

Generating intelligent agent behaviors in multi-agent game AI using deep reinforcement learning algorithm

Rosalina, Axel Sengkey, Genta Sahuri, Rila Mandala

Informatics Study Program, Faculty of Computing, President University, Bekasi, Indonesia

Article Info

Article history:

Received Aug 4, 2023

Revised Oct 9, 2023

Accepted Oct 18, 2023

Keywords:

Artificial intelligence
Deep reinforcement learning
Intelligent multi-agent
Multi-agent game
Q-learning

ABSTRACT

The utilization of games in training the reinforcement learning (RL) agent is to describe the complex and high-dimensional real-world data. By utilizing games, RL researchers will be able to evade high experimental costs in training an agent to do intelligence tasks. The objective of this research is to generate intelligent agent behaviors in multi-agent game artificial intelligence (AI) using deep reinforcement learning (DRL) algorithm. A basic RL algorithm called deep Q network is chosen to be implemented. The agent is trained by the environment's raw pixel images and the action list information. The experiments conducted by using this algorithm show the agent's decision-making ability in choosing a favorable action. In the default setting for the algorithm, the training is set into 1 epoch and 0.0025 learning rate. The number of training iterations is set to one as the training function will be repeatedly called for every 4-timestep. However, the author also experimented with two different scenarios in training the agent and compared the results. The experimental findings demonstrate that our agents learn correctly and successfully while actively participating in the game in real time. Additionally, our agent can quickly adjust against a different enemy on a varied map because of the observed knowledge from prior training.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Rosalina

Informatics Study Program, Faculty of Computing, President University

Jl. Ki Hajar Dewantara, Cikarang Baru, Bekasi, Jawa Barat, Indonesia

Email: rosalina@president.ac.id

1. INTRODUCTION

Technology is developed to help humans perform complicated work that either involves dangerous work or complex computation. Inventions such as computers and smartphones are some good examples of technological development that enables humans to work in a smart, simple, and efficient manner through a variety of smart programs. An example of a smart program is the virtual intelligence assistant developed by Google which can recognize and process our voice as an input, the Google Assistant. This smart program has a trainable intelligence that will get better in recognizing voice and processing tasks as long as it has a decent amount of input and training time. This man-made intelligence is called artificial intelligence (AI).

Google DeepMind and OpenAI are companies that show AI potential in solving problems that can be trained in a simulated environment. Reinforcement learning (RL), one of the machine learning (ML) methods, is utilized by these companies to train an expert agent who outperforms humans in the game. The utilization of the game in training the RL agent is to describe the complex and high-dimensional real-world data [1]–[9]. By utilizing games, RL researchers will be able to evade high experimental costs in training an agent to do intelligence tasks [10]. However, RL application is still impacted by high sample complexity, especially in

multi-agent systems. To solve this problem, Loftin *et al.* [11] formally define the concept of strategically effective exploration in Markov games and use this to create two finite Markov game learning algorithms.

In an end-to-end framework, the combination, known as deep reinforcement learning (DRL), significantly enhances the generalization and scalability of conventional RL algorithms by instructing agents to make decisions in high-dimensional state space, such as playing video games, controlling robots, and making decisions in various real-world applications. Examine the evolution of DRL research with a particular emphasis on AlphaGo and AlphaGo Zero [12], [13]. Research by Li [14] discusses key components, significant mechanisms, and a range of applications while providing an overview of DRL achievements. Meanwhile, Perakam *et al.* [15] introduce the first deep-learning model that can successfully learn control policies from high-dimensional sensory input. The model is a convolutional neural network that was trained using a variation of Q-learning, with raw pixels as its input, and an estimation of future rewards as its output. More recently, the RL algorithm was implemented in [15]–[22] which mostly aims to generate intelligent agent behavior. This research implements RL as the ML algorithm in creating an agent that could outperform humans in the Atari game by using the algorithm that is proposed in [23] and the agent is trained by the environment raw pixel images and the action list information.

