Playing the Snake Game with Reinforcement Learning

Yuhang Pan *[1], Qianli Ma[1], Yiqi Song[2], Bowen Gu[1], Zijun Tang[3], and Junyi Dong[4]

[1]Chien-Shiung Wu College, Southeast University, Nanjing 211189, China
[2]School of Mathematics and Statistics, Beijing Institute of Technology, Beijing 100081, China
[3]School of Science, Jimei University, Xiamen 361021, China
[4]School of Mechanical Engineering and Automation, Northeastern University , Liaoning 314001, China
*Corresponding author: 2441503949@qq.com

June 30, 2023

Abstract

When it is necessary to make several decisions to solve a problem, reinforcement learning works well. In this project, we demonstrate the outcomes of playing with snakes using a Deep Q-Network (DQN). Unfortunately, DQN has a tendency to overfit, which results in a deterioration in terms of scores that are subpar after numerous training sessions. To solve this issue, we enhanced the reward setting and provided the snake with more information about its surroundings in order to raise our score and prevent some overfitting issues. Also, we went through the differences between DQN and conventional evolutionary algorithms as well as possible avenues for DQN optimization.

Keywords:snake game,reinforcement learning,DQN.

1 Introduction

Blockade, a traditional arcade game also known as **Snake**, was published on the Gremlin platform in 1976. It was first only made available on consoles such as the Atari 2600. The game didn't start to become popular around the world until the turn of the twenty-first century, with the release of the Nokia mobile phone. The main objective of the game is to get as many **"apples"** as possible within the given range of the board, thus achieving a higher score. However, as the score increases, the snake's body gets longer, and the board gives it less room to move around, making it increasingly difficult. As shown in Figure 1.

We have summarized the basic rules of this game based on previous research (Brown, de Araujo, & Grichshenko, 2021):

1. The AI can only change the direction of the snake's movement: up, down, left, or right.
2. The snake always moves forward and increases in length by one frame (i.e., one pixel) after eating an apple.
3. If the snake hits its own body or the border of the board, the game ends.
4. The position of the apples on the board is random, and there is only one apple present at any one time.

*Corresponding author: panyh@seu.edu.cn

As shown in the picture and the rules, there are typically two dangerous phases when playing the Snake game: the first happens early on when the snake's body is short and moves slowly, which can cause **distractions** for the human player and result in the snake's death; the second happens later when the human's fingers start to get a little tired and need a break. Then the snake hits its own body, which results in an abrupt end to the game. We wanted to employ AI training to solve these issues because the AI wouldn't become worn out or make mistakes as a result of emotional ups and downs.

In order to realize our goal, we chose a **reinforcement learning** method. In recent years, initiatives such as Go (for instance, AlphaGo) (Silver et al., 2017) have successfully incorporated reinforcement learning. Thus, we tried to adapt this strategy to the snake game. The AI is taught to complete the snake game with a higher score in less time via reinforcement learning. This is one of our group's objectives for the project.

In the Introduction, we outline the fundamental guidelines for the snake game, the cause of the issue that led to this project, and the general approach to finding a solution (a reinforcement learning algorithm). The contributions and accomplishments of our forebears in related domains are presented and summarised in the Background section. Also, we offer a thorough explanation of the fundamental concepts underlying the algorithms we have chosen in the Methods section. After that, we display typical snake scores obtained by our trained AI in the Results section and give a visual representation of our project's outcomes. Lastly, we summarize our contribution to reinforcement learning in video games and provide future directions in the Conclusions & Future Directions section.

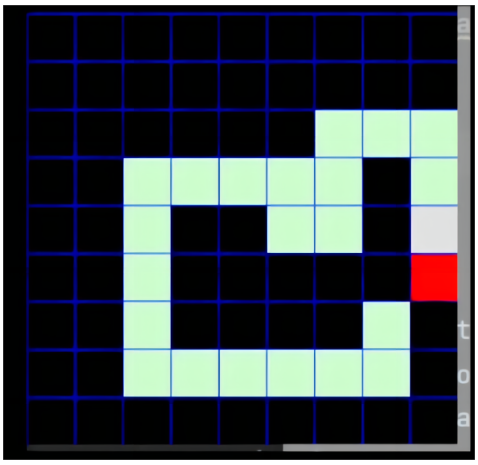


Figure 1: **The Snake Game Demo**: The gray pixel is the head of the snake. The green pixels are the body of the snake. The red pixel is the "apple". And the number of pixels added to the snake's body is indicated by the scores (The figure does not display the scores).

2 Background

In this section, we discuss related work in the field and provide a chronological summary of the contributions.

As shown in Figure 2(a), since **DQN** on the Atari 2600 platform was utilized by the **DeepMind** team to reach levels higher than those of human players (Mnih et al., 2013)(Mnih et al., 2015), there has been some development in the study of **RL** (reinforcement learning) algorithms for use in video games. One of them, Kevin, used DQN to successfully play Flappy Bird in 2(b). However, he also made note of the fact that as the quantity of training sessions rose, **overfitting** began to take place, which led to the AI's playing score gradually declining after a number of training sessions (Chen, 2015). This also occurred in our present project, where the snake's final length (score) began to decline after numerous training sessions and turned into the main challenge to be overcome.

In 2017, world champion Ke Jie was defeated by DeepMind's **AlphaGO Master** (Chouard, 2016), which generated a lot of curiosity in the community and served as the impetus for our team's interest

in artificial intelligence. Since then, teams from DeepMind and OpenAI have successfully used the RL algorithm on Starcraft 2 and Dota 2, respectively (Vinyals et al., 2019) (Berner et al., 2019) in2(c).

Then, the top three teams employed primarily alpha-beta pruning and iterative deepening search, Monte Carlo Tree Search, and A* algorithms in the two "AI Snake" tournaments that Innopolis University organized in 2020 and 2021 2(d). These events pitted teams' AI models against one another in the Snake game. Nonetheless, it was noted that the top three teams primarily used the A* algorithm, Monte Carlo Tree Search, and alpha-beta pruning and iterative deepening search (Brown et al., 2021). Would it be feasible to win the single-player game mode with better scores? This was the starting question of our project.

We thus looked at how **RL** has evolved in video game applications in this section, and we determined the direction of our project to play Snake using the DQN algorithm based on the successes of these predecessors. We give a brief overview of the PPO and DQN approach and present our model optimization for the Snake game in the Methods section that follows.

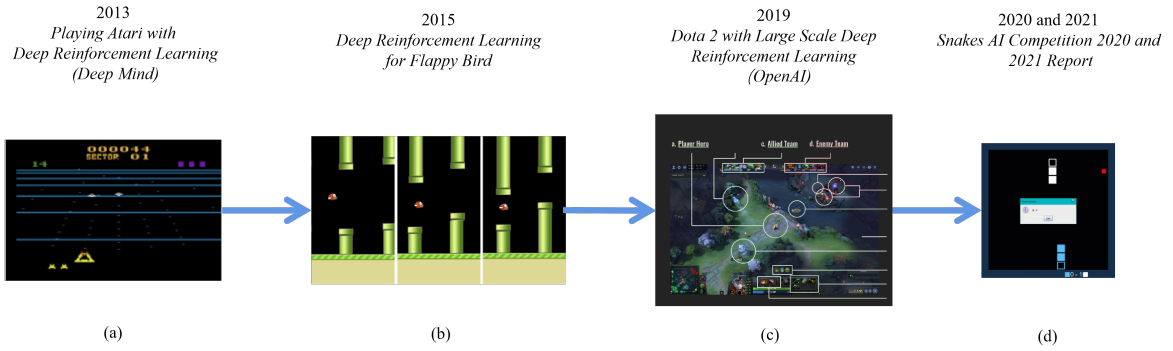


Figure 2: **Development of the application of RL in video games**: (a): An screenshot from the Atari 2600 game Beam Rider, on which DeepMind developed the first deep learning model *to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning*, also known as DQN (Deep Q-network) (Mnih et al., 2013). (b): a collection of screenshots illustrating Flappy Bird's three levels of difficulty (easy, medium, hard), and our team member Kevin showed that deep reinforcement learning has a lot of potential for video games by using the DQN algorithm on Flappy Bird based on the findings of (a)(Chen, 2015). (c): An illustration of the *"Observation Space"* of a person playing Dota 2. In this game, OpenAI used LSTM (Long Short-Term Memory) to build OpenAI Five, which became the first AI to defeat a world champion in eSports (Berner et al., 2019). (d): An illustration of the game's rules from the Snake Competition (Brown et al., 2021).

3 Methods

On the basis of the review of prior work mentioned above, we give a brief analysis of the snake's suitability for use with reinforcement learning in this section. This is followed by an introduction to the category of reinforcement learning, a focus on our two selected algorithms (**PPO** and **DQN**) based on (chuyangliu, 2016) and (Zhou, 2016)., and an analysis of their suitability for use with Snake.

The Snake game presents a typical reinforcement learning problem because it has the following characteristics: 1) The environment is discrete and limited, because the Snake can only move in a limited game area, and can only take limited actions (up, down, left, right); 2) The state of each time step is completely observable because the greedy snake can see its surrounding environment and its own state; 3) The goal is to maximize the game score, that is, get the maximum return.

Therefore, reinforcement learning algorithms are very suitable for training the Snake game. Reinforcement learning involves a machine learning algorithm that interacts with the environment through agents and obtains the maximum return by learning the optimal strategy. The agent in reinforcement learning will make an action according to the state of the environment at each time step and get a reward from the environment. The agent maximizes the total long-term returns through learning, which requires the agent to find an optimal strategy to decide when to take which action.

After conducting literature research, we classified the existing reinforcement learning algorithms, as shown in Figure 3.

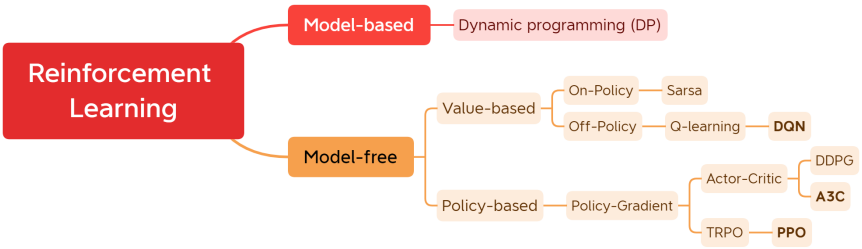


Figure 3: **Classification of Reinforcement Learning**: model-based and model-free are the two main categories. One can further separate the model-free ones into value-based and policy-based categories. The PPO algorithm in value-based and the DQN algorithm in policy-based are the key applications of this research.

Reinforcement learning algorithms can be divided into model-free and model-based categories according to whether or not environmental models are used. The model-based algorithm refers to the use of environmental models in the decision-making process, that is, the ability to learn the dynamic laws and state transition probabilities of an environment, and make decisions on this basis. Model-free algorithms do not require the use of environmental models and learn strategies directly from experience.

Both PPO and DQN algorithms belong to the model-free algorithm. This is because in many practical applications, the state space and action space of the environment are very large, and the dynamic laws of the environment are difficult to model, so the use of model-based algorithms needs to face great challenges. In contrast, model-free algorithms do not require a pre-built environment model and can learn strategies directly from interactions with the environment, so they are more universal.

In the Snake game, it is difficult to build a universal model of the environment because the size of the game map and the location of obstacles can be different. In addition, the states and actions in the snake game are discrete, and the use of model-based algorithms requires a large amount of modeling and calculation of state space and action space, and the computational complexity is very high. In contrast, PPO and DQN algorithms use neural networks to approximate strategy and value functions, which can adapt well to different maps and states, and the calculation speed is also faster. Therefore, it is more appropriate to use the model-free algorithm in the Snake game.

3.1 Algorithm Selection

Based on the above information, we focus on model-free algorithms. They can be divided into policy-based and value-based. We selected the most common PPO and DQN algorithms in these two types of algorithms to experiment on separately.

Algorithms based on value functions, such as DQN, can learn an Action-Value Function (AVF), which can predict the long-term return of each action in each state. Then, the agent can obtain the maximum return by selecting the action with the highest action value. In the Snake game, DQN can predict the return of each action, such as forward, left, right, or down. Then, the agent can select the action with the highest action value to maximize the return. Due to the convergence and stability of the DQN algorithm, in the Snake game, it can obtain higher scores and better game performance.

Algorithms based on policy functions, such as PPO, can learn a policy function, which can select the best action in each state. In the Snake game, PPO can directly learn a strategy function to select the best action to maximize the return. Because the PPO algorithm can achieve faster training speed and a more stable training process while ensuring convergence, the PPO algorithm can obtain higher scores and better game performance in the Snake game.

The PPO algorithm uses a strategy called "clipped surrogate objective" during the training process, which is based on an idea called "proximal policy optimization", the main purpose of which is to maximize the improvement of the strategy while ensuring that each updated policy is not too far

away from the previous strategy. This strategy shows good results in practice, but it also leads to randomness in decision-making.

Specifically, the PPO algorithm will use a probability distribution to represent the strategy during the training process, and then use sampling to update this probability distribution to continuously improve the strategy. At the time of decision-making, the final strategy may also be different because the sample taken each time may be different, thus exhibiting some randomness.

In addition, the PPO algorithm will also use a mechanism called "exploration term" to increase the exploratory nature of the strategy, that is, introduce some randomness into the strategy to explore new states and action combinations. This prevents the algorithm from relying too heavily on existing experience and enhances its ability to explore new states and action combinations, so as to better adapt to different environments.

Therefore, because the PPO algorithm uses random sampling to update the strategy and introduces exploration terms, it shows a certain randomness in decision-making. This randomness can help the algorithm better adapt to different environments and avoid falling into local optimal solutions.

However, when actually trying to use the PPO algorithm, it was found that it was precisely because the PPO still produced randomness when selecting the action at the end, and did not always choose the best action, that the performance was poor. On an 8 by 8 board, the optimal average length is 18. Therefore, we focused on implementing the DQN algorithm and optimizing its performance.

3.2 Deep Q-network (DQN)

DQN (Deep Q-Network) is a reinforcement learning algorithm based on deep learning. It combines a Q-Learning algorithm and deep neural network to deal with reinforcement learning problems in large state space and action space(Mnih et al., 2013)(Mnih et al., 2015).

In the DQN algorithm, we use a neural network to approximate the **Q** function, that is, state and action as inputs, and output the corresponding **Q** value. Specifically, we can express the DQN algorithm as comprising the following steps:

1. Initialize the neural network $\boldsymbol{\theta}$ and use it to approximate the **Q** function $\boldsymbol{Q(s, a; \theta)}$.
2. At each time step $\boldsymbol{t}$, observe the current state $\boldsymbol{s_t}$ and use the $\boldsymbol{\epsilon}$-greedy strategy (that is, select the action that maximizes the **Q value** with a certain probability, and select the random action with a certain probability) to select the action $\boldsymbol{a_t}$.
3. Execute the action $\boldsymbol{a_t}$ and observe the environmental feedback $\boldsymbol{r_{t+1}}$ and the next state $\boldsymbol{s_{t+1}}$.
4. Calculate the target **Q** value $\boldsymbol{y_t}$, namely:

$$\boldsymbol{y_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta^-)} \tag{1}$$

   Where, $\boldsymbol{\gamma}$is the discount factor, and $\boldsymbol{\theta^-}$is a fixed target network for stable training.

5. Update the neural network parameter $\boldsymbol{\theta}$ by minimizing the objective function:

$$\boldsymbol{L(\theta) = \frac{1}{2}(y_t - Q(s_t, a_t; \theta))^2} \tag{2}$$

6. Update the target network $\boldsymbol{\theta^-}$ to the parameter value of $\boldsymbol{\theta}$.

In the Snake game problem, we can use the DQN algorithm to train a neural network to predict the Q value of different actions under a given state. Specifically, we can express the **status** of the snake as a vector, including the position of the snake, the position of the food, the direction of the snake's head, and the length of the snake's body. For an action, we can express it as a discrete value, indicating that the snake moves up, down, left, or right.

In the training process, we can use some techniques to improve the performance and convergence speed of the algorithm. For example, we can use **Experience Replay** to store and reuse previous sample data to avoid over-fitting and improve sample utilization; We can also use a fixed target network to stabilize training and reduce divergence risk; We can also use **Double Q-Learning** to

reduce estimation bias and variance and improve learning efficiency and stability(Van Hasselt, Guez, & Silver, 2016).

In short, the DQN algorithm is a reinforcement learning algorithm applicable to large state space and action space, which can effectively learn an excellent strategy in the greedy snake problem.

In the concrete implementation, we can use the depth neural network to approximate the Q function, for example, and use the convolution neural network (CNN) to process the image input of the game. We can take the pixel value of the game picture as the input of the neural network, and then use the DQN algorithm to train the network to predict the **Q value** of different actions under a given game picture. Through continuous iterative training, we can gradually improve the performance of the neural network, thus realizing the automatic control of the Snake game.

It should be noted that in the Snake problem, the state space and the action space are discrete, which means that we need to use appropriate techniques to deal with this dispersion. For example, we can use **one-pot coding** to represent states and actions so that they can be used as inputs and outputs of neural networks. In addition, in the training process, we need to ensure that the neural network can be effectively updated under different states and action combinations to avoid over-fitting or under-fitting problems.

3.3 Model Optimization

As the input of the agent model, the **states** in the DQN algorithm is expected to contain all information of the game, so the most ideal way is to put the whole game board as the states. However, inputting raw information is not conducive to feature extraction by neural networks. To simulate the vision of the game snake, we first set the block type around the snake head shown in Figure 5(a). Specific parameter settings in the original model are shown in Table 1 below:

Table 1: **The specific parameter settings of the initial original model**

Block Types	Values
Empty	0
Food	1
Wall	2
Body	3

An obvious problem for the original model is that it only represents the local status near the snake head: this makes the output (decision-making) of the agent too localized and it is hard to make long-term optimal decisions. Therefore, it is easy to foresee that the final game score will not be good enough. To solve this problem, it may be useful to increase the amount of information carried by states. Thus, we mainly focus on two directions:

- Replace the type of individual block with information on its direction
- Quantify the **"proximity"** to the body/food/wall using distance

With this in mind, we came up with a new design of states after several trials to contain as much global information as possible. This is the state composition for every direction in total **8 directions** shown in Figure 5(b):

Table 2: **The best status settings after many adjustments and training sessions**

Status	Description	Type
see_body	Whether the snake sees its body in that direction	0,1
see_food	Whether the snake sees food in that direction	0,1
dis_body	distance to its body	Integer
dis_food	distance to the food	Integer
dis_wall	distance to the wall	Integer

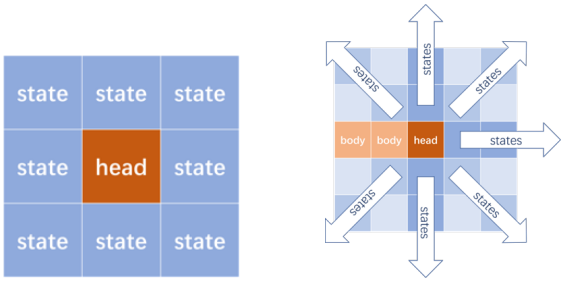
With the above adjustment, the dimension of states has increased significantly from only 8 to $8\times5=40$ which can better represent the game status.

As for the parameters in the model, the DQN uses rewards to evaluate each of the agent's actions and update the weight in the model. We finally found the optimal reward settings by extensive training and adjustment(Godbole, Dahl, Gilmer, Shallue, & Nado, 2023). This ensures the DQN reaches the highest score while training faster:

Table 3: **The best reward settings after many adjustments and training sessions**

Action	Reward
Hit body/wall	-24
Eat one piece of food	+20
Get close to food	+0.3
Do nothing	-0.5

This kind of setting is quite reasonable. The punishment of death should be a little heavier than eating one piece of food, otherwise, the snake grows slower after eating several pieces because death doesn't cost much. At the same time, a small but necessary negative reward for doing nothing works to prevent the snake from hanging around without eating the food.



(a) Original (b) Modified

Figure 4: **The Settings of the State**: The information around the snake's head was initially confined to just 8 pixels, which caused the snake to act in an excessively constrained manner, ultimately lowering our score. This is what is depicted in (a). The state was then optimized, as demonstrated in (b). In order to help the snake make better decisions, we provided information about the two pixels closest to it in seven different directions. The snake now has more knowledge about the area around its head. And we can get higher scores.

In this section, we introduce the classification and PPO principles for reinforcement learning and DQN and analyze how they apply to the snake. The visualization of the outcomes of our play with the Snake using these two algorithms is shown in Section 4.

4 Results

In this section, we will demonstrate the results of the DQN Algorithm. After changing the setting of the state and reward, we finally get the result shown in Figure 5.

As shown in Figure 5, the average length of the snake has increased from 9 in the original model to 22 in the final model. The results show that DQN has better performance than PPO according to our model.

5 Conclusions & Future Directions

5.1 Conclusions

Our results show that the DQN algorithm doesn't exhibit better performance than searching algorithms such as the Greedy algorithm, for example. It has been shown that the average length of a snake can reach 60.15 using the Greedy algorithm (chuyangliu, 2016). In the Snake game, PPO and DQN do not seem to perform better than evolutionary algorithms (Yeh, Su, Huang, & Chiang, 2016).

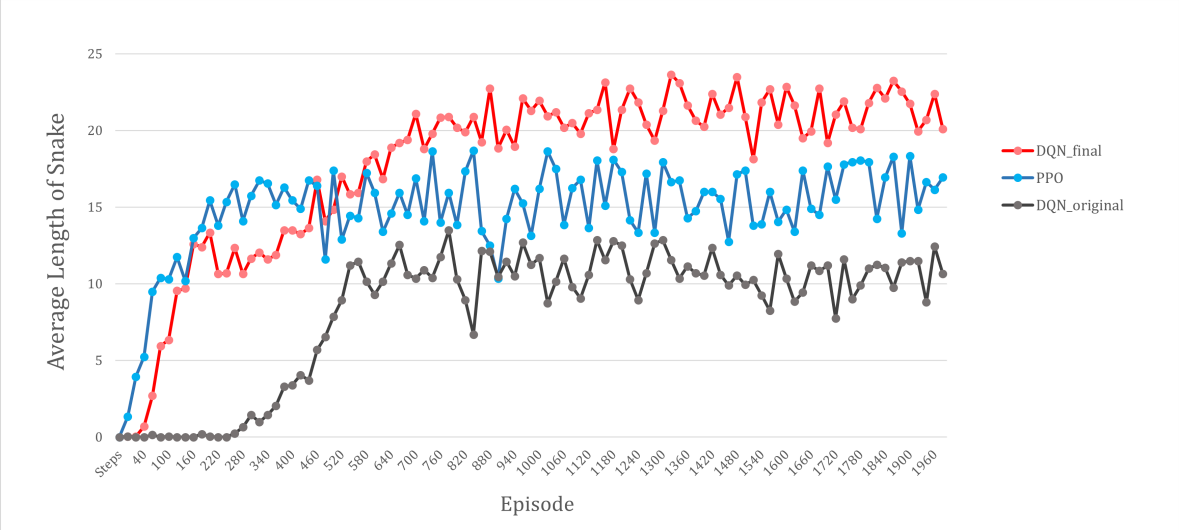


Figure 5: **The results of the research**: The graph's y-axis shows the average length of the snake, while the x-axis shows how many times the algorithm has been run. The figure represents the red curve as the final DQN model, the blue curve as the PPO model, and the grey curve as the original DQN model.

In addition, in machine learning, trials really matter. We gradually improved the performance of the agent by modifying the setting of state and reward. However, our research also has some limitations. Firstly, we only use DQN and PPO algorithms without any comparison to other algorithms. Secondly, our results indicate that our model is over-fitted.

5.2 Future Directions

Based on the limitations mentioned in 5.1, we have identified some future directions to overcome these limitations. Firstly, We plan to improve the performance of DQN by using Bootstrap and Double DQN. Also, we plan to modify the setting of the state by, for example, recording the information of the whole game board and using the convolution neural network (CNN) to extract the features of the information as state. Finally, we plan to compare other reinforcement learning algorithms to DQN with respect to their performance in the Snake game.

Table 4: **Plans and examples of future goals that are anticipated to be achieved**

Plans	Examples
Improvement of DQN	Bootstrap,Double DQN
Modifying of state	The whole board
Other algorithms	PPO,GA

6 Patents

Author Contributions: Conceptualization: Y.P.,Q.M.,Y.S.,B.G.,Z.T. and J.D.; methodology, validation, formal analysis: Y.P.,Q.M. and Y.S.; investigation, resources: Y.P.,Q.M.,Y.S.,B.G. and Z.T.; writing—original draft preparation, visualization: B.G.,Z.T. and J.D.; writing—review and editing, visualization, supervision: Y.P.,Q.M.,Y.S.,B.G.,Z.T. and J.D.; Project administration: Y.P. and J.D.
Funding: This research received no external funding.
Research Guidelines: This study has followed the research guidelines of Advanced Machine Learning and Machine Intelligence, Cambridge Programme 2023.
Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability: Please contact the corresponding author for all reasonable requests for access to the data.

Acknowledgments: We would like to give our thanks to our faculty professors and teachers at Cambridge University for their guidance.

Conflicts of Interest: The authors declare no conflict of interest.

Intellectual Property: The authors attest that copyright belongs to them, the article has not been published elsewhere, and there is no infringement of any intellectual property rights as far as they are aware.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... others (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Brown, J. A., de Araujo, L. J. P., & Grichshenko, A. (2021). Snakes ai competition 2020 and 2021 report. *arXiv preprint arXiv:2108.05136*.

Chen, K. (2015). Deep reinforcement learning for flappy bird. *CS 229 Machine-Learning Final Projects*.

Chouard, T. (2016). The go files: Ai computer wraps up 4-1 victory against human champion. *nature*. Retrieved from `https://www.nature.com/articles/nature.2016.19575`

chuyangliu. (2016). *Snake.* Retrieved from `https://github.com/chuyangliu/snake`

Godbole, V., Dahl, G. E., Gilmer, J., Shallue, C. J., & Nado, Z. (2023). *Deep learning tuning playbook.* Retrieved from `https://github.com/google-research/tuning_playbook` (Version 1)

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... others (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354–359.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 30).

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... others (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Yeh, J.-F., Su, P.-H., Huang, S.-H., & Chiang, T.-C. (2016). Snake game ai: Movement rating functions and evolutionary algorithm-based optimization. In *2016 conference on technologies and applications of artificial intelligence (taai)* (pp. 256–261).

Zhou, M. (2016). *Reinforcement-learning-with-tensorflow.* Retrieved from `https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow`