International Journal of Scientific Research & Engineering Trends
Volume 5, Issue 3, May-Jun-2019, ISSN (Online): 2395-566X

Training an Agent to Play a Game using Reinforcement Learning

Harsh Chhablani **Mohit Karangiya**
Dept. of Computer Science & Engg.
hchhablani96@gmail.com karthicknanmaran@gmail.com
SRM Institute of Science and Technology
Chennai, India

***Abstract*- In this project, we have made a racing game and also train an ML agent to play it, at the end of which it has learnt how to avoid collision with obstacles along the path and reach a goal destination. The resultant AI can then be used to implement intelligent transport systems, driverless cars and assign variable speed limits to the driver for efficient fuel usage. The ML agent was trained by using virtual sensors and raycasting methods using the Unity Engine which recently has acquired integration with Machine Learning.**

***Keywords* – Machine Learning, Artifical Intelligence, Reinforcement Learning, Video Games, Unity**

I. INTRODUCTION

Video games, initially were thought as a means of entertainment but now have been used as platforms for education. Games like Doom are being used to train the military and there are various simulators available which help students in finessing their abilities in the areas of chemistry, surgery. Also, the video game industry has grown exponentially over the last few years with the introduction of eSports, mobile gaming and online streaming. This type of growth also demands frequent release of products from game developers. AI or Artificial Intelligence is one of the most complex elements in any video game.

The reason being that hard coding its behavior to make it serve as a perfect character (an opponent or an ally or a basic NPC) in the video game. It takes a lot of time if a person decides to hardcode the behaviour as a lot of conditions are to be taking in care. This brings us to the concept of machine learning. Machine Learning is a type of learning where we can train agent(s) to behave and work as we want them to. So far, implementation of machine learning has been very small as it required setting up individual environments and the most popular game engines like Unreal Engine or Unity lacked an integration.

This was until mid-2018, where Unity released a new package in the form of an environment known as Unity-ML Agents. This environment allows developers to train various entities or agents to behave accordingly using reinforcement learning. The algorithm used is called PPO or Proximal-Policy- Optimization. This algorithm was released by OpenAI and makes use of a policy and with every iteration refines it to make it more efficient. Normal reinforcement learning requires a lot of fine-tuning factors and variables which take a lot of time and sometimes break the training. PPO makes it easier to tune these variables. We use a typical racing game for our project. The player and the AI opponent have to reach a goal before the other whilst avoiding certain obstacles. This sort of AI can also be used to make Intelligent Transport Systems on which our Literature Survey is about.

II. PURPOSE

The purpose of this project is two-fold.

1. Game Development

Use of machine learning can shorten the development cycle of video games exponentially. The process will be much easier to implement as there will be no need to hardcode the behavior and everything else will come naturally. Major game development studies like Ubisoft are already working on implementing machine learning to video games.

2. Intelligent Transportation Systems

Intelligent Transport Systems have been introduced in many countries and different continents and have resulted in fuel saving and efficiency, prevention of accidents and to avoid traffic jams and choke points. The AI at the end of third phase can be used as a base model for an advanced AI to create simulations and implement the system in real life.

III. METHOD

In this section we describe the general algorithm and methodology used.

1. PPO Algorithm

Natural policy gradient involves a second-order derivative matrix which makes it difficult to scale for

large scale problems. The computational complexity becomes too high for real world tasks. Intensive research is done to reduce the complexity by approximate the second-order method. Proximal Policy Optimization (PPO) uses a slightly different methodology. Instead of imposing a hard constraint, it formalizes the constraint as a penalty in the objective function. By not avoiding the constraint at all cost, we can use a first-order optimizer like the Gradient Descent method to optimize the objective.

2. Optimization

There are two optimization methods

2.1 Line Search

Line Search methods picks the steepest direction from the current location on the curve and moves forward towards it by a step size. If the step size is too small, it will take forever to reach the peak, and if the step size is too large, we can go down the cliff. Policy gradient is mainly an on-policy method. It searches actions from the current state. Hence, we resume the exploration from a bad state with a locally bad policy. This hurts performance badly.

2.2 Trust Region

Trust region is calculation of the maximum step size or the area in which we may want to perform the line search first. Also, the size of the trust region can be dynamically changed as per the requirements.

IV. MACHINE LEARNING IN VIDEO GAME DEVELOPMENT

Machine Learning can be used in video game development to ease the process and reduce the development cycle of the game so as to meet deadlines quickly and more efficiently. [11] Unity, a free game engine has recently integrated machine learning to its engine which allows anyone and everyone to train and implement an agent. This is the engine that we have used for our project.

V. SETTING UP THE ENVIRONMENT

We began setting up the environment. We downloaded the package provided, installed Anaconda Terminal and integrated TensorFlow with Unity. All this was done by referring the document provided in Unity ML's GitHub repository. We then decided to test the environment by using basic examples like the game of Soccer, Tennis, etc

VI. TRAINING BASIC MOVEMENT

We then decided to start with our actual project and start training our agent for basic movement. Initially it was first decided to give it a positive reward for reaching the goal but realized the agent was not having a clear idea as to which direction the agent had to go, so to fix the issue we assigned it a positive reward for moving closer to the goal.

Table 1 Initial Reward System

Action	Reward
Reached Goal	+100.0
Fall Off	-100.0
Move closer to goal	+0.0003

VII. COLLISION AVOIDANCE

We trained the agent to avoid a certain obstacle. We observed that if we assigned a negative reward for colliding with an obstacle, it was still getting a positive mean reward as it was still reaching the goal and not acknowledging individual actions. Hence, we called a reset function called Done() to end the current phase training so that it was getting a negative mean reward.

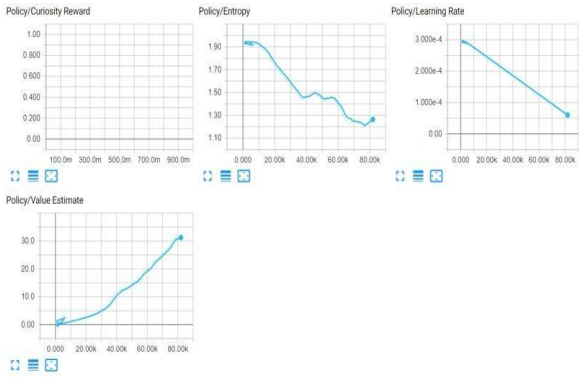

Fig. 1 Multiple graphs showing Policy vs Curiosity Reward, Entropy, Learning Rate, Value Estimate of initial training.

Table 2. Improvised Reward System

Action	Reward
Reached Goal	+500.0
Reached Border	-0.5
Reached Obstacle	-5.0
Fall Off	-100.0
Move closer to goal	+0.0003

We can see that entropy and learning rate are decreasing and the mean value is increasing with successive policy which is feasible. We are not using a curiosity variable and hence that graph has no data.

VIII. MAKING THE GAME

Next, we began making the final game, adding menus, sounds, making models of different cars, etc. We also refined the reward system and created an actual level. We observed that as the complexity of the level increased with addition of more obstacles, the training took a long

time to finish due to our limited computational power. We overcame this by using Nav Mesh surfaces to give aid in collision avoidance.

Table 3. Final Reward System

Action	Reward
Reached Goal	+500.0
Reached Border	-0.5
Reached Obstacle	-5.0
Avoiding Obstacle	+0.1
Fall Off	-100.0
Move closer to goal	+0.0003
Move away from goal	-0.1

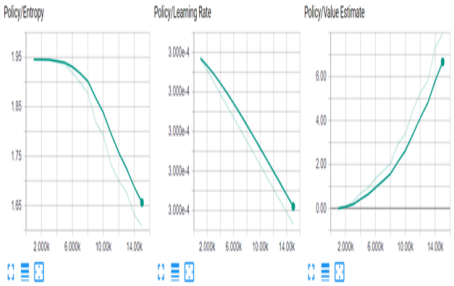

Fig.2 Tensor Flow graphs of the final training showing Policy vs Entropy, Learning Rate, Value Estimate.

IX. HOW IT WORKS

The agent is controlled by a brain which has common parameters like the state space and action space. The agent has a Ray Perception component which emits a raycast of a distance and at angles defined by us. This helps it collect observations and differentiate as to what is a goal and an obstacle by tags and Detectable Objects assigned by us. The training is done using a config file which has parameters like max steps, batch size, etc. The movement is done by the Agent Action function which initially, tries to move in random directions and then see which is gaining a bigger reward. All the data, the progress, etc is shown in the Anaconda Terminal out of which a graph can be created.