2. RESEARCH METHOD

2.1. The Markov property

The state in the RL should satisfy the Markov property. The Markov property defines that a current state completely characterizes the state of the world, hence the current state is independent both towards the future and past state [24]. In the agent's environment, the agent transitions to another state through the taken action. If the next state could be predicted without knowing/dependent on the preceded events, then the mathematical equation of the property is given in (1).

$$Pr\{s_{t+1} = \acute{s}, r_{t+1} = r \mid s_t, r_t, a_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} \quad (1)$$

2.2. Markov decision models

The Markov decision process is the mathematical formulation of the RL defined by the tuple $(S, A, \mathcal{R}, \mathbb{P}, \gamma)$, where S represents the set of possible states, A represents the set of possible actions, \mathcal{R} represents the distribution of the reward given a (state, action) pair, \mathbb{P} represents the transition probability i.e., distribution of the next state given (state, action) pair, and γ represents the discount factor.

The Markov decision process works will be represented as the main task of the RL's agent which is described through the pseudocode: i) The agent initializes by sampling the environment's initial state $s_0 \sim p(s_0)$ and ii) Then, from $t=0$ until done: agent select action a_t , environment samples reward given the state and action given $r_t \sim R(\cdot \mid s_t, a_t)$, Environment sample the next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, and Agent receives reward r_t and move to the next state s_{t+1} .

Based on this, the agent policy can now be stated as $\pi_t(s, a)$ that specifies the choosing action mechanics for the agents in each state. The objective of the RL agent is to find the optimum policy π^* that maximize the cumulative discounted reward $\sum_{t>0} \gamma^t r_t$. The optimum policy should be *stochastic* to fulfill the Markov property. Thus, to handle the randomness, the maximum expected sum of reward is taken.

$$\pi^* = \arg \max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t \mid \pi] \quad (2)$$

Initial state sampled from the initial state distribution $s_0 \sim p(s_0)$, action sampled from the policy given by the state $a_t \sim \pi(\cdot \mid s_t)$, and the next state sampled from the transition probability distribution $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.

2.3. Value function and Q-value function

Finding the optimum policy means that the agent has to learn the goodness of a state and the goodness of a state-action pair. The value function is the expected cumulative reward from following the policy from a state that quantifies the good and bad state.

$$V^{\pi}(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi] \quad (3)$$

The *Q-Value function* at state s and action a is the expected cumulative reward from taking the action a on state s .

$$Q^{\pi}(s, a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi] \quad (4)$$

2.4. Q-learning

The Q-learning uses an action-value function, Q , to approximate the optimal action-value function, Q^* . Q-learning utilizes the Bellman Equation which is a mathematical equation that is mainly used to solve the optimization problem and it is mainly utilized in Dynamic Programming. The equation that is utilized for RL is shown in (5).

$$Q(s, a) = r(s, a) + \gamma \max_a Q(s', a) \quad (5)$$

The (5) is a processed Bellman Equation that is used to fit a state and action pair into it. The $Q(s, a)$, commonly referred to as the Q value, is calculated through the addition of the immediate reward $r(s, a)$ added by the maximum value of the highest possible Q value from the next state (s') in the response of taking an action (a) time a discount factor γ . A discount factor, a number between 0 and 1, is used to control the importance of the short-term and long-term rewards. When given the chance to obtain a short-term reward, an agent will be compelled to act avariciously and grab the largest reward as soon as possible, which leads to the development of exploitation behavior. In contrast, a long-term reward will have the opposite effect. Thus, the goal of Q-learning is to maximize the future cumulative reward that could be achieved. The characteristic of this algorithm made it to be called a greedy algorithm.

The Q-learning algorithm is a reliable solution in the field of RL, especially in uncharted territory. Known for its ease of use, Q-learning has fewer parameters, strong exploratory powers, and a convergence guarantee. Its independence from the requirement for an explicit model of the environment is one of its unique characteristics. Because of this feature, Q-learning is especially useful in situations where it is difficult or impractical to obtain an accurate model [25]. Because of its effectiveness in path planning, an area where outcomes are optimized, Q-learning has attracted a lot of attention and investigation in academic study. The algorithm's adaptability and usefulness in many real-world circumstances are highlighted by its capacity to navigate unfamiliar environments and determine optimal policies without the need for a pre-existing model.

2.5. Deep Q-network

A deep Q network combines the deep neural network with the Q-learning mechanics. The neural network will replace the Q values tables, and as a result, the neural network will replace the table function to approximate Q values. By minimizing the cost function, which will be similar to the mean square error function, the algorithm aims to minimize the difference between the initial learning state and the goal state where the Q value reaches its final converged value. The cost function of the deep Q network is defined as (6):

$$Cost = \left[Q(s, a; \theta) - \left(r(s, a) + \gamma \max_a Q(s', a) \right) \right]^2 \quad (6)$$

where $Q(s, a; \theta)$ is the new state-action value function which takes trainable weights of the neural network (θ).

The main problem of training an agent is to introduce the agent's independence towards state transition. The agent that depends on learning from the exact previous state has a risk of being trapped in the unwanted scenario as the agent does not have any other source to learn. Hence, experience replay is introduced to stochastically handle the problem. The algorithm is shown in Algorithm 1.

Algorithm 1. Deep Q-learning with experience replay

```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode=1,  $M$  do
  Initialize sequence  $s_1=\{x_1\}$  and preprocessed sequenced  $\phi_1=\phi(s_1)$ 
  for  $t=1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t=\max_a Q^*(\phi(s_t), a; \theta)$ 
    Execute action  $a_t$  in the emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
    Set  $s_{t+1}=s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1}=\phi(s_{t+1})$ 
    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$ 
    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
    Set  $Y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_a Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
    Perform a gradient descent step on  $(Y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3
  end for
end for

```

3. RESULTS AND DISCUSSION

Ms. PacMan's environment satisfies the Markov properties, as the agent does not need to know the previous state to predict the next state. For example, the agent does not need to know how the bonus fruit appears in the game, instead, it could predict in the future to approach the bonus fruit when it does appear on the game screen. The system processes high-dimensional pixel data. Therefore, the game's frame is pre-processed by the user-defined function to a smaller size (84, 84) with the grayscale color scheme. Figure 1 illustrates the process of the game frame transformation.

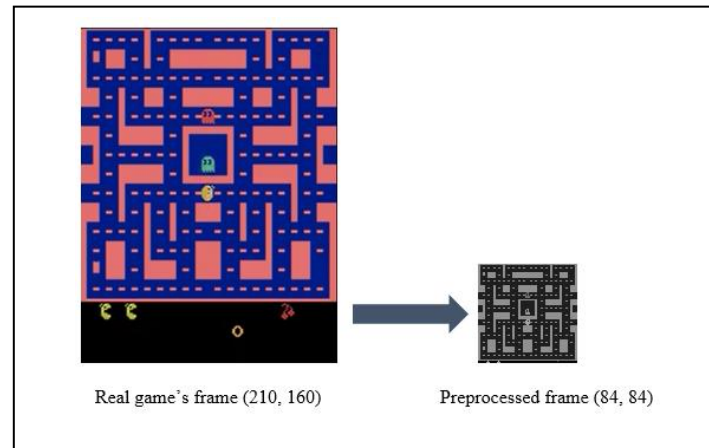


Figure 1. The transformation of the preprocessed game frame

The agent's performance in the initial episode is shown in Figure 2. The initial result when the first episode runs shows that the agent still trying to explore the surrounding area (Figure 2(a)). The agent movement is not smooth which means that the agent did not take 1 action per direction it's heading into (Figure 2(b)). Instead, the agent tries different random actions that cause it to stutter around and end up dying while it gathers a 90 score (Figure 2(c)).

