Autonomous Agents in Snake Game via Deep Reinforcement Learning

Zhepei Wei[†], Di Wang[§], Ming Zhang[†], Ah-Hwee Tan[§¶], Chunyan Miao[§¶], You Zhou[†*]
[†]College of Computer Science and Technology
Jilin University, Changchun, China
[§]Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly
[¶] School of Computer Science and Engineering
Nanyang Technological University, Singapore
[*]Corresponding Author
zhepei.wei@gmail.com, wangdi@ntu.edu.sg, zhangming0722@gmail.com, {asahtan, ascymiao}@ntu.edu.sg, zyou@jlu.edu.cn

***Abstract*—Since DeepMind pioneered a deep reinforcement learning (DRL) model to play the Atari games, DRL has become a commonly adopted method to enable the agents to learn complex control policies in various video games. However, similar approaches may still need to be improved when applied to more challenging scenarios, where reward signals are sparse and delayed. In this paper, we develop a refined DRL model to enable our autonomous agent to play the classical Snake Game, whose constraint gets stricter as the game progresses. Specifically, we employ a convolutional neural network (CNN) trained with a variant of Q-learning. Moreover, we propose a carefully designed reward mechanism to properly train the network, adopt a training gap strategy to temporarily bypass training after the location of the target changes, and introduce a dual experience replay method to categorize different experiences for better training efficacy. The experimental results show that our agent outperforms the baseline model and surpasses human-level performance in terms of playing the Snake Game.**

***Index Terms:* Deep reinforcement learning, Snake Game, autonomous agent, experience replay**

I. Introduction

Reinforcement learning has been applied to play simple games decades ago, one of the probably most well-known models is TD-gammon [1], which plays the backgammon game and surpasses human-level performance. However, this method shows little generalization to other games and did not attract wide attention. As a recent breakthrough in deep learning, DeepMind creatively combined deep learning with reinforcement learning and came up with the prominent deep Q-learning network (DQN) model [2]. DQN outperforms all the prior approaches on six games and surpasses human-level performance on three games. This breakthrough lit up researchers' passion and many similar researches (e.g., [3, 4]) soon emerge.

However, DQN may not be straightforwardly applied to all scenarios because its naive reward mechanism only produces sparse and delayed rewards that may lead to ineffective learning of correct policies [5]. In this paper, we devise a carefully designed reward mechanism to estimate the immediate rewards based on the environmental changes directly collected from the game as effective learning signals for our autonomous agents.

Moreover, in most reinforcement learning problems, a method named experience replay is often adopted to decrease the correlation of sampled experiences when training the network [6]. Lin [6] proved the experience replay smooths the training distribution over a large amount of experiences. However, this method samples previous experiences randomly without considering their quality. To solve this problem, Schaul et al. [7] proposed an improved approach named "prioritized experience replay". Inspired by [7], in this paper, we propose an approach named *dual experience replay* to further distinguish the valuable and ordinary experiences by storing them into two independent memory pools. As such, the training of the agent is better guided.

In this paper, our agent learns how to play the Snake Game based on the screen snapshots. Specifically, we rely on a deep Q-learning network (DQN) to choose the best action based on both the observations from the environment and prior learned knowledge. We use a series of four screenshots of the Snake Game as the network input. Therefore, the network is able to capture the game information including direction and position, and then output the estimated Q-value of each action. Training an agent to successfully learn to play this Snake Game is quite challenging because the restriction of the Snake Game gets stricter as the snake grows in length. Moreover, once an apple is eaten by the snake, a new one is immediately spawned at a random location. To better handle this changing target issue, we introduce a training gap strategy to temporarily bypass training and to provide enough time for the snake to perform necessary maneuvers. The experimental results show that our agent outperforms the baseline DQN model and surpasses human-level performance in terms of both game scores and survival time.

II. Game Environment and Technical Foundation

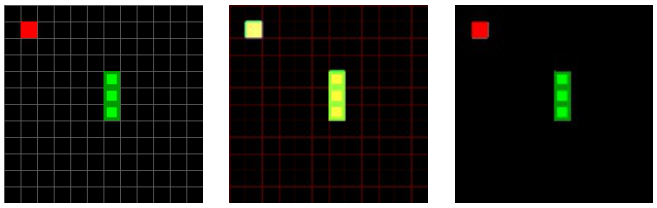
In this section, we introduce the Snake Game environment and the technical foundations of a typical deep reinforcement learning model.

A. Game Environment

Snake Game is a classical digitized game, in which the player controls the snake to maximize the score by eating apples spawned at random places. One and only one apple appears in the game screen at any time. Moreover, because the snake will grow one grid in length by eating an apple, avoiding collision is key to its survival. In this work, we implemented the Snake Game in Python as the testbed of autonomous agents. We adopted the settings of the Snake Game as recommended by [8]. Specifically, the size of the game map is set to 240×240 in pixel and divided into 12×12 equally large grids. The initial length of the snake is 3, initial direction is to the right, and the snake as well as the apple are randomly deployed when a game starts. The game score is initialized to 0 and will increase one as the snake reaches a target. Moreover, the collision of snake will lead to the end of a game and the game score will be reset to 0 at the start of the new game. Due to the re-spawn of a new apple when the previous one is eaten, the target of the snake changes during a game trial upon reaching a previously determined target. Hence, being able to localize new targets in an adaptive manner is crucial to the agents playing the Snake Game. The number of control signals in the Snake Game is four, namely UP, DOWN, LEFT and RIGHT. At each time step, the snake moves one grid forward along its current direction, unless the control signal received is an orthogonal direction. Snake Game is a complex and challenging reinforcement learning environment that has seldomly been studied in the literature. In this paper, we propose a refined DQN model with mainly three technical improvements and apply it to enable an autonomous agent to play the Snake Game.

B. Technical Foundation

Deep Q-Network (DQN) is firstly presented by Mnih et al. [2] to play Atari 2600 video games in the Arcade Learning Environment (ALE) [9]. DQN demonstrates its ability to successfully learn complex control policies directly from raw pixel inputs. Technically speaking, DQN is a convolutional neural network (CNN) trained by a variation of the classical Q-learning algorithm [10], using the stochastic gradient descent method to tune the weights. DQN advances traditional reinforcement learning techniques as it employs CNN to approximate the Q-function, which provides a mechanism to estimate Q-values of possible actions directly from the most recently observed states (pixels). To keep the iterative evaluations stable, DQN uses mini-batches of experiences to train the neural network. Specifically, each experience is manifested as a four-tuple (s, a, r, s'), where s is the state of the observed environment, a is the action that agent performs in state s. After agent executes action a in state s, it receives the reward r from the environment and goes into the next state s'. Along game play, agent stores the experience in memory for subsequent sampling and training of CNN. This is known as *experience replay* [6]. In addition, DQN uses former network parameters to evaluate the Q-values of the next state, which provides a stable training target for CNN [11].



(a) RGB Image (b) HSV Model (c) Final Input

Fig. 1. Three snapshots at different phases during input preprocessing.

In our work, state s is preprocessed and denoted as $\phi(s)$. The agent selects actions using an ϵ-greedy strategy [12], in which the best action determined based on the approximated Q-value function is executed with probability $1 - \epsilon$, or else, a random action is executed with probability ϵ. Moreover, we propose a variant of the memory replay strategy, aiming to distinguish the importance of different experiences. The technical details of our refined DQN model are provided in the following section.

III. Proposed DQN to Play the Snake Game

In this section, we provide the technical details of our refined DQN model in the respective subsections.

A. Sensory Input Preprocessing and Model Architecture

To reduce the dimensionality of the raw sensory input in pixel, we resize the original RGB snapshots (size of $240 \times 240 \times 3$) to $64 \times 64 \times 3$ images. To avoid interferences by the game background, we first convert the RGB image into HSV color model as a temporary intermediate status and then turn it back to RGB as the final input. In HSV color model, we are able to eliminate the unnecessary grids in the game background by applying bitmask and outputting the result to RGB. Thus, as shown in Fig. 1, the final input is more suitable as the input for the agent to learn from.

In practice, we use the most recent four preprocessed frames as the network input. The reason is because a single frame input lacks necessary information and will fail to recognize the moving direction of the snake. Therefore, by stacking four preprocessed frames, the input of CNN is a three-dimensional array of size $64 \times 64 \times 12$. Furthermore, there are three convolutional layers and one fully-connected layer between the input and the output layers of our network. The first convolutional layer (Conv1) comprises 32 7×7 filters with stride 4. Moreover, Conv2 comprises 64 5×5 filters with stride 2 and Conv3 comprises 128 3×3 filters with stride 2. All the three convolutional layers employ the same activation function ReLU. The last hidden layer is a fully-connected layer (Fc1) and comprises 512 rectifier units. Lastly, the output layer is a fully-connected linear layer that outputs the corresponding predicted Q-value of each action. As the number of possible actions in the Snake Game is 4, the output layer comprises 4 corresponding units. The detailed network architecture is summarized in Table I.

TABLE I
NETWORK ARCHITECTURE

Layer	Filter	Stride	Number	Activation	Output
Input					64*64*12
Conv1	7*7	4	32	ReLU	16*16*32
Conv2	5*5	2	64	ReLU	8*8*64
Conv3	3*3	2	128	ReLU	4*4*128
Fc1			512	ReLU	512
Output			4	Linear	4

B. Dual Experience Replay

Because the input to the network is continuous in time, the subsequent states may be highly correlated, which may lead the algorithm to an unwanted local optimum or even divergence [13]. To solve this issue, experience replay [6] has been introduced to break the correlation. However, this method fails to distinguish the importance of different experiences as it samples randomly from the memory pool without considering their quality. In this paper, we introduce two independent experience sets MP_1 and MP_2, in which any experience is stored according to its quality in terms of reward. We define valuable experiences as those with higher reward, i.e., ≥ 0.5, and store them in MP_1. Although has lower reward values, the other experiences stored in MP_2 are still necessary in the training process, because they provide the agent opportunities of further exploration. The selection between the two memory pools is regulated by a proportion parameter η, which controls the experience sampling from MP_1 and MP_2 in a similar manner as the ϵ-greedy strategy, i.e., setting different sampling proportion η and $1-\eta$ for MP_1 and MP_2 respectively. In this paper, we heuristically initialize η to 0.8, which determines that our agent will mostly learn from the valuable experiences stored in MP_1 in the initial game stage. The value of η gradually decreases as the training progresses and finally fixes at 0.5, which means the agent treats valuable and ordinary experiences equally. This strategy is designed to help the agent learn from the valuable knowledge in the initial phase to firstly reach a certain level of performance and then fine tune the action selection strategies by exploring more (relatively speaking) from the ordinary knowledge. However, due to the constraint in terms of physical memory, the size of the experience pool cannot be set to an overly large number. In this paper, we set N to 1,000,000. Furthermore, MP_1 and MP_2 are set to the same size of $N/2$, i.e., 500,000. The detailed dual experience replay strategy is summarized in Algorithm 1.

Algorithm 1 Deep Q-learning with Dual Experience Replay

Require: replay memory MP_1 and MP_2, capacity of memory pool N, state s_t, action a_t, reward r_t, experience e_t at timestep t and approximated action-value function Q
Initialize replay memory MP_1 and MP_2 to capacity $N/2$
Initialize Q with random weights
for all training steps **do**
  Initialize state s_1 for the new episode
  Preprocess $\phi_1 = \phi(s_1)$
  repeat
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_{a \in A} Q(\phi(s_t), a)$
    Decay exploration probability ϵ
    Execute a_t in game then observe r_t and s_{t+1}
    Preprocess $\phi_{t+1} = \phi(s_{t+1})$
    if $|r_t| \geq 0.5$ **then**
      Store $e_t = (\phi_t, a_t, r_t, \phi_{t+1})$ in MP_1
    else
      Store $e_t = (\phi_t, a_t, r_t, \phi_{t+1})$ in MP_2
    Sample minibatch of e_k from MP_1 and MP_2 with proportion η and $(1-\eta)$, respectively
    Decrease sampling proportion η
    if episode terminated at ϕ_{k+1}
      Target value $Q_k^* = r_k$
    else
      $Q_k^* = r_k + \gamma max_{a \in A} Q(\phi_{k+1}, a)$
    Define loss function $loss = (Q_k^* - Q(\phi_k, a_k))^2$
    Update neural network parameters by performing optimization algorithm Adam [14] on $loss$
  until episode terminates
end for

C. Reward Mechanism

To prevent having overly large estimated rewards, reward clipping techniques are adopted by many reinforcement learning models (e.g., [15–17]). In our work, the reward is clipped within the [-1,1] interval. Specifically, our agent receives the highest reward of 1 when it eats an apple and receives the lowest reward of -1 when it crashes into the wall (boundary of the game map) or itself. In the original DQN, the agent receives reward of 0 when neither of the above two cases has happened. Furthermore, we propose a carefully designed reward mechanism to generate an immediate reward at each time step, which is regulated by the following three components: distance reward, training gap and timeout strategy, which are introduced as follows:

1) ***Distance reward****:* This reward is initialized to 0 and adjusted by our defined *distance reward function (DRF)* (see (1)). Specifically, as the agent approaches or moves away from the target, it is awarded or penalized based on its length and its distance from the target, which can be described as follows: $reward = reward + \Delta r$, where Δr is determined by DRF. By intuition, the reward should be inversely proportional to the distance to the target. In other words, the closer the agent is to the target, the higher reward should be given to the agent. In practice, we let one step towards the target produces a fundamental unit of positive Δr and a step away from the target produces a fundamental unit of negative Δr.

In Addition, Δr is also affected by the snake's length. Due to the nature of the Snake Game, as the agent's length grows longer, it gets easier to crash into itself. Therefore, to keep the agent survive longer, more tolerance should be given when it is in the maneuver process, which is necessary to avoid self-

crashing. As such, as the agent's length gets longer, it should receive less penalty when moving away from the target.

Technically, we detect the agent's moving direction by computing the change of distance between the target and the head of the snake. At time step t, we denote the snake's length as L_t and denote the distance between the target and the head of the snake as D_t. According to the snake's moving direction, which is determined by a_t, it is easy to get the subsequent location of the head of the snake. As such, we denote the distance in the subsequent time step as D_{t+1}. Based on D_t and D_{t+1}, the DRF is able to provide positive or negative feedback accordingly, which is mathematically defined as follows:

$$\Delta r(L_t, D_t, D_{t+1}) = \log_{L_t} \frac{L_t + D_t}{L_t + D_{t+1}}. \tag{1}$$

Equation (1) takes the form of the *logarithmic scoring rule*, which is used in decision theory to maximize the expected reward [18]. In our work, we derive (1) to compute the reward in terms of distance. Due to the employment of *log*, our DRF derives positive value when snake approaches the target (i.e., $D_t > D_{t+1}$) and derives negative value otherwise.

Note that in (1), cases when D_t or D_{t+1} equals to 0 are not taken into consideration. Because when the distance equals to zero, it means the snake eats an apple (reaches the target) and the reward should be set to 1 instead of being derived by DRF. Similarly, DRF is bypassed when a_t leads to a termination state, wherein the reward should be set to -1.

*2) **Training gap**:* In Snake Game, certain experiences are not suitable to be learned by the agent, which are referred as the training gap by us. Training gap is conceptually similar to the *refractory period (RP)* in physiology, which refers to a period immediately following the stimulation of a neuron during which the same neuron is temporarily suspended from activation. Following similar mechanism, we implement the training gap by preventing the agent from learning for M time steps after it ate an apple. These M number of experiences are not stored in the memory pools. This training gap is designed to exclude the improper training signals received when the snake just ate an apple and the location of the subsequently spawned new target suddenly appear in a random location. In other words, the training gap provides a warm up stage for the agent to get ready and move towards the new target, i.e., within these M steps, the agent only makes decisions towards the new target without updating its learned knowledge. To make this strategy more adaptive, we relate the value of M to the length of the snake as follows:

$$M(L_t) = \begin{cases} 6, & L_t \leq k, \\ \lceil p \times L_t + q \rceil, & L_t > k, \end{cases} \tag{2}$$

where the values of p, q and k satisfy the following constraint: $\frac{6-q}{p} \equiv k$, so that $M(L_t)$ is continuous at $(k, M(k))$. When $L_t \leq k$, the training gap is heuristically set to 6, which is a half of the game window's width. When $L_t > k$, the training gap is assumed to be linearly proportional to the length of the snake. In this paper, we heuristically set p, q and k to 0.4, 2 and 10, respectively.

*3) **Timeout strategy**:* If the snake failed to eat any apple over the past P steps, it receives a negative reward as punishment, which is defined as follows:

$$\Delta r(L_t) = -0.5/L_t. \tag{3}$$

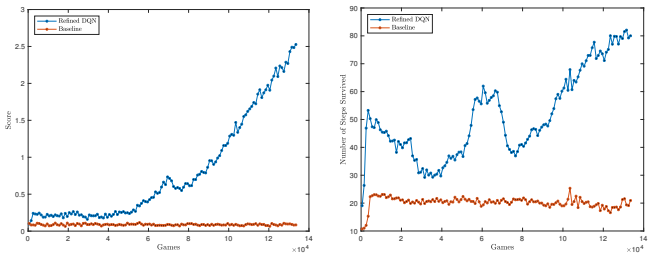
In this strategy, P is a positive integer, heuristically set according to the length of the snake as $\lceil 0.7 * L_t \rceil + 10$. Similar to DRF (see (1)), this timeout penalty decreases as the snake grows and aims to help the snake avoid hanging around meaninglessly with no intention to reach the target. When a timeout punishment is triggered, all the past experiences obtained in the previous P steps are adjusted with the same penalty as defined in (3). Moreover, these experiences are moved to the ordinary memory pool MP_1.

*4) **Overall reward assignment**:* The overall assignment of immediate rewards aggregates the afore-introduced three strategies. Specifically, if the game is not in the training gap period (see (2)), the agent receives rewards based on its distance to the target (see (1)). Moreover, if the timeout punishment is triggered, the recently stored P number of experiences are adjusted (see (3)) and moved to the ordinary memory pool. Moreover, whenever the assigned reward falls out of the [-1, 1] interval, it is clipped at the boundary value of either -1 or 1 correspondingly.

IV. Experimental Results

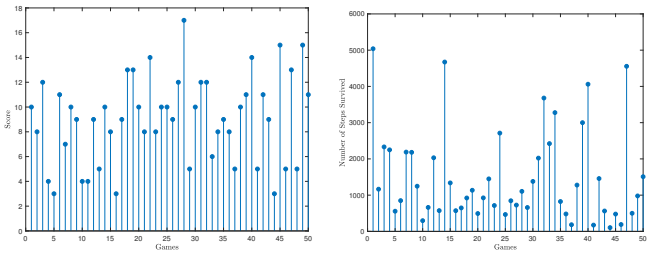
As a summary of the experimental setups, we use the Adam algorithm [14] with mini-batch of size 64 to optimize the network. For input preprocessing, the most recent 4 frames are stacked. The reward is produced by the reward mechanisms described in Section III-C and the discount factor γ (see Algorithm 1) is set to 0.99. The action selection of an agent is regulated by the ϵ-greedy policy with ϵ linearly decays from 0.5 to 0 over the first 50,000 games. The maximum capacity of the memory pools is set to 1,000,000. We train our agent using an Ubuntu server equipped with one TITAN Xp GPU. In total, the agent played 134,000 games, which lasted approximately 7 million steps. To evaluate the performance of our agent, in this paper, we use two evaluation metric, i.e., the game score obtained by the agent and its survival time.

In practice, we start training our refined DQN model until the game has run for a certain number of time steps. This pure exploration phase without training is known as the *observation period*, in which the agent only takes random actions to collect a preliminary set of experiences. When the amount of the collected experiences reaches certain threshold, we start to train the refined DQN according to the *dual experience replay* policy (see Section III-B). In this work, we set the observation period to 50,000 steps. Hence, during the first several games, both the game scores and the survival time are expected to be low because the agent only chooses random actions. After the observation period, the performance of the agent is expected to improve significantly. Note that mostly the end of a game is caused by snake's collision with the wall instead of with its body. As shown in Fig. 2, our refined DQN has a significant performance improvement after the cold start.



(a) Performance evaluation in terms of game score (b) Performance evaluation in terms of survival time

Fig. 2. Visualization of performance comparison. To improve clarity, we only use the averaged values of each 1,000 games.



(a) Performance in terms of game score (b) Performance in terms of the number of steps survived

Fig. 3. The performance of our agent (after being training for 134,000 games) in additional 50 games, wherein $\epsilon = 0$ and training is turned off.

Moreover, for benchmarking purpose, we also conduct experiments using a baseline model, which follows the same strategy used in the DeepMind's groundbreaking work [2] (with the same network structure as shown in Table I). This baseline model is trained in the same manner as our refined DQN model, but without our carefully designed reward mechanism, training gap, and dual experience replay strategy. Fig. 2 clearly demonstrates that our model outperforms the baseline model in terms of both the game score and the survival time. This finding empirically shows the effectiveness of our improvements over the baseline model, i.e., the reward assignment based on distance, the training gap, the timeout punishment, and the dual experience replay strategies. Nevertheless, as shown in Fig. 2, the highest values of the averaged game score and the averaged number of steps survived are seemingly small, i.e., around 2.5 and 80, respectively. However, please note that these numbers are computed as the average of 1,000 games, within which several outlier cases may drastically lower the averaged performance. Furthermore, in the latter part of this experiment section, we compare the performance of our refined DQN model with human performance, trying to further evaluate the capability of our proposed model. As shown in Fig. 2, the performance of our refined DQN model in terms of game score increases slowly over the first 50,000 games along with the decay of ϵ. Moreover, the performance in terms of the number of steps survived even gets decreasing (see Fig. 2(b)). These findings are due to the exploration-exploitation trade-off. As in the exploration phase, wherein ϵ linearly decays from 0.5 to 0, the agent is actually getting familiar with the game environment by accumulating knowledge learned from random exploration. After the exploration phase, the performance of the agent starts to improve by making all the decisions based on the learned knowledge. As shown in Fig. 2(a), the averaged game score generally keeps improving. Similarly, as shown in Fig. 2(b), the averaged number of steps survived also shows improvements in general. There is a noticeable peak in terms of the number of steps survived around 50,000th to 77,000th games. This unexpected peak may be due to the completion of ϵ decay that the performance of the agent starts to improve as it relies purely on the learned knowledge for decision making. However, we suspect that the game play policies learned during the exploration phase may not be optimal or near optimal that after a while (around 27,000 games after ϵ decays to 0), the performance of the agent drops significantly (also shown as a slight drop in terms of game scores in Fig. 2(a)). However, it is encouraging to see that even after the exploration phase, our agent is able to learn more appropriate knowledge and achieves monotonically increasing performance after the performance drop. It seems the period of ϵ decay, i.e., 50,000 games, is not sufficient for the agent to obtain a converged knowledge set. However, due to the limited computing resource we have, we are not able to re-run all the experiments due to the time constraint. Nonetheless, the monotonically increasing performance after 77,000th game empirically shows that our agent is able to learn correctly in the Snake Game. Moreover, in the last paragraph of this section, we show that although pre-converged, our agent can already surpass average human players.

TABLE II
PERFORMANCE COMPARISON AMONG DIFFERENT MODELS

Performance	**Score**	**Survival Steps**
Human Average	1.98	216.46
Baseline Average	0.26	31.64
Refined DQN Average	**9.04**	**1477.40**
Human Best	15	1389
Baseline Best	2	1015
Refined DQN Best	**17**	**5039**

To further justify the performance of our agent, we let the trained agent play additional 50 games with $\epsilon = 0$ and show the results in Fig. 3. In terms of game score, our agent obtains a minimum score of 3, a maximum score of 17, and the averaged score of around 9. The averaged score of 9 is significantly higher than 2.5 shown in Fig. 2(a). Similarly, the averaged number of steps survived is approximately 1,500, which is again significantly higher than that of 80 shown in Fig. 2(b).

To further compare our refined DQN model with human performance, we invite ten undergraduate students to play the Snake Game for 50 games. Before they play 50 games for performance comparisons, each human player played at least 10 games to get familiar with this particular Snake Game implementation. The performance comparisons in terms of game scores and the number of steps survived are shown

in Table II (both the baseline model and our refined DQN model are sampled from the additional 50 games after training with $\epsilon = 0$). As shown in Table II, our refined DQN model obtains the best performance on both the averaged and the best categories. As such, we show that the performance of our agent surpasses human-level performance.

V. Conclusion

In this paper, we delineate how we refine a widely adopted DQN model and apply it to enable an autonomous agent to learn how to play the Snake Game from scratch. Specifically, we propose a carefully designed reward mechanism to solve the sparse and delayed reward issue, employ the training gap strategy to exclude improper training experiences, and implement a dual experience replay method to further improve the training efficacy. Experimental results show that our refined DQN model outperforms the baseline model. It is more encouragingly to find out that the performance of our agent surpasses human-level performance.

Going forward, we shall harvest more computing resources to find out the convergence requirement in this Snake Game and conduct more benchmarking experiments. Moreover, we shall apply our refined DQN model to other similar application scenarios with continuous reallocated targets (such as the respawned apples) and gradually increasing constraints (such as the growing length of the snake).

Acknowledgment

The authors would like to thank Hao Wang and Ruoyi Wang for their contributions in data collection and invaluable suggestions. This research is supported in part by the the National Science Fund Project of China No. 61772227 and Science & Technology Development Foundation of Jilin Province under the grant No. 20160101259JC, 20180201045GX. This research is also supported in part by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative.

References

[1] G. Tesauro, "Temporal difference learning and TD-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv e-prints*, 2013.

[3] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, "Learning to coordinate with deep reinforcement learning in doubles pong game," in *Proceedings of IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 14–19.

[4] S. Yoon and K. J. Kim, "Deep Q networks for visual fighting game AI," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 306–308.

[5] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proceedings of Annual Conference on Neural Information Processing Systems*, 2017, pp. 5055–5065.

[6] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Pittsburgh, PA, USA, 1992, UMI Order No. GAX93-22750.

[7] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *Computing Research Repository*, vol. abs/1511.05952, 2015.

[8] A. Punyawee, C. Panumate, and H. Iida, "Finding comfortable settings of Snake Game using game refinement measurement," *Advances in Computer Science and Ubiquitous Computing*, pp. 66–73, 2017.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[10] P. D. Christopher J. C. H. Watkins, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[11] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep q-network," *ArXiv e-prints*, 2017.

[12] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[13] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computing Research Repository*, vol. abs/1412.6980, 2014.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] D. Wang and A.-H. Tan, "Creating autonomous adaptive agents in a real-time first-person shooter computer game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 2, pp. 123–138, 2015.

[17] H. Y. Ong, K. Chavez, and A. Hong, "Distributed deep Q-learning," *ArXiv e-prints*, 2015.

[18] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.