IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 10, Issue 3, March 2022

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.165

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

Racing Car AI-powered by Deep Reinforcement Learning with Dynamic Difficulty Adjustment for Video Games

Manthan Tupe, Sarvesh Dalvi, Kshitij Khandkar, Prof. Hasib Shaikh

UG Student, Dept. of Computer Engineering, KCCEMSR, University of Mumbai, Thane, India

Assistant Professor, Dept. of Computer Engineering, KCCEMSR, University of Mumbai, Thane, India

ABSTRACT: Demand for computer games has been increasing every year in the entertainment sector. With such increasing demand, it becomes necessary to create engaging and competent artificial intelligence. Traditional hardcoded artificial intelligence in video games often results in monotonous and predictable behaviours. As deep learning-based solutions are often encouraged. Therefore, a model has been developed by utilising Deep Reinforcement Learning and Imitation Learning technologies to create human-like behaviours. The model also scales difficulty based on the player's skill level.

KEYWORDS: Unity, ML-Agents; Deep Reinforcement Learning; Imitation Learning; Artificial Intelligence; Game Development; Dynamic Difficulty Adjustment.

I. INTRODUCTION

Artificial intelligence is becoming more crucial in our daily life. It has been used to automate tasks, help to make better decisions, to solve complex problems[10]. New artificial intelligence methods have been developed to meet this new demand. These methods started as a solution to a single narrow problem but now the goal has shifted to creating autonomous intelligent behaviour. One interesting use case for these new methods is to create intelligent agents in video games.

Reinforcement learning is a promising machine learning method that has been rising in popularity in the video game industry in recent years [9]. The reinforcement learning tools such as the Unity machine learning toolkit have become more available for game developers [4]. Reinforcement Learning has been used to train artificial intelligence to play games such as Chess and Go [1]. The machine learning toolkit makes it possible to train artificial intelligence for games using powerful and modern reinforcement learning algorithms. The purpose of this study is to implement such algorithms into a Racing Car Agent. The main reinforcement learning algorithm used in the study rewards the agent for completing tasks in the environment. With the main reinforcement algorithm, we use the Imitation learning method such as GAIL to help the agent to learn quicker[3]. Imitation learning functions by helping the agent to discover the rewards from the environment by providing the agent rewards for following a demonstration. The study aims to utilise imitation learning methods to increase the training speed of reinforcement agents.

The Unity Machine Learning Toolkit is an open-source add-on package to the Unity game engine. It enables the developers to use game environments to train artificial intelligence for games. These agents can be trained using reinforcement learning to perform tasks in the game. Unity ML-agents toolkit was our choice because it makes reinforcement learning more accessible to game developers. In the dense reward environment, the agent gets rewarded often by the training environment. Our goal was to use a combination of extrinsic and intrinsic rewards to accelerate the training process. The results from this study could help game developers utilise reinforcement learning more effectively, saving time during the training process. The results from this study showcase the use of reinforcement learning and imitation learning methods to build agents that can successfully traverse through a procedurally generated environment while dynamically adjusting difficulty according to the player's performance [6]. Further assists have been added on top of the DL model as a failsafe to ensure smooth working of Agents even in cases of catastrophic failure.

II. RELATED WORK

A comparative study of simulating autonomous driving using various platforms was given here [7]. After selecting Unity we referred to their official research paper [4] and documentation to get better insights on how to utilise different algorithms.

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

A car driving environment using MLAgents was studied while comparing various algorithms to train the driver agent [1]. From this study, we concluded which algorithms are effective for training in the least amount of time.

Algorithms such as Proximal Policy Optimization [5], Random Network Distillation [2] and Generative Adversarial Imitation Learning [3] were studied to implement our working model. Further, we created a custom difficulty adjustment system based on experiences and good practices mentioned here [6]. We also studied what techniques were used by other developers to set up their tracks and related gameplay elements [8].

