An investigation of Tree Search Methods for solving the 2048 puzzle game

ABSTRACT

Tree search methods have been commonly used to solve the 2048 puzzle. Current methods rely on running many simulations until a path which reaches the 2048 tile is reached. It is shown in this report that the expectimax search algorithm has the potential to reach tiles as large as the 512 tiles for tree depths as low as 1, when awarded according to a commonly used human strategy.

KEYWORDS

Minimax, Expectimax, Monte-Carlo Tree Search, Upper Confidence Bounds, 2048 Puzzle Game

1 INTRODUCTION

In March 2014, the game 2048 was released to the public by Gabrielle Cirulli [1, 7]. In this game, the player is presented with a 4×4 grid initially consisting of two randomly placed '2' or '4' number tiles. The player must then choose a direction (up, down, left, right) for the tiles on the grid to move in. After each move, a new tile appears, being either 2, with a 90% probability of appearing, or a 4, with probability 10%.

When the grid is moved such that two cells containing the same number are next to each other, those two tiles are added together to make a singular tile containing the value equal to the sum of the two tiles.

Mathematically speaking, the largest possible number to obtain on the game grid is $2^{17} = 131,072$, however this relies on players obtaining a snake-like grid such that a 4-tile appears in the empty corner so that the 131072-tile can be obtained.

Otherwise, the next largest tile which is commonly sought after by players (and search algorithms) is that given by $2^{16} = 65,536$ [4, 9–13].

In this report, the aim is not necessarily to aim for the highest tile, but to identify whether or not existing algorithms can be modified to follow human strategies.

2 RELATED WORK

To date, plenty of research has been carried out on using tree search methods to solve single-player games, some of which focus on the game of 2048. Many existing methods tend to rely on strategies such as the Minimax, Expectimax, and Monte-Carlo methods, with some going as far to rely on techniques such as bitboards which are used in chess engines [2, 4, 8–13].

In terms of game strategy, players have found that the optimal way to solve the game is by creating what is essentially a snake-like pattern on the game board, where the largest number occupies one of the corners, and each of the tile neighbours decrease by a factor of two, until the opposite corner of the board is reached.

The work illustrated in this report will investigate the usage of what will be later referred to as a "score matrix", which will encourage the AI to generate such a snake-like pattern to solve the game.

3 METHODOLOGY

This report will focus on the usage of three different algorithms for solving the 2048 puzzle, with the first two being the Minimax and Expectimax, and the last being the Monte-Carlo Tree Search method. The report will focus on adapting different algorithms to suit the random nature of how tiles appear on the 2048 game board.

All tree methods will be evaluated over a number of different initialisation parameters, such as search depth and exploration constants. All three methods will be compared on their ability to reach the highest tile possible.

3.1 Minimax & Expectimax

The minimax tree search works as a backtracking algorithm, which evaluates nodes on a tree based on the worst possible outcome. This can be very useful for games such as 2048, where a given move could result in a potential loss, so knowing the worst case scenario for a given move can be used to mitigate this [5].

The expectimax search method works very similarly to this, except it evaluates the **expected** score of a branch, as opposed to the **minimum** score, meaning that it is not so clear which branches may lead to a loss, however can be very useful when the agent wishes to reach a high score quickly [6].

Intuitively, minimax aims to prioritise long-term game strategy, whereas the expectimax will focus on reaching a high score as quickly as possible.

The scoring method for both algorithms is according to a previously mentioned "score matrix", which awards the tree search for producing a snake-like structure. The following score matrix is to be used:

$$S = \begin{bmatrix} -16 & -12 & -8 & -5 \\ -1 & -2 & -3 & -4 \\ 1 & 2 & 3 & 4 \\ 16 & 12 & 8 & 5 \end{bmatrix}$$

The idea is that the largest tile is awarded for being in the bottom left corner, then the next largest adjacent to it, and so on until we reach the top of the board. The idea of using negative numbers is so that as the bottom of the board fills up, the tree search is penalised for keeping numbers at the top, and is encouraged to find combinations of moves which will take them to the bottom of the board.

The above score matrix is combined with the actual board grid through conventional matrix multiplication, and then reduced from $\mathbb{R}^{4\times4}$ to $\mathbb{R}$ by taking the summation of each of the elements of the resultant matrix.

This value is then penalised by the square of the number of cells occupied on the grid divided by the sum of the cells on the grid. We may define the score formula as follows:

$$Score(\omega) = \sum_{i=1}^{4}\sum_{j=1}^{4}(S \times \omega)_{ij} - \frac{M(\omega)^2}{S(\omega)}$$

where

- ω represents a **board state**.
- $M(\omega)$ represents the **magnitude** of the state ω, i.e. the number of occupied cells.
- $S(\omega)$ represents the overall **sum** of the cells in ω.

This score matrix technique does not appear in any popular literature around the topic of tree search methods for solving 2048.

3.2 Monte-Carlo Tree Search

The Monte-Carlo Tree Search (MCTS) method relies on four main principles; selection, expansion, simulation, and back-propagation. We shall refer to these as the tree estimation phase [3, 13, 14].

First, the algorithm selects a node for expansion. On the first iteration, the root node is used, and then the expansion stage is carried out, i.e. it calculates all of the next possible moves based on the current state. Next, a simulation step is performed, picking moves at random until either the game has been "won" or "lost", and then the result is back-propagated through the tree up to the root node (initial state).

Traditionally when using the MCTS approach, the initial game state is observed as the root node, and then further possible states are extended from the root to form their own nodes. However for 2048, we depend on a slightly different structure. For example, take the game state in Figure 1. It is possible for the player to move the tiles in any direction, however once a move has been played, there are 14 possible spaces for either a 2 or 4 tile to appear on, resulting in a further 28 possible game states.

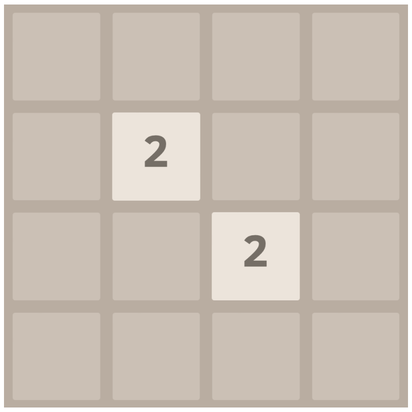

Figure 1: A possible starting state for the 2048 game.

Because of this, the traditional MCTS technique has to be modified slightly in order for it to work with the 2048 game. For each game state, each of its children nodes comes paired with the move played, along with a possible spawn location for the next tile, resulting in up to 56 possible children for each node, as opposed to 4 if each child represented a possible direction.

Further, the MCTS approach used here features the Upper Confidence Bound for Search Trees (UCT) formula in order to select which node is to be expanded next on each iteration. This result is calculated for each child node, and once the MCTS algorithm has exceeded a certain computational budget, usually in the form of a limitation on the number of simulations, the child node with the highest UCT score is selected as the next move.

In the context of 2048, the UCT score is calculated for each child node, and then the child node with the highest score is selected for expansion. However, once the computational phase is complete, the same formula then cannot be used to select the best move at the root node [3].

This is because although the MCTS algorithm may select a move corresponding to a node which has performed well throughout the computational phase, that node only represents one of the final states resulting in that move, and not all of them. It may be the case that a different move is optimal for that Monte-Carlo Simulation, so we need to devise a formula to encapsulate this.

