Advance Sustainable Science, Engineering and Technology (ASSET)
Vol. 6, No.1, January 2024, pp. 02401011-01 ~ 02401011-011
ISSN: 2715-4211 DOI: https://doi.org/10.26877/asset.v6i1.17562

Zonation Method for Efficient Training of Collaborative Multi-Agent Reinforcement Learning in Double Snake Game

Marvin Yonathan Hadiyanto*, Budi Harsono, Indra Karnadi

Department of Electrical Engineering, Krida Wacana Christian University, Jl. Tanjung Duren Raya no. 4, Jakarta 11470, Indonesia.

*marvin.yonathan@ukrida.ac.id

Abstract. This paper proposes a zonation method for training the two reinforcement learning agents. We demonstrate the method's effectiveness in the double snake game. The game consists of two snakes operating in a fully cooperative setting to maximize the score. The problem in this game can be related to real-world problems, namely, coordination in autonomous driving cars and the operation of collaborative mobile robots in warehouse applications. Here, we use a deep Q-network algorithm and the zonation method to train the two agents to play the double snake game collaboratively through a decentralized approach, where distinct state and reward functions are assigned to each agent. To improve training efficiency, we utilize the snake sensory data of the surrounding objects as the input state to reduce the neural network complexity. The results show that after 100 episodes, agents that are trained with the zonation method achieve a game score four times higher than without the zonation method. Furthermore, with the optimized hyperparameters, the agents earn an average game score of 15.4 with just under 140 training episodes. The results demonstrate that the proposed approaches can be used to train collaborative multi-agents efficiently, especially in the limited computing resources and training time environment.

Keywords: Zonation method, double snake game, collaborative multi-agent reinforcement learning, training efficiency

(Received 2023-11-28, Accepted 2023-12-15, Available Online by 2023-12-21)

1. Introduction

Machine learning has been widely used for many applications especially for, prediction, data clustering, and even chatbot [1]–[4]. Methods in machine learning can be classified into three learning paradigms: supervised learning, unsupervised learning, and reinforcement learning (RL) [5]. Supervised learning utilizes data and labels trained in a single process to generate a model that can generate predictions of unseen data. On the other hand, unsupervised learning uses data to train a model that can extract the intrinsic pattern in the corresponding data. In contrast to supervised and unsupervised learning, RL uses trial and error in the training process of an agent to learn the best policy, which can determine an action that can maximize total reward in a particular environment [6]. In RL, we often engage in a complex

environment that is suitable to be solved with deep reinforcement learning (DRL), which is an RL method that uses the prediction ability of neural networks to assist an agent in learning in an environment with a considerable number of actions and states [7], [8].

DRL has been applied to different fields, such as robotics, automobile, and game [9], [10]. The study of DRL in games often aims to minimize the risk of DRL implementation in the real world by creating a virtual world that can simulate actual conditions. This virtual world can be used as the testbed to evaluate the possible DRL solutions and benchmark the agent performance [11], [12]. Different games are commonly used in the study of DRL, including single-player games such as Atari and Super Mario, as well as multi-player games such as Mahjong [13], multi-player Texas Hold'em [14], and StarCraft [15].

The deep Q-Network (DQN) algorithm is typically used in the game to study DRL due to its superior performance compared to the previous algorithms [16], [17]. This algorithm combines neural network and Q-learning algorithms to predict optimal action value and policy in sequential decision-making [5], [7], [18]. DQN has been applied to train agents to play various games, such as multiple classic Atari 2600 games and Snake Game [19]. By utilizing the image pixels of the game screen, DQN-based agents can be trained to play various video games with a decent performance. However, this approach requires a complex neural network and a considerable amount of training time to achieve a sufficient level of performance [5], [20]. To reduce the amount of training time, Sebastianelli et al. proposed an approach that uses sensory data to train the agent under 150 episodes in their snake game environment [5]. The proposed approach works well with a simple neural network, which makes it efficient.

Recently, DQNs have been used in various multi-agent games with different settings, including competitive or cooperative settings. In cooperative setting games, there are generally two approaches for agents' reward functions: common reward function and team-average reward [21]. In the common reward function approach, all agents usually share a common reward function; meanwhile, team-average reward addresses each agent with a different reward function [22]–[24]. Team average reward allowed privacy among agents, recently called the decentralized approach in multi-agent games. Ardi Tampu et al. developed a multi-agent with DQN algorithms for Pong, and Lei Han et al. studied a multi-agent for Starcraft II; both groups successfully used DQN for multi-agent applications [25], [26]. However, both studies require millions of steps to train the agents until they reach a significant performance. Efficiency in multi-agent training time is necessary, mainly when the computing resources and time are limited.

In this paper we proposed a zonation method for training the two reinforcement learning agents to play a double snake game. The proposed double snake game involves two snakes collaborating to maximize the game score. The rules of this game are derived from the classic snake game, such as a point is collected when a snake eats the apple, both snakes are forbidden to hit the edge of the snake field, and a snake is not permitted to strike its own body or another snake. With this double snake game, we can simulate the characteristics of real-world problems, such as the operation of collaborative mobile robots for automated picking systems in warehouse applications and lane coordination and parking coordination for autonomous driving cars [27]–[30]. We train agents by using DQN to play a double snake game collaboratively with the decentralized approach by assigning each agent's different state and reward functions. We use snake sensory data of the surrounding objects to minimize the neural network complexity and training time. The zonation method that we use in this collaborative multi-agent setting is compatible with the simplicity of the neural network and input state that merely needs a small number of episodes to train the multi-agent.

