Beyond 2048: AI models for extended 2048

Han Jiang
Computer Science and Engineering
University of Michigan
Ann Arbor, Michigan, USA
jianghan@umich.edu

Peter Ly
Computer Science and Engineering
University of Michigan
Ann Arbor, Michigan, USA
pmly@umich.edu

Ron Nafshi
Computer Science and Engineering
University of Michigan
Ann Arbor, Michigan, USA
rnafshi@umich.edu

I. Introduction

In 2014, the hit game 2048 became explosively popular, using simple gameplay and stochastic elements to draw millions of players. 2048 is a simple game played on a 4-by-4 board as tiles with value 2 and 4 are randomly placed. Players aim to combine tiles to create 2048 and larger power of 2 tiles to raise their scores. The secret to the game's acclaim is that despite its incredible simplicity, it's deceptively difficult to create the largest powers of 2 and earn the highest scores. Indeed, it's actually difficult to lose the game for hundreds to thousands of moves even with no strategy whatsoever.

Thus, beating the game with the highest scores is incredibly hard for many traditional AI game-playing methods. Even in the simplest case of allowing up to 2048 on a 4-by-4 board, there are more than 100 billion different states; the exponential number of board states combined with the stochastic element of new tiles and most critically that poor decisions may not result in a loss for thousands of moves make this an incredibly difficult search problem.

Motivated by this, we aim to explore existing solutions and heuristics that exist in games with large game spaces where pruning is difficult. We explore expectimax optimization and temporal difference methods on the extended 2048 problem, and evaluate performance the performance of these algorithms against a random player by using two different scoring metrics.

This report gives a comparison of the performance of three different agents (a random agent, an expectimax agent, and a temporal difference learning agent), including a brief study of the environment, the methods used, challenges encountered during the project, our current results, and future work that can be done.

A. Team Member Contributions

All team members contributed to the writing of the reports and presentation. Han Jiang implemented the temporal difference approach. Peter Ly implemented the expectimax algorithm. Ron Nafshi collected data for the implementations on large board sizes.

II. Problem Description

A. Rules of the Game

In the general game, 2048 is played on an initially empty $N \times N$ board. At the start of each turn, a new tile randomly appears on a random empty spot on the board. The cell on which the new tile is placed is chosen uniformly at random, and the number on the tile is 2 with probability 0.9 and 4 otherwise. On each turn, the player may force all tiles on the board to slide left, right, up, or down, until they reach the edge of the board or collide with another tile in the way. If a slide would cause two tiles with the same power of 2 to collide, the tiles merge into the next largest power of two. A tile may not merge more than once in the same slide. The game continues until the board is completely filled and no possible merges can be made. A player may win 2048 by creating the namesake 2048 tile after many merges, but may continue to play until they reach a stage where no possible merges can be made.

B. Environment Implementation

We first implement this environment in Python. We store the game board as an $N \times N$ array, where N is an argument given by the user. The environment is used with a driver program, which prompts a user or agent to select to move all tiles up, down, left, or right. We also implement n-tuple temporal difference learning in C++ for efficiency.

C. Computational Resources

We ran experiments on the smaller board sizes 2×2 to 4×4 on our local systems. We ran experiments on the larger board sizes 5×5 and 6×6 on the Great Lakes computing clusters.

D. Challenge Analysis

In general, on an N-by-N game grid, the largest possible tile is $2^{(N \times N+1)}$, meaning there will be at most $(N \times N+1)^{(N \times N)}$ game states. In other words, the number of game states is exponential in the area of the game board. Searching the entire state space to identify optimal strategies in the game quickly becomes infeasible due to this exponential growth with area of the game board.

The stochastic nature of 2048 further complicates analysis; after every move the player makes, a new tile is introduced to a randomly chosen, empty tile on the grid. This uncertainty has two components: 1) the exact cell on which the tile is introduced and 2) the number on the tile.

There are two different scoring methods in 2048: a player may win by creating a 2048 tile with a variety of board configurations; alternatively, a score is kept by calculating the total sum of the values of all tiles combined over the course of the game. In this case, the maximum score is attained by

Fig. 1. An example of the highest achievable score on a standard 2048 board

a final configuration such as in Figure 1. We aim to explore both the relaxed problem of creating just one 2048 (or larger power of 2) tile, and the more complex task of attaining the largest possible score on boards of arbitrary size.

III. Related Work

Attempts to make probably perfect 2048 models have been limited, largely due to the huge number of game states. Models using Markov Chains were only able to make perfect models for up to the 64 tiles, and even this small problem had more than 40 billion states [7]. Thus, models that can prune many moves in advance or efficiently explore in this huge game space are critical to finding better solutions. As a stochastic game, 2048 has been studied using mainly two different techniques. Initial attempts to apply AI techniques to 2048 used the expectimax algorithm and alpha-beta pruning of the game tree [1], [2]. The expectimax algorithm and alpha-beta pruning are able to create a 32768 tile in 36% of the games sampled on, and yielded a median score of 387222 [3]. Reinforcement learning is the other main technique used to study 2048 in AI, and is the more utilized and effective technique. Szubert and Jaśkowski (2014) utilized a class of model-free reinforcement learning called temporal difference (TD) learning on a N-tuple network to achieve a win rate of 97% [4]. Wu et al. (2014) extended this result so that tiles with large numbers (32768) can be

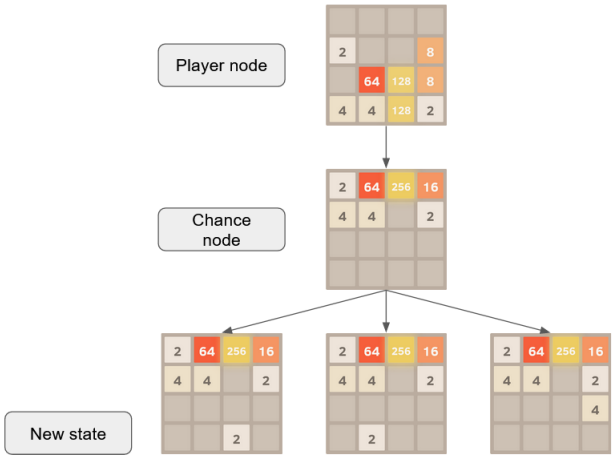

Fig. 2. The arrow between the max node and the chance node represents moving up. The arrows from the chance node to the new state represent some (but not all) of the random outcomes possible after the player's move.

achieved with high probability (31.75%) [5]. Jaśkowski (2017) further refined this technique so that the 32768 tile could be made in 70% of simulated games, and so that the average score was greater than 600,000 [6]. Nearly every study of the game 2048 applying these techniques is limited to variations of 2048 with small 4-by-4 boards. The goal of this project is to identify whether these techniques can be effectively applied to significantly larger search spaces by studying the application of the techniques to 2048 generalized to $N \times N$ boards.

IV. Methodology

We explore three algorithms, a random algorithm, expectimax, and temporal difference learning to implement our 2048 agents. The performance of the random algorithm provides a rough baseline to compare the performance of the expectimax algorithm and the performance of the temporal difference learning algorithm.

A. Random

The random agent is simply a naïve agent which selects a random direction to move on each turn, and terminates when there are no more possible moves remaining.

B. Expectimax

The expectimax algorithm is an analog of the minimax algorithm, where we have a maximizing player as in the minimax algorithm, but replace the minimizing player with a non-deterministic player/node. This induces a game tree with an alternating layer structure in which each node represents a board state. The branches from the nodes of the maximizing player represent the actions of moving the board in the four cardinal directions. The branches from the nodes of the chance player represent all the possible placements of the new tile after the maximizing player agent makes their move. See fig. 2 for a snippet of what this game tree looks like.

We then define the value of a node to be the the sum of several different heuristics applied to the game board. Following [1], we define heuristics based on the number of empty spaces on the board, the closeness of tiles which are similar in value, and the position of large value tiles. Boards which have a large number of empty spaces, closely pack large tiles to each other, and position the large tiles on the edge of the board have higher heuristic value than boards which do not. These heuristics are chosen to implement the strategy of putting large value tiles in the corner, a strategy often used by human players.

A drawback of the expectimax algorithm, and other decision tree approaches is the considerable depth of 2048 game trees. Often, the depth of the tree is considerably large with an exponential number of states, and because decision tree approaches are essentially brute-force searches, we must limit the depth of the search. To process the tree, we perform an iterative deepening search as is performed in several other references such as [1], [2].

C. Temporal Difference Learning

The second method we will use to learn the game temporal difference learning. The basic idea of the learning agent is by making moves and update the value table for the states. The agent will choose the move which will generate the most reward, and after making the move the agent will update the state value of the state before it makes the move based on the value of the reward, the state before the move as well as the state after the move. We do need to make two main modifications to the basic idea talked above because of the special environment the mechanisms of the game 2048, which will be discussed in the following subsections.

1) Evaluating Afterstates: As is introduced in previous sections. In this game, after making a move, besides possible merging of tiles with the same value, a random new tile will also be generated. The randomness of the tile generation adds stochasticity to the state value evaluation. To counter this problem, instead of evaluating the new game state after each move, we choose to evaluate the after state, which is the state after the move and the possible merges but before the new random state is generated. (The "afterstate" is shown in the figure below)

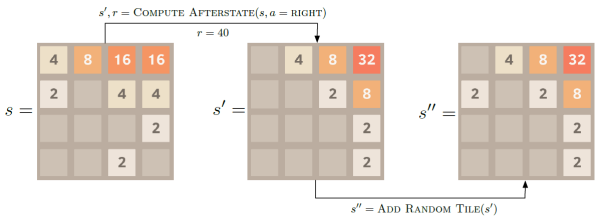

Fig. 3. Example of a afterstate, from [3]

According to several studies ([4], [6], [5]), evaluating the afterstates instead of the new game states significantly improves the accuracy of the temporal difference learning agent.

2) Using Tuples as States: The second modification is about the state representation of the game 2048. Intuitively we would use the entire board as a state, but since theoretically, there are 18 possible tile values in the game 2048, the state space for the game 2048 (if we use the entire board as a state) contains a possible number of $16^{17} \approx 4.7 \times 10^{21}$ states, which far exceed the computational limits. Like many other studies on TD learning of the game, we utilize the value tables of several feature vectors, which is called an n-tuple network, to approximate the value of each state. An n-tuple network is comprised of m n_i-tuples, where m is the number of our choice and n_i denotes the size of the i-th tuple. Figure 4 is an example of a 4-tuple and its look up table The example tuple is

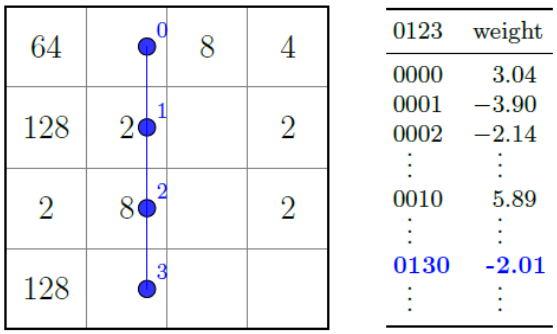

Fig. 4. Example 4-tuple and its lookup table, from [3]

a vertical line consists of 4 tiles, to find the value for this tuple we will look at the entry 0130 because we encode the empty tile to 0, and a tile with value v is encoded to $\log_2 v$. For tuple selection, we will use the same tuples as [5], which are the four 6-tuples shown in Figure 5. Since 2^{17} is ideal and not reachable

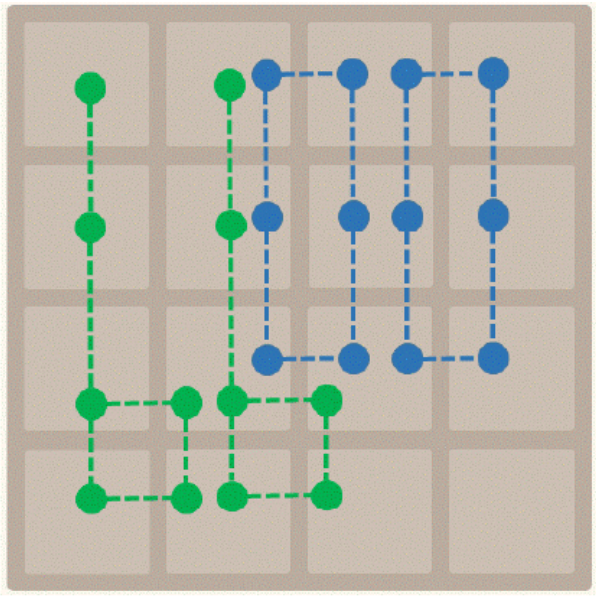

Fig. 5. The four 6-tuples used by [5]

in almost all of the learning scenarios, we will just assume the

possible number of tiles, c, is 16 for easier implementation. In this case, the number of entries in the look up tables of the tuples are $4 \times 16^6 \approx 7 \times 10^7$, which is computationally reasonable. We will then talk about the evaluation of the state space if we choose to use tuples to represent the state space:

3) Evaluation using N-tuples: We will first create a look up table based on the shapes and number of tuples we choose, and initialize all entries to zero. For a state s, we will first choose the action by evaluating the reward of the action and the value of the afterstate reached using the action. Then, we get both the after state s' and the next state s_{next} (afterstate with a random generated tile). By the same method we will get the after state of s_{next}, s'_{next}, as well as a reward r_{next}. We will update the look up table by adding

$$\frac{\alpha}{m} \cdot max\{(r_{next} + V(s'_{next}) - V(s'), 0\}$$

to the 4 entries corresponding to s' (Correspondence illustrated in Figure 4). Note that α is the learning rate (we currently set it to 0.1), and we divide by m (the number of tuples) to "average" the gain on all tuples. The evaluation of a state s, $V(s)$, is by adding up the values of corresponding tuples. Detailed learning procedure is provided by Algorithm 1.

Note that $V(s)$ is the value of the board s, which is calculated by adding up the values of the corresponding tuples. In our implementation, we will first use only the 4 tuples provided in [7] to see how the temporal difference learning behave with only 4 tuples. Then we will add 4 more custom tuples shown in Figure 6 to see how adding more selected tuples will affect the performance of the agent.

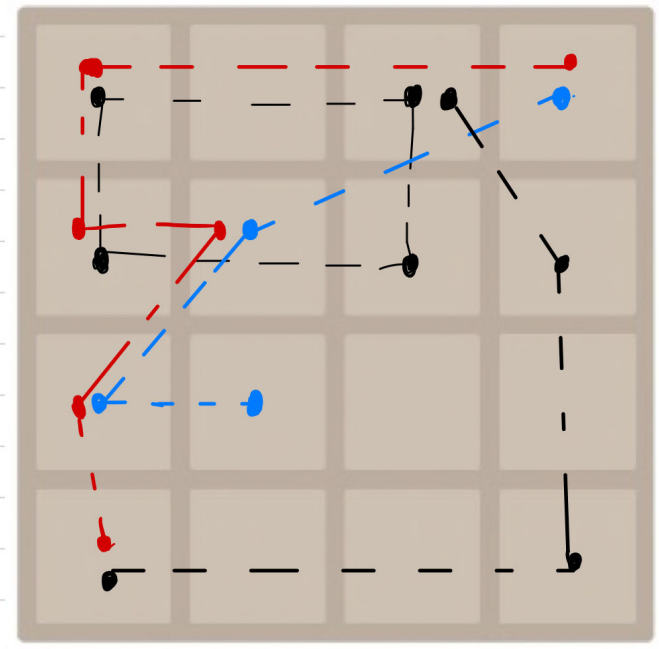

Fig. 6. The four custom tuples

Algorithm 1 Temporal Difference Learning

function PLAYGAME
  $score \leftarrow 0$
  $s \leftarrow Initialized\ Game\ State$

  while s *is not terminal state* **do**
    $a \leftarrow argmax_{a' \in A(s)}$EVALUATE$(s, a')$
    $r, s', s'' \leftarrow$ MAKEMOVE(s, a)
    LEARN(s, a, r, s', s'')
    $score \leftarrow score + r$
    $s \leftarrow s''$

  end while
  return $score$
end function

function MAKEMOVE(s, a)
  $s', r \leftarrow$ COMPUTEAFTERSTATE(s, a)
  $s'' \leftarrow$ ADDRANDOMTILE(s')
  return (r, s', s'')
end function

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTEAFTERSTATE(s, a)
  return $r + V(s)$
end function

function LEARN((s, a, r, s', s''))
  $a_{next} \leftarrow argmax_{a' \in A(s'')}$EVALUATE$(s'', a')$
  $s'_{next}, r_{next} \leftarrow$ COMPUTEAFTERSTATE(s'', a_{next})
  $V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$
end function

V. EXPERIMENTS

A. Evaluation

To evaluate the results, we will learn from 1000 gaming instances, and calculate the win rate, calculated as the percentage of games in those 1000 where the agent achieves a 2048 tile, mean score, max score and max tile number reached of those games. We then repeat the same for the next 1000 games, which means the results of those new games are from an agent which has already been trained for many game iterations. We will iterate this process until the result converges or tend to converge under a reasonable program runtime. In this case we run it 100 times, which means $100 \times 1000 = 100000$ games to learn. For temporal difference learning, we will set the learning rate α to 0.1, let $n = 1000$, and run the report process for 1000 times. The tuple choice for 5×5 and 6×6 boards are of same shapes to Figure 5.

B. Results

Tables I–VI provide the data detailing the percent of games in which the random, expectimax agent, and temporal difference agent achieve a certain value as the max tile in a single game.

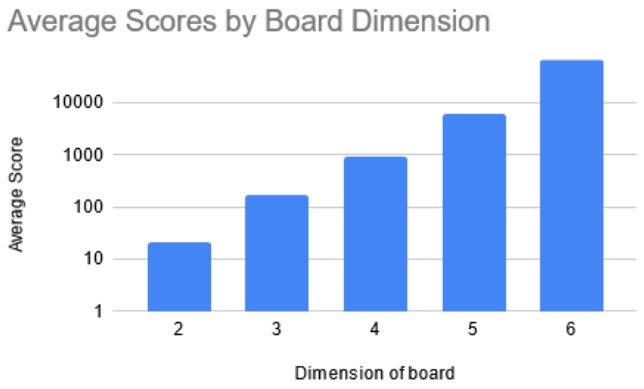

Fig. 7. Average random agent scores increase exponentially with the dimension of the board.

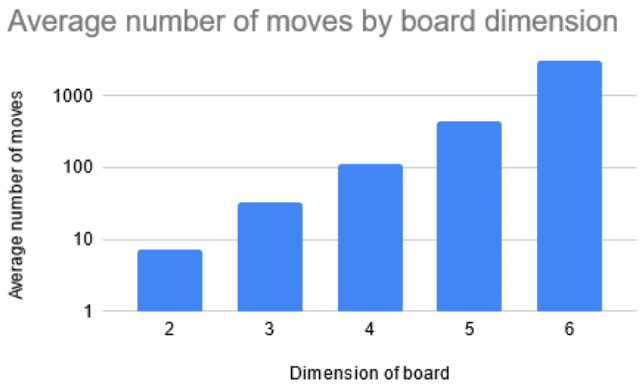

Fig. 8. Average number of moves taken by the random agent increase exponentially with the dimension of the board.

We also more measurements that are specific to method, which will be presented in the following section. Statistics for the temporal difference learning agent on a 6×6 board are not available due to time constraints for testing.

1) Random: The random agent, which serves as out baseline method, performed quite poorly compared to the other two methods as expected, but still exhibits exponential growth in the number of moves and scores as the board dimension increases.

We also collect the average score per game (see fig. 7) and average number of moves made per game (see fig. 8) in our experiments for the random agent.

2) Expectimax: We observed that the average score (cf. fig. 9) and the number of moves achieved by our expectimax agent (cf. fig. 10) increases exponentially with the board size. This matches our initial assumption that the analysis of 2048 would become more complex as the board size increases due to increase in the number of game states. This performance greatly improves on the performance of the random agent.

3) Temporal Difference Learning: Table VII provide more detailed information for statistics of playing 2000 to 200000 games (Using 8 tuples). Figure 11 is the graph comparing the win rate of agent learning with 4 tuples and and 8 tuples

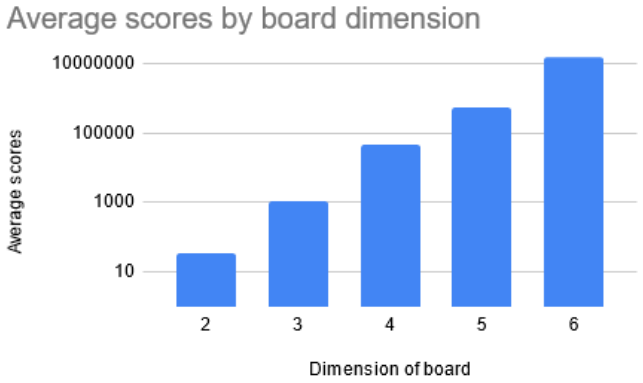

Fig. 9. Average expectimax agent scores increase exponentially with the dimension of the board.

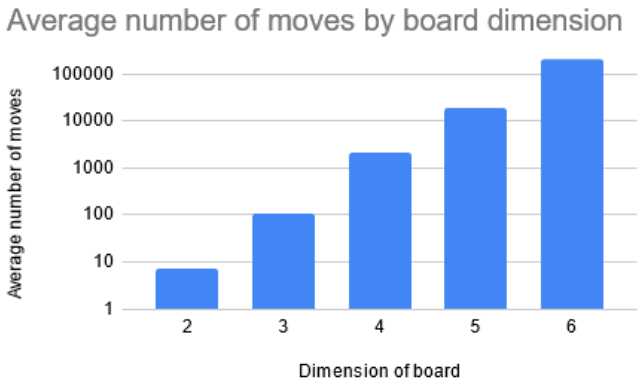

Fig. 10. Average number of moves taken by the expectimax agent increase exponentially with the dimension of the board.

TABLE I: Percent of games tile is the max achieved for 2×2 boards

Max Tile	Random	Expectimax	TD
4	39.1	30.1	22
8	46.1	9.9	10
16	14.7	59.2	67
32	0.1	0.8	3

TABLE II: Percent of games tile is the max achieved for 3×3 boards

Max Tile	Random	Expectimax	TD
4	0.1	0	0
8	6.2	0	0
16	37	0	0
32	48.9	1	1
64	7.7	26.3	19
128	0.1	60	65
256	0	12.7	15

TABLE III: Percent of games tile is the max achieved for 4×4 boards

Max Tile	Random	Expectimax	TD
16	0.5	0	0
32	10.6	0	0
64	44.3	0	0
128	39.4	0	0
256	5.2	0	0
512	0	2	2.3
1024	0	8.9	6.6
2048	0	46.7	23.8
4096	0	42.1	66.4
8192	0	0.3	0.9

TABLE IV: Percent of games tile is the max achieved for 5×5 boards

Max Tile	Random	Expectimax	TD
64	0.9	0	0
128	7.7	0	0
256	35.2	0	0
512	47.6	0	0
1024	8.6	0	0
2048	0	0	0
4096	0	1	1
8192	0	5	3
16384	0	38	31
32768	0	55	63
65536	0	1	2

TABLE V: Percent of games tile is the max achieved by random agent for 6×6 boards

Max Tile	Random
512	0.4
1024	7
2048	28.1
4096	51.7
8192	12.8
16384	0

TABLE VI: Percent of games tile is the max achieved by expectimax agent for 6×6 boards

Max Tile	Expectimax
65536	17.1
131072	9.1
262144	54.5
524288	19.3

(4 tuples from [7] and the 4 custom tuples in Figure 6). We notice that adding 4 additional representative tuples will significantly increase the win rate of the agent, which indicates the importance of choosing tuples that are as representative as possible (discussed further in future works).

VI. Future Work

For expectimax, we noticed for 4×4 boards that the depth limit of the iterative deepening greatly affects the speed of the agent. The branching factor of a game of 2048 is quite large because we must simulate placing a random tile after every possible move. This means that the branching factor is at most $4 \times N^2$ where N is the dimension of the board. To ensure that we were able to complete data collection, we had to fix the depth limit of the iterative deepening to at most 3 (that is, at each root node in the game tree, we had the expectimax agent plan for 3 player-computer pairs of moves). The trade-off between depth limit and expectimax agent performance is not well-understood, at it would be interesting to see how this trade-off behaves.

For temporal difference learning, firstly we noticed that tuple selection will have a significant influence on the performance of the temporal difference learning agent. Thus figuring out how to select the most ”representative” tuples will be a subject of interest, in the future we will study the heuristics of tuple selection and we can even use reinforcement learning to help with the selecting the tuples. Moreover, we will try to improve our TD agent from two perspectives: Selecting and adding more tuples with better representations of the whole state than [5], as well as applying the technique of multistage update specified in [6].

We also find that Expectimax and Temporal Difference Learning are both powerful algorithms in the extended 2048 search problem, performing much better than random in all

TABLE VII: Statistics for the Agent

NumGames	Mean Score	MaxScore	WinRate(%)	MaxTile
2000	8255.45	17 164	0.3	2048
20000	26216.2	71 380	55.3	4096
40000	44620.2	130 276	85.7	8192
60000	53219.6	147 448	87.7	8192
80000	56041	132 704	90.7	8192
100000	58864.1	139 048	91.8	8192
120000	61141	139 024	93.4	8192
140000	61144.9	144 528	93.3	8192
160000	62222.1	151 332	93.3	8192
180000	62108.8	142 764	93.7	8192
200000	64136	173 984	94.9	8192

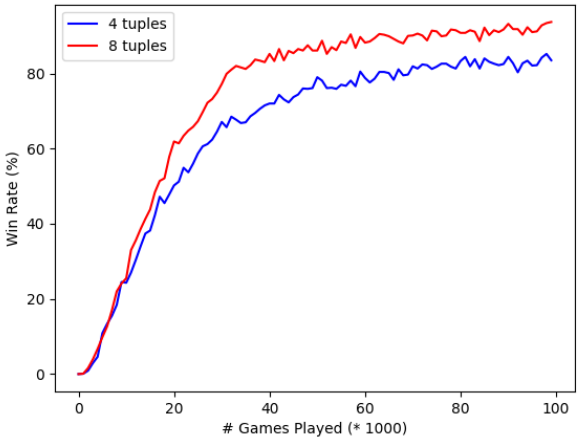

Fig. 11. Win Rate of the agent against games played

scoring metrics used. However, while both algorithms are able to robustly score well as the board size increases, neither algorithm is able to efficiently handle the exponential increase of state space in the size of the board. Memory is not a concern as the entire board can be stores as a NxN array, with the $(N \times N + 1)^{(N \times N)}$ game states being the major issue for extended 2048 algorithms. Using the Great Lakes Slurm HPC Cluster available at the University of Michigan Ann Arbor, completing just one game of 6x6 2048 took over 3 hours using expectimax, and was too computationally expensive for our temporal difference learning agent. On the 7x7 board, one game played by the expectimax would not end in under 8 hours, with similar results for temporal difference learning agent. Future work may fruitfully explore how these two algorithms generalize, as it is beyond of the computational resources used in this study. Novel algorithms exploring this problem would also benefit from analysis in reducing the size of the game state even further.

REFERENCES

[1] John Lees-Miller (2018, Apr 10) Optimal play with markov decision processes. https://jdlm.info/articles/2018/03/18/markov-decision-process-2048.html

[2] Zaky, A. (2022, May 15). Minimax and Expectimax Algorithm to Solve 2048. https://doi.org/10.31219/osf.io/az7vf

[3] nitish172. (2014, Mar 12). What is the optimal algorithm for the game 2048? Stack Overflow. https://stackoverflow.com/questions/22342854/

[4] Wojciech Jaśkowski. (2016, Dec 12). Temporal difference learning of N-tuple networks for the game 2048.

[5] Wojciech Jaśkowski, Marcin Szubert. (2014, Oct 23). Temporal difference learning of N-tuple networks for the game 2048.

[6] Wojciech Jaśkowski. (2017, Jan 11). Mastering 2048 With Delayed Temporal Coherence Learning, Multistage Weight Promotion, Redundant Encoding, and Carousel Shaping

[7] 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.

VII. APPENDIX

We include the data used to generate figs. 7 to 10.

TABLE VIII: Statistics collected for the random agent

Dimension	Average score	Avg. # of moves
2	20.668	7.11
3	170.68	33.206
4	960.136	113.757
5	6023.824	451.989
6	66793.524	3184.35

TABLE IX: Statistics collected for the expectimax agent

Dimension	Average score	Avg. # of moves
2	34.972	7.445
3	1136.832	105.657
4	44487.112	2163.954
5	516711.56	18678.01
6	14745584	204785.909