III. METHODOLOGY

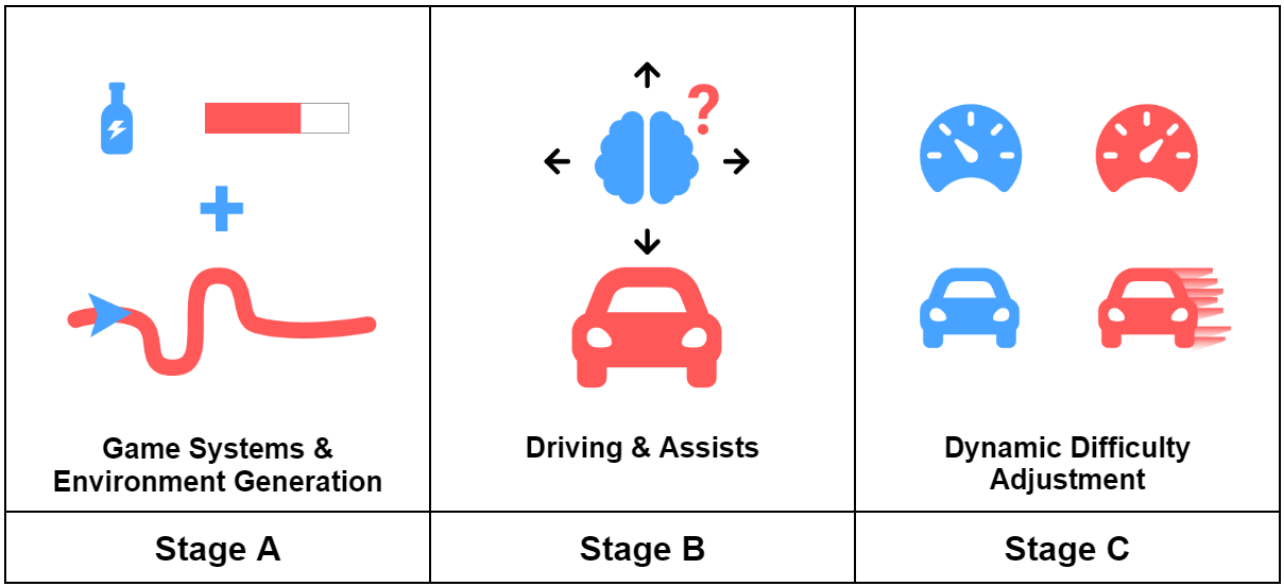

Fig.1. Diagram indicating different stages of the model.

A. *Environment Generation and Game Systems:*

Level Design is a very important part of Game Development. Level Design requires a lot of skill and experience but also comes with drawbacks such as being time consuming for people who aren't designers. This is why a modular procedurally generated road system has been developed. We use tilesets of seven distinct road chunks that result in seamless procedurally generated roads. This is a rule based system which results in sprint tracks that generally end towards positive Z direction in Unity. We use unity's scriptable objects to retain the rules for spawning a chunk and then can spawn it at runtime or before level loads.

Other Game systems setup include:

Arcade Racing Car Controller which relies on simple sphere physics instead of WheelColliders. Various Checkpoints which are made using a trigger component and get destroyed when in contact [8]. Time based power up like NOS is also implemented in similar ways as many popular games. Chase & Free Cameras are toggleable at will to get a better view of the car during runtime. Other than that quality of life features such as Minimaps, HUD and Menu Systems are also implemented using unity.

A. *Driving and Assists:*

In any racing game the NPCs should be able to drive effectively and be able to traverse the entire track. For this purpose MLAgents Toolkit in Unity Engine was utilised to train a model capable of traversing a track. The track is set up with checkpoints which when triggered execute AddReward() which gives the agent a positive reward. Similarly there are walls surrounding the track which the agent is supposed to stay away from and will get penalised if gets in contact with it. It was necessary to find a relatively fast and efficient algorithm for training our model. Thus we decided to use Proximal Policy Optimisation (PPO) as our Reinforcement Learning Algorithm [5]. PPO is an easy-to-use algorithm and relatively stable because it is protected from destructive large policy updates from gradient ascent by constraining policy update maximum change to a smaller range of $[1 - \varepsilon, 1 + \varepsilon]$.

The objective function penalises the policy changes which move the rt (θ) away from 1. The

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

main objective function in PPO is:

$L\ CLIP(\theta) = \hat{E}\ t\ [min(rt\ (\theta)\ \hat{A}\ t\ ,\ clip(rt\ (\theta),\ 1 - \varepsilon,\ 1 + \varepsilon)\ \hat{A}\ t\ ]$

- ε is a hyperparameter, (usually 0.1 or 0.2).
- Êt is the expectation over timesteps.
- rt is the probability ratio of new and old policies.
- t is the estimated advantage at time t.
- θ is the policy parameter.

Additionally we used Random network distillation (RND) to improve the rate of exploration of our agent [2]. RND introduces an exploration bonus to reinforcement learning methods. The bonus is the error of a neural network (NN) feature prediction of the observations from a fixed randomly initialised NN. The method is designed to tackle the sparse rewards problem by acting as a directed exploration method. The agent is rewarded for novel experiences of interacting with the environment or exploring new areas. Unlike most curiosity methods the RND method is not susceptible to getting stuck when observing random noise on the TV for example. This is due to the method using exploration bonus instead of absolute prediction error. The RND is a promising algorithm that is suitable for agents who need to explore complex environments with lots of noise in the collected observation data.

Further to kickstart the training process we utilised the Generative Adversarial Imitation Learning (GAIL) algorithm [3]. GAIL is a model-free imitating algorithm that offers performance gains compared to earlier model-free methods in complex environments. Because the method is model-free it must learn from the environment, and it needs more interaction with it than model-based methods. The method explores actions randomly to find which of the actions allows the policy to shift towards the expert's policy. GAIL allows agents to learn behaviour by observing the expert's behaviour without the knowledge of the reinforcement signal. The expert behaviour is recorded in a demonstration file containing states and actions. The framework can extract behaviour directly from the provided example.

In case our DL model fails we have implemented a validator component that keeps track of velocity of the vehicle. Since on collision or in case of low confidence inference the validator will detect failure based on a specific threshold value. When this threshold is reached we discard all actions from the DL model and switch to a traditional hard coded AI that can still complete the race safely so players won't be disappointed.

B. *Dynamic Difficulty Adjustment:*

Difficulty in videogames is subjective. But it has been observed that there should be a balance between too hard and too easy [6]. The NPC should be able to behave in challenging ways but also not too punishing. In order to achieve this we implemented our own dynamic difficulty adjustment system. Our system uses the distance between the player's car and agent to determine which state to trigger. As of right now we have three distinct behaviour states that result in three difficulty levels. We buff or nerf agents' performance by multiplying speed stats with a multiplier depending on behaviour. This system can further be expanded with even more behaviour.

IV. RESULTS - SCREENSHOTS + ACCURACY STATS + GRAPHS

We trained our model for a total of 5 Millions steps using combinations of different algorithms and hyperparameters. Our training was divided into five consecutive training sessions on 1 Million steps each. After every 1 Million step we initialised the previous model and trained it further. It took 5 hours of training per million steps on an Nvidia 940mx GPU paired with Intel i5 7200U. Around 5 Millionth step our Cumulative Reward reached around 22.

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

Fig.2. Splash Screen.

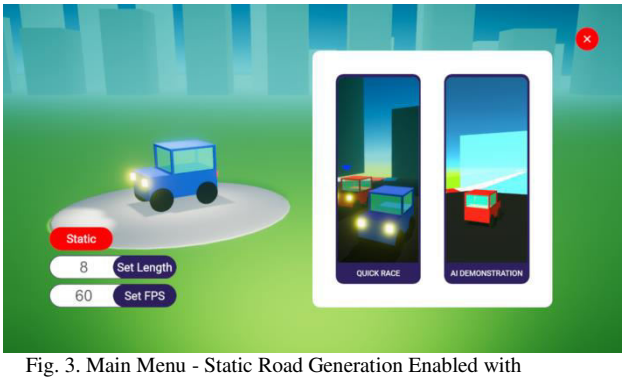

Fig. 3. Main Menu - Static Road Generation Enabled with Track Length set to 8 chunks.

Fig.4. Main Menu - Random Road Generation Enabled with Track Length set to 10 chunks.

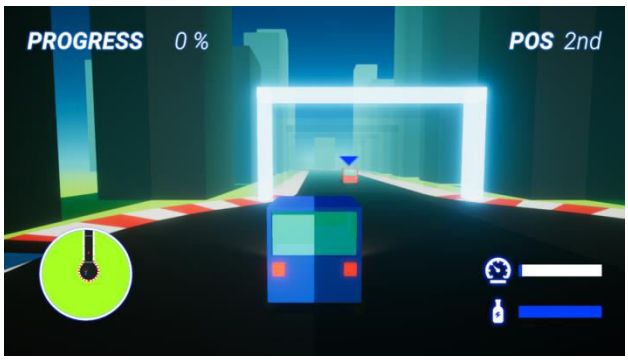

Fig. 5. Quickrace Scene showcasing UI and various Gameplay functionalities.

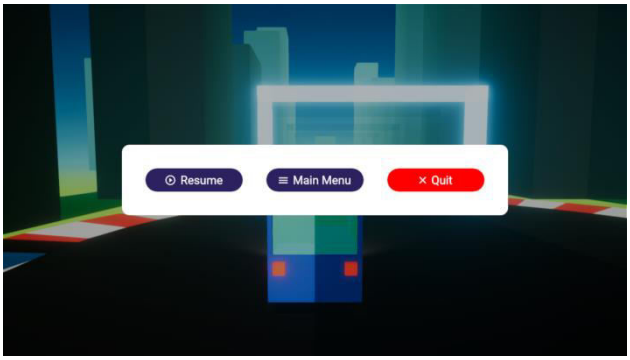

Fig.6. Pause Menu.