2. Methods

2.1. Double Snake Game

The classic snake game in which the player controls the snake to eat the spawned apple at a random place to maximize the game score. The snake’s body will increase by one pixel and the game score will increase by one point when the snake eats the apple. Moreover, in this classic snake game, the player needs to avoid the collision of the snake’s head either with the wall or with the snake’s body. In this

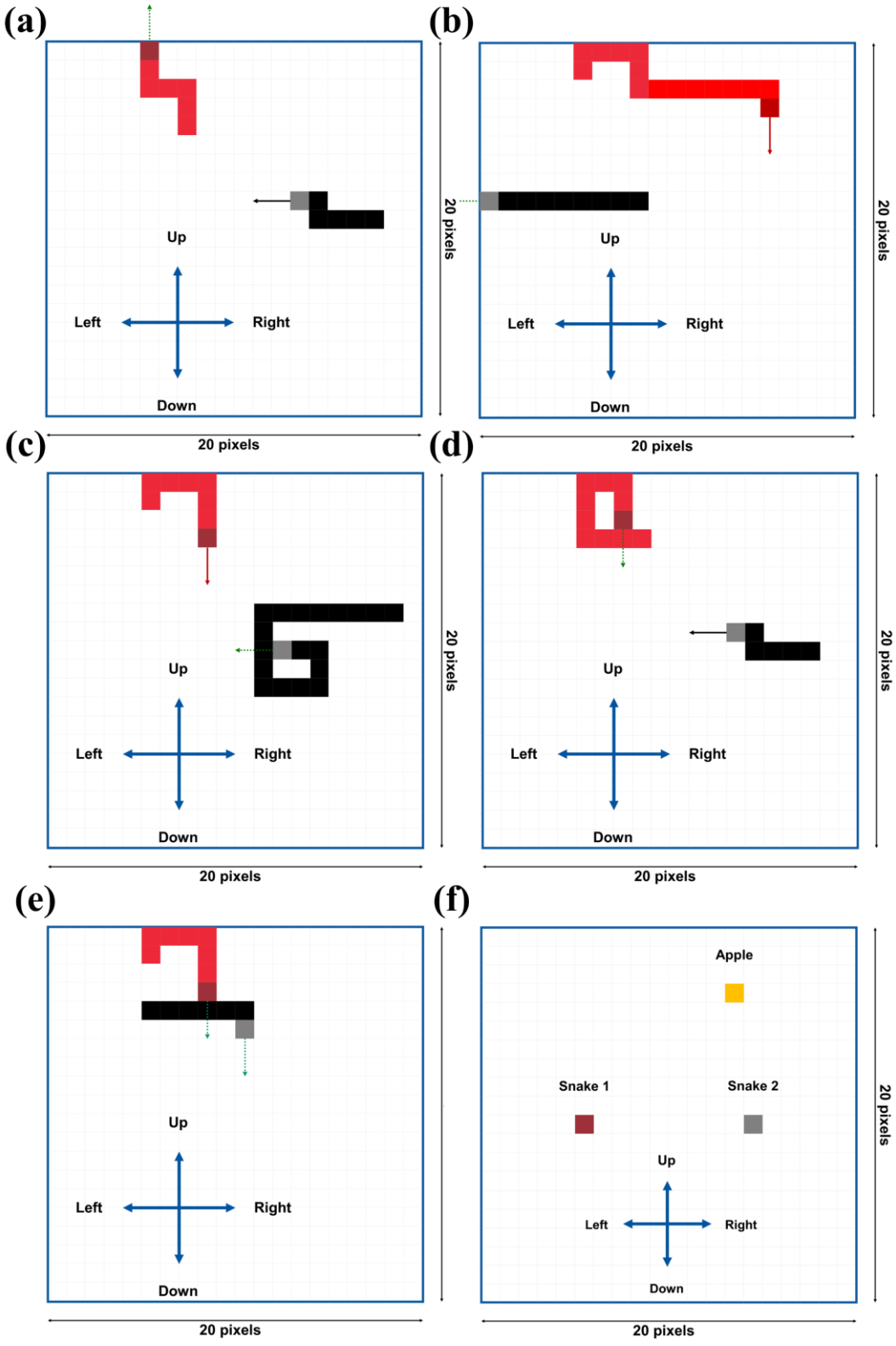
(a)
(b)
(c)
(d)
(e)
(f)
Up
Left
Right
Down
20 pixels
Apple
Snake 1
Snake 2

Figure 1. The game will be terminated **(a), (b)** if a snake head hits the edge of the snake field, **(c), (d)** if a snake hits its own body, or **(e)** if two snakes collide with each other. **(f)** Initial conditions of double snake game.

paper we propose the double snake game which is derived from the classical snake game. The double snake game consists of two snakes, an apple, and snake field with the size of 20 × 20 pixels. Each pixel in the field can only be occupied by an apple or a single snake part (either body or head), if there are two parts of snake (either the same snake or different snake) in the same pixel then the game will be over. When a snake hits the edge of the snake field the game will be over as well. The conditions that can make the double snake game terminate can be seen in **Figure 1**.

