IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 11, 2022, accepted May 2, 2022, date of publication May 11, 2022, date of current version May 16, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3174361

Enhancing Reinforcement Learning Performance in Delayed Reward System Using DQN and Heuristics

KEECHEON KIM
Department of Computer, Information, and Communications Engineering, Konkuk University, Seoul 05029, South Korea
e-mail: kckim@konkuk.ac.kr

This work was supported by Konkuk University, in 2020.

ABSTRACT This paper suggests and implements how to apply the reinforcement learning on delayed reward system which is known to be complex to apply the machine learning technology such as Q-learning. Such games as Tetris game is known to be a delayed reward system because of its characteristics of generating sparse reward in learning process. Tetris game requires the actor's quick judgment ability and speed of response because the blocks must be stacked in an optimal location quickly, considering the random shape and rotation of appearing blocks. Also, since the number of cases is very large due to the various block types and order, if a human-being is playing the game, the performance is limited by simply relying on human memorization capability. Therefore, we applied a reinforcement learning implemented in this study for this delayed reward system. We find that the general legacy reinforcement learning method shows its limitation in improving the performance. Hence, we apply the heuristic to increase the decision accuracy as the weighting method of reward. As a result, we were able to obtain high scores in games. Although it is not yet possible to say that this heuristic(rule-based) approach has completely conquered the game. In several experiments, this hybrid reinforcement learning shows better playability than human in terms of learning speed, as well as high scores. In this paper, it is shown that general Q-learning is not suitable for delayed reward system. And a hybrid learning that adds prioritized experience replay tactics, and the related techniques and algorithms are introduced to increase the reinforcement learning performance.

INDEX TERMS Machine learning, reinforcement learning, heuristics, Q-learning, delayed reward system, Tetris.

I. INTRODUCTION

Recently, artificial intelligence has applied to many different fields of applications and the various research have led to a better learning performance. Therefore, it is a great challenge for those who study those fields which have more irregular and complex characteristics of the problems.

A various type of approaches and techniques are being studied, and among them, the most notable one is reinforcement learning, which recognizes the current state at every moment and autonomously makes optimal decisions.

Reinforcement learning is an algorithm that defines the environment of a problem to be solved as a state, an action, and an expected reward so as to make autonomous decision making in the direction to maximize the value expected in the future [1]. This decision-making structure is expected to solve the complicated problem of choosing the optimal policy in a sudden change of situation because it recognizes the state of data and makes the decision for each situation unlike the other learning methods such as map learning [2].

However, the reinforcement learning when it is applied to real environment shows many problems and limitations. In certain industries or environments, unexpected situations arise in many circumstances, or high-dimensional data that are not able to be inferred or learned is often utilized for better decisions. In these unexpected and irregular environments, reinforcement learning shows clear limitations and various factors that negatively affect the learning performance. Delayed reward system such as Tetris game is one of the application that we can not use the conventional value

The associate editor coordinating the review of this manuscript and approving it for publication was Huiyan Zhang.

functions based on state-action pairs since the expected return has to be predicted from every state-action pair [3]. A single prediction error might hamper learning.

As an example, the most commonly used Q-Learning algorithm in reinforcement learning has a decision-making structure that evaluates the Q-value of an action and selects the action with the highest value when performing the learning in the environment based on the Markov Decision Process (MDP) theory [4]–[6]. It is a learning paradigm with learning by rewards/penalties with some interesting applications, so as to maximize numerical performance measures that express a long-term objective. This decision-making structure is most similar to the basic philosophy of reinforcement learning, but when this structure is applied to an environment with poor regularity, it has been found that the learning ability is significantly deteriorated [2]. If the agent knows enough facts about its environment, the corresponding logical approach allows him to formulate plans that are guaranteed to work. Such an organization of the functioning of the agent is very convenient. Unfortunately, agents almost never have access to all the necessary information about their environment. Therefore, agents must act under conditions of uncertainty, which makes applying the traditional Q learning algorithm difficult to sparse reward environment [7], [8].

As various factors affecting performance are suggested, research for improving them have been actively carried out for various purposes. This paper is one of those research effort, which identifies various factors that affect learning performance by combining reinforcement learning with a complex game environment (Tetris) [6]–[10]. We have set up the Q-Network based on HDQN(Hierarchical Double Q-Network) which use some heuristics (prioritized experience replay) to improve the performance of the learning process [11], [12]. Episodes acquired by each actor performing actions are stored and shared in the replay buffer, and in this process, various reinforcement learning improvement factors such as prioritized experience replay and segmentation of reward acquisition sections are applied.

From the experience of the previous researches and this work [4], [13], it is identified that heuristic and experience selection results in high performance, we intend to implement a more sophisticated model that further improve the selection and utilization of heuristics and the selection of experience.

In section 2, we introduce the related work, and section 3 shows our hybrid reinforcement learning system. Performance evaluation is in section 4, and the conclusion follows in section 5.

II. RELATED WORK

A. TETRIS

Tetris game has been popular for a long time considering its intuitive rules and simple manipulation. However, the fact that good accessibility does not mean that "the level of difficulty is low." The actor who plays the game must quickly build up the blocks considering the shape and rotation of the block appearing randomly in a short time. This requires the actor to have quick judgment and fast reaction speed. If the actor makes a mistake in the process of filling the block, the number of change cases that need to be computed grows a lot more. This means that the cumulative error or misjudgment makes the situation much more difficult.

The basic rule of the Tetris game is to perform the action of filling the block with no empty space as shown in Figure 1. If you move the block to the empty space correctly, the block of that line disappears and gains a certain amount of compensation. However, the most optimal space for placing a block should consider the circumstances. And many places can be an optimal place.

This feature means that the Tetris game is able to keep the game indefinitely if the actor is skillful to remove blocks and the blocks can be removed indefinitely. However, it is almost impossible to play depending on simple memory, because it depends heavily on reflexes and judgment. Therefore, it is suggested that the difference between the performance of skilled and non - skilled actors is obviously clear.

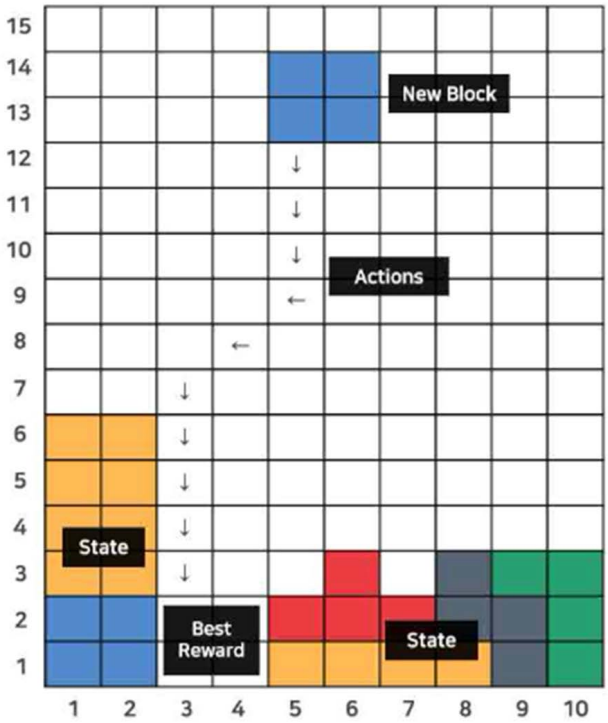


FIGURE 1. Visualization of basic rules of Tetris.

B. TETRIS THROUGH REINFORCEMENT LEARNING

In this section, direct factors and limitations of reinforcement learning performance identified in various previous studies and cases for such games as Tetris are presented.

This paper cites the core contents of two different previous studies. Previous work [6] defines the content and results of reinforcement learning through DQN algorithm for Tetris game environment.

The structure of reinforcement learning through DQN is to process the state of game environment as input values in image format. Since this structure is processed through virtually all pixel values, it exhibits limited performance in many areas.

Previous work suggested a way to combine Feature Columns with CNN structure as an approach to solve this problem [14]–[16].

This result shows that by applying the column-based fetch at the stage after the convolution layer and improved prioritized sweeping, it is possible to conduct efficient learning by selecting the experiences with the largest errors. In addition, the study shows that when the reward is given as a new block fits the lowest space among the accumulated blocks, the performance is significantly improved as compared with the case of applying rewards randomly.

Previous research [7] presents a theoretical foundation for learning strategy by analyzing and verifying the rules of Tetris game through various formulas. The following three learning strategies have the greatest impact on performance improvement in the paper [17]–[19].

1) Maximize the number of 4-block size blocks that fill the empty block and cause the rows to be erased.
2) Minimize the height of stacked blocks.
3) Maximize the number of blocks placed before the game ends.

These hypotheses have proved to be the most influential factors in performance improvement through various experiments [20]. At the same time, it is known that the approach to learn Tetris games is the most NP-complete directional problem. Figure 2 shows a Tetris board used in the previous research.

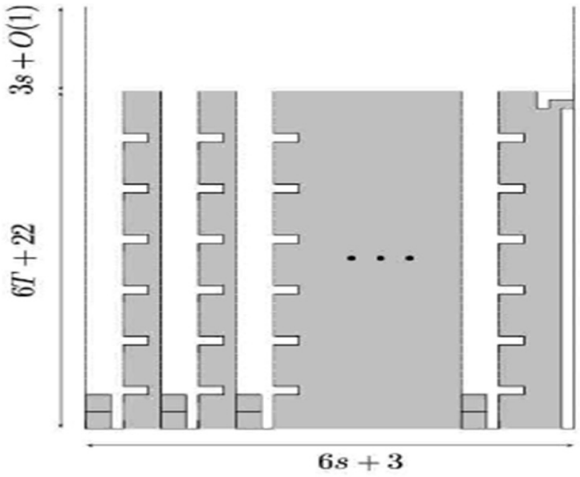


FIGURE 2. Tetris board used in the research.

III. DOUBLE Q-NETWORK WITH HEURISTICS REINFORCEMENT LEARNING SYSTEM

This chapter describes in detail the heuristics, architectures, and technologies used to implement reinforcement learning for the Tetris game environment.

The methodology and the factors of the performance enhancement were verified in the simulator Tetris board that is shown in Figure 2. However, in the actual game environment, the speed of dropping block is faster than that of the simulator in order to maximize the interest, and various weights are given in calculating the score.

In this paper, we focus on the interpretation of how to solve the relation of state due to block shape and rotation by heuristic. In addition to previously proven performance enhancement factors, we define more methods for possible performance enhancement. We show our hybrid reinforced learning and heuristic approach as algorithms described in pseudo code.

A. TETRIS GAME ENVIRONMENT

The Tetris game environment is a typical sparse-reward model and has a relatively low probability of acquiring compensation for any untrained behavior. This probabilistic case is caused by applying the reinforcement learning to the Tetris game, because it has the same operation process as Figure 3 in general. This means that it is necessary to consider the number of randomly occurring block shapes and rotation pattens to check the best expected compensation for each action in each state. Therefore, in order to solve such many cases through reinforcement learning, it is necessary to rely on ''intuition'' and ''experience'' of a person or to optimize decision making by combining heuristics. In this study, we define Table 1. below by reinforcing the heuristics presented in the previous study.

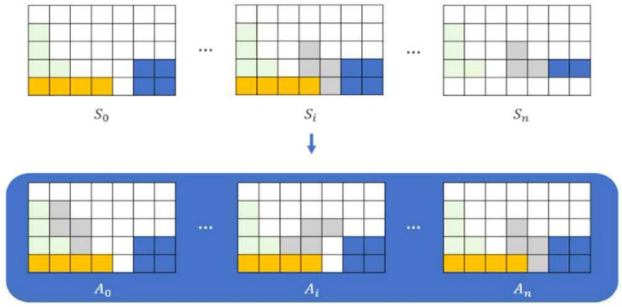


FIGURE 3. Visualization of reinforcement learning in Tetris.

TABLE 1. Basic heuristics of Tetris.

No.	Rule
1	Prevent blocks from being stacked high
2	Minimize empty space between blocks
3	First, remove the blocks that block the empty space.

To obtain intuitive experimental results, the environment of the Tetris game used in the research was implemented in Python. Blocks are randomly generated through a random algorithm and have five shapes as shown in Figure 4. A block can be rotated through the control keys and defined to rotate

only in one direction (right). The size of the game screen is 10×15 (horizontal/vertical), and all the rules of the game are the same as those of general Tetris games.

FIGURE 4. Types and shapes of block in Tetris.

In the case of the scoring in the game, it is implemented to obtain differentially based on the height of the entire game screen size (10×15). In a block matching game, basically 10 points of compensation can be obtained as a score, but as in other games, compensation points are varied depending on the state and height of the block significantly from 5 to 20 points.

B. PROPOSED ARCHITECTURE

We have implemented a structure that uses multiple actors in parallel to perform faster learning in the complex cases. Looking at the architecture of Figure 5, we can see that we use four Sub Actors and one Main Actor. Each Sub Actor is actually the subject of a Tetris game and shares its result (behavior, state, compensation, etc.) with one replay memory as in Algorithm 1 [17], [21]. The shared results have a structure in which the main actor learns the neural network by selecting the higher value experience through the preferential experience reproduction in the replay memory [22].

Algorithm 1 Replay Memory Sharing

Procedure: Training
Initialize: Shared replay memory D to size N ∗ n
Initialize: N action-value function Q with random weights
Loop:
Episode = 1, M do
Initialize state s_1
For t = 1, T do
For each Q_n do
With probability ε select random action a_t
Otherwise select $a_t = argmax_a Q_n(s_t, a; \theta_i)$
Execute action a_t in emulator and observe r_t and s_{t+1}
Store transition (s_t, a_t, r_t, S_{t+1}) in D
Sample a minibatch of transitions (s_t, a_t, r_t, S_{t+1}) from D
Set $y_j :=$
r_j for terminal s_{j+1}
$r_j + \gamma$max_(a^/) $Q_n(s_{j+1}, a'; \theta_i)$ for non-terminal s_{j+1}
Perform a gradient step on $(y_j - Q_n(s_t, a; \theta_i))$^2 with respect to θ

C. TRAINING FEATURE

1) STATE

State is defined as the real-time status information of the board in a Tetris game environment having a size of 10×15 (width/height). The game of Tetris game can be changed in real time as the blocks accumulate and disappear as shown in Figure 6. In this study, since it is possible to directly access Tetris game environment implemented in Python, it receives the information of the board in real time and processes the information as an array without unnecessary image processing.

2) ACTION

Action is defined as a direct motion of a block in a Tetris environment. Blocks can change shape through rotation and require precise movements to the desired location. Actor performs Action by selecting one of 5 behaviors defined in Table 2.

TABLE 2. List of actions actor can perform.

No.	Action	Command key
1	No Action	
2	Move to the left	←
3	Move to the right	→
4	Rotate	↑
5	Vertical Descent	↓

3) REWARD

Reward defines the state information as a comparison with the previous state and how the current state changes through various factors. The reward computation is performed by assigning a basic initial value according to the Equation (1) and assigning weights by polices varying with heuristic techniques. The initial reward value and the weight value described in the formula are used for better understanding, and the exact numerical value differs depending on the environment.

$$Reward = init\ (1000) \tag{1}$$

The weighting of compensation is largely performed in three situations. The first occurs in ''the action of matching a block to an empty space'' which has the greatest objectivity of the reinforcement learning. If the actor achieves the goal of filling all one line as shown in Figure 7, we assign a large value as a weight to the compensation as shown in Equation (2). This leads the actor to perform the act of matching the block which is the basic rule of the game.

$$Reward+ = goal\ (10000) \tag{2}$$

Since the probability of losing the game increases as the block height increases, the second weight is subtracted to obtain less compensation based on the height of the block, considering the characteristics of the Tetris game. Equation (3) below defines this compensation structure. In this way, the actor is prevented from accumulating the block height, which is reinforced by some of the methodologies proven

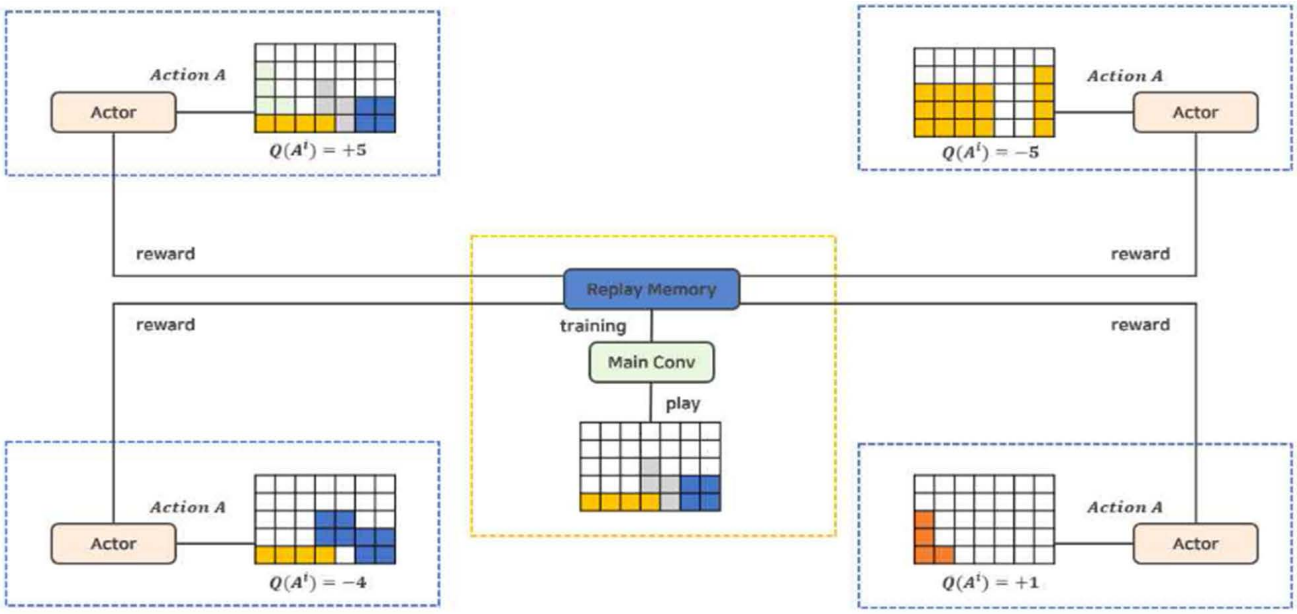


FIGURE 5. Proposed architecture in Tetris learning.

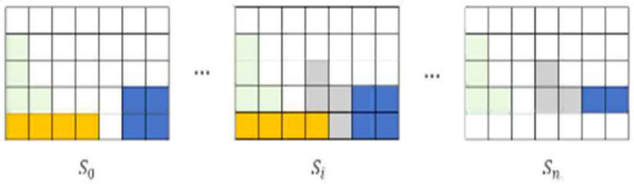


FIGURE 6. State feature as defined in this research.

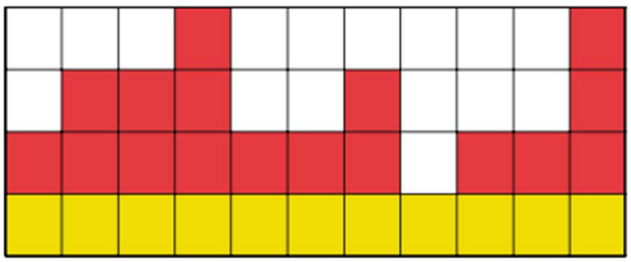


FIGURE 7. Situation in which a line is completed through blocks.

in the previous researches.

$$Reward -= round((1/Height) * 1000) \quad (3)$$

The height-based compensation computation structure defined by Equation (3) can be implemented in the same manner as in Algorithm 2 below. The desired weight can be set based on the height (h) and width (w) information of the game environment of the state, and subtraction can be performed on the compensation. Weights are important to find the optimal value through several experiments, and it is better to implement so that it is not too large in comparison with the compensation value. Too much too large a weight can cause additional problems.

Algorithm 2 Height-Based Reward Calculation

```
Input: Board_height h, Board_width w, Array arr
for i in range(h):
    For j in range(w):
        if I != 0:
            arr[i][j] = (1/i) * 100
Output: Reward -= round((numpy.sum(arr))
```

The final weight is generated by various factors such as the calculation of the size of the empty space, the calculation of whether or not the block blocks the empty space, and the weight thus obtained is directly applied to the compensation.

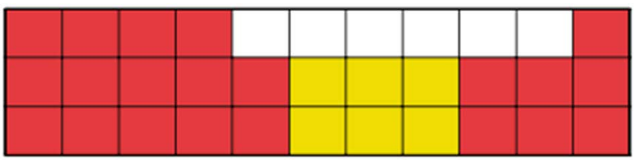


FIGURE 8. Situation in which empty space appears between blocks.

FIGURE 9. Removing blocks blocking an empty space.

First, the size of the empty space between the block and the block is calculated as shown in Figure 8, and additional subtraction is performed on the compensation. Equation (4) defines this compensation structure. This leads the actor not to create as much free space as possible

$$Reward-=round((1/space)*15) \tag{4}$$

As shown in Figure 9, heuristic selection is performed to remove blocks blocking an empty space. Equation (5) defines a block that blocks this empty space and further subtracts from the compensation. This leads the actor not to block the empty space.

$$Reward-=round((1/blocked)*5) \tag{5}$$

In addition, various weighting algorithms are implemented. The blocks stuck on the left/right wall and the weighting of the vacant space priorities result in adding weight. The sum of the reward till the game is ended is obtained as in Equation (6). In addition, we implement a structure that weights are discounted each of the steps with the argument of y. In addition to this, we implemented the system to obtain the weight on compensation from information other than state such as the number of times the game was played, and the time spent.

$$R_t = \sum_{t'=1}^{T} Y^{t'-t} r_t \tag{6}$$

D. REINFORCEMENT LEARNING

1) Q-LEARNING

Learning algorithms are based on Q-Learning, which is most commonly used in reinforcement learning. Equation (7) finds the predicted value of the action in a given state through Q-Learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times \left(R_{t+1} + \gamma_{\frac{max}{a}'} Q\left(S_{t+1}, a'\right) - Q(S_t, A_t)\right) \tag{7}$$

However, in an environment where the regularity is low, such as a Tetris game, the application of one-dimensional Q-Learning is inefficient. This is because the Q-Learning uses the same value when evaluating and selecting each behavior, and thus poses a positive-bias problem in which the future value of the behavior evaluated at the highest level is overestimated. Therefore, in this study, we use the double-DQN learning method to solve these problems.

$$y = R(s, a) + \gamma(\hat{Q}(\emptyset\left(s'\right), a' : \theta^{-}) \tag{8}$$

$$a' = argmax_a \hat{Q}(\theta\left(s'\right), a : \theta) \tag{9}$$

In equations (8) and (9), θ^{-} and θ represent parameters. One is a slowly updating neural network, and the other is a fast-updating neural network. θ^{-} performs the update every predetermined number of repetitions and selects the best Q-value among the estimated Q-values so far. When the target-value for the behavior is obtained, the existing updated parameter is used for the estimated Q-value. In this way, parameters are slowly updated to perform stable learning.

We did not perform any experiment to compare before and after applying the learning structure of double-DQN [4]. This does not address the exact performance improvement factors and effects. However, many studies have failed to reinforce learning of Tetris games through Q-Learning [3], [6], [7].

2) PRIORITIZED EXPERIENCE REPLAY

The implemented reinforcement learning architecture uses many actors to accumulate experience quickly, so the value of stored experience and determining the best experience have the greatest influence on learning speed [21].

The importance of experience is proportional to the amount of learning available in the present state through the experience. Quantitative values for this can be obtained by the difference between the TD target and the actual V(S) expressed as TD-Error. The implementation that calculates the corresponding TD-Error actually uses the function defined in Tensorflow provided by Google.

$$\delta_j = R_j + \gamma_j Q_{target}\left(S_j, max_a Q\left(S_j, a\right)\right) - Q(S_{j-1}, A_{j-1}) \tag{10}$$

The value obtained through Equation (10) is applied by the same method as Algorithm 3. Bidirectional Deque is structured in order of state, action, reward, td-error, and sorted by td-error. This can be expected to increase the efficiency of learning.

Algorithm 3 Prioritized Experience Replay

```
Input: State s, Action a, Reward r, TD-Error e,
ReplayBuffer.append(s, a, r, e)
ReplayBuffer = Deque(sorted(ReplayBuffer,
key = lambda x:
    x[−1], reverse = true))
Output: ReplayBuffer.pop()
```

IV. SIMULATION RESULTS

A. EVALUATION RESULT AND COMPARISON

Table 3 show the evaluation result of the learning network compared with human actor. We use the learning network of 4 actors as in Figure 5. The network spent 1 hour of leaning time for a brand-new customized Tetris game, which is surprisingly fast at learning. It only took 20 outer loops of Algorithm 1 to reach this level of play. During the reinforcement learning period of 1-hour, human actor can play the game 5 times, which implies that human actor has achieved some kind of experience that can be used next time. After the reinforcement learning period, the machine scores a lot higher, mostly more than 15 times higher than human actors. The scores vary every time, because of the playing time of the machine. As you can see from the table 3, it is verified again that the benefit of the reinforcement learning is the speed of the play. Machine actors can achieve very high score in a very short playing time. Playing time and the score

TABLE 3. Results of Tetris of human and R/L.

Case	Human		Reinforcement Learning	
	Time	Score	Time	Score
1	4 min	910	Training (1 hour) with 4 actors	
...	...			
5	7 min	3,125		
6	13 min	4,582	45 sec	66,200
7	4 min	1,266	2 min 17 sec	192,549
8	6 min	2,980	51 sec	70,451
9	10 min	3,560	1 min 42 sec	215,897
10	1 min	871	28 sec	46,689

are not necessarily proportional to each other because of the heuristics and the randomness of the reward.

B. DISCUSSION

1) DISCUSSION OF PERFORMANCE

Figure 10 shows that the architecture implemented through reinforcement learning, which shows the results of the human and machine actors. This is due to the speed difference in behavior decision and repetition. When you reach the goal of completing a block in the game, you will get 5-20 points by yourself.

Therefore, to get 3,606 points in Figure 10, you must complete at least 180 lines. This depends on the understanding and proficiency of the game, but it takes about 10 minutes on average. However, reinforcement learning plays a game at a very high speed because it performs the game through graphic operation. It takes about 30 seconds to acquire 51,120 points of figure 10.

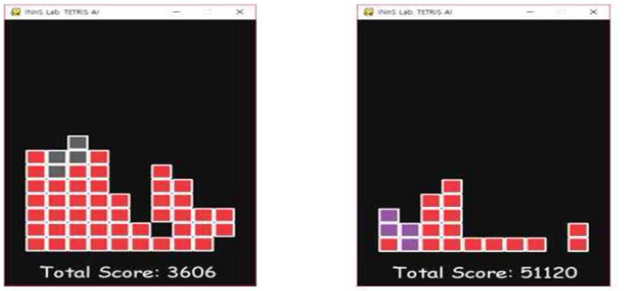
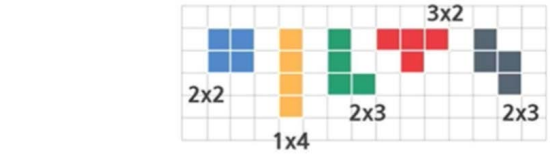


FIGURE 10. Game results of human and R/L.

Implemented reinforcement learning shows good results not only in terms of speed but also in accuracy of decision making. Through our reinforcement learning with prioritized experience replay tactics, we achieved over 200,000 points during the 5th game and over 40,000 points in the worst case. In order to achieve these scores, it takes at least a few tens of minutes arithmetically, and a lot of concentration is required because mistakes must be minimized.

But these results can never be assured that reinforcement learning has conquered the field. This is based on the fact that there are some skilled game users who actually acquire more than 3,000,000 points, and that performance does not increase dramatically even after additional learning.

The limited cause of this performance improvement comes from the randomization-nature of the Tetris game. The block of Tetris implemented in this game has 5 types as shown in Figure 4. If the rotation is performed considering the square and the stick shape, there are 15 cases. This can be defined as Figure. 11 below.



Game Board in 10x15	9x14	7x15, 10x12	8x14, 9x13	8x14, 9x13	8x14, 9x13
Rotate Shape	1	2	4	4	4

FIGURE 11. Features of Tetris environment.

In this environment, the number of locations where blocks can enter is assumed to be the size of the block (n * w) in the size of the game environment (i * j).

As a result, we can conclude that the number of locations for the blocks as follows.

$$(i-n)*(j-m) \quad if\ n=m$$
$$(i-n)*(j-m) \quad and\ \ (i-m)*(j-n) \quad if\ n\neq m$$

However, since the type of block is randomly generated among the rotate shape in Figure. 11, the number of cases for completing one episode becomes very large by calculating the number of such a series of cases. Therefore, it is practically impossible to receive a reward by learning the number of all these cases and removing one line at a time.

In addition, DQN and Q-Learning are not perfect learning when one episode is performed, and it is necessary to perform several episodes because it gradually increases expectation compensation. It can be inferred that the present performance is achieved by compensating more performance improvement factors in heuristics than learning. That is, reinforcement learning is good at speeding up the learning, but we can't say that it is suitable for achieving high scores.

This presumption is also presented in the previous study, which is the reason why heuristic is mainly focused on games such as Tetris. Therefore, if the performance is to be further improved, it is better to focus on heavily algorithmizing the heuristic weighting algorithm to the decision-making area.

2) DISCUSSION OF PARALLEL PROCESSING

In this study, to confirm the relation between parallel processing and learning performance, additional experiments were performed to expand the number of actors to 8 as shown in Figure 12. As the number of Actors increases, the share of various resources increases. However, it is confirmed that

the share of the resources is decreased greatly by rendering for the visualization of the game screen. This is because the resources consumed in the visualization rendering of the game screen are larger than those used in the reinforcement learning. If you use a large number of actors without turning off rendering, a significant load on graphics and memory is expected.

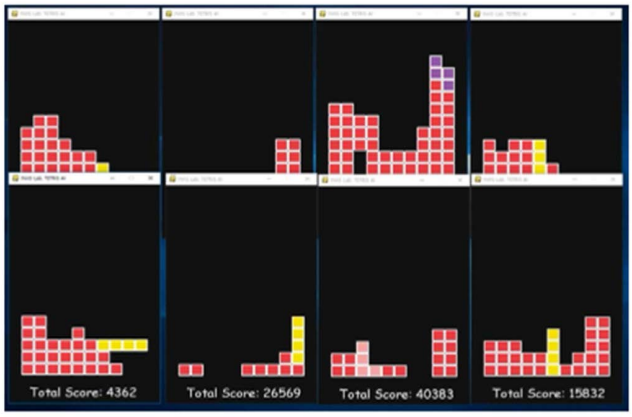


FIGURE 12. Screen implementing parallel processing.

After returning to the main text, learning was performed for 10 minutes from 1 to 10 actors in the environment where all learning data was initialized. The amount of experience data accumulated in the replay buffer was, of course, much higher than the Actor can use in the environment. However, in order to learn two neural networks, there is a problem that the main actor can not accept all of information because the speed is almost fixed in the process of selecting and learning experiences in the replay buffer. This problem depends on the performance of the device and the efficiency of the learning mechanism. However, in this research environment, where learning through three or five actors was the most ideal, it was no longer meaningful. This problem presents a need to further refine the learning process of the algorithm.

3) COMPARISON WITH GENERAL Q-NETWORK

We evaluated the performance of our algorithms using two metrics: game length and game score. Game length is the number of moves the player makes before they reach the end of the game, and game score is the score based on the number of lines they clear. While the game score is ultimately more important, it can only be used to judge the performance of sufficiently advanced networks because there is a large initial barrier to overcome before the algorithm can score with any degree of consistency. Thus, game length, which was typically correlated with game score, is a good metric for showing intermediate performance since an algorithm can incrementally learn to survive longer.

In order to make our results more interpretable, we compare game performance to general Q-Learning. Both networks were given 1 hour of learning period.

TABLE 4. Comparison of Tetris between Q-Network and DQN with heuristics.

Case	Q-Learning		DQN with Heuristics	
	Time	Score	Time	Score
1	35sec	12,350	45 sec	66,200
2	1 min 45 sec	56,300	2 min 17 sec	192,549
3	1 min 5 sec	35,651	51 sec	70,451
4	1 min 25 sec	46,800	1 min 42 sec	215,897
5	50sec	25,600	28 sec	46,689

According to Table 4, our DQN with modified heuristics performs faster and score more. The difference is due to the fact that randomness of Q-learning tends to concentrate pieces in the middle of the game board, whereas our model sample across all positions on the game board, which avoids building tall layers of blocks too soon.

If we compare the how those algorithms react to new episode, learning results are compared through training steps. As we can see from the figure 13 and figure 14, both the algorithms successfully perform the learning through many training steps, there are often interval in which the performance deteriorates. Especially, general Q-Network in figure 13 shows inefficient performance because it relies on more randomness in the process of exploring and creating new episode.

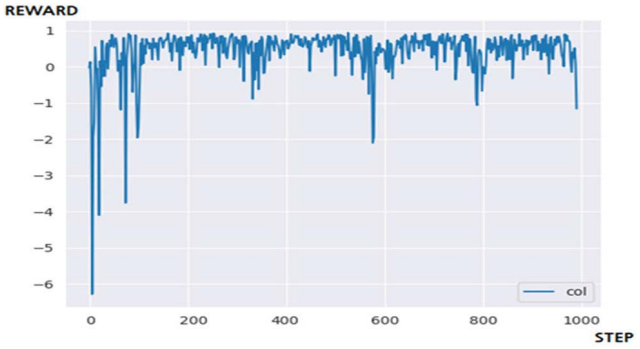


FIGURE 13. Performance of Q-Learning.

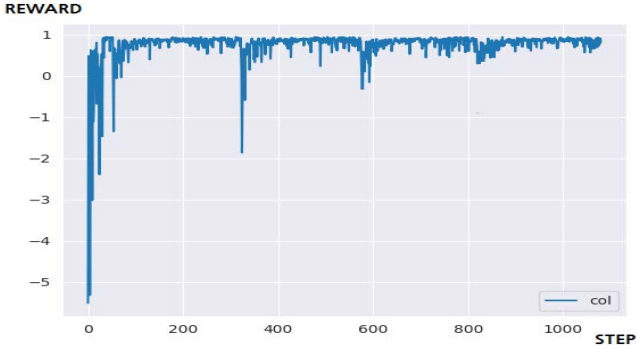


FIGURE 14. Performance of DQN with heuristics.

However, Figure 14 shows our DQN with heuristics successfully performs learning and it guarantees a certain level of flexibility to the new episode patterns. This also, however, does not significantly reduce the number of steps required for learning than expected and it shows performance deterioration in some sections due to the randomness of the episode. We need to further investigate this issue by improving the learning network itself to be more suitable for the delayed reward system.

V. CONCLUSION

In this paper, we show our reinforcement learning implementation on delayed reward system such as Tetris. game to verify that we can't use conventional Q-learning technique for sparse reward environment.

Our results show that the key factor in the difficulty of a reinforcement learning problem is the time lag between the action and the reward. When training with grouped actions off of the heuristic reward function, the reward for a given action was immediate, and this network showed the best performance. Intuitively speaking, a longer gap between action and reward means that a game involves a lot of strategy, and a shorter gap between action and reward means that a game is more reflex-based. We found that learning converges much more quickly for reflex-based games than for games that involve a large amount of strategy, where the DQN network with some level of heuristics performed much better on games like Tetris. Our results show that using an explicit AI and heuristics to bridge the gap between strategy and reflex is a useful approach for improving gameplay performance. In order to get better learning result, the learning architecture need to be changed to accept some heuristics in learning process. The implemented reinforcement learning architecture uses many actors to accumulate experience quickly, so the value of stored experience to determine the best experience are prioritized for the replay. In addition, our algorithm used for learning also has a better expectation of using the advanced Actor-Critic algorithm [23] than the general algorithm like Q-Learning.

The performance of reinforcement learning in Tetris game environment is shown in this paper as well as the description of various changes used to implement our reinforcement learning. Even though reinforcement learning has achieved universally surpass the people in terms of play result and speed, it has been identified that the performance improvement is limited from above a certain level. This is because of the delayed reward environment of Tetris game, which is more affected more by the heuristic nature of the learning model than the learning network itself.

However, we also identified the disadvantage of this approach because we can't prove how heuristics have influenced performance in our learning model, which will be solved by considering improving a learning network itself to get rid of the randomness.

VI. FUTURE WORK

Because of the sparseness of rewards, and especially the difficulty of getting any rewards since random action is highly unlikely to lead to cleared rows, the actual game performance of the network trained with the proposed heuristics remained further research, as it failed to clear lines and was well within a standard deviation of the game length (time) to be expected from playing randomly. With further modifications, we could achieve convergence on a heuristic reward function for actions [24]. Modifications to the prioritized experience replay algorithm could potentially reduce training time and thus allow more improved performance. Therefore, in order to further improve the learning performance on the delayed reward system, it is necessary to further refine the algorithm for weight calculation used in the heuristic, or to further refine the decision-making process by utilizing the MCTS algorithm or the like. This research may lead us to XAI(Explainable AI) problem since it may be necessary to identify how these heuristics rule-based approach influence the total performance of the reinforced learning process.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[2] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Proc. Adv. Neural Inf. Process. Syst.*, 1995, pp. 369–376.

[3] H. Lu, X. Wei, N. Lin, G. Yan, and X. Li, "Tetris: Re-architecting convolutional neural network computation for machine learning accelerators," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[4] H. V. Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[5] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.

[6] M. Stevens and S. Pradhyan, "Playing tetris with deep reinforcement learning," Convolutional Neural Networks for Visual Recognition CS23, Stanford Univ., Stanford, CA, USA, Tech. Rep., 2016.

[7] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell, "Tetris is hard, even to approximate," in *Proc. Int. Comput. Combinatorics Conf.* Berlin, Germany: Springer, 2003, pp. 351–363.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[9] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*.

[10] J. Oh, X. Guo, H. Lee, R. Lewis, and L. S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2863–2871.

[11] V. Mnih, A. Badia, P. M. Mirza, A. Graves, T. Lillicrap, T. Harley, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[12] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for Q-learning in random stochastic Mazes," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.

[13] I. C. Osband Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4026–4034.

[14] M. A. Wiering and H. van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Trans. Syst., Man, Cybern., B (Cybern.)*, vol. 38, no. 4, pp. 930–936, Aug. 2008.

[15] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 271–278.

[16] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.
[17] A. Geron, *Hands-on Machine Learning With Scikit-Learn & Tensorflow*. Sebastopol, CA, USA: O'Reilly Media, 2017.
[18] S. Masashi, *Statistical Reinforcement Learning*. Boca Raton, FL, USA: CRC Press, 2015.
[19] M. Lapan, *Deep Reinforcement Learning Hands-On*. Birmingham, U.K.: Packt, 2018.
[20] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 1038–1044.
[21] S. Lyu. *Prioritized Experience Replay*. Accessed: Jan. 11, 2018. [Online]. Available: https://lyusungwon.github.io/reinforcement-learning/2018/03/20/preplay.html
[22] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.
[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[24] G. Beruvides, C. Juanes, F. Castano, and R. E. Haber, "A self-learning strategy for artificial cognitive control systems," in *Proc. IEEE 13th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2015, pp. 1180–1185, doi: 10.1109/INDIN.2015.7281903.

KEECHEON KIM received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 1992. He is currently a Professor with Konkuk University. He is also the Dean of the Graduate School of Information and Telecommunications and the Director of the Konkuk Software and Security Research Center and the Intelligent Network and Security Laboratory, Konkuk University. His research interests include AI convergence networking, network security, the IoT, SDN, AI cryptography, and C-ITS.