Jet Racing Bot

Fall 2020

Zaiyan Alam
alamm@usc.edu

Hanieh Arabzadehghahyazi
arabzade@usc.edu

Hunny Vankawala
vankawal@usc.edu

Raunaq Porwal
raunaqra@usc.edu

Abstract

The goal of this paper is to summarize our findings while training a game playing agent for our Jet-Bot racing game. We have deployed a Deep Reinforcement Learning policy gradient method, Proximal Policy Optimization (PPO) with a Deep Neural Network (DNN) i.e Policy Network to train our ML agent (aircraft) to fly and dynamically interact with the 3D environment. The training is done through a system awards and penalties to provide the agent incentives in order to make decisions and complete the navigation in optimal way.

I. Objective

We aim to develop a Human player vs ML Agent 3D racing game. The autonomous jet racing bot is trained as an opponent to a Human player through different world scenarios driven by a physics engine. We hope to publish our game in future through a WebGL build so that it can provide a fun and challenging game-playing experience to the users.

II. Motivation

Jet-Bot Racing game burgeons heavily from the Human vs AI games genre. Thus, this factor along with the growing Deep Learning methodologies aspect 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.

III. Background

Our Jet-Bot racing game demands multiple aircrafts to compete with each other inorder to win the race. In order to make the game more challenging and involved we have enriched the aircrafts with capacities to enhance their capabilities if they successfully avoid certain obstacles or fly through all checkpoints without any collisions. Apart from conventional flying actions, we have provided the Boost capability to the aircrafts in order to fly 'smart' not just correctly. Integrating the reward based learning not only makes the interaction with the aircraft more intuitive but also helps aircrafts make optimized decisions while navigating.

As shown in the figure below, our Race path and 3D environment is composed of certain buildings on the green terrain, which the ML agent must avoid colliding with. There are a number of checkpoints which the ML agent must pass through in order to win the race.

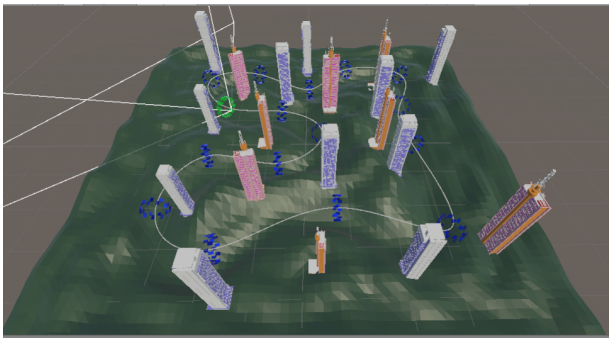

Fig. 1. Race Path

IV. Related Work

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.

V. Design Approach

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. 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).

A. *Proximal Policy Optimization*

As mentioned earlier 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.

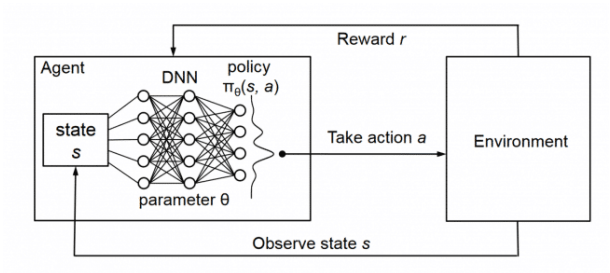

Fig. 2. PPO

B. *States*

In our gaming environment, a particular state is defined through a collection of observations : velocity, vector distance to the next checkpoint and 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.

C. *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 respawns on the last visited checkpoint.

D. *Rewards*

The following reward system is used for training the agent:

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

$$R(Checkpoint) = 0.5 \tag{1}$$

(ii) Negative reward regarding collision with solid objects -

$$R(Collision) = -1 \tag{2}$$

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

$$R(Stationary) = -1/(Max\_step\_value = 5000) \tag{3}$$

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

$$R(Checkpoint\_timeout) = -0.5 \tag{4}$$

Therefore, Total cumulative reward :

$$R(Cumulative) = R(Checkpoint) + R(Collision)$$

$$+R(Stationary) + R(Checkpoint\_timeout) \tag{5}$$

E. *Results*

We used certain hyperparameters from Unity-ML agents config file in order to tune the PPO model performance and achieve better results. 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* and *num_epochs = 6*.

These parameters 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.

Furthermore, we can observe from figure 3 that the Red line corresponds to training over 3 million steps and the Blue line regarding 6+ million steps. It's evident from the figure that the cumulative_reward increment is drastically climbed in the first 500k steps and by 1M steps the agent has sort of peaked it's learning experience with cumulative_reward = 18.43 . Moreover, it's evident that from 3M to 6M steps that the agent's cumulative_reward stays in the range of 16 to 19, thus exhibiting that agent has exploited most of the learning experience.

From figure 4, we can see that the learning rate of the agent decreases linearly with the timesteps which signifies that the agent has learned most of the environment and exhausted most of the observations. One major factor to be considered over here is the fact that the environment is simple, deterministic and static.

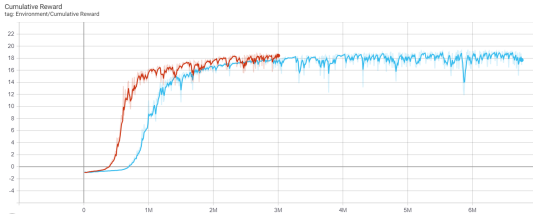

Fig. 3. Cumulative Rewards per Episode

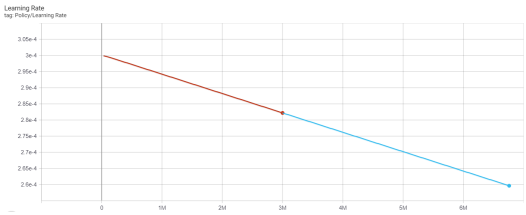

Fig. 4. Learning Rate

VI. Future work

For our final work we will incorporate Curriculum Learning over a no. of lessons in order to make our aircraft agent more efficient and 'smart', in the sense that it knows when to accelerate and decelerate etc. For Curriculum learning we will use more complex environments with non-deterministic and dynamic state rather than static. We will implement the Human player feature and publish a WebGL build so that users can compete with our ML agent and have fun.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
[2] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.
[3] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: arXiv preprint arXiv:1809.02627 (2018).
[4] Kai Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: IEEE Signal Processing Magazine 34.6 (2017), pp. 26–38.
[5] Yuxi Li. "Deep reinforcement learning: An overview". In: arXiv preprint arXiv:1701.07274 (2017).
[6] Gavin A Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
[7] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: Nature 518.7540 (2015), pp. 529–533.
[8] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: arXiv preprint arXiv:1509.02971 (2015).
[9] John Schulman et al. "Trust region policy optimization". In: International conference on machine learning. 2015, pp. 1889–1897.
[10] John Schulman et al. "Proximal policy optimization algorithms". In: arXiv preprint arXiv:1707.06347 (2017).
[11] Yoshua Bengio et al. "Curriculum learning". In: Proceedings of the 26th annual international conference on machine learning. 2009, pp. 41–48.
[12] Tambet Matiisen et al. "Teacher-student curriculum learning". In: IEEE transactions on neural networks and learning systems (2019).