2048 Game using Deep Reinforcement Learning

1st Kedarisetty Vishnu Sainadh, 2nd Kukkadapu Satwik, 3rd Vadde Ashrith
[1,2,3] *Department of Computer Science and Engineering*
Amrita School of Computing ,Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie19028@bl.students.amrita.edu,bl.en.u4aie19034@bl.students.amrita.edu,bl.en.u4aie19066@bl.students.amrita.edu

***Abstract*—Applying both Deep Learning and current Reinforcement Learning techniques together has the potential to greatly advance difficult applications that require nuanced perception and careful policy-selection. Players have reported losing countless hours of their lives as a direct result of the recent spread of the puzzle game 2048 across multiple platforms, including the internet and mobile devices. In this study, we address the concept of constructing a game-playing agent that can win this single player puzzle without the input of human skill or the dynamics of the game. In order to accomplish this, we are using two approaches DQN and Dueling DQN for playing the game 2048. The primary focus of this effort is on developing a more capable real-time Atari game-playing agent.**

***Index Terms*—2048, DQN, Dueling DQN, Reinforcement Learning.**

I. Introduction

In recent years, reinforcement learning has seen widespread use in the playing of a variety of strategic games containing uncertainty. These games include backgammon (Tesauro 1995), card games (Zha 2019), and go, among others (Silver, 2017). Reinforcement learning has been shown to be capable of achieving superhuman performance in strategic games without the need for prior human expertise. This was demonstrated by AlphaGo's consistent victory over the reigning world champion in the game of Go, Jie Ke, in 2016.

Our goal is to figure out how to play the game 2048 in the most effective way possible, and we were motivated to do so by earlier applications of reinforcement learning. Gabriele Cirulli came up with the moniker 2048 for a single person game in the year 2014. The playing surface is a square grid that measures 4 by 4. The numbers that are shown on the screen, which range from 2 1 to 2 15, each represent a power of 2. A player may move all tiles in one direction, and if two adjacent tiles have the same value, their values will be added to form a new tile. When the move is finished, a brand new tile will be added to the grid as soon as possible after it has been completed. The purpose of this game is to get the tile that has a number that is higher than 2048.

II. Literature Survey

Over the course of the last ten years, Reinforcement Learning has had a meteoric rise in popularity as a method for managing how agents can make optimal decisions when confronted with uncertainty. The first and most well-known story is undoubtedly that of TD Gammon, a Reinforcement Learning algorithm that became a backgammon master after playing Tesauro [1995]. Recent advances in Deep Learning have enabled Reinforcement Learning techniques to be applied to a wider variety of situations, which has led to more media success. DeepMind, which is part of Google, is responsible for developing algorithms that have been effective in playing Atari games (Mnih et al., 2013) and beating the world Go champion (Silver et al., 2016). This new achievement in artificial intelligence is believed to be a significant step forward in the field of artificial intelligence due to the fact that the algorithm must search across a massive state space before making a decision.

In 2014 M. Szubert and W. Jaśkowski [1] The 2048 game was modelled as a Markov Decision Process (MDP), and then temporal difference learning (TD) was applied to the MDP to learn the optimal strategy. The model outperforms human capabilities by a significant margin, reaching 2048 at a rate of 97 percent.

In 2015 K. -H. Yeh, I. -C. Wu, C. -H. Hsueh, C. -C. Chang, C. -C. Liang and H. Chiang [2] Enhances the model by making use of multi-stage temporal difference learning in conjunction with three-ply expectimax search (game theory algorithm).

In 2017 Amar and Jonathon [3] Utilizes policy network to figure out the most effective approach for playing 2048 without using any human expertise. A neural network with two convolutional layers with ReLU serving as the activation function makes up the policy network.

In 2021 Li, Shilun, and Veronica Peng [4] They utilised two methods: deep Q-learning and beam search, which is a greedy search algorithm that traverses a tree and picks the nodes that appear to have the most promise. Beam search was more successful than deep Q-learning, reaching 2048 28.5

III. RL Formulation

A. State Space

Each tile in the game is represented by the dimensions (4, 4), changed into 1x16 vector then its is changed to One-hot encoded vector (16 cells * 18 possible states for each). while the digit shown on each tile is represented by the initial dimension which is 16. Because each tile can store either the number 0 (which indicates that there is no number in the tile) or the number 2**i where i is in the range of 0 to 16, the initial dimension is 16.

B. Action Space

At each state, there are theoretically four different actions that can be taken: 1) Move all of the tiles on the left side of

Fig. 1. 2048 Game

the board, 2) Move all of the tiles on the right side of the board, 3) Move all of the tiles on the up side of the board, 4) Move all of the tiles on the down side of the board. On the other hand, not all of the actions are accessible in every state. In the game shown in Figure 1, for instance, a player is unable to move all of the tiles to the left or all the way down. An action list is used throughout our game to denote each and every one of the possible responses to a given state.

C. Reward

Sum of log2 values of collapsed cells on the board at the current step minus the penalty for moving the cell and the penalty for an incorrect move.

At first, the reward was determined by adding up the values of each cell on the board. This was the most basic method of calculation. It turns out that it wasn't. And the answer may be found in the game's underlying mechanisms. Let's imagine you have two cells right next to each other that both have the same value. You collapse them, but the score that is displayed on the board does not alter as a result. Because the sum total of their separate values is the same as the value of the new cell. Therefore, despite having successfully completed the desirable behaviour, the agent will not be rewarded with positive reinforcement and will not acquire any new skills. In addition, following each action, a new random cell receives either a value of 2 or 4, depending on the case. Therefore, regardless of the action that the agent chooses to carry out, it will invariably get a value in return that corresponds to [count to step plus 2 or 4]. It should be obvious that the information provided is insufficient to determine how successfully the agent picked the action.

Coming to the penalty, the agent is given penalty when the cells move (change their position) after an action. So more the cells shifted, more the penalty. We also understood that if accumulate a higher value in any corner of the board and try not move it then the reward increases that is when it causes bigger cells to collapse, as well as when it causes the fewest amount of cell movements.

IV. Formulization

A. Reinforcement Learning

RL, also known as reinforcement learning, is an area of machine learning influenced by behaviourist psychology. It focuses mostly on the subject of how software agents should act in an environment to maximise some notion of cumulative reward. Due to the generic nature of the topic, it is studied in several different academic disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, genetic algorithms, and statistics. In the academic literature on operations research and control, reinforcement learning is also known as neuro-dynamic programming or approximation dynamic programming. Optimal control theory has also investigated the difficulties that are of interest to reinforcement learning. In the absence of a mathematical model of the environment, the theory of optimal control is less concerned with learning or approximation than with the existence and characterisation of optimum solutions and methods for the accurate calculation of such solutions.

B. Deep Q-Learning

DeepMind came up with the idea for the Q-Network (DQN), which became the first method of its sort to be used for deep reinforcement learning. In Deep Reinforcement Learning, determining an estimate of the value of Q is achieved by the utilization of a neural network. Reinforcement learning can use neural networks as a function approximator when it is applied to large datasets. This allows the learning method to estimate functions. An agent can learn how to trade a single stock or asset with the use of Deep Q-learning (DQN) and its upgrades. Deep RL relies heavily on DQN as an essential algorithm. It establishes the framework for the area, with the principles outlined in the paper still being used today in various contexts.

In Deep q-learning, it combines the concept of Reinforcement Learning and Deep Neural Networks. The violations of the IID assumption, which means that the data is not independently and identically distributed, and the stationarity of targets are the two most prominent difficulties that appear consistently in value-based deep reinforcement learning. Changing the nature of the problem so that it may be solved using supervised learning is the approach that is taken for learning in DQN. The recursive character of the Bellman equation for Q-values comes in handy for us at this point. As indicated by the Bellman equation

$$Q(s,a) = r + \gamma \cdot max_{a'} Q(s',a')$$

Having a distinct network that we can fix for numerous steps and reserving it for the calculation of more stationary targets is an easy technique to make target values more stable. In DQN, the network that serves this function is referred to as the target network. We prevent the issue of "chasing your own tail" by utilising a target network to fix targets. This is accomplished by artificially producing a series of small supervised learning tasks that are presented in sequential order to the agent. Our

targets are locked in place for the same number of steps as our target network is locked in place. This enhances our odds of convergence, but not to the ideal values because non-linear function approximation does not allow for such things to exist; however, convergence in general is improved as a result of this. However, and this is the most essential benefit, it drastically lessens the likelihood of divergence, which is something that frequently occurs in value-based deep reinforcement learning approaches.

The current reward of a state and the expected reward from taking an action are combined in a weighted algorithm to create a q value of each action, the action with the highest q value will be the one taken at that step in the game and this process repeats for the entire game. Deep Q-Learning uses a neural network to find an approximation Q(s,a,theta) of Q*(s,a) . theta is the hyper-parameter of the neural network.

Experience Replay or Replay Memory:

The best solution to the problem of data not being IID(Independently and identically distributed) is called experience replay. Across all episodes played by the agent, the replay memory records all of the agent's experiences at each time step. Actually, we'll normally see the replay memory set to a finite capacity limit in practice, so it'll only store the most recent encounters. We'll randomly take some samples from this replay memory to update the weights of the network. When we take an action and complete a step in order to collect a reward, the network does not learn from this final step, instead we will add the experience to the replay memory. By selecting a random mini batch from the replay memory we will perform back-propagation using a gradient descent step. Using experience replay can result in a variety of positive outcomes. We can increase the likelihood that the changes we make to the neural network will have a low variance if we take samples at random.

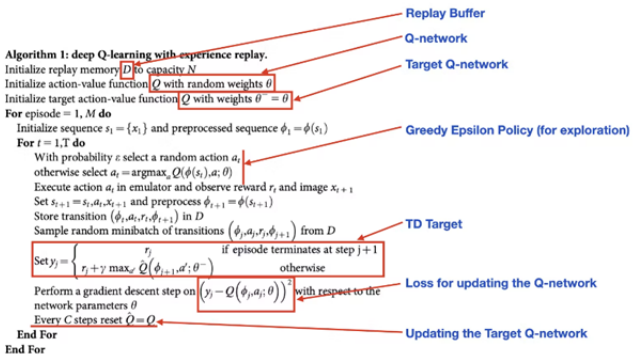

Fig. 2. DQN Algorithm

C. Dueling DQN

In comparison to DQN, Dueling DQN demonstrates an alteration in the structure of the network. It is less crucial to choose the action to do since the values associated with the various actions are relatively comparable in many situations. This is of utmost significance in contexts in which a diverse range of actions is available for selection. In DQN, the 'Q' values for each of the states in the batch are updated during each training iteration solely for the specific actions that are performed in those states. This is done for all of the states in the batch. Since of this, our learning is slowed down because we do not acquire knowledge of the Q values for actions that have not yet been carried out. When using duelling architecture, on the other hand, learning occurs more quickly since we begin learning the state-value even if only a single action has been performed at this state. This allows us to learn more quickly.

In Dueling DQN, we make use of a non-sequential architecture for deep learning. This architecture is characterised by the fact that, following the convolutional layers, the model layers branch off into two distinct streams (sub-networks), with each sub-network containing its own fully-connected layer as well as output layers. Because of this, the model is able to learn in a way that is more effective than it would be in a more conventional sequential design. The first of these two networks corresponds to the Value function, which may be used to "guess" the value of a certain state and only has one node in its output layer. This network can be thought of as the more straightforward of the two. The Value function may be thought of as matching to this branch or network. The second network is referred to as the "Advantage" network, and it calculates the value of the "advantage" of carrying out a certain action in comparison to the base value of the present state.

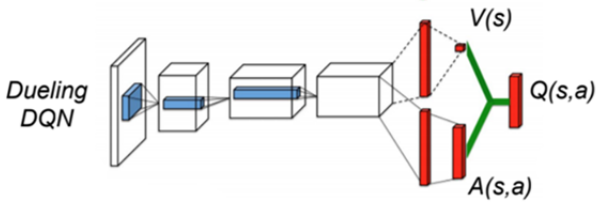

Fig. 3. Dueling DQN

V. System Architecture

A. DQN

The DQN design that we have chosen can be seen in Figure 4, and it consists of nine layers. The first layer accepts the one hot encoded vector of size 288 (16 cells * 18 possible states for each) as input, and it generates outputs of size 4, which are the Q-Values of our action space. Figure 3 depicts how this architecture works. The design includes 3 batch normalisation layers, 4 dense layers, and 3 relu-Activation layers in total.

B. Dueling DQN

Figure 5 shows the Dueling DQN architecture that we have used, which consists of 13 layers in which the first layer takes the one hot encoded vector of size 288 as input (16 cells * 18 possible states for each) and gives output of size 4 which are the Q - Values of our action space. The architecture contains 8 dense layers, 5 batch normalization layers and 5 relu-Activation layers.

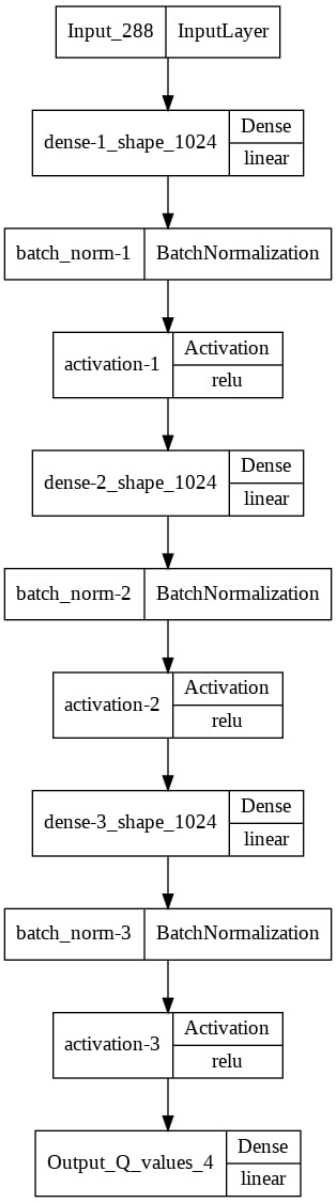

Fig. 4. DQN Architecture

VI. Training

The training has been completed for 80,000 episodes. The training procedure makes use of hyper parameters such as a batch size of 1024, a learning rate of 0.0005, a buffer size of 100000, and an initial epsilon value of 0.05, which is then steadily decreased until it reaches 0.00001.

A. Hyper-Parameter Tuning

Hyper parameter tuning is performed on 3 hyper parameters which are buffer size, batch size and learning rate. In the course of our study, we explored a package that goes by the name optuna. Optuna is a software framework for automated hyperparameter tuning that was developed specifically for Machine Learning and Deep Learning. This may be in conjunction with other frameworks such as PyTorch, TensorFlow, Keras, SKlearn, and so on. The define-by-run

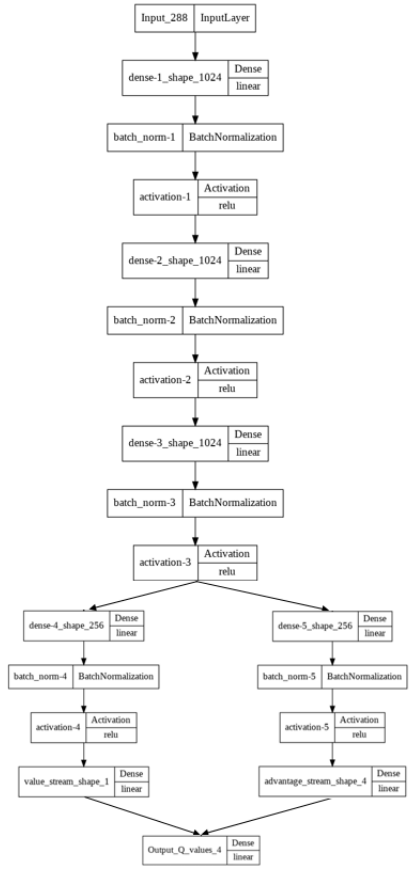

Fig. 5. Dueling DQN Architecture

application programming interface (API) is something that Optuna makes use of. This API enables the user to create highly modular code and dynamically generate the search areas for the hyperparameters.

For the purpose of tuning, we have considered three hyperparameters which include three different values of batch size (64,256,1024) , two different values of buffer size (100000,200000), and two different values of learning rate(0.005,0.0005). we have analysed these hyper parameters using optuna and each combination of the provided values are being trained with 250 episodes, we found that the optimal values for hyperparameters for high rewards are as follows: a learning rate of 0.0005, a buffer size of 100000 and batch size of 1024.

Figures 6, 7, 8 and 9 indicate the results of hyper parameter tuning

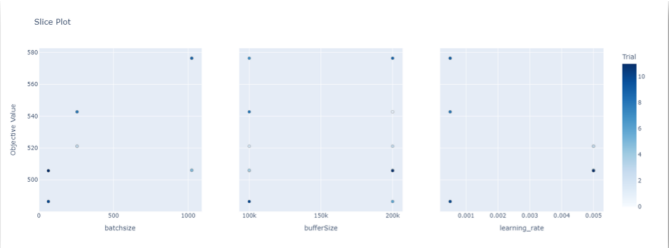

Fig. 6. Slice plot of DQN Hyperparameters

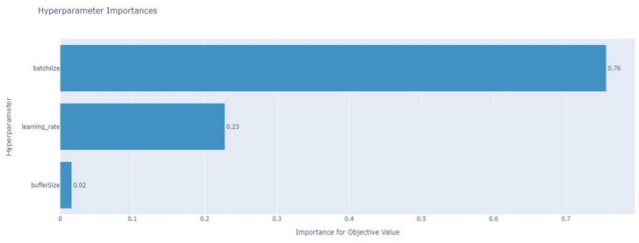

Fig. 7. DQN Hyperparameters Importance

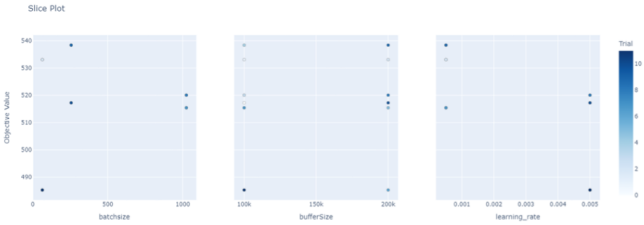

Fig. 8. Slice plot of Dueling DQN Hyperparameters

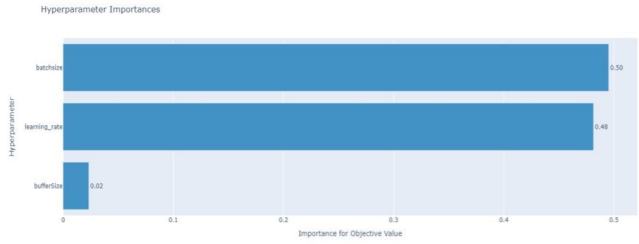

Fig. 9. Dueling DQN Hyperparameters Importance

VII. Results

In figures 10 and 11 the red line indicates the mean total rewards over last 50 episodes, blue line indicates mean steps over 50 episodes and green line indicates max cell value seen on board and as number of episodes increases 1024 is achieved consistently in both models also if the model is trained with more episodes 2048 can also be achieved. We also noticed that Deuling DQN demonstrated far better outcomes than DQN when it came to consistently reaching 1024.

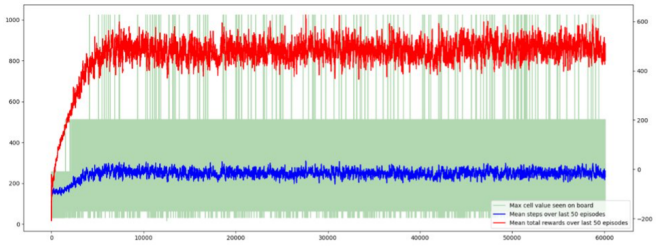

Fig. 10. Episodes vs Max Tile DQN

Figures 12 and 13 represent the loss functions in DQN and Dueling DQN architectures.

The Table 1 shows the comparative study with 2021 paper[5] where both of our models performed better than the model proposed in the paper.

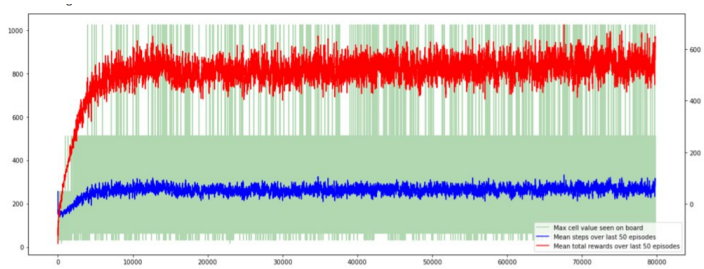

Fig. 11. Episodes vs Max Tile Dueling DQN

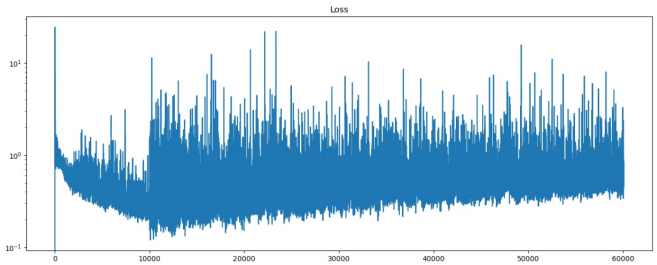

Fig. 12. Loss function of DQN

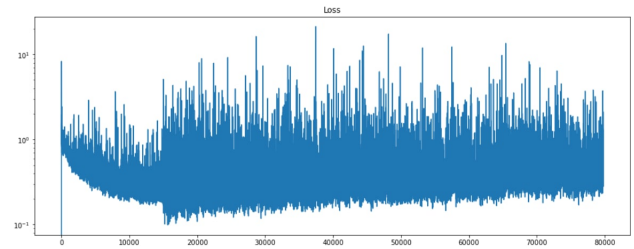

Fig. 13. Loss function of Dueling DQN

TABLE I
Comparitive Results

Max Tile	Models		
	Deep Q-Learning 2 [5]	*DQN*	*Dueling DQN*
16	0%	0%	0%
32	0%	0.77%	0.54%
64	5.7%	10.14%	8.31%
128	34.2%	36.19%	33.8%
256	56.1%	40.74%	42.29%
512	3.9%	11.88%	14.61%
1024	0.1%	0.27%	0.44%
2048	0%	0%	0%

VIII. Conclusion

A DRL agent is implemented to reach higher tile values. We achieved 1024 tile after training the model for 80000 episodes.We have also achieved better results when compared to previous works[5]. With higher GPU we can reduce time and train the model for better results while maintaining higher scores. We can train the model to reach higher tiles like 2048, 4096, 8192 and so on.

IX. Decision Points

- First decision point: Selecting a value function to approximate

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

the above functions represents the value function for the given algorithm.

- Second decision point: Selecting a neural network architecture - the architecture we are using is state in action value-out architecture, the state variables for our application is a numpy array which represents the present state of our grid which is also our environment and action variables are move the tiles up, down, left, right.
- Third decision point: Selecting what to optimize - in our application our objective is to obtain a high score for a single game. We need to minimize the loss w.r.t optimal action value function.
- Fourth decision point: Selecting the targets for policy evaluation - in our application we use off policy TD target for policy evaluation.
- Fifth decision point: Selecting an exploration strategy - In our project, we used the epsilon-greedy method throughout agent training. Epsilon is initially kept as 0.05 and then gradually decreased to 0.00001 with a decay factor of 0.009 and after every 5000 steps we double the epsilon value in order to maintaion the agents exporation
- Sixth decision point: Selecting a loss function – We used MSE (Mean Squared Error) as our loss function
- Seventh decision point: Selecting an optimization method – we used Adam as an optimiser.

REFERENCES

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

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

[3] K. -H. Yeh, I. -C. Wu, C. -H. Hsueh, C. -C. Chang, C. -C. Liang and H. Chiang, "Multistage Temporal Difference Learning for 2048-Like Games," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 4, pp. 369-380, Dec. 2017, doi: 10.1109/TCIAIG.2016.2593710.

[4] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[5] Li, Shilun, and Veronica Peng. "Playing 2048 With Reinforcement Learning." arXiv preprint arXiv:2110.10374 (2021)