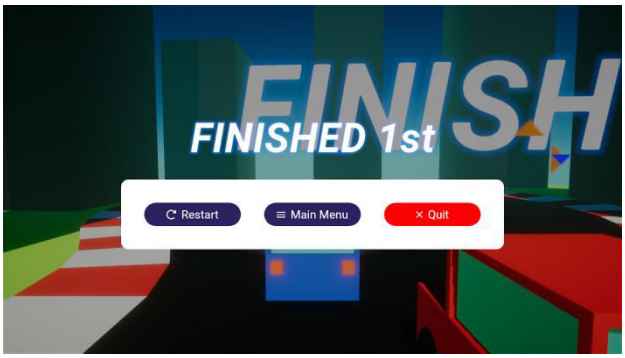

Fig. 7. Race Finished Screen.

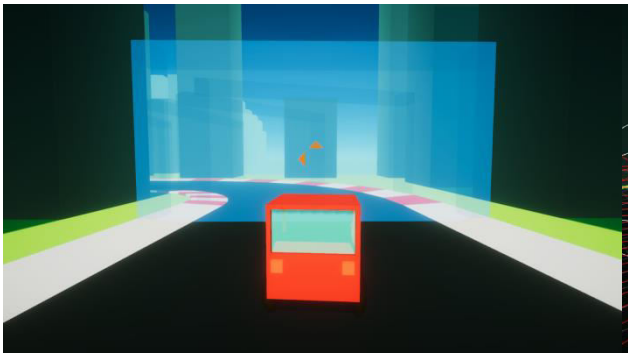

Fig.8. AI Demonstration Scene.

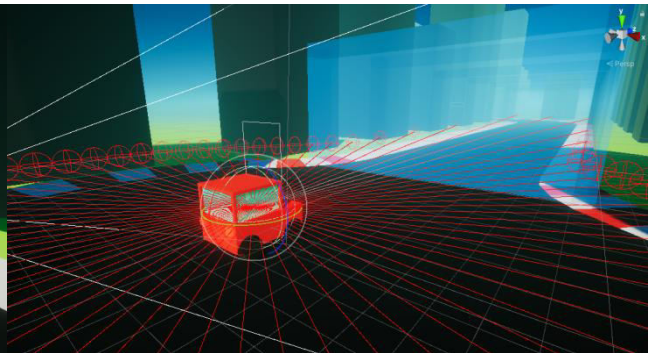

Fig. 9. Agents Perception Sensors and Scene setup.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

