Autonomous player to maximize performance in 2048 Game

Authors: Xuanyu Chen, master in computer science department,
Yimin Lin, master in data science department, Jiaoyan Chen, master in data science department

Abstract—**2048 is a sliding puzzle game release in 2014, which was quite popular on mobile devices. This paper discussed the theory and applicability of Minimax, Expectimax, Monte-Carlo Tree Search (MCTS), and reinforcement learning techniques. While Expectimax shows the best performance with strong domain knowledge, minimax follows with fewer domain knowledge. Currently MCTS and Reinforcement Learning has the worst performance with no domain knowledge provided. but they still have large possibility to improve.**

Keywords—2048 Game; Tree Search; Reinforcement Learning.

I. INTRODUCTION

2048 is a single-player, nondeterministic sliding block puzzle game developed by Gabriele Cirulli in 2014, when he was 19 years old[1]. The core concept of it comes to similar sliding block games like Three and 1024[2]. Once it's released, it catched public eyes and was very popular at Android and IOS app store. The aggregated time of playing the game online by people during the first three weeks after release was over 3000 years[1]. Since Gabriele made the code open-source, various versions of the puzzler games are developed.

The main characteristic of 2048 game is that user does not even need any illustration to get started. But it shows a steeper learning curve as the game going on. Usually it is very hard to reach the goal. Difficult to lose for the first several minutes, but hard to reach the goal ultimately[3]. Such characteristics make the game so popular and also draw a wide attention in artificial intelligence field. Various playing strategies and algorithms have been developed and discussed how to achieve maximum result of 2048 game[4]-[5].

The main challenge of this project is to find a right way to define the appropriate heuristic function and cost function, so that the agent can make decisions that sacrifice the short term reward but gain more reward in the long term.

In this project, we create a 2048 GUI template and a console temple, implement different strategies by applying tree search algorithm, minimax, minimax with alpha-beta pruning, expectimax and reinforcement learning to beat 2048 game, as well find the best solution which maximizes the performance. Different agents such as model-based reflex agents and learning agents are implemented to explore the possibility. Tree search algorithms have been notably employed in game play. One of the supper cool things about tree search is that the same core algorithm can be applied for a whole class of games: Chess, Go, Othello, and almost any board game you can think of. Reinforcement Learning has enjoyed a great increase in popularity over the past decade by controlling how agents can take optimal decisions when facing uncertainty.

As the strategies to play 2048 are proved to be useful and valid. We can easily apply our strategies to the similar 2048 games such as Three and 1024. Also, we can further explore the performance in similar single player situation, such as the automated warehouse distribution system.

II. BACKGROUND

A. Rules of 2048

The 2048 game represents a 4 X 4 grid where the value of each cell is a power of 2. An action can be any of the 4 movements: up, down, left and right. When an action is performed, all cells move in the chosen direction. Any two adjacent cells with the same value (power of 2) along this direction merge to form one single cell with value equal to the sum of the two cells (i.e. the next power of 2). After each move the board will randomly generate 0 or 1 new blocks with value 2 or 4 understand given probability in available space.

The score for each move is accumulated. If all spaces on the board are full and the board cannot take any action to move, then the game ends. The objective of the game is to combine as many cells as possible until reaching 2048 or even a higher value. The performance are measured by five factors, depth of the tree, average movements till game ends, largest tile in the board, average total score, occurrence of large tiles(larger than 128)[6]. Detailed description for our 2048 environment is presented in Section III.

B. Related works

There are various theoretical and practical researches explored in previous work. The minimax algorithm is an iterative deepening depth-first tree-search algorithm in game theory[7], where Min player and Max player interact with each other. Knuth Donald analyzed the alpha-beta pruning minimax algorithm further in AI field, which helps prune the search depth of the tree[8]. Such algorithm is firstly proposed in 2014 by Ovolve[9] to solve 2048 game, which regards the player to be the max player that try to maximize the score. However, the minimax algorithm doesn't perform well to achieve the long term goal rather than short term rewards.

To enhance the performance of minimax algorithm, David G[10] applied expectimax algorithm on 2048 game, which requires researchers to manually weigh states. Although expectimax algorithm provides more stable performance, it requires domain knowledge to weigh the states and has to search for a full expand tree.

To explore the possibility for AI to play without human knowledge, Monte Carlo Tree Search and Reinforcement Learning are introduced in 2048 Game. Fenix Chen[11] implemented Monte Carlo Tree Search, which randomly moves for a given action and calculates accumulated reward to choose the action that gives the largest rewards. Although the algorithm doesn’t need any human knowledge, it just makes decision based on random movements. The evaluation function is quite simple and hard to say that it’s ‘smart’.

In 1998, Erev proposed that the reinforcement learning can be applied in game playing[12], which build a learning-based agent that automatically explores and exploits the game without being told which action to take. (Some previous works have examined its performance on 2048[13-15]. However, the reinforcement algorithm takes a long time to train.)

Although various approaches have been proposed, there is not much systematic investigation and comparison for all these algorithms. Rodgers, P[5] compared the result of Monte-Carlo Tree Search with the Average Depth-Limited Search and explored the effect of depth. Mehta and A Abdelkader[17-18] summarized and explored the theoretical strategy to beat the game, but no comparison for different strategies provided.

In this Project, we formulated the logic of 2048, built the GUI and console templates. Based on our understanding, we developed all above algorithms and compared the performance of those algorithms via five performance measurements mentioned above. For Reinforcement learning, to reduce the computation complexity, we built our own learning agent. For expectimax, we weighed the states in our own way.

We structure the paper in the following order. Section I introduces the problem domain. Section II formulates the problem and introduces current approaches to address the problem. Section III presents the game 2048, as well introduces methods we apply to play 2048 game autonomously. In Section IV, the experiment results by applying methods in section III in game 2048 will be analyzed and presented. We will conclude our work in section V.

III. Methods to Apply

A. Description of 2048 environment

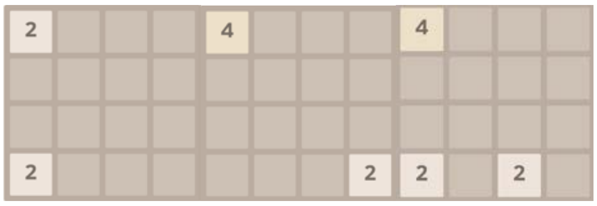
2048 playouts are quite straightforward and can be unfolded as follows (see Figure 1 below) :

1. **beginning**: at the very beginning of the game, only two tiles are given 2s or 4s, which location is chosen at random as seen in Figure 1(a) below.

2. **player's turn** : the player chooses a direction in set of (up, down, left, right) and the whole board is moved accordingly.

3. **randomness** : after the player's actions, a new tile (either 2 or 4, with unequal probabilities) is added at a random free location. Fig 1(b) and (c) illustrates player and randomness successive moves : first player moves up (tiles merge into 4) and then a new tile is added in the bottom right corner, followed by a left move from player (previous 2 is pushed in bottom left corner) and a randomly added tile (again a 2, but on the third column, last row this time).

4. **termination** : the game ends when no move is left available for the player.



(a) Initial board (b) up, randomness (c) Left, randomness

Fig 1: Typical beginning of game: *the game will initial 2 tiles randomly as shown in (a), player can swipe all tiles up, down, right or left, if swipe (a) up, the board will look like (b) and another tile generated randomly. Swipe board in (b) to the left will lead to board in (c)*

A key operation that allows to obtain tiles with increasingly larger values consists in merging adjacent tiles. When making a move, each pair of adjacent tiles of the same value is combined into a single tile along the move direction. The new tile is assigned the total value of the two joined tiles. Additionally, for each such merge, the player gains a reward equal to the value of the new tile. Figure 1c shows the board resulting from combining two 2-tiles in the upper row when sliding the tiles in the LEFT direction. Another LEFT move leads to creating an 8-tile (see Fig. 1d). The moves generate rewards of 4 and 8, respectively.

The game is considered won when a 2048-tile appears on the board. However, players can continue the game even after reaching this tile. The game terminates when there are no legal moves, i.e., all squares are occupied and there are no two adjacent tiles sharing the same value. The game score is the sum of rewards obtained throughout the game.

B. Heuristic

Although using heuristic functions simulates human thoughts on the game, it improves performance for several algorithms. Several heuristics we used are monotonicity, emptiness and smoothness.

Monotonicity evaluates how monotonicity the board is. It is a human strategy when playing 2048 Game, that, put the tile with large number in the corner following with smaller tiles. In this way, when smaller tiles are swiped out and added up to large tiles, it can be merged with large tiles easily, resulting in total less number of moves to merge tiles. For simplicity, we choose to calculate in two directions, in a row, from left to right, if in column, from top to bottom.

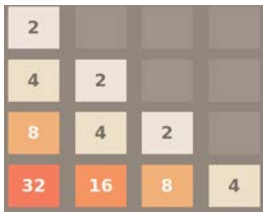


Fig. 2. Monotonicity Example. In human trial *We tend to put the large tile in one corner, and follows the smaller corner, in this way, when an smaller tile has swiped out, the larger one can be swiped out naturally.*

Emptiness measures how empty the current board is. The board is dead when there is no next move can be executed. If the board more empty, it is obvious that more moves can be executed. Consider this situation, that two board with same largest tile and same total score, and relatively same monotonicity, the board with less empty tiles left are more easily to die, and the board with more empty tiles can execute more moves, increasing the chances of achieving larger tiles and total score.

Smoothness measures how smooth the board is. By smooth, we mean the number of moves to swipe two neighbor board.

C. *Monte Carlo*

Monte carlo strategy is totally random strategy. We define one step for a current board for one direction is, first moves one step to the direction is if can make a move, and then totally randomly simulates moves until the board can no longer make a move. Compute the total score for one step. Monte carlo simulates situation for four directions, when each move, simulates each direction for a large number of steps, and choose the direction with maximum score / steps.

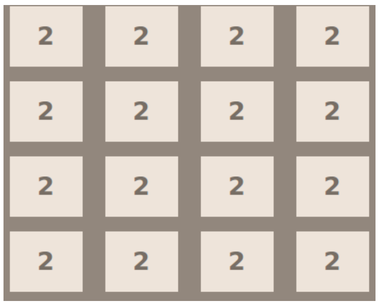


Fig. 3. Smoothness example. *This board is fake, but it shows a perfect smooth board that we can swipe all the tile in one move.*

D. *Minmax*

The Minimax was firstly proposed in game theory, and widely applied in two-player situations, where the game existing two player. The max player always wants to maximize the final performance(make himself win) while the min player always wants to minimize the final performance(make max player lose).

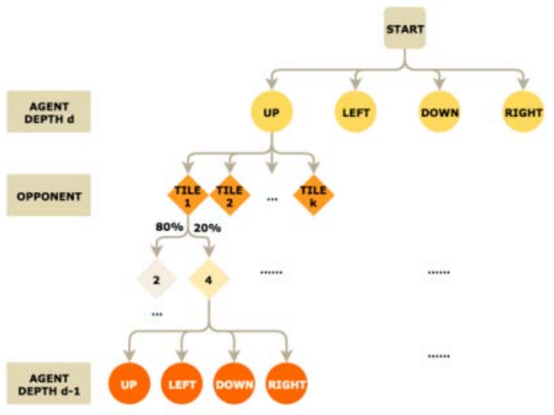


Fig. 4. Minimax Search Tree. *This image comes from a student project in Stanford[19] which illustrate the search tree of minimax player.*

To apply the minimax algorithm, the 2048 game formulate the problem as following:

For each step, we regard the system agent to be max player, whose task is to evaluate which action to take. Then we define the min player to be the opponent that random generate new tiles on empty spaces. For each step, the max player implement a tree search with depth d, then from the leaf back to the root, the perspective of max and min players are changed by turns. Then at the root, the max player decides in which action he will get the highest performance.

E. *Expectimax*

The Expectimax algorithm is pretty the same as the minimax algorithm mentioned above, except that we add chance nodes between agent nodes and opponent nodes. An expectimax search tree is illustrated in Fig. 5. Then the pseudo code for the algorithm is introduced in Table 1.

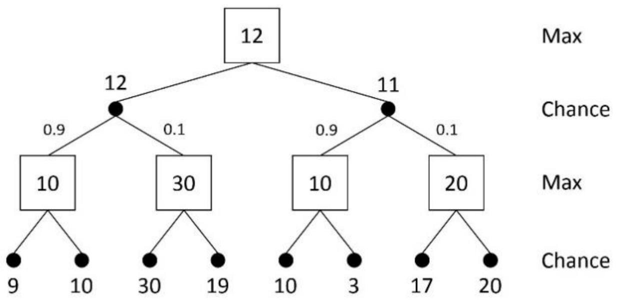


Fig. 5. An expectimax search tree. *Expectimax search is similarly to minimax search tree, except that it add a chance node.*

Algorithm Expectimax

```
if state s is a Max node then
    return the highest Expectimax-value of succ(s)
end if
if state s is a Min node then
    return the lowest Expectimax-value of succ(s)
end if
if state s is a Chance node then
    return the average Expectimax-value of succ(s)
end if
```

Table 1. pseudo code for Expectimax *In this table, it shows how to implement expectimax.*

F. *Reinforcement learning*

Reinforcement learning is an important technique in training an agent to learn how to respond to a given environment. A Markov decision process (MDP) is a model commonly used in reinforcement learning, modeling the problems in which an agent interacts with the given environment through a sequence of actions according to the change of the state and the rewards, if any. The game 2048 can be naturally formulated as an MDP. Actions change the state of the environment and typically result in both immediate and delayed consequences that can be quantified as rewards for the agent.

Formally, an MDP is a discrete time stochastic control process, and can be defined as <S, A, R, P>, where:

1. S is a set of possible states of the environment,
2. A is a set of actions and $A(s) \subseteq A$ denotes a set of actions available in state $s \in S$,
3. $R : S \times A \rightarrow R$ is a reward function, where R(s, a) provides the reward for making action a in state s,
4. $P : S \times A \times S \rightarrow [0, 1]$ is a stochastic transition function, where P(s, a, s'') denotes the probability of transition to state s" in result of taking action a in state s.

For the MDP in this game, we set policy:

$$\pi(s) = \mathrm{argmax} P s,a(s) V(s)$$

With the winning tile of 2048, a board could have as many as possible states. Thus we tried to discretize the states by extracting important features of a given board:

1. Decreasing pairs in snake shaped board

To emphasize that we want to approximate a snake shaped board, we check whether the value in each tile decreases in the following direction. If the value in successor tile is less than previous tile, they are considered as decreasing pair. If there is a empty tile, the value for this empty tile is taken as 0. We expect to see number of pairs in range (0, 15). In figure X, there are 11 decreasing pairs.

2. Number of empty tiles

We deem it as an important feature, since it imposes a sort of reward or penalty on current board: the more freedom tiles, the more likely to last longer until death. The best situation is when game just begins and only two tiles are occupied and the other fourteen are free, while the worst is when the board is full and game is over. For other situation in between, we expect to see number of tiles in range (0, 14).

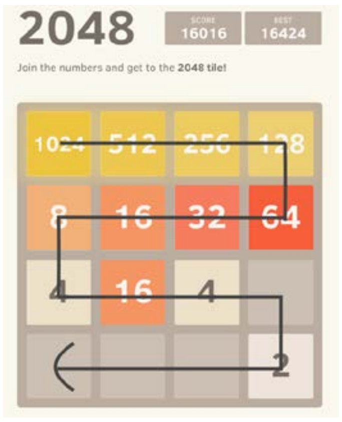


Fig. 6. snake-shaped like board *Go through the board in the direction shown in this figures. It is represented as snake-shaped like board.*

3.Number of possible merges

This metric measures the efficiency of potential swipes from current to next state: the more merges, the more tiles to be freed. Even for boards whose monotonic characteristics and number of empty tiles are the same, their states could be largely different due to the number of merges available. A board that could merge tiles more than once in either direction is clearly more flexible and promising than a board that only moves but not merge tiles. Since each row or column allows at most merges, this metric should be an integer between 0 and 8, inclusive.

The above three features give 16 x 15 x 9 = 2160 states, which are less computationally expensive but also sacrifice some information of the board. For each action change state s to s”, it will earn 10 * value after merge if any two cells merge. The objective of an agent placed in an environment defined as an MDP is to learn such a decision making policy π : S → A that maximizes the expected cumulative reward.

Temporal difference (TD) learning, a kind of reinforcement learning, is a method for adjusting state values from the subsequent evaluations. Since 2048-like games can be easily modeled as MDP, TD learning can be naturally applied to AI programs for these games. As an intermediate step towards learning decision making policies, TD methods compute value functions. The state value function V^{π} : S → R estimates the expected cumulative reward that will be received by the agent if it uses policy π to make actions starting from a given state s ∈ S. In the context of playing games, the state value function predicts how much points the agent will get from the given state till the end of the game. The simplest TD employs the following update rule:

$$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$$

This rule attempts to minimize the difference between the current prediction of cumulative future reward V (s) and the one-step-ahead prediction, which involves the actual reward r and is equal to r + V (s’’). Consequently, the error between the successive predictions δ = r + V (s’’) − V (s) is used to adjust the value of state s. The size of correction is determined by the learning rate α ∈ [0, 1].

For some problems, it might be more effective to learn the action value function Q^{π} : S × A → R, which estimates the utility of a given state-action pair, i.e., the expected sum of future rewards after making action a ∈ A(s) in state s ∈ S. One of the most recognized algorithms for learning action value functions is Q-LEARNING [17], which operates in a similar way to TD. Upon observing a transition (s, a, r, s’’), it updates the action value estimates as follows:

$$Q(s, a) \leftarrow Q(s, a)+\alpha(r + \max Q(s'', a')-Q(s, a))$$

In our project, we use Q-LEARNING to update the value for each state.

IV. Results and Analysis

In the result analysis sections, currently three performance measurements are employed, including max tile in the board, the score and movements when game ends. We firstly compare the effect of depth of tree search, then compare different tree search algorithms. Finally, we compare the tree search algorithm with the reinforcement learning.

A. Comparison of Tree Search

(1) Monte-Carlo Tree Search

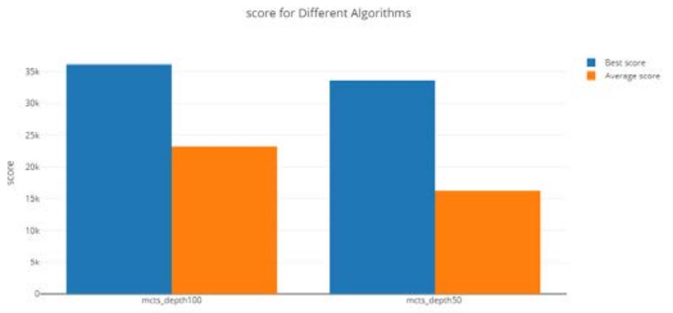


Fig. 7. Average and Best Score for MCTS *This figure shows the difference of average and best score gained for MCTS in depth 50 and depth 100 for 20 runtimes. MCTS with 100 depth has a higher performance in both average and best score.*

From the Fig.7 we can see that the performance improved as the depth increase in MCTS. In Fig.8 the performance also shows an improvement as the depth of the tree increases.

(2) Minimax and Expectimax

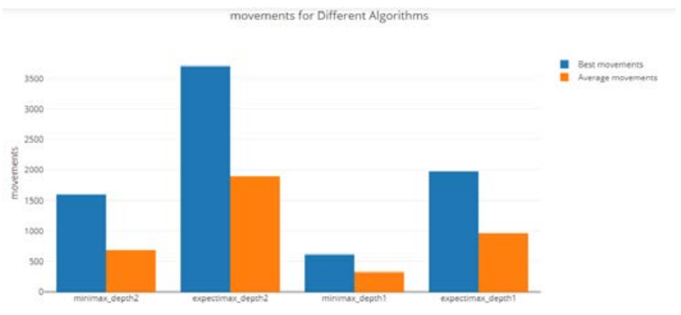


Fig. 8. Average and Best Movements for Minimax and Expectimax *This figure shows the difference of average and best movements till game ends for minimax and expectimax in depth 1 and depth 2 for 100 runtimes. The grouped bar from left to right are minimax with depth 2, expectimax with depth 2, minimax with depth 1, expectimax with depth 1. Expectimax with deeper depth has a higher performance in both average and best score.*

In Fig.8, in general expectimax algorithm always perform better than the minimax algorithm because it contains much stronger domain knowledge compared to minimax. However, because the opponent randomly generate the tiles on empty state, the performance of expectimax is not so stable. In Fig. 9 we can see although the expectimax algorithms in general have better performance, sometimes the minimax algorithm can be able to outperform it.

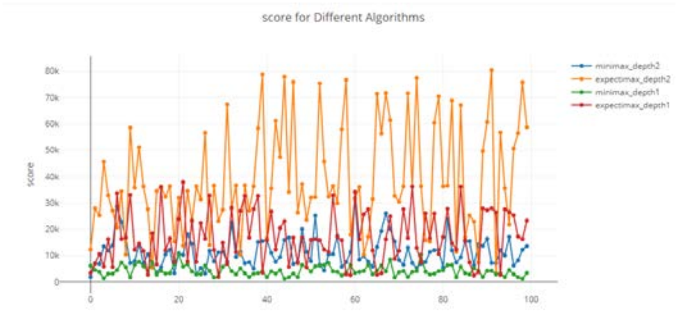


Fig. 9. Score for Minimax and Expectimax *This figure shows the scores for minimax and expectimax in depth 1 and depth 2 for each runtimes. The line from top to bottom are expectimax with depth 2, expectimax with depth 1, minimax with depth 2, minimax with depth 1.*

In Fig. 10 we can see that the expectimax algorithms usually have darker stack, which means it reaches higher max tile than minimax algorithm. When we compare the expectimax with depth 2 and depth 1. We can see that expectimax with depth 2 has darker stack and the size of the second dark stack is larger than the stack with same color in expectimax with depth 1.

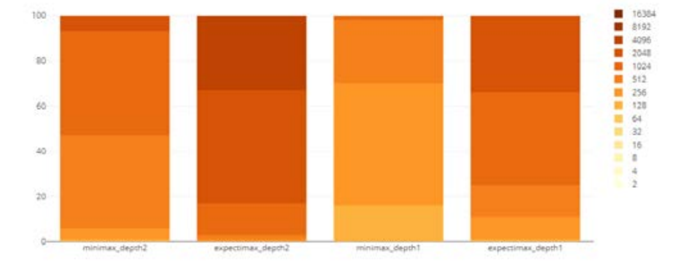


Fig. 10. Max Tile Count for Minimax and Expectimax *This figure shows the max tile counts for minimax and expectimax in depth 1 and depth 2 for each runtimes. The darker stack on the chart, the higher score it presents. The larger the stack size is, the more times it achieved in the experiment. The bar from left to right represents, minimax with depth 2, expectimax with depth 2, minimax with depth 1, expectimax with depth 1.*

B. Comparison between Tree Search and Reinforcement Learning.

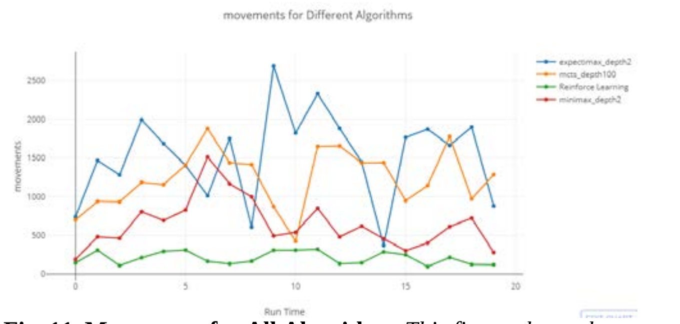


Fig. 11. Movements for All Algorithms *This figure shows the movements for 20 run times. The line from top to bottom represents expectimax with depth 2, MCTS with depth 100, minimax with depth 2, and reinforcement learning.*

From the Fig. 11 we can see that in most cases the expectimax and MCTS outperform the minimax and reinforce learning.

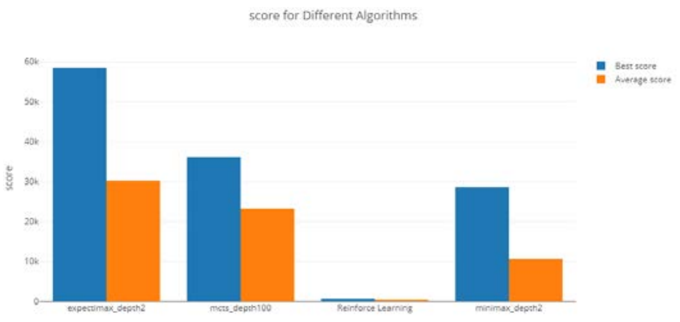


Fig. 12. Average and Best Score for All Algorithms. *This figure shows the best and average score for each algorithm. The bar from left to right represents expectimax with depth 2, MCTS with depth 100, reinforcement learning and minimax with depth 2.*

From the Fig. 12 we can see that although the expectimax outperform other algorithms, but the best score of MCTS can be higher than the average score of expectimax. The expectimax shows a large gap for its best score and average score, which means that the algorithm is not so stable.

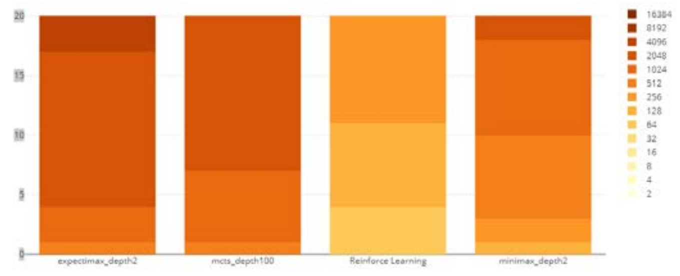


Fig. 13. Max tile Count for All Algorithms *This figure shows the max tile counts for all algorithms. The bar from left to right represents expectimax with depth 2, MCTS with depth 100, reinforcement learning and minimax with depth 2.*

From the Fig. 13 we can see that the expectimax algorithms has a larger counts of times that reaches 2048. And currently it is the only algorithm that can be able to reach 4096.

V. CONCLUSION

This paper implements, compares and analyzes different algorithms to beat 2048 game. The expectimax shows the best performance while the reinforcement learning has the worst performance, which might because we simplify the reinforcement learning algorithm to reduce the computational complexity and the lack of training times. The performance of Qlearning is expected to be better as the training time increases.

Currently, most of the algorithm we implemented are able to reach 2048. While the minimax and expectimax algorithms still need the domain knowledge to help it to decide which action is more preferable. MCTS shows a stable performance to reach without domain knowledge.

In this project, minimax is implemented via the heuristic of monotonicity, smoothness and emptiness, and expectimax is implemented via the heuristic of S shape. The result of S-shape wins the combination of monotonicity, smoothness and emptiness because it provides a better scheme to merge tiles. Therefore, for algorithms that requires domain knowledge, the design of heuristic is critical.

For minimax and expectimax, the game are dominated by domain knowledge. Although the system can reach a satisfying result though the carefully designed heuristic function, it is hard to generalize such algorithm because the heuristic in different field are quite different. The algorithms that do not require domain knowledge shows a wider possibility for solving different problem. Although their performance in this project is still limited. A bright future can be expected.

Due to the limitation of our computing device, we sacrifice some heuristic for states and the storage of possible states for learning in order to get the algorithms run. In the future work, a better heuristic and learning scheme can be designed as the capacity of computing improves. Also, a more detailed exploration is desired for the algorithms that do not need domain knowledge since they are more genetic to solve problems in different field. What's more, since the 2048 Game contains a large element of randomness, any research that explores the approaches to minimize the risk and randomness is also desired.

REFERENCES

[1] Cirulli, Gabriele. (2014). "2048." (nd) . http://ovolve.github.io/2048-AI/

[2] Dickey, Megan Rose (2014). "Puzzle Game 2048 Will Make You Forget Flappy Bird Ever Existed". Business Insider.

[3] Nick Statt (2014, March). '2048 starts easy; gets hard. Here's how to make it easy again'. https://www.cnet.com/news/2048-starts-easy- gets-hard- -heres-how -to-make-it-easy-again/

[4] "What is the optimal algorithm for the game, 2048?" (2014, March). http://stackoverflow.comlquestionsI22342854.

[5] Rodgers, P., & Levine, J. (2014, August). An investigation into 2048 AI strategies. In Computational Intelligence and Games (CIG), 2014 IEEE Conference on (pp. 1-2). IEEE.

[6] Taiwan 2048-bot, http://2048-botcontest.twbbs.org/

[7] Fan, K. (1953). Minimax theorems. Proceedings of the National Academy of Sciences, 39(1), 42-47.

[8] Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. Artificial intelligence, 6(4), 293-326.

[9] Ovolve. (2014). '2048-AI'. https://github.com/ovolve/2048-AI.

[10] David Greydanus, nneonneo. (2014). "Expectimax algorithm in 2048". https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048

[11] Fenix Chen. (2014, Nov). "AI Designing for the game 2048", https://sites.google.com/a/umn.edu/fenixshrine/ai-projects-homeworks/ai-for-2048-with-mcts

[12] Erev, I., & Roth, A. E. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. American economic review, 848-881.

[13] Wen Zhong. (2015). "AI for game 2048". https://github.com/johnzw/AI-for-game-2048

[14] Georg Wiese. (2016). "2048 Reinforcement Learning". https://github.com/georgwiese/2048-rl.

[15] Tjwei. (2016). "2048 Neural Network". https://github.com/tjwei/2048-NN.

[16] C. J. C. H. Watkins and P. Dayan, "Q-Learning," Machine Learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] Abdelkader, A., Acharya, A., & Dasler, P. (2015). 2048 is NP-Complete.

[18] Mehta, R. (2014). 2048 is (PSPACE) hard, but sometimes easy. arXiv preprint arXiv:1408.6315.

[19] Yun Nie, Wenqi Hou, Yicheng An. (2016). "AI Plays 2048". http://cs229.stanford.edu/proj2016/report/NieHouAn-AIPlays2048-report.pdf