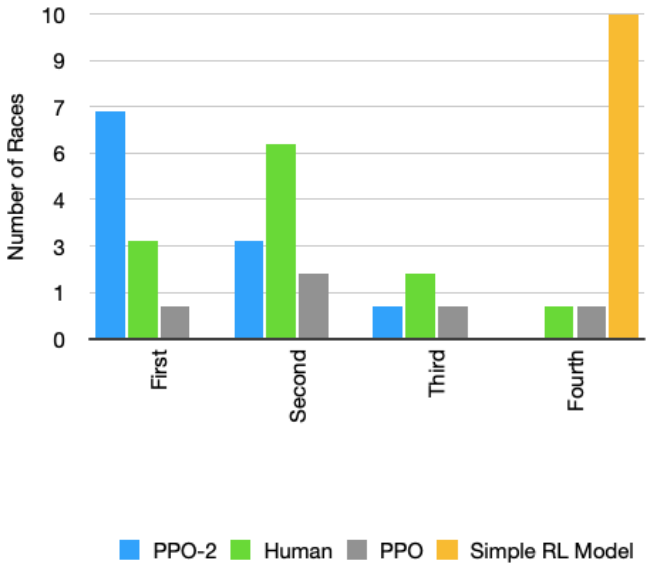

Figure.12 Play-testing results

6. Results

From figure 13 we can compare the baseline PPO model and our proposed PPO-2 model. It's evident from the plot that our **PPO-2** model performs better than the PPO as it attains highest ***cumulative_rewards = 18.2*** much faster, i.e within ***2 Million*** timesteps as compared to ***3 Million*** timesteps used by **PPO**. This higher performance by our PPO-2 model arises because of the memory power through LSTMs and optimized training through Curriculum Learning.

Figure.13 Cumulative Reward per Episode

From figure 14, we can see that PPO-2 model learns pretty quickly compared to PPO. The graph corroborates the fact that at any given iteration of training in early steps, say at ***600k***, the episode length of ***PPO-2 (310)*** was longer than ***PPO (170)***. Henceforth, our model performs better than the baseline PPO learning and exploring the environment pretty quickly. Therefore, PPO-2 ML agent gives a very strong competitive edge against the human, making it a challenging experience.

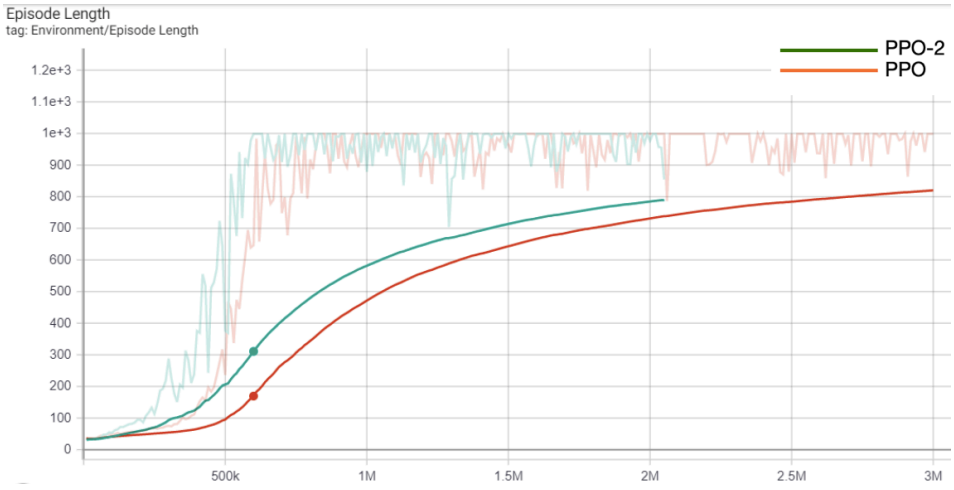

Figure.14 Episode Length Per Iteration

7. Conclusion and Future Work

During the course of this project we faced multiple challenges and came up with out of the box solutions in order to tackle them. One of the main challenges was the occurrence of repetitive behavior of the agent across one specific check point. This occurred due to same initial starting point of the ml agent in each episode, thus in order to fix it we randomized the initial starting point of the agent at the start of each episode.

Another risk we took was to make the training more penalty heavy than positive reward heavy, this is evident from aforementioned **Rewards** section where we have chosen 3 negative rewards rather and only one positive reward. For example, we could have chosen to give +0.5 reward to the agent to cover the distance between 2 checkpoints within 300 timesteps, but this would not have pressurized the agent to act on incentive and perform better. Thus, we chosen the other way around of giving a penalty of doing the above objective, and observed significant improved in the performance.

We have proposed a better model PPO-2 as compared to PPO, which can be used to give a Hard level gaming experience to the Human player. Moreover, in our experiment, we carried out competitive ML agents (PPO-2, PPO and Simple RL model) vs Human races where in around 80% of the cases PPO-2 won.Thus, we have shown that using multiple ML agents in a game with different underlying ML models gives a unique challenging experience. In future, work on experimenting how the behavior of other advanced Deep RL methods in the game can be observed.

7. References

1. The Simulation of Autonomous Racing Based on Reinforcement Learning

   https://etd.ohiolink.edu/!etd.send_file?accession=osu1523566974378793&disposition=inline

2. Continues Reinforcement Learning for playing racing games

   https://fse.studenttheses.ub.rug.nl/21333/1/bAI_2019_HolubarMS.pdf

3. A Survey of Deep Reinforcement Learning in Video Games

   https://arxiv.org/abs/1912.10944

4. PPO Dash: Improving Generalization in Deep Reinforcement Learning

   https://arxiv.org/abs/1907.06704

5. Recent Advances in Convolutional Neural Networks

   https://arxiv.org/pdf/1512.07108.pdf

6. Build your own RL environments w/ Unity ML agents

   https://medium.com/@marksaroufim/building-your-own-game-simulations-for-reinforcement-learning-with-unity-ml-agents-a-code-deep-e69a7bbc601e