The double snake game starts with an apple in a random position and two motionless snake heads, as depicted in **Figure 1(f)**. When a snake eats the apple, when the snake's head is in the same pixel as the apple, the associated snake will elongate by one pixel. At the same time, the apple will be respawned at a new random position, and the score will be increased by one point. Correspondingly, the agent who plays the associated snake will receive 10 reward points. During the game, there is only one apple in the snake field. When a snake hits its own body, or hits another snake, or the edge of the snake field, the game will be over and restarted, and the snake will get -10 points as the punishment.

The input states used to train the agent consist of a vector with 18 elements. Elements of the input states are:

states = [apple up, apple right, apple down, apple left,
danger up, danger right, danger down, danger left,
move up, move right, move down, move left,
flag up, flag right, flag down, flag left,
game over, zone].

Every element in the input state is filled by a binary value of 0 or 1. The first to the fourth element of the input state represents the position of the apple relative to the snake head, and these states will assist the agent in finding the position of positive reward. The fifth to the eighth element of the input state shows the sides of the snake head that are in touch with an object (its own body, another snake, or the wall). These states will assist the agent in avoiding the position of negative reward. The ninth to the twelfth element of the state shows the direction of the snake head movement. This information will assist the agent in calculating the current movement toward the direction of reward. The thirteenth to the sixteenth element of the state shows the position of the zone flag relative to the head of the snake. This zone flag position assists the agent in staying in the zone while waiting for the apple to spawn in the agent's zone. The seventeenth element represents whether the game is still on or over. This is the state that is triggered if the agent takes the wrong action when one or more of the danger states is on. The eighteenth element of the state shows whether the snake is in its zone. When the agent is out of the zone, it will be rewarded negatively. The description of the zone and zone flag will be explained in section 2.3.

2.2. RL Agent

The double snake game is played by RL agents, which consist of neural network that can predict the value of actions when a particular input state is given. In the training process of an agent, the neural network learns a policy that is a function of an optimal action value. This policy is obtained by training the neural network through trial and error to maximize total reward while playing the double snake game. Each time the agent chooses an action, it will obtain a reward that can be a positive or a negative value, depending on the impact of the corresponding action. The agent training process aims to maximize the sum of rewards that can be achieved in a given state.

Figure 2 shows the agent architecture comprising of five fully-connected neural network layers. The first layer is an input layer with 18 nodes to proceed with the 18 input state elements. The subsequent three layers extract the intrinsic features in an input state. These features are then used as the parameters to predict optimal action value. Each layer in this layer group is followed by a Rectified Linear Unit

(ReLU). The last layer is an output layer, which consists of four nodes that generate the prediction value of each action. The agent will execute the action with the highest value.

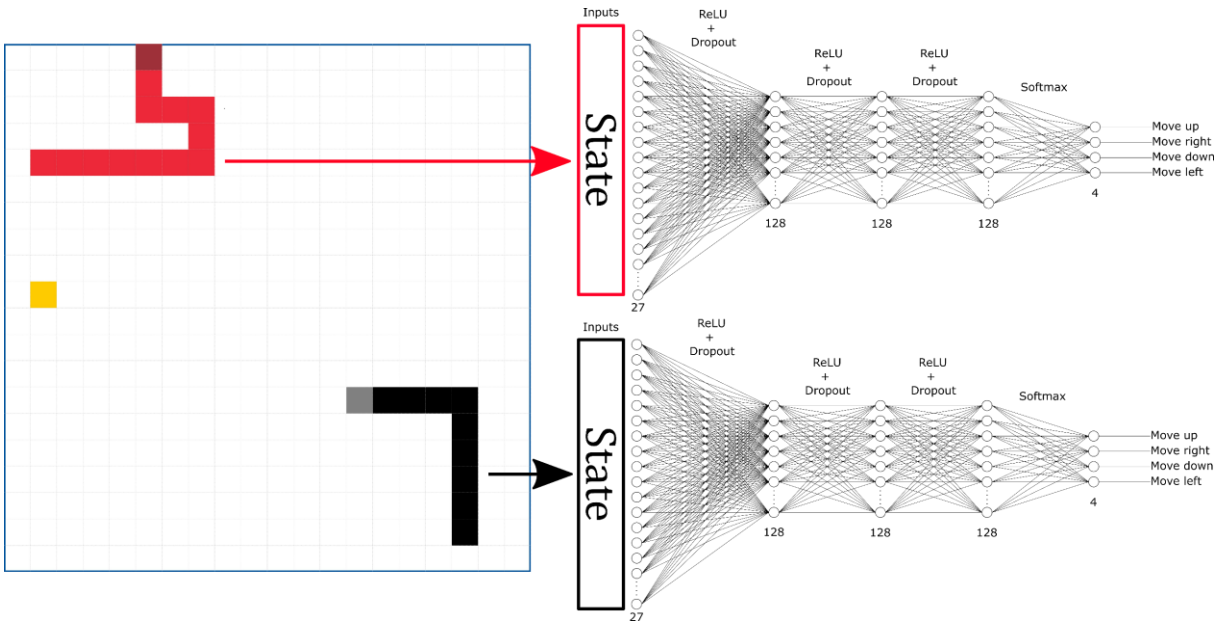

Figure 2. Neural network architecture of the RL agent.

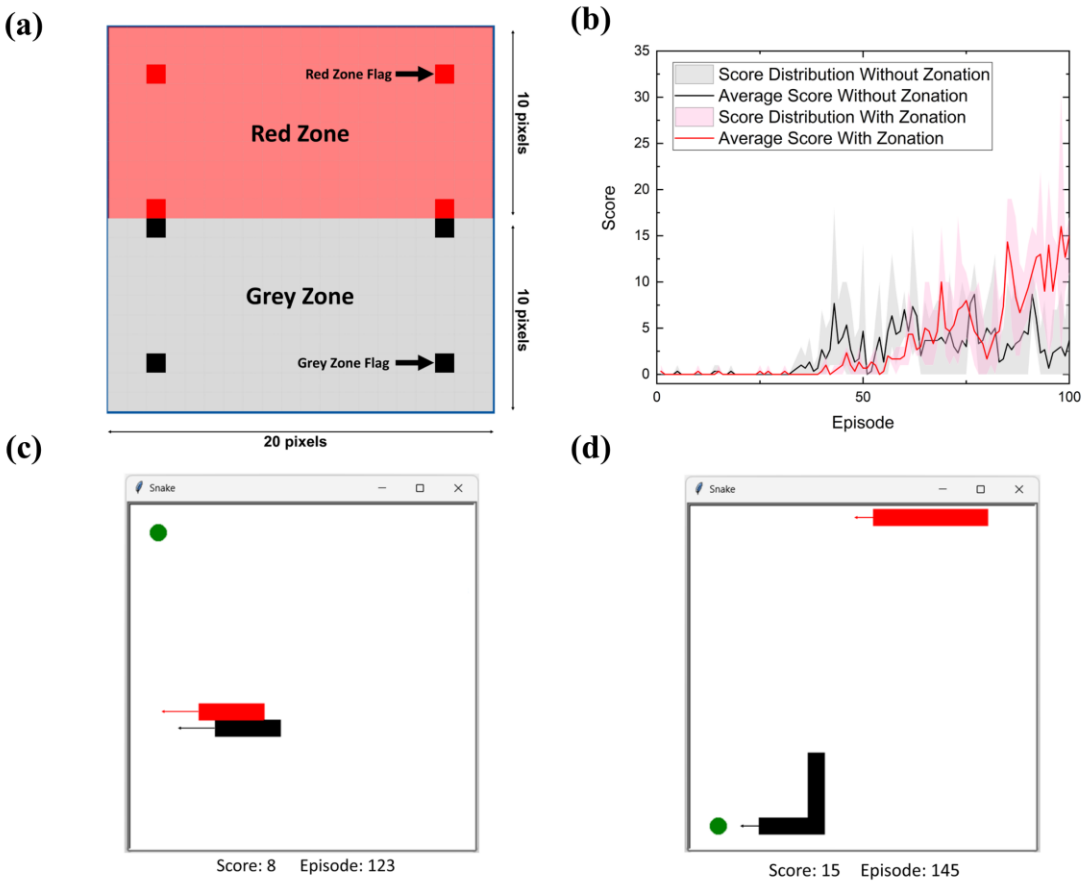

Figure 3. (a) Snake zone coordination in apple consumption for each snake during the game. **(b)** The game score comparison with and without zonation. Behavior of the snakes **(c)** without zonation and **(d)** with zonation.

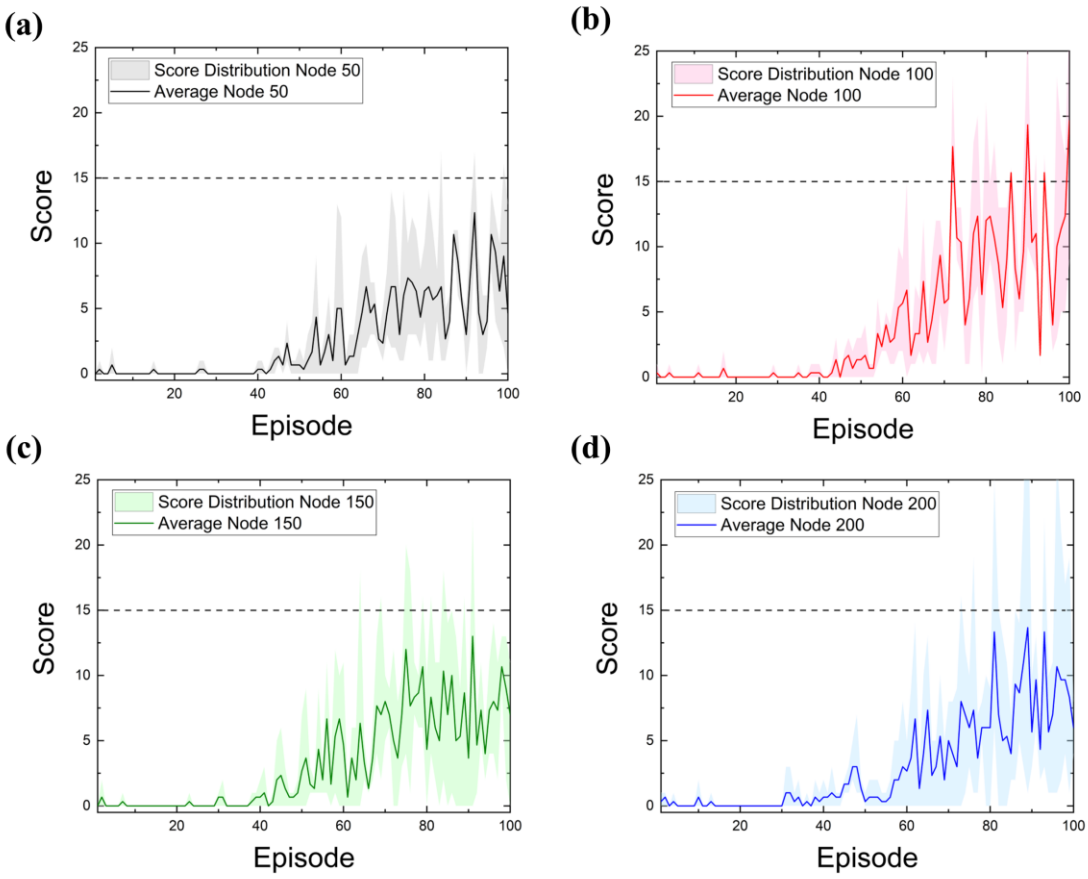

Figure 4. Score distribution of node **(a)** 100, **(b)** 150, **(c)** 200, and **(d)** 250.

2.3. Snake Zonation Method

The snake zonation method is proposed in this paper to optimize the collaboration between the two agents in playing the double snake game. As shown in **Figure 3(a)**, the snake field is divided into two zones: red zone for the first snake and grey zone for the second snake. Each agent is responsible for controlling the snake to eat the apple in their own zone; respectively, the first snake will eat the apple in the red zone, and the second snake will eat the apple in the grey zone. When the apple is spawned in the red zone, the second snake will be waiting in the grey zone by looping around the flag in the respective zone. Conversely, the first snake will be looping around the red zone flag when the apple is spawned in the grey zone. To measure the effectiveness of this method, we compare the agent performance in terms of the game score with and without the zonation method.

3. Results and Discussion

In training the MARL for the double snake game, the results may differ from one trial to another. To visualize the difference, we provide the shade color in the data to elucidate the score distributions during training.

3.1. Zonation

Agents' performance with and without the zonation method can be seen in **Figure 3(b)**. It is shown that the game score without the zonation method is higher in the early episodes, while the agents with the

zonation method show a higher game score in the later episodes. This situation can be caused by the smaller number of states owned by the agents without the zonation method, so fewer training processes are required in neural networks to reach convergence. On the other hand, in the case of the zonation method, the agents have a larger number of states to train. In the higher episodes, we can see that the agents without the zonation method show a steady performance without further improvement. The lower game score resulting from the agents without the zonation method in the higher episode can be attributed to the behavior of the snakes that move closely to each other while searching for the apple as can be seen in **Figure 3(c)**. On the other hand, the agents with the zonation method show a better game score due to the better coordination of snake positioning in the respective zone, as shown in **Figure 3(d)**. We have shown that with the extra input states, including zone flag states and zone states, the training process is much more efficient especially when the training reach more than 70 episodes that resulting the efficient coordination of snakes to search for the apple in their respective zone.

3.2. Hyperparameters Optimizations

The hyperparameters tuned in the optimization are the number of nodes in neural network layers, memory, sampling memory, and episodes. As shown in **Figure 4**, by training the agents for 100 episodes with node variation of 50, 100, 150, 200. We can see that the agents with 100 nodes show the highest game score compared to other node variations, which is the most optimum trade-off between prediction accuracy and neural network complexity. The agent with 50 nodes shows a lower game score due to a lack of complexity in the neural network to be able to predict an appropriate action in each state. Variations with more nodes, such as 150 and 200, suffer from excessive neural network complexity, resulting in a lower game score.

The following optimizations are memory and sampling memory; the memory variation is 1000, 1500, 2500, and 3500, while the sampling memory is 500 and 1000 for every memory variation. To simplify the data analysis, we calculate the average score for every ten episodes and visualize it in a bar chart, as can be seen in **Figure 5** and **Figure 6**. In **Figure 5**, we can see the comparison of average scores for

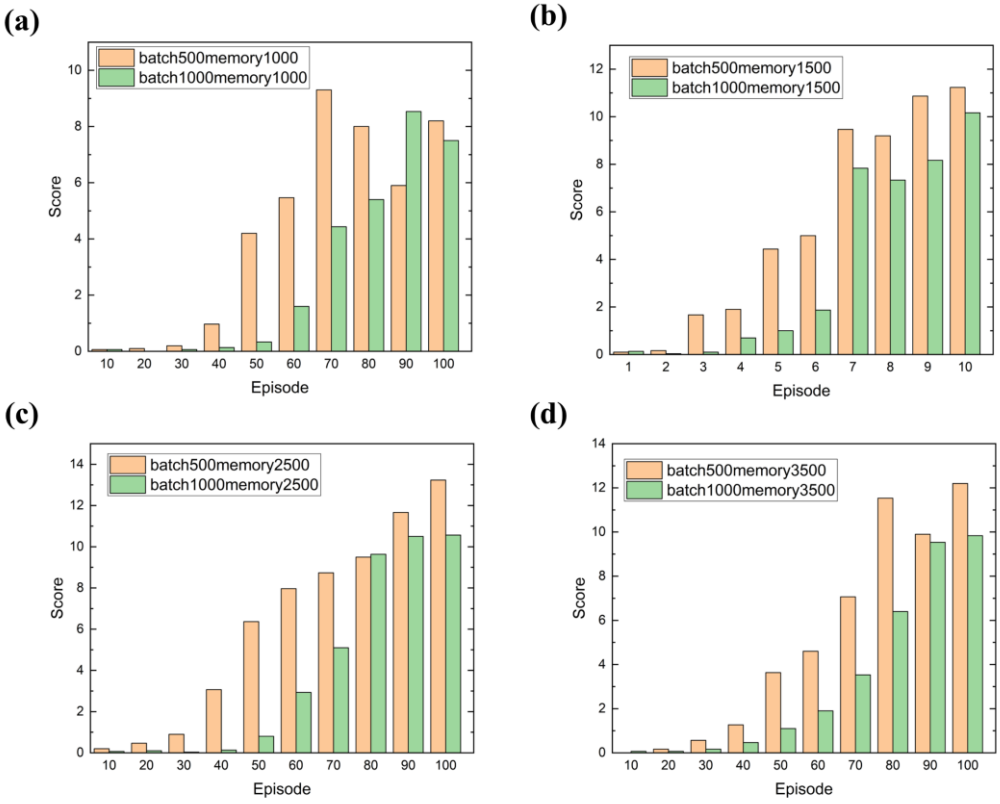

Figure 5. Comparison of average game scores for batches 500 and 1000 in every ten episodes for memory of **(a)** 1000, **(b)** 1500, **(c)** 2500, and **(d)** 3500.

different batches in respective memory sizes. For all memory sizes, it is shown that a batch size of 500 exhibits a higher average score than a batch size of 1000. Furthermore, we can see the average score difference between a batch size of 500 and a batch size of 1000; the average score difference is smaller as the episode increases. The batch size is the size of experience that the agent memorizes; in the lower episode, a batch size of 1000 keeps ineffective experiences, and as the episode increases, the batch size will eventually replace the earliest experience with the latest experience due to the size constraint. At this time, the batch size of 1000 starts to collect more effective experiences, resulting in better agent performance and increasing the average score. However, the batch size of 1000 still holds more ineffective experiences than effective experiences, resulting in a lower average score than the batch size of 500. Apart from the fact that the batch size of 500 holds a sufficient number of effective experiences, it also collects experiences faster than the batch size of 1000. These reasons yield a superior performance of batch size 500 than batch size 1000.

Figure 6 compares the average score for memory 1000, 1500, 2500, and 3500 for a batch size of 500 and 1000, respectively. For batch sizes 500 and batch sizes 1000, the memory size of 2500 exhibits an overall higher average score compared to memory sizes of 1000, 1500, and 3500. From the batch size of 500 and batch size of 1000, we can see a similar trend of the average score when the memory size increases from 1000 to 3500; the average score increases as the memory size increases to 2500, then the average score decreases when the memory size increase to 3500. The improvement of the average score when the memory size increases to 2500, can be attributed to a better prediction from the network of the agent as the network complexity increases, while the reduction of the average score when the memory size increases to 3500 is due to the excessive complexity of the agent's network, this complexity makes the agent learns less efficient than the agent with the memory size of 2500. We can see that when the episodes increase, the average memory score size of 3500 is nearly the average score of 2500.

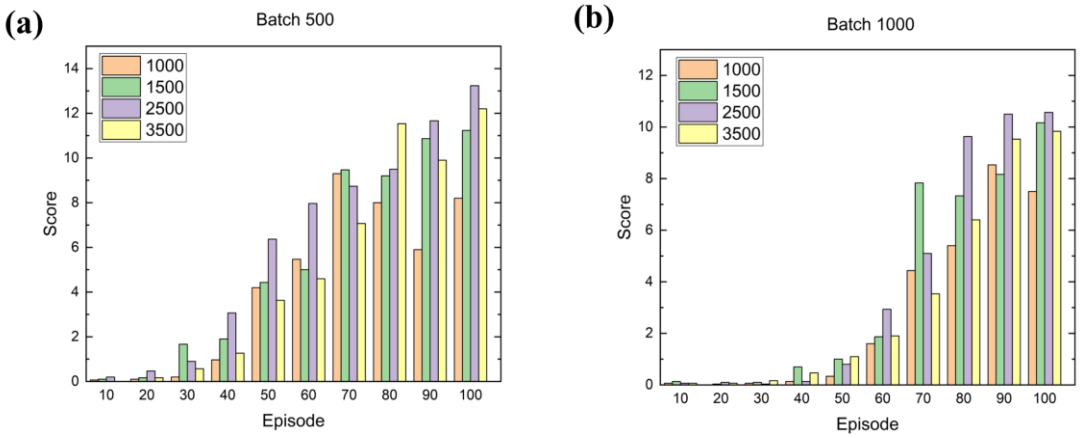

Figure 6. Comparison of average game score for memory 1000, 1500, 2500, and 3500 in every 10 episodes for **(a)** batch of 500 and **(b)** batch of 1000.

