USING ARTIFICIAL INTELLIGENCE TO CREATE A SOFTWARE AGENT FOR SOLVING THE 2048 GAME

Mikloš Pot[1], Milovan Milivojević[2], Srđan Obradović[3]
[1]Visoka Tehnička Škola Strukovnih Studija, Subotica, pmiki@vts.su.ac.rs
[2]Akademija Zapadna Srbija, Odsek Užice, milovan.milivojevic@vpts.edu.rs
[3]Akademija Zapadna Srbija, Odsek Užice, srdjan.obradovic@vpts.edu.rs

Abstract *– The 2048 game was invented in 2014. It is characterized by the fact that the moves of the opponent player are almost completely random. The aim of this paper is to generate a software agent and to create a strategy that will successfully complete the game. Because of the huge number of combinations and possible steps, it is impossible to evaluate all the states the game can go through, and that is the reason to create an algorithm that will shorten the time needed to make a quality decision for the following move.*
The minimax and expectimax techniques were used along with alpha-beta pruning that limited the depth of the search space while not influencing the quality of the move. Heuristic functions were also used that guided the agent towards the optimal solution. The achieved results show that the implemented strategy gives much better results than an average human player.

Keywords: *game theory, minimax algorithm, expectimax algorihm, alpha-beta pruning*

1. INTRODUCTION

In artificial intelligence game theory studies mathematical modeling of the strategic interaction between agents who make racional decisions.
Earlier only games with zero sum were considered where players could make utilities only on the expense of their opponents. In the 21st century game theory refers to the scientific discipline which is occupied by making racional decisions in human, animal and computer world.
The 2048 game is a logical one-person puzzle game developed by Gabriele Cirulli in 2014. The player moves his tiles on the 4x4 board, and tiles have values that are powers of number 2. Tiles are moved in *Left*, *Right*, *Up* and *Down* directions using the arrow keys on the keyboard. The movement refers to all tiles on the board. When two tiles have the same value and are moved in the same direction, they will be merged and the new tile will have the value that is equal to the sum of the tiles' values. The newly created tile can not be merged with the neighbouring tile with the same value in the same move. After each move of the human player, on one of the empty places of the board a new tile will appear. In 90% of the cases the new tile will have value 2, and in 10% of the cases the new tile will have value 4. The goal of the game is to make one tile to 2048, but the game can last even longer. than that. The game ends when all 16 board tiles are populated without playing a legal move for the human player. Figure 1 shows a board when a game is won.
The goal of this paper is to create an agent who will automatically play the 2048 game and will maximize the score, and to play each move in a reasonable amount of time. Points are earned by merging the tiles with the same value; the resulting tile will have the value that equals to the sum of the merged tiles. It can be calculated that achieving a total score of 20000 corresponds to making a tile with value 2048.

2. PROBLEM DEFINITION

The intelligent agent have to take into consideration all 16 board tiles (empty board places are assumed to have value 0) as inputs, and to determine the move that will maximize the possibility of winning the game. Since the board consists of only 16 tiles, the game could look easy, but the space limitation is what make this game hard.

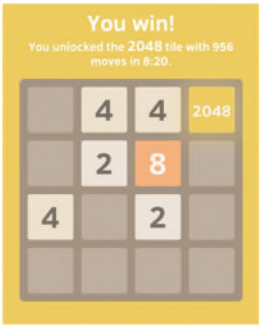


Figure 1. Possible look of the won game

By simple calculation it can be concluded that a tile with value 2048 is made by merging 1024 tiles with value 2, what again means that 1024 moves are needed to do on board with only 16 tiles. So the agent is required to play the whole game without making an error, what make generating the agent even harder.
The search space is extremely huge, there are around 10^{16} possible states prior to generating the tile with value 2048, and this number is even bigger if we continue the game after winning, what is also a possibility. To illustrate the complexity let's examine Figure 2. For this board state there are 3 possible moves: *Up*, *Left* and *Right*. After those moves the board will have 10, 11, and again 11 empty tiles, respectively. After that the computer randomly places a tile with either value 2, or value 4 onto one of the empty board places. It can be seen that after only one move of the human player ond one move of the computer the possible number of states is 64 (1x20+2x22). After one more move of both players this number increases to 3600, assuming that the branch factor is 3 and if 6 tiles are populated on the board. The mentioned assumptions are realistic because the branch factor can vary between 1 and 4 (branch factor 0 in any moment would mean that the game is finished and lost), and depending on the phase of the game the board is populated between 2 and 16 tiles. As we see, the growth of the possible states is exponential, and is multiplied by 60 in each move. It again means that it is impossible to construct the complete search tree because the exponential growth does not allow exhaustive search because of the time and the space limitation of the computer. So for finding a quality move we will need a good heuristic function that will guide us towars the solution. That's why the correct strategy is of crutial importance.

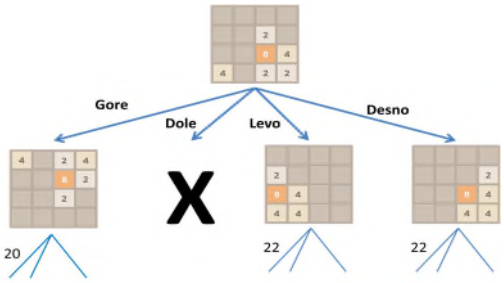


Figure 2. Illustration of the complexity of the 2048 game

3. STATEGY AND HEURISTIC FUNCTION

In this section we'll concentrate on the heuristic function that will be used in the algorithm. As it is known, the heuristic function is a value that assigns diferent values to different alternatives in heuristic search algorithms. This value will be greater if the alternative is "better". But the question that arises is how to determine which alternative is better? The heuristic function will be constructed using the observations about the 2048 game: to get a tile with value 2048, 2 tiles of 1024 are needed, to get a tile with value 1024, 2 tiles of 512 are needed, etc. So it would be desirable to transform the board to have the look that is presented in Figure 3.

3.1. Empty tiles

It is obvious that it is desirable to have as many empty board tiles as possible, so moving the tiles could be easier. Then there are more possibilities for merging, and creating tiles with higher values.

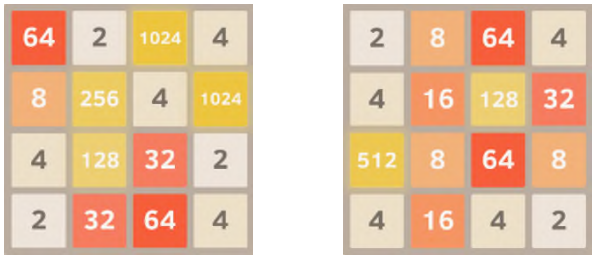
Figure 3. Desirable layout of board tiles for winning the game

3.2. Smooth board with moderate differences

The goal of the game is to generate a tile with value 2048, and for getting a tile with a higher value, 2 tiles with lower values are needed; to merge the tiles, they have to be neighbours. When values of the neighbouring tile are similar, it is said that the board is smooth. In this case smooth transitions are determined by calculating the difference between neighbouring tiles. The smaller the differences, the smoother the board. Figure 4(a) shows a board that is not smooth. Both 1024 tiles are neighbours with tiles 2 and 4. On Figure 4(a) there are no more legal moves, and the game ends.

3.3. Board with high values

The consequence of the smooth board is that high values are placed near each other, and empty tiles are left so they could be merged into higher values. But smooth board is not enough on its own. Figure 4(b) shows a smooth board where no more legal moves are available.



(a) (b)

Figure 4. (a) Board with sharp transitions, (b) Smooth board

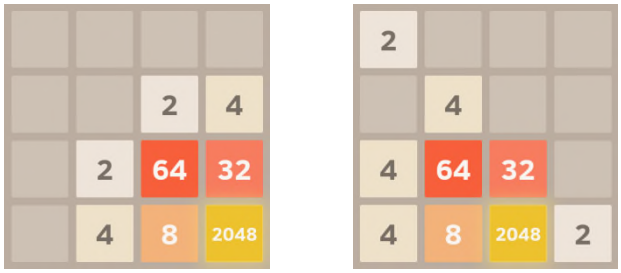
In Figure 4(b) high value tiles are positioned in the middle of the board. To get a board similar to the board in Figure 3, it is needed to keep high values on the same edge of the board so they could be merged to even higher values. One other observation showed that the best place for storing the highest board value is the corner of the board. It is logical because the highest board value has the least possibility to be merged with another tile (the corner tile has only2 neighbours).

3.4. Forced moves

In some cases the board is completely in accordance with the board from section 3.1., 3.2. and 3.3., but the player is forced to play a move that will spoil the desirale bord characteristics. such an example is showed in Figure 5(a). The game is won (a 2048 tile exists on the board) and continued, the board is with mild transitions, but a move to be played will move the 2048 tile from the corner (possibilities are *Left* or *Up*) and decrease the possibility of merging other tiles what can quite quickly lead to the end of the game. Figure 5(b) shows the state of the board after 2 moves where the 2048 tile has neighbours 2 and 8.

3.5. Heuristic function

Now observations from the previous sections will be formally presented using a heuristic function *H*. If *E* denotes the *number of empty tiles*, if *D* denotes the *sum of all differences of the neighbouring tiles*, and *P* denotes the sum of *distances to the closest border*, then the heuristic value can be represented using the formula



(a) (b)

Figure 5. (a) Board with desirable characteristics, (b) board with sharp transitions

$$H = A \times E - B \times D - C \times P \quad (1)$$

where *A*, *B* and *C* are suitably selected constants. During the implementation the following values were used: *A=4096*, *B=10* and *C=10*. The choice of constant *A* is arbitrary, and it has to be big enough to dominate when evaluating the heuristic function.

4. SEARCH TREE, MINIMAX AND EXPECTIMAX

Search tree is a basic method for solving problems in artificial intelligence. Minimax and expectimax are algorithms for determining the best move in two player games. Because of that reason both algorithms are called *decision rules*. There are other edcision rules, but they will not be used in this paper.

4.1. Search tree

Search tree is a basic method for solving problems in artificial intelligence. This method examines possible future states and determines which has to highest probability of winning the game. In our implementation recursive depth forst search was used with different depths to search all states. A value was assigned to each path, and the path with highest value was selected. The next step of the agent is the first step of the best path. Tests showed that this method solves the game with very low percentige of success, this is showed in Table 1.

Table 1: Success of the depth first search algorithm with different depths

Depth	1	2	3	4	5
Highest result	5216	16132	23676	35268	32112
Average result	1826	7319	10934	18092	16976
Percentage of games won	0%	0%	2.9%	28%	25%

4.2. Minimax algorithm

The minimax algorithm is a decision rule for minimizing the posible loss for the worst case scenario, or in other words for minimizing the maximum loss. This algorithm is used in two player games with zero sum. The value of the current state is calculated by traversing the tree to the leaf nodes, but because of high complexity and exponential growth, in many cases it is imposible to reach the leaf nodes. That is the reason why the minimax algorithm is often implemented only with limited depth. Since it is not possible to exactly determine the value for each leaf node, the value will be estimated using a heuristic function. The original (naive) minimax algorithm requires all nodes of the search tree to be expanded what make this method very complex. That is why *alpha-beta pruning* is used. Alpha-beta pruning gives the same result as the basic minimax, but it does not have to examine the whole search tree.

4.3. Expectimax algorithm

In the minimax algorithm the next move is chosen based on the maximum value gained (minimal loss). It means that only the minimal or maximal value of the ancestor nodes are needed. In expectimax algorithm when evaluating the opponents' nodes, all possible moves are taken into consideration and they are weighted with their

probability of occuring. In other words, the *expected values* of all possibilities are calculated. For most of the cases the probability of each node is the same. In our case (the 2048 game) the probabilities are known . The choice between the empty tiles is uniform, the probability of appereance of number 2 is 90%, while the probability of appearence of number 4 is only 10%.

5. RESULTS

While examining the algorithm 4 cases were tested: minimax algorithm with depths 4 and 8, and expectimax algorithm with depths 2 and 3. Depth to which the search tree is examined and execution time are inversely proportional. Tables 2-5 shows the percentages of the highest tiles the algorithm achieved. Let's remember that the game is won if the highest tile has value 2948 or higher.

Table 2: Minimax algorithm, search tree depth limit is 4

Highest value	Percentage of games won
256	1%
512	17%
1024	45%
2048	33%
4096	4%

Table 3: Expectimax algorithm, search tree depth limit is 2

Highest value	Percentage of games won
256	5%
512	8%
1024	31%
2048	53%
4096	3%

Table 4: Minimax algorithm, search tree depth limit is 8

Highest value	Percentage of games won
512	5%
1024	25%
2048	55%
4096	15%

Table 5: Expectimax algorithm, search tree depth limit is 2

Highest value	Percentage of games won
1024	20%
2048	40%
4096	40%

6. CONCLUSION

Both minimax and expectimax algorithms showed good performances in solving the 2048 game. As expected, the percentage of games won increases when the depth of the search tree increases. Minimax algorithm with depth 8 has a success rate of 70%, while expectimax with depth 3 has an even higher success rate of 80%, and a probability of 40% to reach tile 4096. By improving the heuristic function even better results could be possible.

REFERENCES

[1] Rodgers, Levine: *An investigation into 2048 AI strategies*, 2014 IEEE Conference on Computational Intelligence and Games.
[2] Russel, Norwig: *Artificial Intelligence: A modern Approach*, Pearson Education, Inc., 2003.
[3] https://gabrielecirulli.com
[4] Ahmad Zaky: *Minimax and Expectimax Algorithm to Solve 2048*, 2014.