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. It is similar in style to rush-hour and the 15 puzzle, but is a very different type of game. It is not a puzzle, in the traditional sense, as you cannot look at the board and calculate the optimal path to the goal. The stochastic nature of the game requires one to develop a *strategy* for maximising the score.

II. Previous Work

A. Mini-Max

The first published AI for 2048 appeared on the internet in March 2014 [2]. 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 seams 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, introduce an evaluation function based largely on *monotonicity* of the rows and columns of cells.

B. Expectimax

Robert Xiao implemented an Expectimax algorithm to play 2048 [3]. 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.

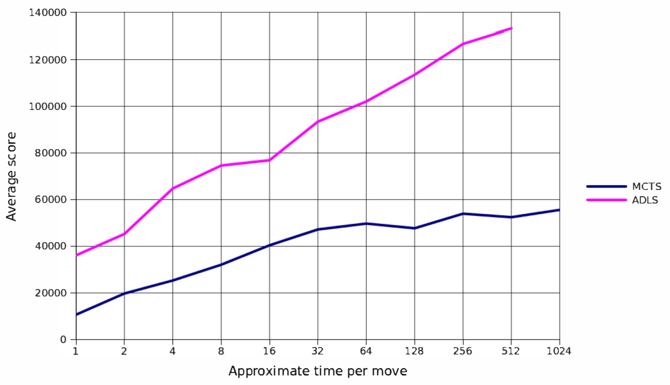

Figure 1: Comparison of average scores of MCTS vs ADLS.

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.

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. The more simulations that are run, the more likely it is that the outcomes will match the calculated Expectimax.

IV. Results

The chart in figure 1 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.

978-1-4799-3547-5/14/$31.00 ©2014 IEEE

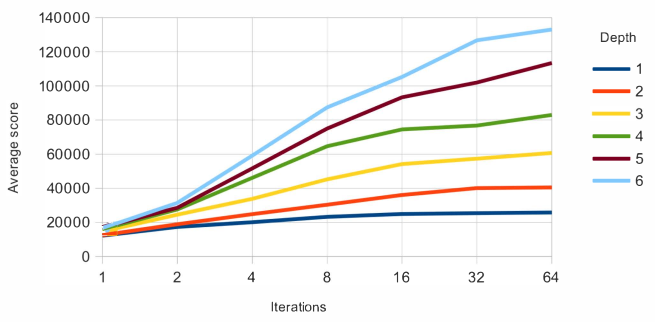

Figure 2: ADLS Results.

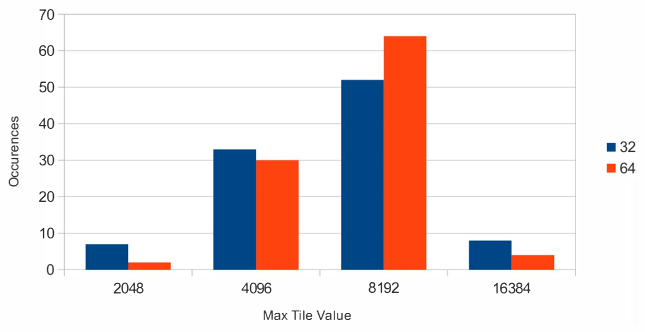

Figure 3: 64 iterations gives fewer lows, but also fewer highs.

Robert Xiao posted some results on StackOverflow for his Expectimax player [4]. 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. In that time frame ADLS could only search to a depth of six, but Xiao's algorithm can reach a depth of eight, thanks largely to his efficient use of transposition tables.

Extrapolating from the results shown in figure 2, 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.

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 3. This can be explained as a tightening of the probability curve.

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 [5]. 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 [6] for creating this phenomenon, seemingly by accident. We would also like to thank Veewo Studio [7] and Asher Vollmer [5] for creating the games upon which 2048 is heavily based. We would also like to thank Robert Xiao [3] for providing us with a target.

References

[1] "2048," http://gabrielecirulli.github.io/2048.
[2] "Mini-max ai," http://ov3y.github.io/2048-AI.
[3] "Expectimax ai," https://github.com/nneonneo/2048-ai.
[4] "2048 ai discussion," http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048.
[5] "Threes," http://asherv.com/threes.
[6] "Gabriele cirulli," http://gabrielecirulli.com/.
[7] "1024," http://www.veewo.com/games/1024.