AI Course Project —— Optimization of 2048 Game

Ruitao Zong
ShanghaiTech University
2021533028
zongrt@shanghaitech.edu.cn
Yixuan Liu
ShanghaiTech University
2021533021
liuyx6@shanghaitech.edu.cn
Zikang Chen
ShanghaiTech University
2021533026
chenzk@shanghaitech.edu.cn
Yige Gao
ShanghaiTech University
2021533137
gaoyg@shanghaitech.edu.cn
Kehao Wang
ShanghaiTech University
2021533025
wangkh@shanghaitech.edu.cn

January 26, 2024

Abstract

2048 is a single-player sliding tile puzzle video game written by Italian web developer Gabriele Cirulli and published on GitHub. The objective of the game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048; however, one can continue to play the game after reaching the goal, creating tiles with larger numbers. In this project, on the basis of realizing the game interface and basic functions of 2048, our team also used different AI search algorithms, evaluation functions and movement strategies to explore the possibility of optimizing the game and making the game more challenging and interesting.

1 Introduction to basic rules

2048 is played on a plain 4×4 grid, with numbered tiles that slide when a player moves them using the four arrow keys. The game begins with two tiles already in the grid, having a value of either 2 or 4, and another such tile appears in a random empty space after each turn. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move. Higher-scoring tiles emit a soft glow; the largest possible tile is 131,072.

If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine. If all four spaces in a row or column are filled with tiles of the same value, a move parallel to that row/column will combine the first two and last two. A scoreboard on the upper-right keeps track of the user's score. The user's score starts at zero, and is increased whenever two tiles combine, by the value of the new tile.

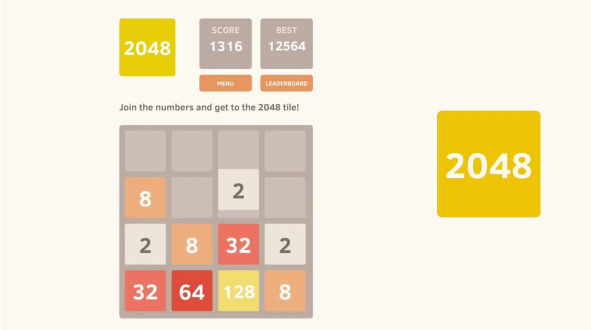

Figure 1: An example of 2048 game

The game is won when a tile with a value of 2048 appears on the board. Players can continue beyond that to reach higher scores. When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends.

2 Problem Statement

The current 2048 game has some pass-through skills that allow players to easily obtain high scores. For example, the player can always place the largest number on either side of the top, bottom, left, or right, and only move in the other three directions.

In this regard, our team's project hopes to upgrade and optimize game strategies through the exploration of AI algorithms, and increase the difficulty and challenge of the game.

3 Basic Methodology

3.1 Search Algorithms and Corresponding Strategies

Greedy Search. Greedy just consider the current state and next possible states and takes the most favourable step for the current state.

Minimax Search. The player will assume that every new tile will occur in the worse place (actually tiles occur randomly). Player's actions are based on the above assumption.

Breadth-first Search. BFS is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

Depth-first Search. DFS is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. In this case, we will simulate the next four steps and choose the first step which results in the best evaluation.

Expectimax Search. For four actions of each state, play the game completely randomly until win or lose for many times to get the expected score of each action. After that, choose the action with the maximum score. We use calculate() to calculate the score after 100 random operations. Then we choose the movement which results to the state has the highest score by calculate().

3.2 Evaluation Function

We evaluate the current state based on our own experience playing 2048, and the criteria are as follows:

1) We hope to have more empty spaces in order to avoid losing the game.

2) Minimize the gap between adjacent tiles.

3) Place the tile with the largest number in the corner to reduce the effect of moves to its position.

4) Make the values if tiles between the same row or column as monotonous as possible to make it easier to combines the tiles. For example, 248 will receive a better evaluation than 284.

4 Experiment

4.1 Basic Work

In the early stages of the project, our team completed the reproduction of the basic functions of the 2048 game, allowing players to play the game and pass the levels normally.

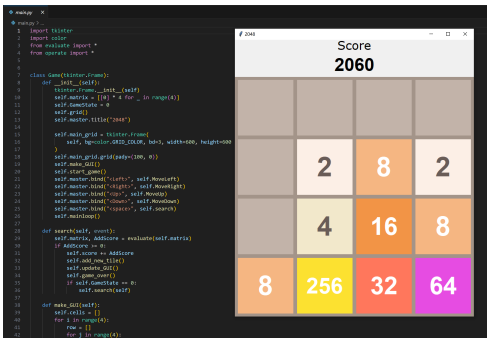

Figure 2: Realizing basic functions of 2048 game

4.2 Results of Algorithms

Table 1: Results of Minimax Search with 100 trials

	step size=1	step size=2	step size=3	step size=4
number of trials achieving 128	15	1	0	0
number of trials achieving 256	59	8	4	2
number of trials achieving 512	25	25	22	10
number of trials achieving 1024	1	53	45	32
number of trials achieving 2048	0	13	29	56
number of trials achieving 4096	0	0	0	0
Average score	3873.04	12209.32	13743.92	16667.44
Success rate	0%	13%	29%	56%

Table 2: Results of other methods with 100 trials

	Expectimax Search	Depth First Search	Greedy Search
number of trials achieving 128	0	3	1
number of trials achieving 256	1	21	8
number of trials achieving 512	4	63	18
number of trials achieving 1024	59	13	44
number of trials achieving 2048	36	0	28
number of trials achieving 4096	0	0	1
Average score	15925.64	5831.52	15685.84
Success rate	36%	0%	29%

Across all run trials during our experiments on Minimax Search Algorithm, the probability of 4096 occurring was about 3% when the step size was 3, and about 10% when the step size was 4.

Figure 3: An example of achieving 4096

4.3 Analysis

For Greedy Search: Greedy can only find local optimal solutions. But not very time-consuming. However, a temporary compromise can lead to a better solution. But because our valuation function tries to simulate real human operations as much as possible, Greedy Search surprisingly performs well.

For Minimax Search: Performs best in the test, but takes a lot of time.

For Breadth-first Search: Since the target is difficult to handle, we can hardly decide when to stop. Thus, BFS is not suitable for this problem.

For Depth-first Search: Restricted by time complexity, we can only use DFS for a small depth and find the action with maximum value. However, due to the uncertainty of square generation, DFS is unstable. It may be far from real situation when the depth gets bigger. Moreover, the small depth is very likely to trap us in local optima rather than global optima. Thus, DFS does not perform very well.

For Expectimax Search: In our previous experiment, we find that after each step, generating a new square randomly will cause significant interference to our estimation and the adverse effect in accuracy of algorithm. To solve the problem, this algorithm uses the experience of Markov Decision Process. When we simulate playing the game, we introduce a discount. Therefore, we prefer score with few steps than that with many steps in order to reduce the influence of the randomness of square generation. Also, we play the game for every state many times. This is like approximating probability using sampling methods. However, for each state we need to play the game too many times in four directions. It not only causes the program runs slowly, but also limits the number of games played for each state.

4.4 Improvements on Algorithms

For Greedy Search: The following algorithms can be a better solution of Greedy Search. Sacrificing some time to get a more stable solution.

For Minimax Search: As time is positively correlated with step size, we can distribute step size according to the complexity of the current state. As we are doing now, before 32 occurs, we set step size to 3. Before 512 occurs, we set step size to 4. And when 1024 occurs, the step size is set to 5. We

believe there would be a optimum distribution of step size according to the complexity of the current state that balances the result and time consumption.

For Depth-first Search: We can apply some prunes to save the time which will increase the depth of searching.

For Expectimax Search: We can use the evaluation function for some state rather than completely playing the game until win or lose. It can reduce the time complexity, allowing us to play more games and increase the accuracy at each state.

5 Conclusion and Future Directions

5.1 Conclusion

In this project, we presented a complete process of optimizing 2048 game. We first completed basic functions of the game. Then we used different algorithms and their corresponding evaluation functions and movement strategies to find better game strategies. Finally, we analyzed the advantages and disadvantages of each algorithm based on the running results of the algorithm, and provided corresponding improvement methods.

5.2 Future Directions

The work of this project also provides some ideas for the future development of 2048 Game.

Change the way the game is played. Expand the grid size of the 2048 game, or change the shape of the overall interface (such as rectangle, triangle) to create greater challenges and changes.

Man-machine game. The machine acts as a player in an information-equivalent game with human players, making the game more challenging and interesting.

6 External Resources

As required, at the end of our report, we will list all the external resources (e.g., code, library, tools) that we used.

- `https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048`
- `https://ovolve.github.io/2048-AI/`
- `https://www.youtube.com/watch?v=96ab_dK6JM0`
- `https://www.cnblogs.com/mumuxinfei/p/4305981.html`