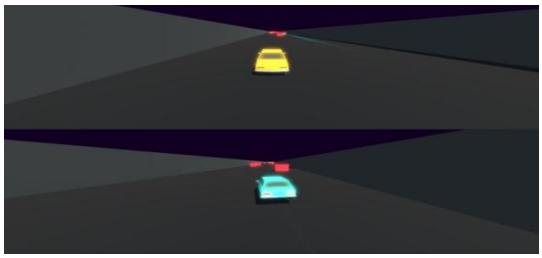

Fig 1. Player competing agaisnt training AI to avoid obstacles and finish first.

X. LIMITATIONS

Reinforcement Learning depends mostly on the environment, so change the environment and you have to re-do all the training. One way to overcome this, can be to implement path finding algorithms or have a heuristic brain to make the decisions. In our case, if we had to change an obstacle's location then we'll have to repeat the training process. This somewhat restricts us to only static environments which are fixed by us, the developers which is not a bad thing since the environments are created by us and are not randomly generated.

XI. CONCLUSION

The final game 'Speedsters' was successfully made using Unity Engine. The AI was competing extremely well with the player and avoiding obstacles as expected. We even put different difficulty level based on the training it received and other attributes like acceleration, etc. We added menus, music and gave the game a visual appeal and polished the final build. However, the training is still static meaning it won't work for dynamic environments. To counter this, one can either use a heuristic brain which decides on its own with the help of some hard coding or to implement other algorithms such as a path finding algorithm to aid the training and in turn, the AI.

REFERENCES

1. C Perera, S. . D. (2017). "Applying intelligent speed adaptation to a road safety mobile application – Driver Safe Mode." 10.1109/ICTER.2017.8257790.
2. C. Wang, J. L. and G. Teng, S. Chang, Y. Z. (2007). "Study on intelligent speed adaptation impact of driving safety based on simulation." Second International Conference on Innovative Computing, Information and Control (ICICIC 2007).
3. Carsten, O. and Tate, F. (2005). "Intelligent speed adaptation: accident savings and cost benefit analysis, accident analysis & prevention." Volume 37, Issue 3,2005,Pages 407-416,ISSN 0001-4575.
4. EA. "Seed - https://www.ea.com/seed".
5. Ebot and Booysen (2016). "Auditory intelligent speed adaptation for long-distance informal public transport in South Africa." IEEE Intelligent Transportation Systems Mag- azine. 8. 10.1109/MITS.2016.2533979.
6. Gamez Serna, C. and Ruichek., Y. (2017). ""Dynamic speed adaptation for path tracking based on curvature information and speed limits.." Sensors 17.6 (2017): 1383.
7. Lai, F. and Magnus Hjalmdahl, Kathryn Chorlton, M. W. (2010). "The long-term effect of intelligent speed adaptation on driver behavior." Applied Ergonomics, Volume 41, Issue 2, 2010, Pages 179-186, ISSN 0003-6870.
8. Open AI (2017). "Ppo (proximal policy optimization) https://openai.com/blog/openai- baselines-ppo/.

9. Qureshi & Abdullah, H. (2013). "A survey on intelligent transportation systems. middle-east journal of scientific research." Middle-East Journal of Scientific Research. 15. 629-642. 10.5829.
10. Stephan (2014). "The effectiveness of an advisory intelligent speed adaptation (isa) system for victorian repeat speeders." Australasian Road Safety Research Policing Education Conference,
11. Stephenson, J. (2018). "Ways machine learning will be used in game development.
12. Sven Vlassenroot, S. B. (2007). "Driving with intelligent speed adaptation: Final results of the belgian isa trial, transportation research Part A: Policy and Practice." Volume 41, Issue 3, 2007, Pages 267-279, ISSN 0965-8564.
13. Ubisoft. "Machine learning in Ubisoft."
14. Unity Technologies. "Unity ml - https://unity3d.com/machine-learning".
15. Zhao, Y. (2017). "Green drive: A smart phone-based intelligent speed adaptation system with real-time traffic signal prediction." 2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS).