Heuristic Functions in Solving 2048 Game

Bao Tran NGO, Student, University of Information Technology

Abstract

This research provides a better understanding of how effiecient heuristic functions contribute to a good game playing. In this research, we used a popular game called 2048 to describe and implement heuristic functions. The biggest aim of the AI using these functions is firstly winning the game, and then achieving the highest score as possible. We mainly use best-first algorithm with some effective heuristic functions. These functions are proved to enhance efficiency in game results to some extent. In conclusion, this research shows that with good heuristic functions, AI can be a powerful performer in game playing.

1 Introduction

Artificial Intelligence has been a fundamental part of games for a long period of time until today. Implementing effective AI remains the most interesting challenge for game creators as well as players from time to time. In this research, we attempt to build a solver to 2048 game using two different methods: a naive, non hard-coded method called priority-based method and best-first method using efficient heuristic functions.

1.1 What is a 2048 game?

2048 is a single-player sliding block puzzle game designed by Italian web developer Gabriele Cirulli. 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.

2048 is played on a gray 4×4 grid, with numbered tiles. Players can slide in four directions including down, up, left, right. This will cause all tiles to move in the chosen direction to the farthest edge. If two tiles have the same value and are slid to the same direction, they merge and form a new tile whose value are equal to the sum of two original tiles' value. A tile can only merge once in a slide. After every slide, a new tile will appear randomly on one of the empty grid with value of two at 90% chance or four at 10% chance. Players can

continue sliding until reaching the 2048 tile (winning state) or until there is no empty place for new tile and no possibility to move or merge (losing state).

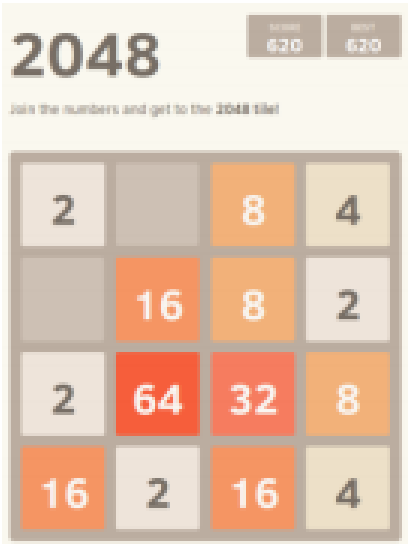

Figure 1: A 2048 Game.

1.2 The problem

In order to achieve the 2048 tile, players need to merge 1024 tile with value 2, which means players need to move at most 1024 times on a limited 4x4 board. This requires optimum and accurate movements, or else players will end up having a full board with no possibility to merge or move.

The 4x4 limited board as well as the randomness of new tiles make it even more difficult to evaluate all the game states. There are around 10^{16} of states possible before getting a 2048 tile, and there are even more if players continue the game after winning. Therefore, some techniques such as Q-learning are impossible.

2 Methods

2.1 Some strategies for 2048 game

In order to have better performance in 2048 game, there are some strategies that can be used to accomplish higher score and tiles. Some of these are used in our heuristic functions.

First, keep the highest tile in the corner. This is the most fundamental strategy, which allows players to have more space to merge and create new tiles.

The second one is the monotonicity of the board. In a monotonic board, the tiles are aligned in either increasing or decreasing order in terms of their values. This ensures the tiles can be merged in an easier way.

Figure 2: A monotonic board.

Beside monotonicity, a snake-like patterned board can also improve the board's quality, especially when players have tiles with big values. This optimizes the space needed and make the merging easier.

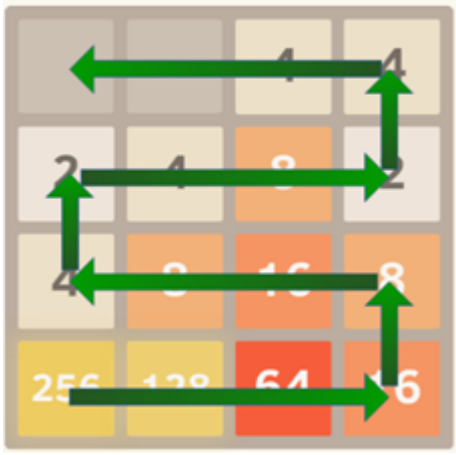

Figure 3: A snake-like patterned board.

There are some other criteria to evaluate board's quality such as: empty tiles, the edges contained the highest tiles... but we mainly focus on those mentioned above to implement our heuristic functions.

2.2 First Approach: Priority-Based, An Easy and non Hard-coded Function

In this approach, we simply move tiles in one-way direction. As a results, the highest numbered tiles will gather on one edge, thus give us more opportunity to achieve larger tiles and have more space to move around.

To do so, we set priority to each direction. The direction which has higher priority is tend to be chosen unless there is no change to the board when we slide in that direction. In our implementation, we choose the priority in decreasing order respectively: down, left, right, up. That means our agent will tend to slide down and left, place the largest numbered tile in the bottom left and the higest numbered tiles on the bottom edges.

Results

This approach seems ideal but unfortunately it did not perform well. In 100 runs, it only achieved the value 512 as the highest value, therefore, could not succeed in reaching the winning state.

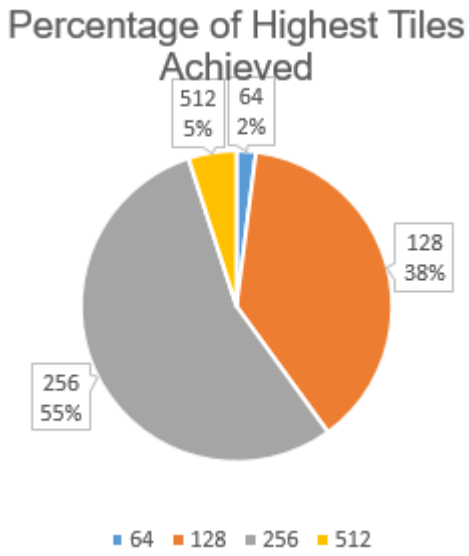

Figure 4: Priority-Based's percentage highest tile achieved in 100 episodes

Figure 4 shows that over 100 runs, our agent successfully achieved the 256 tile 55 times, the 128 tile 38 times, the 512 tile 5 times and the 64 tile twice.

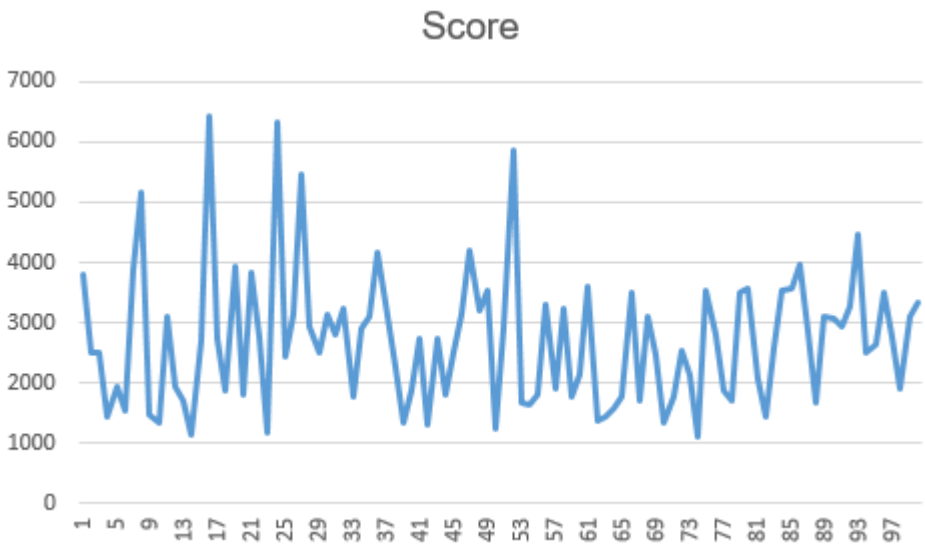

Figure 5: Priority-Based score achieved in 100 episodes

Figure 5 demonstrates the score of each run. Our agent achieved the highest score of over 6000 and the lowest at about 1000.

	Priority-Based
Highest score	6416
Lowest score	1096
Average score	2706
Highest number value	512
Lowest number value	64
Most achieved value	256

Figure 6: Priority-Based's overall results

2.3 Second Approach: Best-first Algorithm with Heuristics

As the results shown above, priority-based method is not efficient as it gave pretty poor performance. Hence, we tried to approach in a different way: choose next move as the best move, also called as best-first algorithm. We used three

different heuristic to calculate the evaluation score for the move. If the board did not change after executing the move, the score equals zero.

Heuristic 1: Score Distribution

For each evaluation, we build a score distribution table based on the maximum tile's location. The score distribution table has a snake-like pattern. The corner which is closest to the maximum tile will have the largest weight, which equals 16. Then the weight value decreases in a snake-like pattern. This encourages our agent to move to the direction which returns maximum score; keep the highest value tiles at the highest score distribution; hence improve the board quality.

Furthermore, we also used a dynamic score distribution table, which means it will change if the max tile's location change. We initialize the table with the largest weight at the bottom left and during the execution, it can flexibly alter, depending on where the max tile is placed.

The score value is evaluated by the sum of every tile's number multiplies by its respective score distribution. That means a tile will have higher score if it is placed in the higher priority rows or columns than the one in the lower priority rows or columns. Therefore, the higher numbered tiles will likely to be in the same direction and toward the edges.

Heuristic 2: Corner Check

This heuristic is simple yet pretty important in evaluating the board. It gives bonus if the maximum tile is placed at one out of four corners with the bonus value equal largest number multiply by 16. If the maximum tile is not at the corner, there is no bonus, which means the bonus equals zero. The bonus contributes pretty much to the score; thus, our agent tends to keep the largest tile in the corner.

Heuristic 3: Monotonicity Heuristic

This heuristic will check each row and column for its monotonicity. If a row/column is in increasing or decreasing order, a bonus is given with the value equals to the smallest value of the row/column multiplies by the row/column priority index. The row/column priority index is ranged from 1 to 4 according to the score priority table, which means if that row/column contains the highest numbered tile, its priority will be the maximum and so on. This ensures the monotonicity value will likely not outscore other heuristic values and also represents that the row or column that nearer to the maximum numbered tiles will have more weights than the one that is farther away.

Score Evaluation

The score evaluation is calculated by the equation below:

*scoreEvaluation = scoreDistribution * weight + monotonicityScore + cornerScore*

The weight that we multiplied to the scoreDistribution guaranteed that the score add up to a right amount of score, which means it will not be too low to be outscored and too high to outscore other values. Unfortunately, there is no certain way to calculate this weights but to try different values and find the best one. Out of all the values we have tried, we found that the *weight = 2 +*

maxTile/512 had the best performance. There is no ensurance that this value is the best one, there may be some others that is more ideal.

Results

After 100 runs, we received results as figures below.

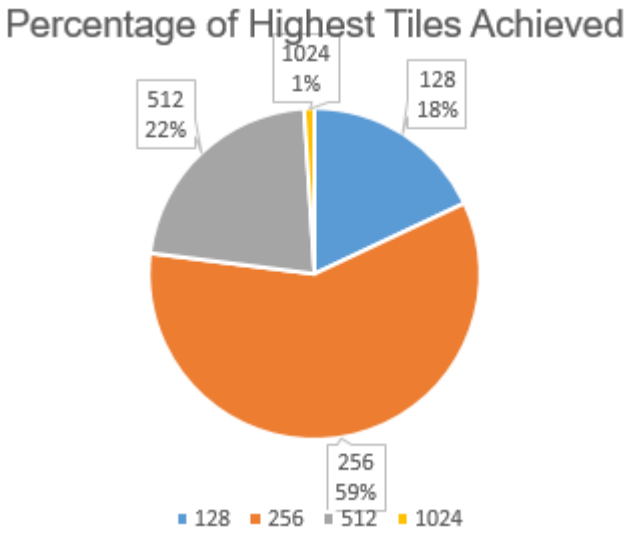

Figure 7: Best-first's percentage highest tile achieved in 100 episodes

Figure 7 shows that over 100 runs, our agent successfully achieved the 256 tile 59 times, 512 tile 22 times, 128 tile 18 times and 1024 tile only once.

Figure 8 demonstrates the score of each run. Our agent achieved the highest score at over 10000 and the lowest at around 1000.

Overall, best-first with heuristics gave better results compared to priority-based method. Our agent could achieve 1024 value as the highest tile in some rare cases. The highest score and average score are also higher than ones of priority-based method. To some extent, we can state that best-first method is more efficient than priority-based method.

3 Conclusion

In conclusion, best-first method with some efficient heuristic functions are proved to perform well to some extent. Our agent is likely to form a good-quality board in terms of capability to move or merge, snake-like pattern and monotonicity.

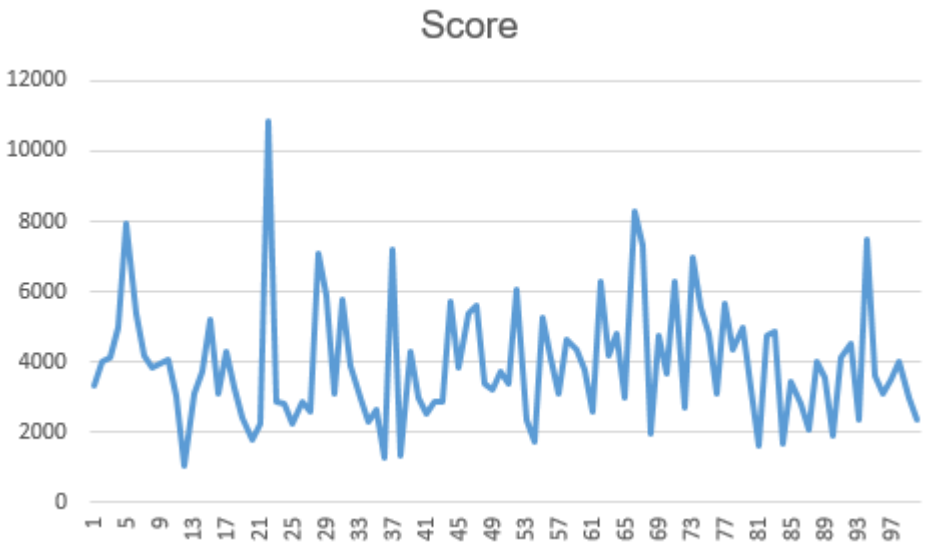

Figure 8: Best-first score achieved in 100 episodes

	Best-first with Heuristics
Highest score	10880
Lowest score	1008
Average score	3942
Highest number value	1024
Lowest number value	128
Most achieved value	256

Figure 9: Best-first's overall results

Although it could not win the game (reach 2048 numbered tile), its highest results is really close to the winning state.

The heuristic functions have a significant contribution to the effectiveness of the method. However, finding good heuristics is not an easy job. As a result, our agent did not perform as well as we expected at the first place.

4 Future Work

Both methods are pretty ineffective in terms of score and highest tile achieving as there is no method that can actually win the game. We believe that there is no best solution, but only better ones. Therefore, there is still room for improvement on the heuristics.

Beside that, as our agent is only able to see forward one move, sometimes it will greedily choose the best move at that time without knowing that other moves can produce a better results in some more finite moves (but not the next move). Some approaches using minimax or expectimax with depth can simply solve this problem and give more fruitful performance in game playing.

5 Acknowledgment

We would like to thank Dev Tutorials on YouTube to build a full functional 2048 game and leave it as an open source. We also appreciate those contributors in this stackoverflow thread to provide us a better understanding about the game and give us ideas about the solution to the game playing.