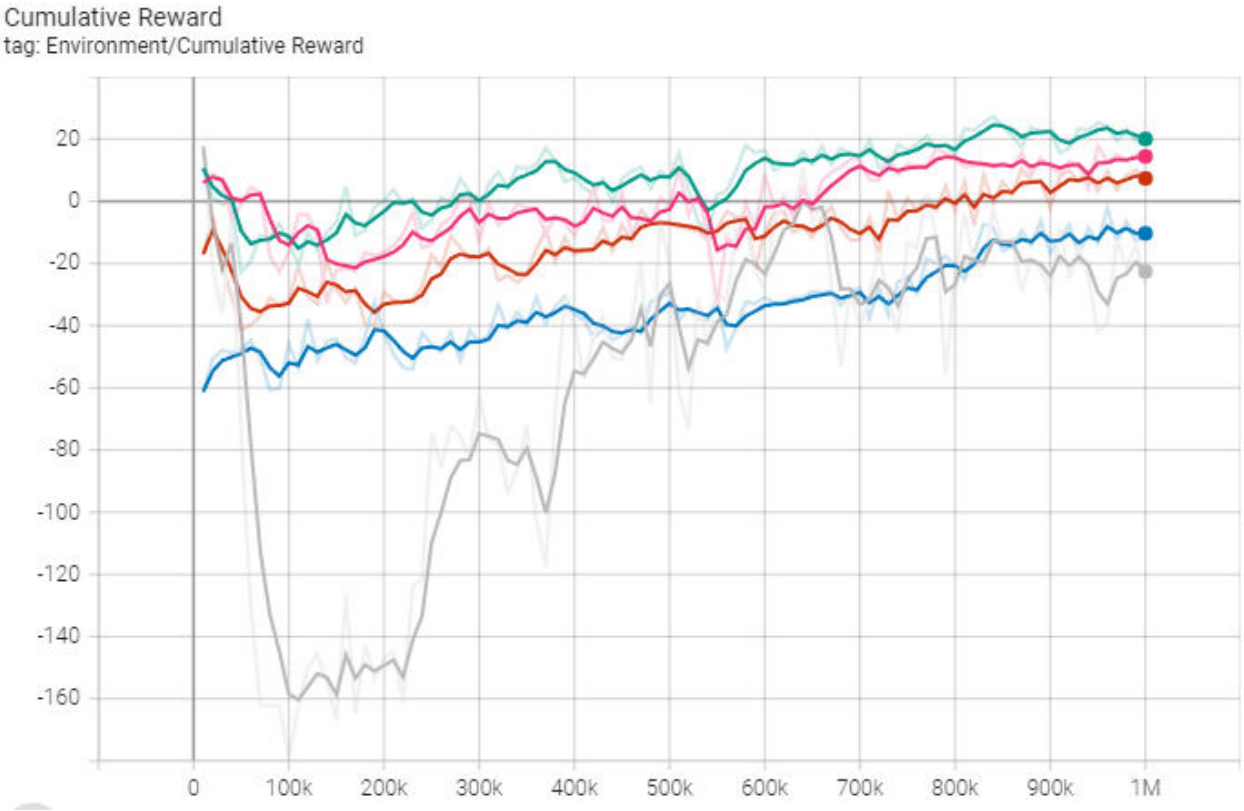

Fig.8. Maximisation of Cumulative Reward in every training run.

- BLUE - Second Run (PPO + RND)
- RED - Third Run (PPO + RND)
- PINK - Fourth Run (PPO + RND)
- GREEN - Fifth Run (PPO + RND)

We gauge that accuracy of our model on a static track is around 72% while on procedurally generated tracks is very low around 4%. This is to be expected with extremely limited training data and computation power.

V. CONCLUSION AND FUTURE WORK

We proposed a method of implementing Racing Game AI systems which utilise deep learning methodologies with dynamic difficulty adjustment features.

Further, we propose the use of a system that acts as a failsafe when the deep learning model fails.

We are also looking forward to training on more complex road environments generated using spline-based solutions, implementing object avoidance systems and a more simulation-based vehicular controller in future builds.

We hope our project will be a helpful addition to the arsenal of Indie Game Developers who don't necessarily have the infinite computing power for Deep learning but want to create competent AI behaviour on a race track like environment.

REFERENCES

1. Jari Hanski, Biçak and Kaan Baris,'An Evaluation of the Unity Machine Learning Agents Toolkit in Dense and Sparse Reward Video Game Environments', UPPSALA Universitet, May 2021.
2. Yuri Burda, Harrison Edwards, Amos Storkey and Oleg Klimov,'Exploration by random network distillation', ICLR 2019 Conference , Sept 2018.
3. Jonathan Ho and Stefano Ermon,'Generative adversarial imitation learning', NIPS'16, December 2016.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165 |

|| Volume 10, Issue 3, March 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1003087 |

4. Juliani, Arthur & Berges, Vincent-Pierre & Vckay, Esh & Gao, Yuan & Henry, Hunter & Mattar, Marwan & Lange, Danny. (2018). Unity: A General Platform for Intelligent Agents.
5. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms.," CoRR, vol. abs/1707.06347, 2017.
6. Zohaib, Mohammad. (2018). Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. Advances in Human-Computer Interaction. 2018. 1-12. 10.1155/2018/5681652.
7. Huynh, Anh-Tuan, Ba Nguyen, Hoai-Thu Nguyen, Sang Vu and Hien D. Nguyen. "A Method of Deep Reinforcement Learning for Simulation of Autonomous Vehicle Control." ENASE (2021).
8. Chan, Marvin & Chan, Christine & Gelowitz, Craig. (2015). Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques. International Journal of Computer Games Technology. 2015. 1-6. 10.1155/2015/839721.
9. W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), 2011, pp. 1143-1146, doi: 10.1109/MEC.2011.6025669.
10. C. Shetty, A. Khan, T. Singh and K. Kharatmol, "Movie Review Prediction System by Real Time Analysis of Facial Expression," 2021 6th International Conference on Communication and Electronics Systems (ICCES), 2021, pp. 1109-1113, doi: 10.1109/ICCES51350.2021.9489171.

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor: 8.165

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details