ISSN No:-2456-2165

AI Soldier using Reinforcement Learning

Simran Bhojwani
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Sagar Ganiga
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Harsh Jain
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Shravan Jain
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Riya Karia
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Nausheen Sayed
Student, Department of Information Technology
Vivekanand Education Society's Institute of Technology, Mumbai

Jayashree Hajgude
Assistant Professor, Department of Information Technology
Vivekanand Education Society's Institute of Technology
Mumbai

Abstract:- This paper describes the implementation of Reinforcement Learning algorithms on FPS Games by applying AI Agents. The steps undertaken to build an FPS Game using various game engines and reinforcement learning are outlined. The game consists of player AI and opponent AI competing with each other in which the goal is to kill the opponent. The algorithm has been tested on different maps of vizDoom Opensource Library.

Keywords:- *Reinforcement Learning, FPS Game, ViZDoom, Unity ML Agents, A2C, Q-Learning.*

I. INTRODUCTION

➢ *First Person Shooter Games*

As per Glavin and Madden[7], First Person Shooter games are famous game genre wherein the player experiences the perspective of a character (the first person) and is involved in a competitive, three-dimensional environment. In this paper, the main concern is to develop a bot that can play any match with the main objective to defeat the other players in the Environment.

➢ *Reinforcement Learning*

Reinforcement learning is a powerful sort of Machine Learning algorithm where the agent figures out how to work in a situation by performing activities and watching the results. The Action Space is a list of all potential actions that can be performed by the agent and the list of all states is known as the state space representing the agent's view.

A feedback is given to the agent depending on the action which it has performed and in which state. The feedback might be a reward (positive points) or a penalty (negative points). State-action pairs are stored by the policy of the learner. These represent how useful it is to carry out a particular action in a given state.

II. RELATED WORK

There exist various inquiries about methodology choice in FPS recreations as of late. For example, the bot modeling in Modelling a Human-like Bot in a First Person Shooter Game[5] considers several parameters to be considered & list of weapons used along with their utility. They proved that E-Bot outperforms the standard bots, even in the hardest difficulty level. The work by McPartland and Gallagher[4] embraces the neural systems for their bot so as to improve the choice of fight methodologies by impersonating human techniques. The best choice to create bots in an FPS game is to use Interactive Learning.

III. OUR PROPOSED SOLUTION

A. Platform

We have experimented with different game engines for developing the testing environment of our algorithms. Below is the list of different game engines which we have analyzed.

➢ *Unreal Engine*

The Unreal Engine is developed by Epic Games. With its code written in C++, the Unreal Engine features a high degree of portability and is a tool used by many game developers today as per Tremblay, Schneider and Cheston[8]. Unreal provides with its own scripting language so that we can quickly integrate our AI models with the built environment. These scripts are high level and simple to use. Plugins can be used with the Unreal Scripts to integrate the AI and ML.

➢ *Unity*

The Unity Editor features various tools that allow fast editing and iteration in your development cycles, including Play mode for quick previews of the work in real-time("Products-Unity", n.d.)[9].

ISSN No:-2456-2165

➢ *OpenAI Gym*

This python library gives a huge number of test environments to work on our RL agent's algorithms with shared interfaces for writing general algorithms and testing them as evident in the works of Rana[10]. OpenAI gym provides the current state details of the game means the environment. It also handles when to do an action which we want to perform based on the current state/situation.

➢ *VIZDOOM*

VIZDOOM facilitates testing AI bots that operate DOOM using the visual data. It is intended for research in machine visual learning and deep reinforcement learning. VIZDOOM is based on ZDOOM, the most popular modern source-port of DOOM. This means it is compatible with a large range of tools and resources that can be utilized to custom scenarios.

B. Environment

➢ *Weapon Selection*

In this progression, the incapability of every weapon the bot is holding is determined and the one with the smallest value is chosen. Such a value is determined by the supreme estimation of the contrast between the separation to the foe and the best scope of a weapon of intrigue. However, the pistol is so much ineffective than the other weapons that it is excluded from consideration.

➢ *Shooting and Moving*

In the end, the bot shoots and moves as per the decisions made before in the past dimensions. At the point when bot loses perspective on its adversary, in the event that it has ammunition for its weapons, other than for the essential gun, and the health level over 30, it will pursue the rival to the last observed spot; otherwise, its state will be changed to the item-collection state.

➢ *Aim-Point Determination*

On a very basic level, the bot goes for the present position of the rival. In any case, there is some randomness introduced in this so as to restrain the bot from winding up excessively powerful and not working as a human player. This adds to the rival's x, y, and z-pivot positions some arbitrary numbers produced corresponding to the separation between a bot and the enemy.

➢ *Procedure Selection*

The system technique is to keep the bot moving in the challenge. On the off chance that it only from time to time stops or moves with no arrangement amid the fight, its rival can make a decision on the minute that it is a bot. Then again, if the bot moves with a fitting system, it is not really conceivable to be recognized as a bot by the rival. In this manner, the bot must choose an appropriate procedure. The bot chooses one from the accompanying four systems: pick the proximal weapon, pick the proximal curative item, approach the adversary, and move far from the rival. Table 1 shows how the technique is characterized.

Situation	Strategy
The bot does not have ammunition for its weapons.	Pick the nearest weaponry.
Bots' strength is under 80.	Pick the nearest health package.
The distance from the opponent is less than 100.	Approach the opponent.
Otherwise.	Move away from the opponent.

Table 1:- Strategy for Situation

IV. ALGORITHM

A. DQN

As mentioned by Yu[11], Q Learning is a traditional reinforcement learning algorithm first introduced in 1989 by Walkins. It is a widely used off the shelf method that is usually used as a baseline for benchmarking reinforcement learning experiments. Q-Learning is an RL technique that is aimed at choosing the best action for given circumstances (observation). This implementation exercise allows us to have a firmer grasp and more intuitive understanding of Q Learning and Policy Gradients. Each possible action for each possible observation has its Q value, where 'Q' stands for the quality of a given move. In practice, we usually use a deep neural network as the Q function approximator and apply gradient descent to minimize the objective function LL. A close variant called Double DQN (DDQN) basically uses 2 neural networks to perform the Bellman iteration, one for generating the prediction term and the other for generating the target term. This will assist in reducing the bias added by the errors of Q network at the starting phase of training.

inputs:
S : set of states
A : set of actions
γ : discount
α : step size
internal state:
real array Q[S,A]
previous state : s
previous action : a
begin
initialize Q[S,A] arbitrarily
observe current state s
select action a using a policy based on Q
repeat forever:
carry out an action a
observe reward r and state s'
select action a' using a policy based on Q
$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a] - Q[s,a])$
$s \leftarrow s'$
$a \leftarrow a'$
end-repeat
end

B. A2C

Actor-Critic models are a common form of Policy Gradient model. The Actor-Critic model is an optimized score function. Instead of waiting till the termination of the episode as we do in Monte Carlo REINFORCE, we make an update at each step (TD Learning).

Input: a differentiable policy parameterization (a|s,)
Input: a differentiable state-value parameterization $\hat{v}(a|s,\boldsymbol{\theta})$
Parameters: step sizes $\alpha\boldsymbol{\theta} > 0$, $\alpha w > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}d'$ and state-value weights $w \in \mathbb{R}d$

Repeat forever:
Initialize S (first state of the episode)
$I \leftarrow 1$

While S is not terminal:
$A \sim (.|S,)$
Take action A, observe S', R
$\delta \leftarrow R + \gamma\hat{v}(S',w) - \hat{v}(S,w)$ (if S' is terminal, then $\hat{v}(S',w) = 0$)
$w \leftarrow w + \alpha w I\delta\nabla w\hat{v}(S,w)$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\theta} I\delta\nabla\boldsymbol{\theta}\ln(A|S,\boldsymbol{\theta})$
$I \leftarrow \gamma I$
$S \leftarrow S'$

❖ *Implementations*

We started with unreal and later shifted to unity for better python support for building ML agents. The steps undertaken during the development are:

➢ *Requirement Gathering for an AI Bot in an FPS Game*

We contacted many gamers to understand and gather enough information about the current usage of AI in modern FPS game. We also discussed the requirements of playing arena to make it more interesting for the players.

➢ *Implementation of AI Algorithm in Python*

We started our algorithm implementation in python and trained our model on 2D graphics of OpenAI gym environments. We trained thousands of episodes on different 2D maps to get better accuracy along with different learning rates. Along with this we also tested our model with other Reinforcement Algorithms like SARSA and A3C.

➢ *Testing AI Algorithm in Doom Game on vizDOOM Platform*

We tested our trained model on different maps and environments of Doom Game in the vizDoom platform. This allowed us to test the accuracy of our implementation and resolve the issues involved in strategic planning and better the implementation. At the end of testing, we were able to play our AI agent on different maps over the DOOM game.

V. RESULTS

A. DQN

Below is the mean reward performance chart of DQN over 5,000 episodes on VizDoom Basic Map.

As we can see from the graph, DQN initially explores the environment and takes random action from the q_table. But after some time of training, it learns to perform better and starts exploiting the q_table for getting good performance continuously. It shows that DQN takes less experimental steps and focuses more on exploiting the q_table.

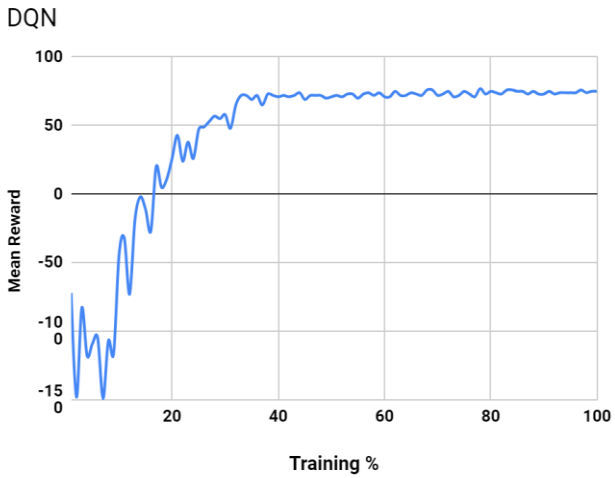

Fig 1:- DQN Performance

B. A2C

Below is the mean reward performance chart of A2C over 10,000 episodes on VizDoom Basic Map.

As we can see from the graph, A2C initially explores the environment and takes random action from the q_table. But after some time of training, it learns to perform better and starts exploiting the q_table for getting good performance continuously. It shows that A2C takes more experimental steps and focuses more on frequently updating the q_table.

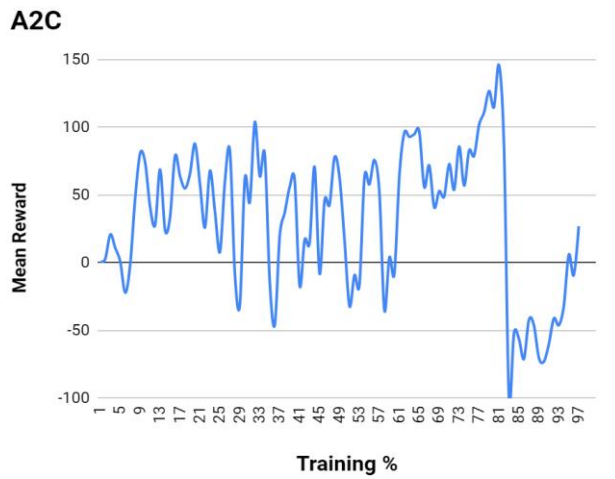

Fig 2:- A2C Performance

VI. CONCLUSION

In this paper, we presented the use of Reinforcement Learning to train an AI Soldier to play FPS Games. We tested two different Reinforcement Learning Algorithms named DQN and A2C to train the AI Soldier to move and shoot in a partially observable environment of VizDoom and to observe its behaviour. The aim of the AI Soldier is to kill the enemies and survive as long as possible and based on its performance, it is given awards. We observed that DQN algorithm provided the best result once it found the optimum strategy it continued using it to exploit the enemy's weakness. Whereas in A2C algorithm, the agent didn't stopped at optimum strategy and continued exploring for other strategies. The output of DQN shows that it is better algorithm for exploiting the weakness whereas A2C is better algorithm for exploration. We also tried testing these algorithms on Unity and Unreal environment but failed to get result as it was too heavy and lacked native python support. These results shows that Reinforcement Learning is the best learning method for playing FPS games.

REFERENCES

[1]. Amar Bhatt, "Teaching Agents with Deep Apprenticeship Learning", 2017
[2]. Partha Sarathi Paul, Surajit Goon, Abhishek Bhattacharya, "History and comparative study of modern game engine", International Journal of Advanced Computer and Mathematical Sciences, Vol 3, 2012.
[3]. Yong Ding, "Research on Operational Model of PUBG", SMIMA, 2018
[4]. Michelle McPartland and Marcus Gallagher, "Game Designers Training First Person Shooter Bots", 2012.
[5]. A.M. Mora, F. Aisa, P. García-Sánchez, P.A. Castillo, J.J. Merelo, "Modelling a Human-like Bot in a First Person Shooter Game", 2015
[6]. Sule Yildirim, "Serious Game Design for Military Training", 2010.
[7]. Frank G. Glavin, Michael G. Madden. "DRE-Bot: A hierarchical First Person Shooter bot using multiple Sarsa(λ) reinforcement learners" , 2012 17th International Conference on Computer Games (CGAMES), 2012.
[8]. Tremblay, Paul J; Schneider, Kevin; Cheston Grant. "*Exam Prep Flash Cards for Software Development in an Object-Oriented*", 2019. Retrieved from https://books.google.com
[9]. "Products-Unity", n.d. Retrieved from https://unity3d.com/unity
[10]. Ashish Rana. "Introduction: Reinforcement Learning with OpenAI Gym", 2018. Retrieved from https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2
[11]. Felix Yu. "Deep Q Network vs Policy Gradients - An Experiment on VizDoom with Keras", 2017. Retrieved from https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html