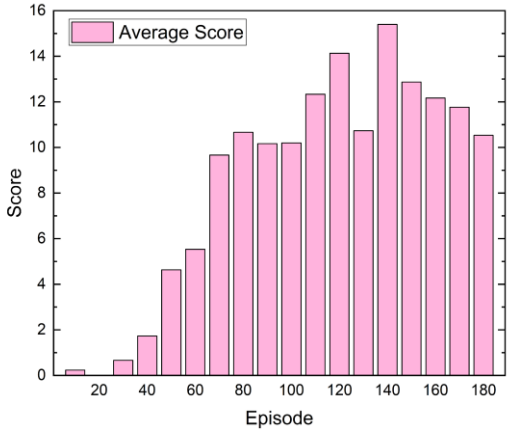

Figure 7. Average game score for every 10 episodes from 1st to 180th episode.

Another hyperparameter we optimize is episode; since we aim to minimize the training process time, the episode is limited to 180 episodes at maximum. By observing the average game score for every ten episodes, we can see that the highest score lies between 120-140 episodes, with an average score of around 15. As can be seen from **Figure 7**, the average game score increases due to more experience in the memory as the episode rises. However, when the memory is full, new memories will replace the early memory, and the agents will eventually forget most of the early memories, especially how to avoid collision; hence, the score will reduce as the episodes increase.

4. Conclusion

In summary, we have demonstrated the zonation method in the double snake as a simple method to improve the collaborative performance between two agents in the game. We have shown that by dividing the field into two zones, red zone, and grey zone, the agents offer a significantly better positioning to avoid collision between two snakes. Furthermore, the hyperparameters can be made efficient by employing sensory data as the input state, that is, under 180 episodes. Under the optimization of hyperparameters, we have shown that the game score increases as the number of nodes, memory size, and episodes increase to an extent, and then the score decreases as those hyperparameters increase. With the optimized hyperparameters, we obtain the highest average game score of 15.4 with respective hyperparameters: the number of nodes of 100, memory size of 2500, batch sampling of 500, and episode of 140. These results suggest that our approaches can be used to train collaborative multi-agents efficiently, especially when computing resources and time are the main considerations. In future work, we will develop methods for agents to play a competitive game and then further develop the complete setup game, including collaborative and competitive game strategy.

Considering the application of MARL in the physical world, we need to ensure the safety issues due to the nature of these agents learning by trial and error. The exploration nature of MARL must be designed carefully to ensure the agents are safe when deployed. Moreover, we need to design a proper environment to safely train the agents.

Acknowledgements

The authors wish to express their gratitude to Lembaga Penelitian dan Pengabdian kepada Masyarakat Universitas Kristen Krida Wacana (UKRIDA) for providing financial support for this publication.

References

[1] M. T. Anwar, S. Nugrohadi, V. Tantriyati, and V. A. Windarni, "Rain Prediction Using Rule-Based Machine Learning Approach," *Adv. Sustain. Sci. Eng. Technol.*, vol. 2, no. 1, May 2020, doi: 10.26877/asset.v2i1.6019.

[2] R. L. Islami and P. R. Sihombing, "Application Biplot and K-Medians Clustering to Group Export Destination Countries of Indonesia's Product," *Adv. Sustain. Sci. Eng. Technol.*, vol. 3,

no. 1, p. 0210105, Apr. 2021, doi: 10.26877/asset.v3i1.8451.
[3] P. R. Sihombing, "Implementation of K-Means and K-Medians Clustering in Several Countries Based on Global Innovation Index (GII) 2018," *Adv. Sustain. Sci. Eng. Technol.*, vol. 3, no. 1, p. 0210107, Apr. 2021, doi: 10.26877/asset.v3i1.8461.
[4] A. S. Dewantara and J. Aryanto, "Implementation Of A Web-Based Chatbot Using Machine Learning For Question And Answer Services In Universities," *Adv. Sustain. Sci. Eng. Technol.*, vol. 6, no. 1, pp. 0240106-01 ~ 0240106-10, 2024.
[5] A. Sebastianelli, M. Tipaldi, S. L. Ullo, and L. Glielmo, "A Deep Q-Learning based approach applied to the Snake game," in *2021 29th Mediterranean Conference on Control and Automation (MED)*, Jun. 2021, pp. 348–353, doi: 10.1109/MED51440.2021.9480232.
[6] R. S. Sutton and Andrew G. Barto, *Reinforcement learning: An introduction*, Second Edi. MIT Press, 2018.
[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.
[8] D. P. Bertsekas, "Feature-based aggregation and deep reinforcement learning: a survey and some new implementations," *IEEE/CAA J. Autom. Sin.*, vol. 6, no. 1, pp. 1–31, Jan. 2019, doi: 10.1109/JAS.2018.7511249.
[9] Y. Li, K. Fu, H. Sun, and X. Sun, "An Aircraft Detection Framework Based on Reinforcement Learning and Convolutional Neural Networks in Remote Sensing Images," *Remote Sens.*, vol. 10, no. 2, p. 243, Feb. 2018, doi: 10.3390/rs10020243.
[10] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 2, pp. 712–733, Feb. 2021, doi: 10.1109/TITS.2019.2962338.
[11] M. Tipaldi, L. Feruglio, P. Denis, and G. D'Angelo, "On applying AI-driven flight data analysis for operational spacecraft model-based diagnostics," *Annu. Rev. Control*, vol. 49, pp. 197–211, 2020, doi: 10.1016/j.arcontrol.2020.04.012.
[12] G. D'Angelo, M. Tipaldi, F. Palmieri, and L. Glielmo, "A data-driven approximate dynamic programming approach based on association rule learning: Spacecraft autonomy as a case study," *Inf. Sci. (Ny).*, vol. 504, pp. 501–519, Dec. 2019, doi: 10.1016/j.ins.2019.07.067.
[13] J. Li *et al.*, "Suphx : Mastering Mahjong with Deep arXiv : 2003 . 13590v2 [cs . AI] 1 Apr 2020," pp. 1–28, doi: https://doi.org/10.48550/arXiv.2003.13590.
[14] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," *Science (80-.).*, vol. 365, no. 6456, pp. 885–890, Aug. 2019, doi: 10.1126/science.aay2400.
[15] M. Samvelyan, T. Rashid, C. Schroeder, and D. W. Gregory, "The StarCraft Multi-Agent Challenge," no. NeurIPS, pp. 1–14, 2019, doi: https://doi.org/10.48550/arXiv.1902.04043.
[16] S. Yoon and K.-J. Kim, "Deep Q networks for visual fighting game AI," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2017, pp. 306–308, doi: 10.1109/CIG.2017.8080451.
[17] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, "Learning to Coordinate with Deep Reinforcement Learning in Doubles Pong Game," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2017, pp. 14–19, doi: 10.1109/ICMLA.2017.0-184.
[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
[19] J. Wang, D. Xue, J. Zhao, W. Zhou, and H. Li, "Mastering the Game of 3v3 Snakes with Rule-Enhanced Multi-Agent Reinforcement Learning," in *2022 IEEE Conference on Games (CoG)*, Aug. 2022, pp. 229–236, doi: 10.1109/CoG51982.2022.9893608.
[20] Z. Wei, D. Wang, M. Zhang, A.-H. Tan, C. Miao, and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," in *2018 IEEE International Conference on Agents (ICA)*, Jul. 2018, pp. 20–25, doi: 10.1109/AGENTS.2018.8460004.
[21] K. Zhang, Z. Yang, and T. Başar, "Multi-Agent Reinforcement Learning: A Selective Overview

of Theories and Algorithms," 2021, pp. 321–384.
[22] W. Zhao, E.-A. Rantala, J. Pajarinen, and J. P. Queralta, "Less Is More: Robust Robot Learning via Partially Observable Multi-Agent Reinforcement Learning," 2023, [Online]. Available: http://arxiv.org/abs/2309.14792.
[23] K. Zhang, Z. Yang, and T. Başar, "Decentralized multi-agent reinforcement learning with networked agents: recent advances," *Front. Inf. Technol. Electron. Eng.*, vol. 22, no. 6, pp. 802–814, 2021, doi: 10.1631/FITEE.1900661.
[24] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Başar, "Fully decentralized multi-agent reinforcement learning with networked agents," *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 13, pp. 9340–9371, 2018.
[25] A. Tampuu *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS One*, vol. 12, no. 4, p. e0172395, Apr. 2017, doi: 10.1371/journal.pone.0172395.
[26] L. Han *et al.*, "Grid-wise control for multi-agent reinforcement learning in video game AI," *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019-June, pp. 4558–4571, 2019.
[27] O. Tanner, "Multi-Agent Car Parking using Reinforcement Learning," *arXiv*, no. June, pp. 1–122, 2022, [Online]. Available: http://arxiv.org/abs/2206.13338.
[28] F. D'Souza, J. Costa, and J. N. Pires, "Development of a solution for adding a collaborative robot to an industrial AGV," *Ind. Robot Int. J. Robot. Res. Appl.*, vol. 47, no. 5, pp. 723–735, May 2020, doi: 10.1108/IR-01-2020-0004.
[29] H. Lee, J. Hong, and J. Jeong, "MARL-Based Dual Reward Model on Segmented Actions for Multiple Mobile Robots in Automated Warehouse Environment," *Appl. Sci.*, vol. 12, no. 9, p. 4703, May 2022, doi: 10.3390/app12094703.
[30] Troullinos and M. Dimitrios and Chalkiadakis, Georgios and Papamichail, Ioannis and Papageorgiou, "Collaborative Multiagent Decision Making for Lane-Free Autonomous Driving," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021, pp. 1335–1343, doi: doi/10.5555/3463952.3464106.