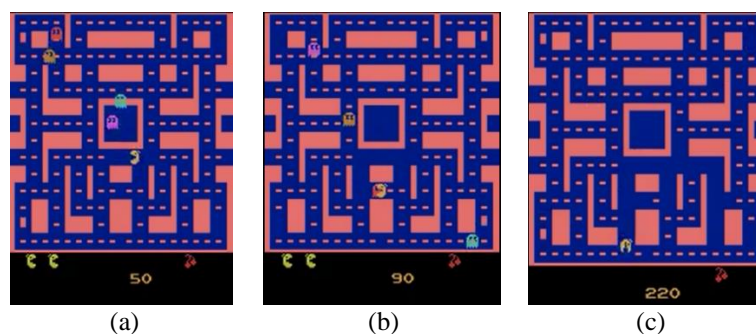


Figure 2. The agent's performance at the initial episode, (a) the interface of the agent's initial behavior, (b) the agent's collision with a ghost, and (c) the agent out of lives

The states that will be used in the paper will be the pre-processed game frames. The state space consists of a lot of states as Ms. PacMan has 1293 distinct locations in the maze. A complete state of Ms. PacMan's model consists of the location of Ms. PacMan, the ghosts, the power pills, along with the ghost's previous move, and the information on whether the ghost is edible.

The agent has nine actions that could be performed at the game which are represented by a single integer. These actions are ['NOOP', 'UP', 'RIGHT', 'LEFT', 'DOWN', 'UPRIGHT', 'UPLEFT', 'DOWNRIGHT', 'DOWNLEFT']. Meanwhile, Ms. PacMan's reward could be obtained by gathering the foods (Pac-Dot), bonus fruit (Fruit), power-up items (Power Pellet), eating a ghost, and chain-eating the ghosts. The reward list is shown in the Tables 1 and 2.

Table 1. Ms. PacMan's reward space—the foods and ghost-eating scores














Image	Name	Score
	Pac-Dot	10 points
	Power Pellet	50 points
	1 Ghost	200 points
	2 Ghost	400 points
	3 Ghost	600 points
	4 Ghost	800 points

Table 2. Ms. PacMan's reward space—the bonus fruits

Image	Name	Score
	Cherry	100 points
	Strawberry	200 points
	Orange	500 points
	Pretzel	700 points
	Apple	1000 points
	Pear	2000 points
	Banana	5000 points

The testing was conducted in the local machine in the PyCharm IDE. The observation is conducted by observing the output of the neural network per game episode, analyzing the video output, and creating a gameplay graph. The initial result when the first episode runs shows that the agent still trying to explore the surrounding area. The agent's movement is not smooth which means that the agent did not take 1 action per direction it's heading into. Instead, the agent tries different random actions that cause it to stutter around the hall and end up dying. At the end of its life, the agent manages to gather 220 episodes without any utilization of power pills which can power up Ms. PacMan to eat the ghost without dying.

In the default setting for the algorithm, the training is set into 1 epoch and 0.0025 learning rate. The number of training iterations is set to one as the training function will be repeatedly called for every 4-timestep. However, the author also experimented with two different scenarios in training the agent and compared the results. The full training scenario is shown in Table 3.

Table 3. Testing scenario

	Epoch	Learning rate
Case A	1	0.3
Case B	100	0.025
Default	1	0.025

The first scenario, case A, is to set 1 epoch and 0.3 learning rate. The time used to train 100 episodes with 1 epoch and 0.3 learning rate is 6 hours. During its 100th gameplay, the agent has undergone 12,595 training sessions. The agent got a 180 score when it reached the 100th episode. The scores from episode 80 to episode 110 mostly dominated around the range of 200–250, the highest score that the agent can reach occurred at episode 97 with the score 590. In this case, the agent still tends to act like the initial test.

