Artificial intelligence in game 2048

Maj Gaberšček
mg5781@student.uni-lj.si
Faculty of Mathematics and Physics,
University of Ljubljana
Jadranska ulica 19
SI-1000 Ljubljana, Slovenia

ABSTRACT

I implemented a simple game of 2048 along with user interface in programming language Java. Along with single player option to play the game, I also implemented some algorithms, which try to solve the game. All algorithms have a different fundation, some base on positional play (position evaluator) and some are based on Monte Carlo methods (simulator and dynamic simulator). Overall, I would say that best algorithm is dynamic simulator, as it best compensates between move time and success.

KEYWORDS

2048, puzzle game, artificial intelligence, problem solving, Monte Carlo methods, algorithms

1 INTRODUCTION

I have played the game 2048 and been a fan of it ever since high school. The game is quite simple, however it requires smart planning in order to beat it. Since I haven't found many articles, that describe algorithms, which are successful at playing the game, I decided to implement some methods myself. The goal was to find different types of algorithms, that are all fairly good at playing the game. Of course, all the algorithms have to make a move reasonably fast.

2 ABOUT 2048

2048 is a single player video game, invented by an italian software developer Gabriele Cirulli as a weekend project and later published on `Github`. The goal of the game is to merge smaller numbers on the board in order to reach higher numbers (or more specifically, 2048). Some variantations of the game of course also allow infinite play, so that the game tries to reach higher numbers. The board is usually of size 4x4, but there are many different variantations as well (3x3, 5x5 or even 8x8). [2]

The game is not deterministic and it involves probability. Therefore it is important to understand, that theoretically, even if played perfectly, there exists a probability that the game will be lost. With a reduction to 3-SAT problem, it has been proved that the problem of 2048 is NP-complete [1].

3 ALGORITHMS

This section of the article describes all the algorithms and how they work.

3.1 Random moves algorithm

First algorithm is very simple. It is playing completely random moves, no matter the position. This algorithm is not very good at solving the game, however it serves as a base for *Simulator* and *Dynamic simulator* algorithms, which are described later.

3.2 Empty spaces algorithm

Another simple algorithm checks all the moves and plays the one, that results in most empty board. If more moves result in the same number of empty spaces, a certain move priority is specified. This algorithm is of course too simple to achieve anything. However, the its main goal was to analyse, how much better then simple random moves it can be.

3.3 Position evaluator algorithm

The position evaluating algorithm contains a parameter of depth. It uses depth first search and searches all the possible positions for the specified depth. Then it evaluates all the positions and plays the move, that results in best expected position evaluation. The problem with this is, that because a new number is spawned to a random position on a board after each move, the breaching factor is very big. So realistically, we can only use depth 2.

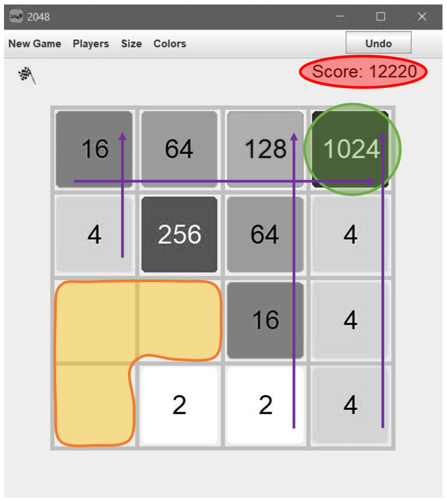

Figure 1: Visualization of position evaluation

Position evaluation explanation is visualized in Figure 1. First contributor towards evaluation is the score of the game (color red). By doing this, we force the game to merge bigger tiles as fast as possible. Second contributor towards evaluation is the most important one. We always check if the tile with the biggest number is in the top right corner (green color). Then we check first row and every column. If they are ordered (in the direction of the purple arrow), it further contributes towards position score. Number of empty spaces also affect the evaluation of the position.

3.4 Simulator and dynamic simulator

This algorithm is well known in Game theory as **Pure Monte Carlo game search**. Algorithm has input integer k. Then it looks for all possible moves in the given position. For each of this moves, it then plays the move and simulates rest of the game (until game over) with the *Random moves algorithm*. It does this k times for each possible move and returns a move, that averages highest game score across k games.

The only problem with Simulator is, that for bigger k, not only performance increases, but also move time. So we want as small k as possible, for which the algorithm still successfully plays. After some testing, I found out, that algorithm only has trouble right before building another biggest number. That is because right before building, for example 1024, there must be a 512, 256, 128, 64, 32, 16, 8, 4 and double 2 already on the board.

So, to optimize move time and success combination, I introduced another algorithm, which dynamically changes k, based on the game's score. If the score indicates, that we are about to reach a complex position (as described above), it increases k. Otherwise, we compromise k and rather focus on playing as fast as possible. k in regard to score is visualized in Figure 2.

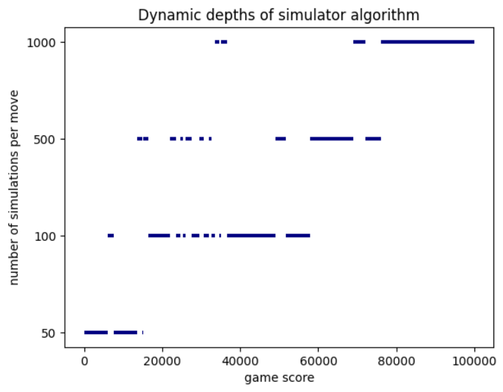

Figure 2: Depth in regard to score for dynamic simulator algorithm

4 EVALUATION OF ALGORITHMS

Of course the main evaluation of the algorithms has to be percentage of winning, which is shown in Figure 3. Best is Simulator with $k = 500$, but dynamic simulator is far faster and almost equally successful.

In the Figure 4, we can see distribution of highest number reached (before losing) of each algorithm. This provides more insight into Evaluator algorithm of depth 2. It seems to reach at least 1024 most of the times, but then fails to reach 2048.

5 CONCLUSIONS

Overall, I think that best algorithm is the Dynamic simulator. However, the position evaluator's playstyle very much imitates how humans plays the game. If parameters of this algorithm (for position evaluation) would be better tuned or they could dynamically change (in regard to score), winning chances could increase. This is one of the further work possibilities.

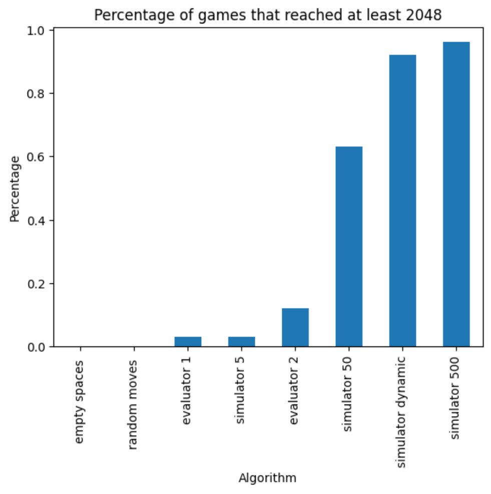

Figure 3: Percentage of times that algorithm reached 2048

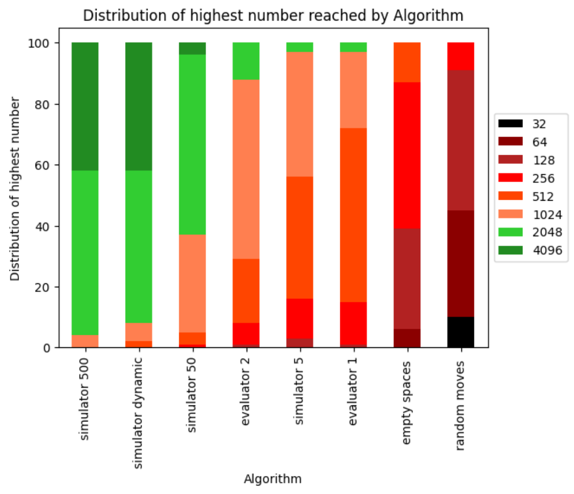

Figure 4: Distribution of highest reached tile when losing

REFERENCES

[1] Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. 2015. On the Complexity of Slide-and-Merge Games. *CoRR* abs/1501.03837 (2015). arXiv:1501.03837 http://arxiv.org/abs/1501.03837
[2] Wikipedia. 2022. 2048 (video game) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=2048%20(video%20game)&oldid=1083118206. [Online; dostopano 25.4.2022].