

An Investigation into 2048 AI Strategies

Philip Rodgers and John Levine
 Department of Computer and Information Sciences
 University of Strathclyde
 Glasgow, UK
 Email: {philip.rodgers,john.levine}@strath.ac.uk

Abstract—2048 is a recent stochastic single player game, originally written in JavaScript for playing in a web browser but now largely played on mobile devices [1]. This paper discusses the applicability of Monte-Carlo Tree-Search (MCTS) to the problem, and also Averaged Depth Limited Search (ADLS).

While MCTS plays reasonably well for a player with no domain knowledge, the ADLS player fares much better given an evaluation function that rewards board properties. Attempts to guide the roll-outs of MCTS using an evaluation function proved fruitless.

I. INTRODUCTION

2048 is a sliding tile style game played on a four-by-four grid. Each game starts with two cells populated with a new tile, each with a 90% chance of being a 2 and a 10% chance of being a 4. The player usually has four moves available—up, down, left and right—unless a move would have no effect, in which case that move is not available. The game ends when there are no moves available.

When the player makes a move, all tiles slide in that direction as far as they can. A tile is blocked from sliding by either being at the edge of the board or up against a tile of a different value. If two tiles of the same value are adjacent in the direction of the slide, the tiles will merge and the merged tile will have the combined values of the two original tiles. Each tile can only merge once per slide, and precedence is given to the tile nearest the edge of the board toward which they are sliding. See figures 1, 2, 3 and 4.

Although 2048 is a single player game it can also be considered a two player game, but one where the opponent has no control of its moves. In this respect it is similar to Black Jack, where players play against the dealer but the dealer has no choice in their play. Every time the player makes a move, a new tile will appear in one of the empty cells, with, again, a 90% chance of being a 2 and a 10% chance of being a 4. As each merge frees up a cell and each move causes a cell to become occupied, the player must try to average one merge per move in order to continue playing.

2048 is not like other sliding tile games, such as Rush Hour [2] and 15 Puzzle [3], as it does not have an optimal path from the start state to the goal state. Although it does officially have goal states—a tile with the value 2048—this goal is generally removed for AI players, leaving the game open-ended.

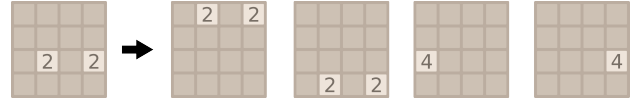


Figure 1: Results of slides with merging.

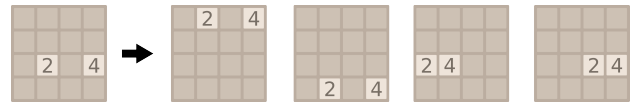


Figure 2: Results of slides without merging.

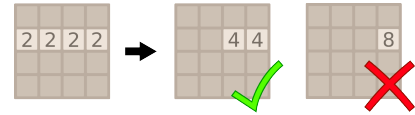


Figure 3: Tiles can only merge once per slide.

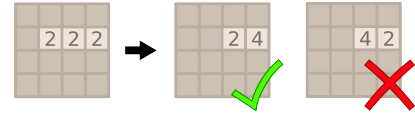


Figure 4: Tiles nearest the edge merge first.

In theory, the maximum attainable tile value is 131072 (2^{17}). This is because in order to make the tile 2^n , you need all the tiles $[4, 8, \dots, 2^{n-1}]$ on the board, so generating a 2^{18} tile would require seventeen tiles, but the board only has sixteen cells [4].

II. PREVIOUS WORK

A. Mini-Max

The first published AI for 2048 appeared on the internet in March 2014 [5]. The approach taken was to treat the game as adversarial; the AI does traditional depth-limited Mini-Max searching, assuming the opponent will place the worst possible tile. While this approach seems reasonable—plan for the worst—the AI is making decisions based on events that may be highly unlikely happen, which is clearly sub-optimal.

The author of this AI does, however, introduce an evaluation function based largely on *monotonicity*. Boards where each row and column of tiles are either increasing or decreasing in value have a higher monotonicity value

than those that are not (ignoring empty cells). Rewarding monotonicity in this way has the effect of bunching high value tiles together in a corner. The evaluation function also rewards the *smoothness* of the board, where adjacent tiles are of the same, or near same, value. The number of empty cells and the highest value tile are also factored into the final evaluation. This evaluation function was duplicated exactly for our own experiments.

B. Expectimax

Robert Xiao implemented an Expectimax algorithm to play 2048 [6]. It is similar in style to Mini-Max in that it is a recursive, depth-limited tree search algorithm, and uses a similar evaluation function. The difference is that the player chooses not the maximum of the minimums from the opponent, but the maximum of the *expected* scores of the next states. The expected score of a state is the sum of the Expectimax value of each of the next states, multiplied by probability of that state occurring. This means that it makes decisions based on what is likely to happen.

Expectimax is a computationally expensive algorithm, so Robert Xiao has used numerous techniques to make the player more efficient; he uses a very efficient model representation and makes great use of caching where possible. He also prunes branches of the tree that drop below a probability threshold, so that the algorithm does not waste time calculation scores that will add very little to the overall score.

There is an interesting discussion of 2048 AI strategies on the Stack Overflow website, which includes both of these algorithms [7].

III. OUR STRATEGIES

A. Monte-Carlo Tree-Search

MCTS has three key properties that make it interesting.

- It can run for virtually any amount of time and still return a result; the longer it runs, the better the result.
- It does not require an evaluation function, as the roll-outs do this job. The random nature of the roll-outs ensure no moves are ruled out.
- It produces asymmetric trees, effectively pruning poor paths allowing for deeper searching of the paths with greater potential.

It is the asymmetry of the trees produced that allows MCTS to concentrate its effort on the potentially good moves, but we found that the trees produced whilst playing 2048 were symmetrical. Our assumption is that each move may or may not look good, depending on the random tile placement that comes after it. This makes it difficult for MCTS to calculate which moves to concentrate on.

B. Averaged Depth-Limited Search

ADLS approximates Expectimax by running multiple simulations. It does not try to calculate all possibilities, instead likely outcomes will appear more often in the simulations.

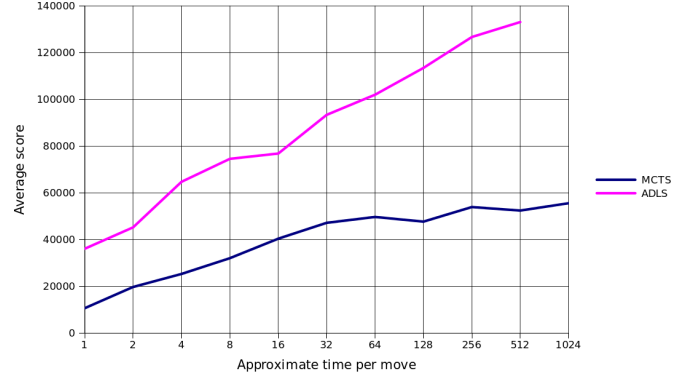


Figure 5: Comparison of average scores of MCTS vs ADLS.

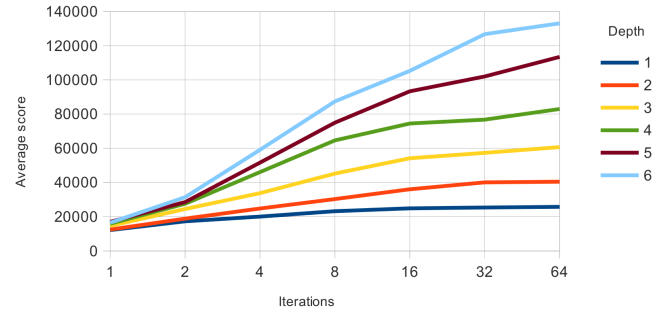


Figure 6: ADLS Results.

The more simulations that are run, the more likely it is that the outcomes will match the calculated Expectimax. How deep the algorithm searches, and how many iterations performed per search, impact the time taken to select a move. We ran many experiments to try to determine the best compromise between time taken and results.

IV. RESULTS

The chart in figure 5 shows that ADLS is far better than vanilla MCTS at playing 2048 for any given time frame. This is mainly down to the use of the evaluation function that rewards tidy boards.

Robert Xiao posted some results on StackOverflow for his Expectimax player. His code took about 150ms per move and achieved an average score of 157652, which is much better than the 120000 score for ADLS at that time frame, but the results are not directly comparable. The experiments were run on different machines, using different model representations and different evaluation functions. It is impossible to say at this stage *why* the Expectimax player is better than ADLS, only that it is. The maximum search depth will undoubtedly be a factor; the Expectimax player searches to a depth of eight, while ADLS can only search to a depth of six in a similar time frame. This may be alleviated to some extent by the use of a more efficient model and evaluation function, allowing ADLS to search deeper.

Extrapolating from the results shown in figure 6, we can assume that ADLS, given a depth limit of eight and thirty

two iterations, would meet, if not exceed, the Expectimax average score of 157652, but our implementation as-is would take about three seconds per move to search that deep—some twenty times slower than Xiao’s Expectimax.

A. Human Players

There is, as yet, no official leader board for human 2048 players, and although there are many videos on YouTube showing massive scores, their authenticity cannot be verified. With the code for the game available on GitHub [8], anyone could create their own version of the game and include cheats. One video shows someone achieving the 131072 tile [9]—but it is highly suspect, showing only the last four minutes of the game.

A more believable video shows a player achieving a 65536 tile [10], but even this video is suspicious. The full game is split over three videos, with the first two at one hour each. By the end of the second part, there is a 32768 tile on the board and a score of 466880. However, a large part of the game is missing, with only the last eight minutes shown in the final video where the 65536 tile is made.

V. FUTURE WORK

A. Evaluation Functions

We have conceived of several evaluation functions that may or may not get better results. The most promising is based on one-dimensional monotonicity, whereby we reward boards that are monotonic in an ‘S’ shape, rather than each row and column being monotonic. This seems to be the strategy favoured by human players. We have implemented this this evaluation function, and it does generate the correct board patterns, but there has been no time to run the required experiments.

B. Being brave

It seems obvious that a lot of luck is involved in getting the very high scores, and both Expectimax and ADLS aim for the maximum trade-off between reward and likelihood. This is fine for maximising the average score, but who wants to be average? If we want to create AI that gets 32K+ tiles, we may need to create an algorithm that takes riskier decisions rather than playing safe. A professional snooker player may find himself in a similar position; he could pot the easy pink and go on to take the frame, or he could go for the much harder black. If he misses the black, his opponent may be able to take the frame, but if he pots the black, he will go on to achieve a maximum 147 break.

When running the experiments for ADLS with a search depth limit of six, we noticed that doubling the number of iterations from 32 to 64 resulted in a small increase in the average score, but fewer 16384 tiles. See figure 7. This can be explained as a tightening of the probability curve.

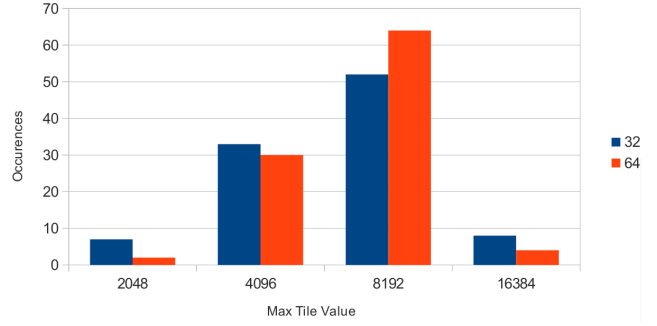


Figure 7: 64 iterations gives fewer lows, but also fewer highs.

C. Competition

This game is perfect for official competition, both for human and AI players. A web page could be set up that allows games to be played and recorded, controlled by either keyboard or API, with statistics taken. Leader boards could be made where the games can be replayed.

D. Adaptation for playing ‘Threes’

Threes is a similar sliding tile game, upon which 2048 is based [11]. There are some significant differences in the game play, so it would be interesting to know if ADLS would have similar success playing it.

VI. CONCLUSION

2048 is an intriguing and addictive puzzle game. The AI for this game is still in its infancy, and while much progress has already been made, there is still plenty of room for improvement. The game has a large element of luck involved, so a good AI must minimise the risk, but not so much as to rule out greatness.

ACKNOWLEDGMENT

The authors would like to thank Gabriele Cirulli [12] for creating this phenomenon, seemingly by accident. We would also like to thank Veewo Studio [13] and Asher Vollmer [11] for creating the games upon which 2048 is heavily based. We would also like to thank Robert Xiao [6] for providing us with a target.

REFERENCES

- [1] “2048,” <http://gabrielecirulli.github.io/2048>.
- [2] “Rush hour,” <http://www.thinkfun.com/rushhour>.
- [3] “15 puzzle,” http://en.wikipedia.org/wiki/15_puzzle.
- [4] “Maximum board position in 2048 game,” <http://math.stackexchange.com/questions/716469/maximum-board-position-in-2048-game>.
- [5] “Mini-max ai,” <http://ov3y.github.io/2048-AI>.
- [6] “Expectimax ai,” <https://github.com/nneonneo/2048-ai>.
- [7] “2048 ai discussion,” <http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048>.
- [8] “2048 source code,” <https://github.com/gabrielecirulli/2048>.
- [9] “Video of achieving the 131072 tile,” <https://www.youtube.com/watch?v=MDkZkweB5IM>.
- [10] “Video of achieving the 65536 tile,” <http://www.youtube.com/watch?v=lfwF25VKf94>.
- [11] “Threes,” <http://asherv.com/threes>.
- [12] “Gabriele cirulli,” <http://gabrielecirulli.com/>.
- [13] “1024,” <http://www.veewo.com/games/1024>.