The author uses 100 epochs to test whether the agent could run well if the number of training is increased. The second scenario, case B, is to set 100 epochs and 0.025 learning rate. The time used to train 100 episodes is 1 day and 2 hours, or 26 hours in total. During its 100th gameplay, the agent has undergone 13,023*102 training sessions. In this scenario, the agent behavior has resulted in a more consistent exploration, detecting that a lot of the road is emptied from the food that is previously lying around. In this scenario, there exist three high scores achieved by the agent which are found at episodes 85, 100, and 104 with a value of 940, 1,040, and 1,340. Seeing the amount of resources that are exhausted in this testing scenario, the author decides that this testing scenario is not feasible to be tested.

For the default case, the training is conducted in three different phases because the author encountered a technical error during the training session which interrupted the training session. The first phase has 547 episodes which is running for 29 hours and 27 minutes. The second phase has 601 episodes which is running for 9 hour and 43 minutes. The third phase has 1,001 episodes which is running for 20 hours and 8 minutes. Hence, in total, the default scenario was going through 2,149 episodes in 59 hours and 19 minutes. In this scenario, the agent can reach 100 episodes/hour. However, as the training went on, the training speed declined to around 35-40 episodes/hour starting from the 700th–1,001th episode of the second phase. The training speed declined as the agent consumed a lot of memory and storage throughout the training session. At the start of a training session, the agent only needs less than 2 GB of RAM whereas at the halfway of the training, around the 600th episode, the agent consumes around 6.5–8.7 GB of RAM.

In the first phase, the agent explored any possible actions it could take which is shown by the low score achieved (ranging around 200-400 points) and stuttering behavior. Thus, the number of high scores achieved at this point cannot be said as a result of an intelligent decision. The average scores graph, depicted in Figure 3, shows the trends of improvement in the agent's performance through 547 episodes.

The agent started to frequently achieve scores of more than 400 points as shown in the scores history graph depicted by Figure 4. The number of high scores achieved also shows an improvement as the score graph is updated to more than 2,500 points. The epsilon is set to follow the progress that has been made in the first phase with a 0.05 value addition.

At the third phase of the training session, the agent is also loaded with the second phase training's weight and the latest epsilon of the second phase. The agent's performance at the beginning is much more improved than the agent's performance at the beginning of the first phase. At the end of the third phase, the agent reached a maximum score at the 462nd episode with 4,020 points as shown in Figure 5. Around the 700th episode, the maximum average score peaked at 877.7. The second evaluation agent gained 651.8 evaluation points.

The agent performance in the default case shows promising behavior (indicated by the agent's performance improvement in achieving a new higher score). The average scores history graph is presented to give more details on the agent's performance through Figure 6. The figure shows that throughout the learning the agent has shown a good trend of improvement. However, after the agent passes the 500th episode, the agent shows stagnant performance that almost leads to performance deterioration.

An additional experiment was carried out on a simpler game environment, Breakout. Breakout is a game where a layer of bricks is stationed on the top third of the screen and the goal of the game is to destroy all of the bricks using a bouncing ball that the player can hit using a panel that can move horizontally. The agent shows similar performance in the game of Breakout as shown by the average score graph in Figure 7. In 1000 episodes, the agent still iterates through the exploration phase as shown by the fluctuation of the average score graph.

In this research, the author implements the algorithm featured in [9], where the result of this paper is shown in Figure 8. However, due to the experimental parameters, which are a combination of experimental and recommended values, the code's performance may not be optimal. There are a lot of sources that give a better approach to either construct the algorithm or set the recommended parameters, but these approaches require more expert knowledge that needs to be understood and studied which requires a lot of time.

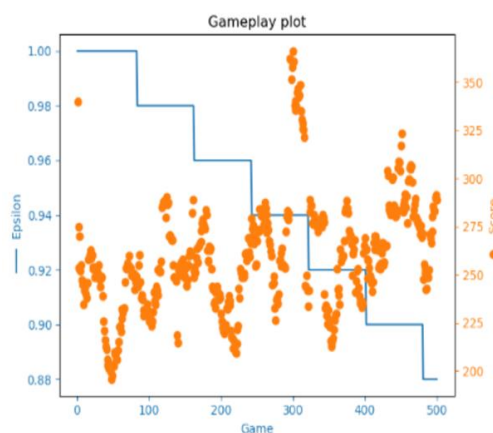


Figure 3. The first phase score graph

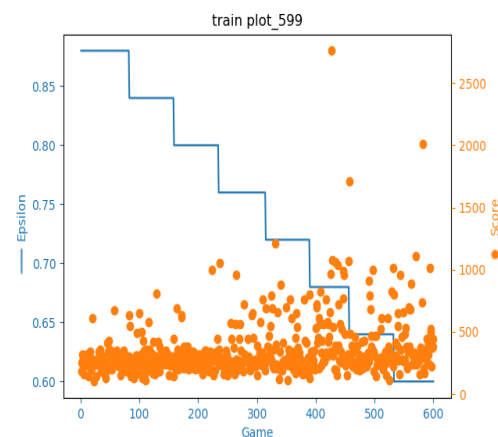


Figure 4. The second phase score graph

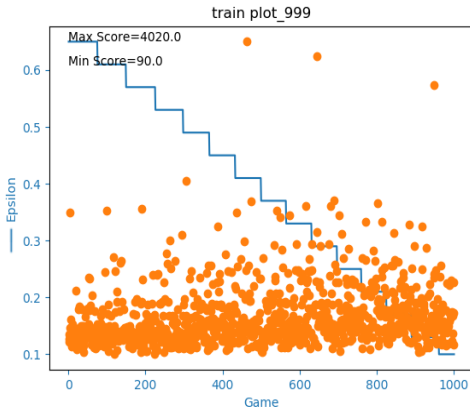


Figure 5. The third phase score graph

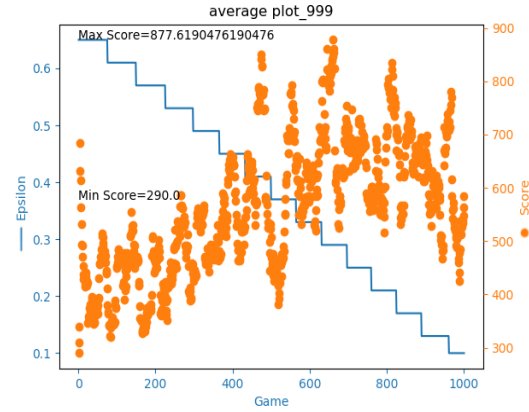


Figure 6. The third phase average score graph

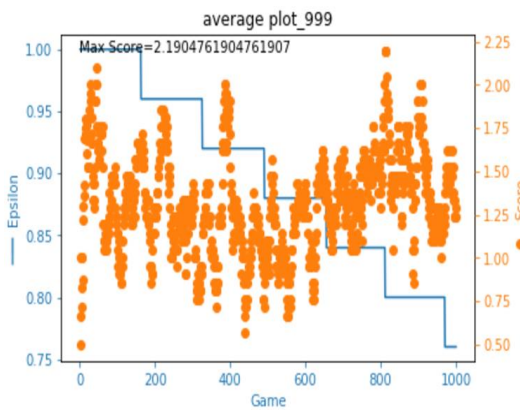


Figure 7. The Breakout average score graph

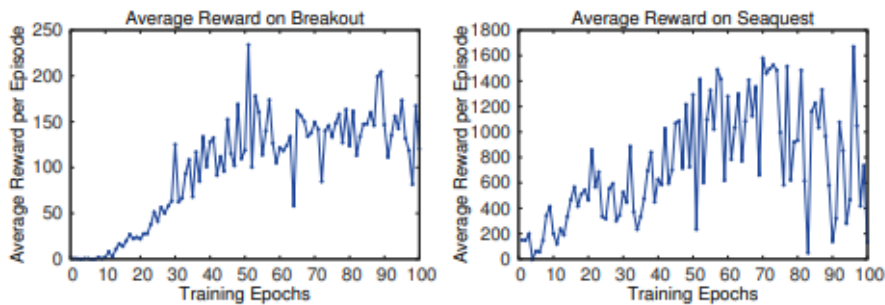


Figure 8. The average reward per episode on Breakout and Seaquest result in paper [9] respectively during training

4. CONCLUSION




Even though the agent training progress shows a good trend of improvement, this research cannot be said successful in creating an agent that could master and exploit Ms. PacMan. The author found that the choosing action mechanism works as intended. The agent first discovers new possibilities until the agent exploits a good approach to get a better result. The agent's got a stagnant score after passing the 500th episode followed by the downfall in the latest episode. This pattern is assumed to happen because of the agent's behavior to exploit the action that is not resulting in the desirable result as the agent's performance and behavior around 1000 episodes is not perfect which means the agent is still learning. Hence, the agent that exploits the non-desired action got a downgrade in its performance.

At the end of the experiment, the agent can move with less stuttering per frame explore a large amount of area, and utilize some power pills before it runs out of life. Looking at the results of the experiment, the author believes that the RL could be a great approach to solving real-world problems. The author encourages other students to study this field. It should be noted that further knowledge is required as the real-world problem is more complicated than a game which means that a basic RL algorithm will only be fundamental to solve the problem.




REFERENCES

- [1] I. A. Kash, M. Sullins, and K. Hofmann, "Combining No-regret and Q-learning," *ArXiv-Computing Science*, 2019, doi: 10.48550/ARXIV.1910.03094.
- [2] S. Levine, "Unsupervised reinforcement learning," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, in AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2020, pp. 5–6, doi: 10.5555/3398761.3398766.
- [3] P. Knott *et al.*, "Evaluating the robustness of collaborative agents," *ArXiv-Computing Science*, 2021, doi: 10.48550/ARXIV.2101.05507.
- [4] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, pp. 1–10, doi: 10.1609/aaai.v32i1.11791.
- [5] J. Beck, K. Ciosek, S. Devlin, S. Tschitschek, C. Zhang, and K. Hofmann, "Amrl: aggregated memory for reinforcement learning," in *International Conference on Learning Representations*, 2020, pp. 1–20.
- [6] I. Szita, "Reinforcement learning in games," in *Reinforcement Learning*, vol. 12, M. Wiering and M. van Otterlo, Eds., in *Adaptation, Learning, and Optimization*, vol. 12, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 539–577, doi: 10.1007/978-3-642-27645-3_17.
- [7] K. Hofmann, "Minecraft as AI playground and laboratory," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, Barcelona Spain: ACM, Oct. 2019, pp. 1–1, doi: 10.1145/3311350.3357716.
- [8] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," *arXiv:1611.06256*, pp. 1–12, 2016, doi: 10.48550/ARXIV.1611.06256.
- [9] L. Zintgraf *et al.*, "Exploration in approximate hyper-state space for meta reinforcement learning," in *International Conference on Machine Learning (ICML)*, vol. 139, pp. 12991–13001, 2020, doi: 10.48550/ARXIV.2010.01062.
- [10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May. 2015, doi: 10.1038/nature14539.
- [11] R. Loftin, A. Saha, S. Devlin, and K. Hofmann, "Strategically efficient exploration in competitive multi-agent reinforcement learning," *arXiv:2107.14698*, 2021, doi: 10.48550/ARXIV.2107.14698.
- [12] D. Zhao *et al.*, "Review of deep reinforcement learning and discussions on the development of computer Go," *Control Theory and Applications*, vol. 33, no. 6, pp. 701–717, Jun. 2016, doi: 10.7641/CTA.2016.60173.
- [13] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," *Control Theory and Applications*, vol. 34, no. 12, pp. 1529–1546, Dec. 2017, doi: 10.7641/CTA.2017.70808.
- [14] Y. Li, "Deep reinforcement learning: an overview," *arXiv:1701.07274*, 2017, doi: 10.48550/ARXIV.1701.07274.
- [15] S. K. U, P. S. G. Perakam, V. P. Palukuru, J. Varma Raghavaraju, and P. R., "Artificial intelligence (AI) prediction of atari game strategy by using reinforcement learning algorithms," in *2021 International Conference on Computational Performance Evaluation (ComPE)*, Shillong, India: IEEE, Dec. 2021, pp. 536–539, doi: 10.1109/ComPE53109.2021.9752304.
- [16] B. Setiaji, E. Pujastuti, M. F. Filza, A. Masruro, and Y. A. Pradana, "Implementation of reinforcement learning in 2D based games using open AI gym," in *2022 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, Jakarta, Indonesia: IEEE, Nov. 2022, pp. 293–297, doi: 10.1109/ICIMCIS56303.2022.10017810.
- [17] J. Nogae, K. Ootsu, T. Yokota, and S. Kojima, "Comparison of reinforcement learning in game AI," in *2022 23rd ACIS International Summer Virtual Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Summer)*, Kyoto City, Japan: IEEE, Jul. 2022, pp. 82–86, doi: 10.1109/SNPD-Summer57817.2022.00022.
- [18] A. Das, V. Shroff, A. Jain, and G. Sharma, "Knowledge transfer between similar atari games using deep Q-networks to improve performance," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India: IEEE, Jul. 2021, pp. 1–8, doi: 10.1109/ICCCNT51525.2021.9580091.
- [19] C. Hu *et al.*, "Reinforcement learning with dual-observation for general video game playing," *IEEE Transactions on Games*, vol. 15, no. 2, pp. 202–216, Jun. 2023, doi: 10.1109/TG.2022.3164242.
- [20] P. Xu, Q. Yin, J. Zhang, and K. Huang, "Deep reinforcement learning with part-aware exploration bonus in video games," *IEEE Transactions on Games*, vol. 14, no. 4, pp. 644–653, Dec. 2022, doi: 10.1109/TG.2021.3134259.
- [21] C. Fan and Y. Hao, "Precise key frames adversarial attack against deep reinforcement learning," in *2022 10th International Conference on Intelligent Computing and Wireless Optical Communications (ICWOC)*, Chongqing, China: IEEE, Jun. 2022, pp. 1–6, doi: 10.1109/ICWOC55996.2022.9809899.
- [22] Y. Zhan, Z. Xu, and G. Fan, "Learn effective representation for deep reinforcement learning," in *2022 IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, Taiwan: IEEE, Jul. 2022, pp. 1–6, doi: 10.1109/ICME52920.2022.9859768.
- [23] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv:1312.5602*, 2013, doi: 10.48550/ARXIV.1312.5602.
- [24] S. Pitis, "Rethinking the discount factor in reinforcement learning: a decision theoretic approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 1, pp. 7949–7956, Jul. 2019, doi: 10.1609/aaai.v33i01.33017949.
- [25] A. Wong, T. Bäck, A. V. Kononova, and A. Plaata, "Deep multiagent reinforcement learning: challenges and directions," *Artificial Intelligence Review*, vol. 56, no. 6, pp. 5023–5056, Jun. 2023, doi: 10.1007/s10462-022-10299-x.




BIOGRAPHIES OF AUTHORS

Rosalina    is a lecturer in the Informatics Study Program at President University. Strong education professional who graduated with a master's degree in informatics from President University. She can be contacted at email: rosalina@president.ac.id.






Axel Sengkey    is a student in the Informatics Study Program at President University. His research interest includes programming, web development, game development, data science, and machine learning. He can be contacted at email: axel.sengkey@student.president.ac.id.



Genta Sahuri    is a lecturer in the Information System Study Program at President University. Strong education professional who graduated with a master's degree in informatics from President University. He can be contacted at email: genta.sahuri@president.ac.id.



Rila Mandala    is the dean of the Faculty of Computing at President University. Strong education professional in Informatics, graduated from Tokyo Institute of Technology (Ph.D.). He can be contacted at email: rilamandala@president.ac.id.