The Upper Confidence Bound (UCB1) formula has been modified in this case so that it takes all of the states with a given move into consideration. Mathematically speaking, let us define $M(\omega)$ as the move required to get from the parent state to that particular child state. Further, we may define the set of child states which are the result of a move m as

$$\mathcal{M}(m) = \{\omega : M(\omega) = m\}$$

Then, the UCT score for a move m can be calculated as follows:

$$UCT = \frac{\sum_{\omega \in \mathcal{M}(m)} S(\omega)}{\sum_{\omega \in \mathcal{M}(m)} V(\omega)} + \varepsilon \times \sqrt{\frac{2\log(\sum_{\omega \in \mathcal{M}(m)} V(\omega))}{\sum_{\omega \in \mathcal{M}(m)} V(\omega)}}$$

where

- $S(\omega)$ is the **score** associated with state ω.
- $V(\omega)$ is the **number of visits** associated with state ω.
- ε is a constant known as the **exploration factor**.

4 RESULTS & ANALYSIS

Both the expectimax and minimax search algorithms were evaluated over a range of maximum tree depth values and exploration constants.

The tree depth controlled how much computational power was spent on each game state, with a higher value taking more time on average.

The exploration constant represented the probability that on any given run, the search tree picked a random route.

From Figure 2 it is evident that the expectimax was the winning algorithm when the two were pitched against each other, reaching much larger tiles, and averaging much higher scores than the Minimax algorithm in most cases.

Note that a tree depth of zero plays all possible moves on the root state, and calculates the scores of each branch based on the resulting state. Then, a depth of 1 will evaluate the same procedure on each of the resultant states, and so on for larger depth values.

Figure 3 illustrates that the expectimax algorithm still dominates, reaching scores of up to 5812 and tiles as large as the 512 square even for a depth as low as 0.

Conversely, the MCTS algorithm proved to be a weak candidate for solving the 2048 puzzle, only reaching scores of up to 1492 in

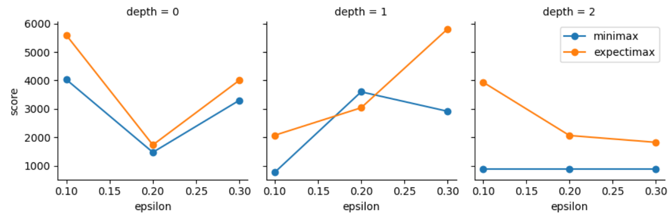

Figure 2: Scores for both the Minimax and Expectimax algorithms for different depth values and exploration constants.

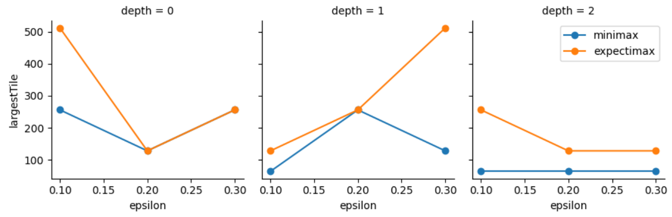

Figure 3: Largest tile values reached by both the Minimax and Expectimax algorithms for different depth values and exploration constants.

simulations, and no higher than the 128 tile. The scores for a range of tree depth and simulation count values is visualised in Figure 4, where it performed optimally at a simulation depth of 10, with a maximum number of 10 simulations.

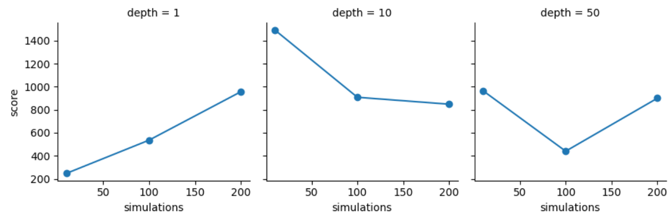

Figure 4: Score values reached by the MCTS algorithm for different depth values and simulation counts.

Further, the maximum tile counts obtained by the MCTS algorithm were much lower than those obtained by the expectimax and minimax algorithms. Figure 5 shows that a depth of 10 allowed the tree search algorithm to obtain the largest tiles.

5 DISCUSSION

Despite the initial belief that the minimax would be the better search method due to evaluating the worst-case scenario for each node, it turned out that evaluating the expected score was more beneficial when the aim was to reach the highest node as possible.

Further, having a "strategy" in the form of a score matrix gave the minimax and expectimax algorithms a huge boost in performance, proving the usefulness of the snake-like strategy used by human players.

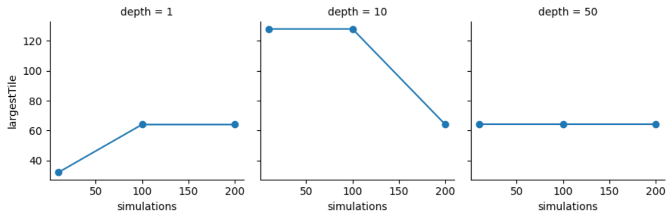

Figure 5: Largest tile values reached by the MCTS algorithm for different depth values and simulation counts.

Figure 3 seems to support this concept, where even for a depth as low as 0 and exploration factor as low as 0.1, the expectimax algorithm was still able to reach large-valued tiles.

On the contrary, the MCTS algorithm equipped with no strategy proved to be a weak choice for the task at hand. Although a unique approach was adopted in order to use MCTS for the game of 2048, it was not as successful as the simpler methods.

6 CONCLUSION

In conclusion, the expectimax algorithm proved consistent at reaching higher valued tiles during simulations.

Although the expectimax proved to be the dominant tree search method in this study, tree search methods can be very computationally expensive. So, with more time and processing power, it could possible that one of the other methods may reach the 2048 tile first.

REFERENCES

[1] [n.d.]. 2048 (Video Game). *Wikipedia* ([n. d.]). https://en.wikipedia.org/wiki/2048_(video_game)#:~:text=Higher%2Dscoring%20tiles%20emit%20a,largest%20possible%20tile%20is%20131%2C072.

[2] [n.d.]. *Bitboards.* https://www.chessprogramming.org/Bitboards#:~:text=also%20called%20bitsets%20or%20bitmaps,chessboard%2C%20one%20bit%20per%20square.

[3] [n.d.]. *Monte Carlo tree search.* https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[4] 2020. *A puzzle for AI.* https://towardsdatascience.com/a-puzzle-for-ai-eb7a3cb8e599

[5] 2022. *Minimax Algorithm in Game Theory.* https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[6] AftaabZia. 2021. *Expectimax Algorithm in Game Theory.* https://www.geeksforgeeks.org/expectimax-algorithm-in-game-theory/

[7] Gabriel Cirulli. 2018. *2048: A small clone of 1024.* https://github.com/gabrielecirulli/2048

[8] CodeBullet. 2018. *AI learns to play 2048.* https://www.youtube.com/watch?v=1g1HCYTX3Rg

[9] Graham Cox. 2023. *What is the optimal algorithm for the game 2048?* https://www.baeldung.com/cs/2048-algorithm#:~:text=When%20we%20start%2C%20the%20board,a%20is%20a%20âĂœ2âĂİ.

[10] Nathaniel Goenawan, Simon Tao, and Katherine Wu. [n.d.]. *What's in a game: Solving 2048 using reinforcement learning.* https://web.stanford.edu/class/aa228/reports/2020/final41.pdf

[11] Hung Guei. 2023. *On Reinforcement Learning for the Game of 2048.* https://arxiv.org/pdf/2212.11087.pdf

[12] Mmiermans. 2014. *Bitboard AI for the game 2048.* https://github.com/mmiermans/solver2048

[13] Gabriel Romualdo. 2020. *Using the Monte Carlo tree search algorithm in AI to beat 2048 (and other games).* https://gabrielromualdo.com/articles/2020-09-12-using-the-monte-carlo-tree-search-algorithm-in-an-ai-to-beat-2048-and-other-games

[14] Sagar Sharma. 2018. *Monte Carlo Tree Search.* https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa