The Implementation of Reinforcement Learning Algorithm for AI Bot in Fighting Video Game

Adi Aiman Bin Ramlan
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia
emailadi97@gmail.com

Azliza Mohd Ali
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia
azliza@tmsk.uitm.edu.my

Nurzeatul Hamimah Abdul Hamid
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia
nurzeatul @tmsk.uitm.edu.my

Rozianawaty Osman
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia
roziana@tmsk.uitm.edu.my

2021 4th International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR) | 978-1-6654-3632-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISAMSR53229.2021.9567749

***Abstract*—A video game is a natural and valuable medium to test AI algorithms since this virtual environment is considered safe and controllable. The finite state machine is a common technique used in developing AI in games, using a predetermined action for each situation. The problem with a predetermined action is that the scripted behaviours are predictable and easily exploit by human players. Such settings cause the game to have repetitive gameplay, leading the player to lose interest since the player knows how the AI will behave. The primary approach of this project is to use reinforcement learning to train the agent for the game. Reinforcement learning algorithms learn what to do and map the situations to maximize the cumulative reward signal from the agent environment. The agent in this project will receive raw data from the environment as input, and the agent will then act based on the environment rather than predetermined action. The results show that lowering the learning rate in deep reinforcement learning can increase the cumulative reward for the agent. This project's findings will be helpful for the game developers in developing AI for their games and benefit the game players who will interact with the AI in those games.**

Keywords— Artificial Intelligence; Reinforcement Learning; Fighting Game; Unity ML-Agent

I. INTRODUCTION

The video game is interactive digital entertainment that people play through multiple medium such as a computer, a game console, or a smartphone. There are different genres of video games, ranging from complex strategy games to simple arcade games. It is stated in a book titled "Artificial Intelligence and Games" that video games are an ideal testbed for artificial intelligence (AI) methods and are also becoming a growingly crucial application area [1]. Games from different genres could provide various problems for agents to solve, making video games an excellent environment for AI research. Not only that, but these virtual environments are also actually safe and controllable [2].

In the 1950s, Arthur Samuel is the first person to introduce a machine learning method known now as reinforcement learning and used that method to develop a Checkers playing program that could learn from experience [3]. The recent developments in machine learning that involved reinforcement learning that act as a massive milestone for the artificial intelligence field of study are the achievements of AlphaGo and OpenAI Five. In March 2016, in a five-game match and with a score of 4-1, AlphaGo had won against one of the best Go players, Mr Lee Sedol, the winner of 18 world titles, and can be the greatest Go player of the past decade [4]. OpenAI Five successfully wins back-to-back games against Dota 2 world champions Team OG in 2019 and becomes the first AI to beat the world champions in an eSport match [5].

The most used method in developing game AI is the Finite State Machine algorithm started in the 1990s [6]. In the Finite State Machine method, the game developers generalize all possible situations game AI may encounter and then create a predetermined action for each situation [6]. One of the issues with a predetermined action in an AI bot is that the scripted behaviours can be predictable and easily be exploited by human players [7]. This can cause the game to have repetitive gameplay, which leads to the player losing interest in the game since the player will know how the AI bot will behave.

II. LITERATURE REVIEW

A. Fighting Game Environment

Fighting games are usually games that are designed for two distinct players. Those two players control their game character in the game and fight each other by acting that deals damages to their opponent and fighting games are an actual time type of games. In fighting games, possible actions that could be made contain a triangular relationship like the rock-paper-scissors game; this is to make sure the players cannot take the most dominant action over the other action repeatedly [8].

Each fighting game environment is also slightly changed with a different game, there may still be two players, but there will be additional input action and various observations. The observation in the environment can be about the state of the character, position, velocity, general game status, and enemy status [9].

B. Deep Reinforcement Learning

Deep reinforcement learning combines artificial neural network modelling with reinforcement learning, a set of approaches for learning from penalties and rewards instead of more explicit instructions [10]. After decades as an aspirational rather than practical idea, deep reinforcement learning has become one of the most popular areas of AI research within the past couple of years. Deep reinforcement learning techniques have given outstanding advances in the artificial intelligence field by surpassing human performance in domains ranging from no limit poker to Go to Atari [10].

Deep reinforcement learning, which synthesizes deep neural networks and reinforcement learning methods to tackle complex problems, came from the advancement in deep learning, making the researchers rethink the potential application for deep neural networks in diverse domains [11]. The synthesis between these two techniques can mitigate data reliance without introducing convergence problems via efficient data-driven exploration based on deep neural network output.

C. Proximal Policy Optimization

Proximal policy optimization is one of the policy gradient methods in reinforcement learning. The objective in policy gradient methods is to learn a parameterized policy that directly selects an action from the state space. During training, such methods calculate an estimation of the policy gradient in a stochastic gradient ascent algorithm [12]. In proximal policy optimization, the steps between sampling the data via interaction with its environment and optimizing a surrogate objective function using stochastic gradient ascent. It utilizes the advantages of trust-region optimization in stability and reliability by defining a clipped surrogate objective. Through that, the clip function can avoid substantial policy updates and decrease the problem of catastrophic steps (Melo & Maximo, 2019) [12]. Proximal policy optimization was introduced by the OpenAI team in 2017 [13]. It has a similar objective as trust region policy optimization, which utilizes a trust-region constraint to update the policy to ensure the new policy is not too far off from the old policy.

III. METHODOLOGY

A. Game Development

Game development is required as a testbed to test the learning in AI. Developing a game is better by using a game engine. A game engine provides game creators with the necessary set of features to build games quickly and efficiently.

The game engine used to assist with fighting game development is the Unity game engine. Unity is a cross-platform game engine that Unity Technologies developed. Unity game engine also comes with a built-in IDE for easy scripting and a physic engine to add physic features to handle physical simulation of the game assets. One of the reasons this game engine was selected is because Unity also provides an online platform known as Unity Learn to ease the game engine's learning process, which simultaneously will speed up the game development process.

The fighting game for this project will consist of two fighters, and each of them will have their health and stamina gauge to increase the complexity of the game environment. Game states that is currently available for a fighter are:

i. Idle state

ii. Move State (Moving to the left and right)

iii. Slash Attack State

The perception for the agent in the game environment is the agent will be able to observe their health and stamina, the opponent's health and stamina, and the distance between the agent and their opponent (see figure 1). The fighter in the environment will be assigned the same health and stamina to fight with the same condition.

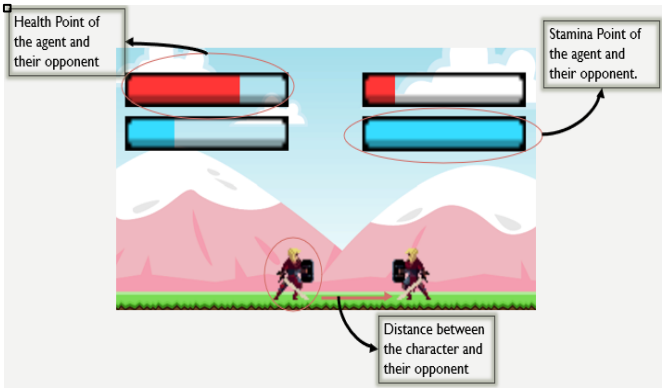

Figure 1: Agent Perception in the Environment

B. Model Development for the Agent

The development of game agents using the deep reinforcement learning technique will be developed using the Unity ML-Agent toolkit that Unity Technologies also developed. Unity ML-Agent toolkit is an open-source tool that can train intelligent agents with deep reinforcement learning and evolutionary methods using a simple Python API. In this project, the models of game AI will be trained with deep reinforcement learning using a proximal policy optimization algorithm for the policy update.

For the agent's learning process, the agent will observe the state from the environment (see Figure 2). In this case, it takes its own and its opponent's health, stamina, and distance between two fighters as input and gives a particular action such as attack or movement as output. Then the action is performed in the environment; if something positive happens due to the action, like hitting the opponent, the environment sends back a positive response in reward. The agent receives a negative reward if the fighter's health is deducted because of getting hit by the opponent. Based on the rewards received, the current policy will be compared to the new policy and decide which policy to keep

for the agent. The proximal policy optimization algorithm was used to update the policy for the agent.

Figure 3 shows the deep neural network architecture for this project. It has six neurons in the input layers. It represents the agent's health, the agent's stamina, the opponent's health, the opponent's stamina, direction of the opponent and the distance between the agent and the opponent. The range of values for the neurons in the input layer is shown in Table 3. Before all the values of that feature are fed into the network, it will first undergo normalization. The process of normalization is to convert the data into appropriate forms for training. All those values will be rescaled to the range of 0 to 1. It has three hidden layers, and each layer contains ten neurons. The output layer contains two neurons. It represents the actions which the agent can make. The first neuron is for attack action; if the output is 0, the agent will not perform an attack. If the output is 1, then the agent will perform an attack. The second neuron in the output layer is for movement. If the output value for the second neuron is 1, the output is 0; the agent will not move.

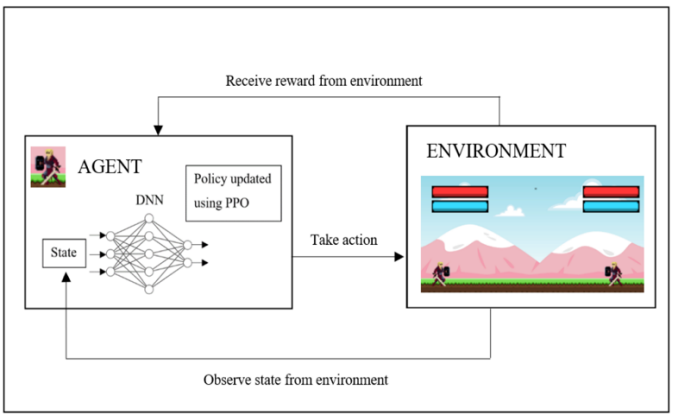

Figure 2: Deep Reinforcement Learning Process

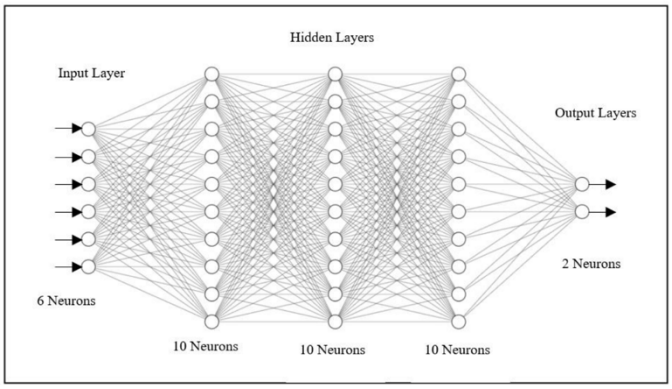

Figure 3: Deep Neural Network Architecture

To train a model for the agent, a reward function needs to be designed for the algorithm. For designing the reward function, values from the game environment such as player and opponent's health point, stamina point, the distance between player and opponent are observed. The reward function is used to provide a learning signal to the agent. It can be defined or modified at any time during the simulation using the Unity scripting system. The rewards allocation for the agent is as shown in Table 1.

Table 1: Reward Function

Action	Reward
Move closer to the opponent	+0.1
Hitting an opponent	+5
Perform attack action but missed the opponent	-5
Win the fight	+50
Lose the fight	-50

C. *Analysis*

The last phase is the analysis phase. The model will be analysed based on cumulative reward throughout the iteration. This method is suitable to implement because it will show the bot learning convergence in the given environment. Changing parameters is also important to train the network to achieve the expected performance and accelerate the learning model's convergence. The parameters manipulated in this project are the number of neurons in the hidden layers and the learning rate.

The hidden unit parameter corresponds to how many neurons are in each hidden neural network. For a simple environment where the action is straightforward, a small number of neurons are sufficient for the model development. The learning rate refers to the initial rate for the gradient descent, and it corresponds to how quickly or slowly a model learns about the problem.

IV. Results

A. *Hyperparameter Setting*

The base parameter setting for this project is shown in Table 2. The value for max steps was chosen based on several test training sessions, which shows that most training models start to converge at the 50,000 steps. The max step for the base parameter setting is twice those values. The other parameters value set in the base parameter set is based on the default value set by the Unity ML-Agent toolkit. In this project, the parameters that will be manipulated are the learning rate and the number of neurons in the hidden layer. The learning rate will be increased to 0.03 and lower value to 0.0003. The neurons in the hidden layer will be decreased to 6, which is equivalent to the number of neurons in the input layer. Then, the number of neurons in hidden layers will be reduced to 2, which is the number of neurons in the output layer. The experimentation regarding these two parameters is expected to show changes in the training model of the agent. The details about different parameter settings can be seen in Table 3.

In this project, the training for the agent will stop when the number of steps taken by the agent is the same as the maximum step's parameters, which is 100,000 steps. The results of the training model for all parameter settings are shown in Figure 4.

V. Discussion

Figure 4 shows that the training model with parameter setting 2 has the best result compared to others. In parameter setting 2, the learning rate is set to 0.0003. Lowering the

learning rate manages to increase the agent's learning convergence point, and it stays above 140 lines. Because the learning rate is low, the agent's steps need to be taken before convergence is extended at 70,000 steps. Next, parameter setting one based on figure 4 shows the most stable training model compared to others. Parameter setting three from figure 4 shows that lowering the neurons in the hidden layer to the same number of input neurons negatively impacts this project since it shows that the training model has the worst consistency in training sessions. Lastly, from figure 4, it can be concluded that the learning convergence for the agent in this project environment is located between the cumulative reward value of 140 to 120.

Table 2: Base Parameter Setting

Parameter	Value
Max Steps	100,000
Learning Rate	0.003
Batch Size	64
Buffer Size	12,000
Hidden Layers	3
Hidden Units	10
Beta	0.001
Epsilon	0.2
Lambda	0.99
Gamma	0.99

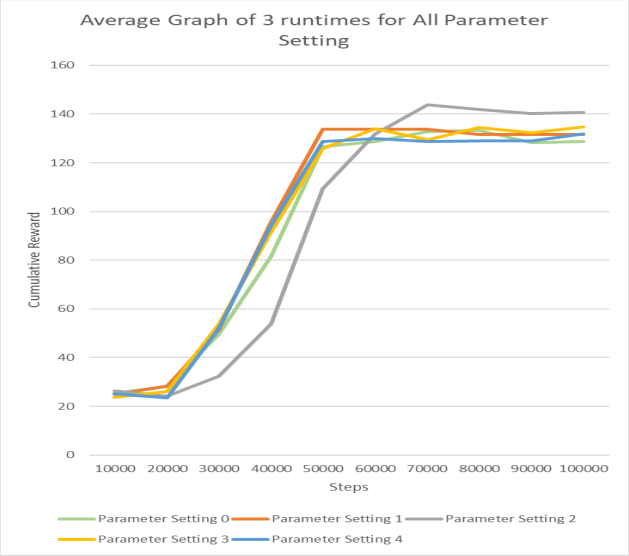

Figure 4: Average Graph of 3 Runtimes for All Parameter Settings

Table 3: Multiple Parameter Settings

Parameter	Parameter Setting 0 (Base)	Parameter Setting 1	Parameter Setting 2	Parameter Setting 3	Parameter Setting 4
Max Steps	100,000	100,000	100,000	100,000	100,000
Learning Rate	0.003	0.03	0.0003	0.003	0.003
Batch Size	64	64	64	64	64
Buffer Size	12,000	12,000	12,000	12,000	12,000
Hidden Layers	3	3	3	3	3
Hidden Units	10	10	10	6	2
Beta	0.001	0.001	0.001	0.001	0.001
Epsilon	0.2	0.2	0.2	0.2	0.2
Lambda	0.99	0.99	0.99	0.99	0.99
Gamma	0.99	0.99	0.99	0.99	0.99

VI. Conclusion

An AI bot for the fighting game was successfully developed using a deep reinforcement learning algorithm in this project. For this project, a fighting game was developed using the Unity game engine and C#. AI bot was developed using Unity ML-Agent toolkit, which makes use of the deep reinforcement learning algorithm and proximal policy optimization algorithm for the policy update. One of the important lessons learned from this project is that lowering the learning rate in deep reinforcement learning can increase the cumulative reward for the agent. However, it will increase the steps the agent needs for the model to converge.

This project's findings will prove helpful to the game developers in developing AI for their games and benefit the game players who will interact with the AI in those games. After all the obstacles and challenges faced throughout this project, the lessons learned will be helpful for those interested in pursuing the same type of problem domain. The project also has room for improvement for future works. For those that are interested in improving this project further, some future improvements that can be made are:

i. Use another form of machine learning techniques and make a comparison.

ii. Develop a more complex game environment where multiple agents need to cooperate.

iii. Develop agent for the 3D game environment.

REFERENCES

[1] G. N. Yannakakis and J. Togelius, Artificial Intelligence and Games, Springer, 2018.

[2] K. Shao, Z. Tang, Y. Zhu, N. Li and D. Zhao, "A Survey of Deep Reinforcement Learning in Video Games," 2019.

[3] J. Togelius, Playing Smart: On Games, Intelligence, and Artificial Intelligence, MIT Press, 2018.

[4] F.-Y. Wang, J. J. Z. X. Zheng, X. Wang, Y. Yuan, X. Dai, J. Zhang and L. Yang, "Where Does AlphaGo Go: From Church-Turing Thesis to AlphaGo Thesis and Beyond," *IEEE/CAA Journal Of Automatica Sinica, Vol. 3, No. 2,* pp. 113-120, 2016.

[5] C. Berner, B. C. Greg Brockman, P. ". D. Vicki Cheung, D. F. Christy Dennison, S. H. Quirin Fischer, R. J. Chris Hesse, C. O. Scott Gray and M. P. Jakub Pachocki, "Dota 2 with large scale deep reinforcement learning," 2019.

[6] H. Lou, "AI in Video Games: Toward a More Intelligent Game," 28 8 2017. [Online]. Available: http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/.

[7] S. Petrakis and A. Tefas, "Neural Networks Training for Weapon Selection in First-Person Shooter Games," *Artificial Neural Networks -- ICANN 2010,* 2010.

[8] N. Sato, S. Temsiririrkkul, S. Sone and K. Ikeda, "Adaptive Fighting Game Computer Player by Switching Multiple Rule-based Controllers," *3rd International Conference on Applied Computing and Information Technology,* 2015.

[9] M. Azzinaro, "Teaching AI To Play a Platform-Fighting Game Using Genetic Neural Networks," 22 November 2017. [Online]. Available: https://medium.com/@mikecazzinaro/teaching-ai-to-play-a-platform-fighting-game-using-neural-networks-ef9316c34f52.

[10] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell and D. Hassabis, "Reinforcement Learning, Fast and Slow," *Trends in Cognitive Sciences,* 2019.

[11] T.-R. Lin, D. Penney, M. Pedram and L. Chen, "A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study," 2020.

[12] L. C. Melo and M. R. O. A. Maximo, "Learning Humanoid Robot Running Skills through Proximal Policy Optimization," *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE),* 2019.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," 2017.