Backtracking Algorithm for Solving Star Battle Puzzle

Ricky Fernando - 13518062
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518062@std.stei.itb.ac.id

***Abstract*—The Star Battle Puzzle is an interesting puzzle where the main purpose of the puzzle is to put a certain number of star in NxN grid puzzle in every row, column, dan region. This puzzle is similar to one of the classic problems on backtracking which is N-Queens. So the author of this paper tried to solve this puzzle with backtracking approach.**

Keywords—Star Battle Puzzle; Backtracking;

I. INTRODUCTION

Puzzle is a game for people of all ages. Puzzle games are not just games but a way of culture that has been passed down through generations. From the jigsaw puzzle that we all know until some puzzle that aren't many people know about it like Star Battle Puzzle. Star Battle Puzzle isn't a famous puzzle because it is quite simple and some people say it's not challenging enough.

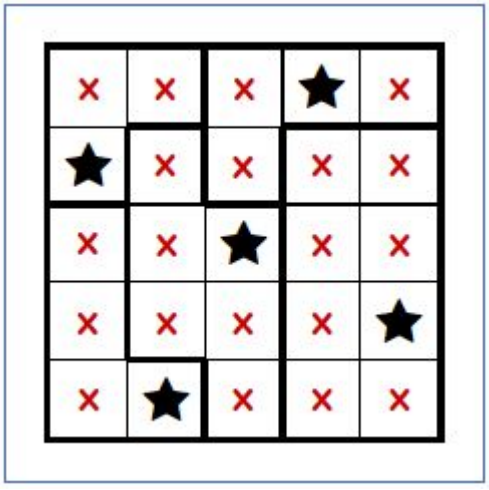

Figure 1 : Example of Star Puzzle

source: https://www.puzzle-star-battle.com/

Star Battle Puzzle is a logical grid puzzle where the purpose is to put a star (1 ✷ Star Battle Puzzle) in every row, column and Block. For a 1 ✷ puzzle with small grid, it is easy to solve. What if the size became larger? For example 50x50? Or not 1 ✷ but 2 ✷, or 3 ✷, or even 10 ✷? This will be a hard puzzle to solve.

Because recently the author likes to play this puzzle and found that problem, the author is eager to solve this puzzle through algorithms. That's why the author chose this as a topic for this paper.

II. BASE THEORY

A. *Recursion*

Recursion is a process which a function calls itself directly or indirectly. The function which does recursion is called recursion function. Recursion can help solve many problems. For example, Tower of Hanoi, Dynamic programming, and DFS.

Recursion function consists of the base and recursion part. The base is where the function is supposed to call itself, and the recursion part is where the function calls itself.

The advantage of this process is that it can be used to solve problems that can't be solved by a regular loop. The drawback of this process is it consumes a lot of memory because it calls itself and stack it on the memory stack. This process may cause stack overflow.

B. *Backtracking Algorithm*

Backtracking is an algorithm introduced in 1950 by D. H. Lehmer. Backtracking is an algorithm to solve a problem recursively by exploring every possible solution one by one. Backtracking itself is an improvement from exhaustive search. While exploring the solution, backtrack will remove those solutions that fail to satisfy a certain condition at any point at that time.

Backtracking can be categorized as 3 types, which is:

1) Decision Problem – Used to search for a feasible solution.

2) Optimization Problem – Used to search for the best solution.

3) Enumeration Problem – Used to find all feasible solutions.

The general properties of backtracking is the solution of the problem, generating function and bounding function. Solution of the problem is defined as a vector with n-tuples. Generating function (defined as T(k)) is used to generate a value for x_k which is an element of Solution of the problem. Te bounding function (defined as $B(x_1, x_2,..., x_k)$) will return true if $B(x_1, x_2,..., x_k)$ leads to a solution and false otherwise. If the result of the bounding function is true then backtrack will generate x_{k+1} and continue the search, otherwise $B(x_1, x_2,..., x_k)$ will be removed.

Every possible solution from the problem is called solution space. Solution space is represented with a tree structure and called state space tree. Every node of the tree representing a state and every edge representing value of x_i.Path from root to leaves represent the possible solution and create a solution space.

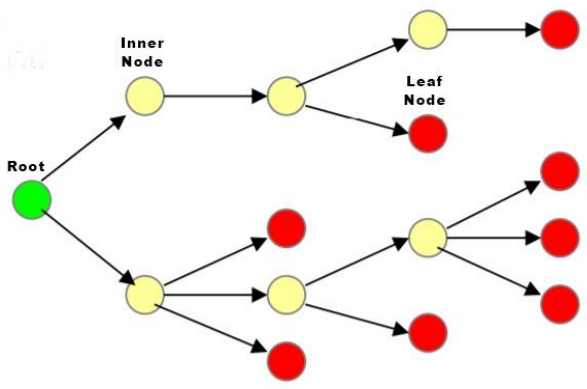

Figure 2 - Representation of state space tree
Source:
http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Runut-balik-(2018).pdf

The Principle of Finding a Solution with Backtracking method :

- Solution is searched to build a path from root to leaf
- Live node is node that has been generated
- Expand-node is node that currently expanded
- Dead node is Expand-node that doesn't lead to solution and killed and never be expand again
- Generate every possible child node
- Next live node become current expand-node
- Searching continue until found a goal node

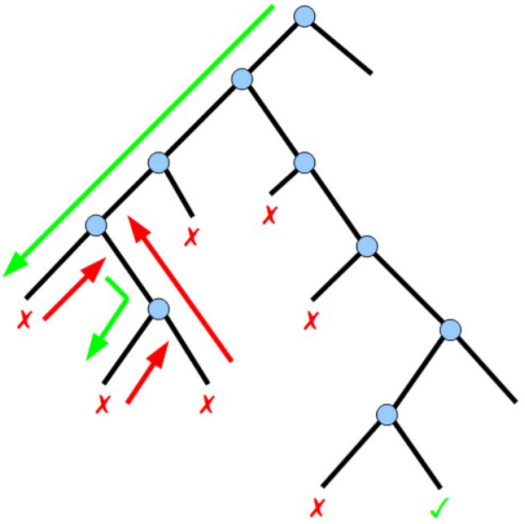

Figure 3 - Representation of how Backtracking works
Source:
http://www.w3.org/2011/Talks/01-14-steven-phenotype/

Pseudocode for backtracking using recursion:

```
procedure Backtrack(input k:integer)
{Find every solution with backtrack
(recursion)
input: k, index element of solution
vector, x[k]
output: Solution x = (x[1], x[2], …,
x[n])
}
Algorithm:
 for every x[k] which has not been tried
in such a way
 ( x[k]T(k)) and B(x[1], x[2], ...
,x[k])= true do
 if (x[1], x[2], ... ,x[k])is a solution
path
 then
 print(x)
 endif
 Backtrack(k+1) { generate x[k+1]}
 endfor
```

If the number of nodes in the state space tree is 2^n or n!, then worst case scenario for backtrack algorithm will have time complexity of $O(p(n)2^n)$ or O(q(n)n!), where p(n) and q(n) is a polinom with n degree presenting every node's complexity time.

C. The N-Queens Problem

The N-Queens Problem is a problem where given a chessboard with size of N x N and N queens that we must

place in a certain way so that no 2 queens attack each other. Example of figure 4.

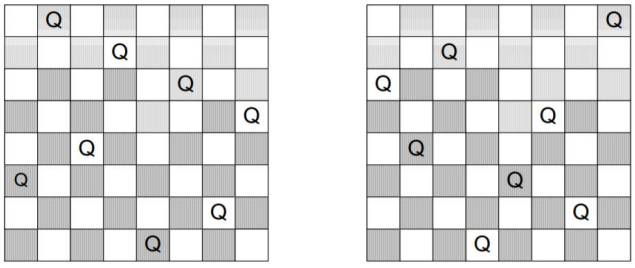

Figure 4 - Example of a N-Queens Solutions

Source
:http://www.w3.org/2011/Talks/01-14-steven-phenotype/

This problem was originally proposed in 1848 by the chess player Max Bezzel, and over the years, many mathematicians, including Gauss, have worked on this puzzle and its generalized N-queens problem. The first solutions were provided by Franz Nauck in 1850. Nauck also extended the puzzle to n-queens problem (on an N×N board—a chessboard of arbitrary size).

This classic problem can be solved with many algorithms for example with bruteforce, exhaustive search, and backtracking. The brute force method uses the idea of placing every possible queen in the chessboard, which is 4.426.165.368 possible solutions (C(64, 8)). And can be improved to 16.777.216 possibility solution (8^8) by placing every queen in different rows.Exhaustive search used the idea of permutation. If the solution is a 8-tuple vector ($X = (x_1 , x_2 , \dots , x_8)$), so the solution is a permutation of 1 to 8, the total possible solution is P(1,8)=8!= 40.320.

The idea to solve this problem using backtrack is to place each queen one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

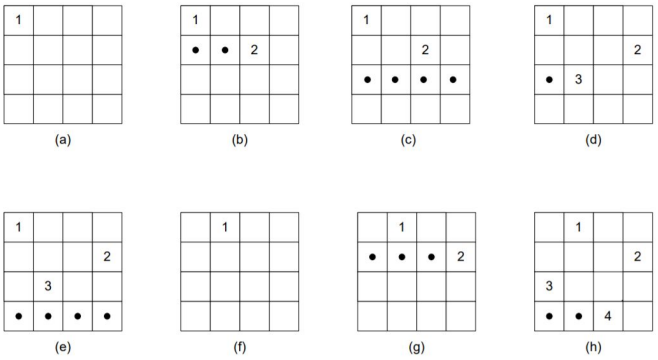

Figure 5 - Example of 4-Queens Solutions using backtrack

Source
:http://www.w3.org/2011/Talks/01-14-steven-phenotype/

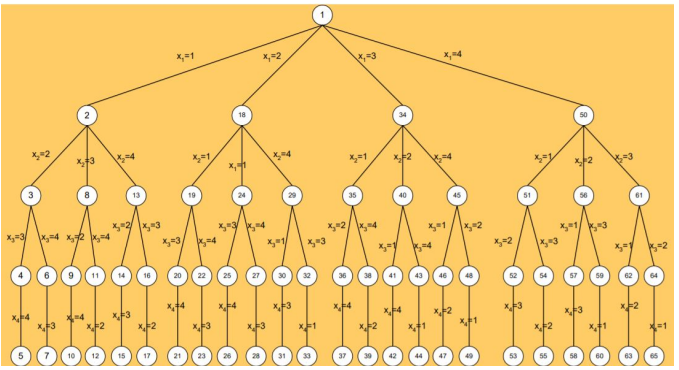

Figure 6 - 4-Queens state space tree

Source
:http://www.w3.org/2011/Talks/01-14-steven-phenotype/

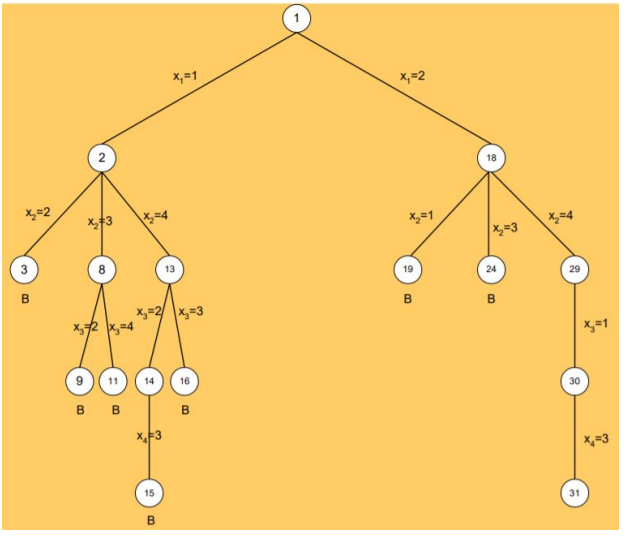

Figure 7 - 4-Queens backtrack generated while searching

Source
:http://www.w3.org/2011/Talks/01-14-steven-phenotype/

N-queen solver algorithm with backtrack :

```
function Place(input k:integer)boolean
{true if queen can be place in x[k],
false otherwise}
Declaration
 i : integer
 stop,canPlace : boolean
Algorithm:
 canPlace ← true
 i← 1
 stop← false
 while (i<k) and (not stop) do
 if (x[i]=x[k]) or
(ABS(x[i]-x[k])=ABS(i-k))then
 canPlace← false
 stop← true
 else
 i← i+1
 endif
 endwhile
 return canPlace

procedure N_RATU_R(input k:integer)
{ place queen in k-row
 prerequisite: x is an array of integer,
size 8 and initialized with 0
 input: N, total queens
 output: print every solution x = (x[1],
x[2], …, x[N]).
}
Declaration:
 stop : boolean
Algorithm:
 stop← false
 while not stop do
 x[k]← x[k]+1
 while (x[k] <= n) and (not Place(k)) do
   x[k]← x[k]+1
 endwhile
 if x[k]<=N then
 if k=N then
 print(x,N)
 else
 N_RATU_R(k+1)
else
 stop← true
 x[k]← 0
 endif
 endwhile
 {stop}
```

D. Star Battle Puzzle

Star Battle is categorized as an object placement puzzle. This puzzle is not so different from more common puzzles like Sudoku and Battleships that have its own intriguing logic to deduce. The objective of this puzzle is to place an object (star) in every rows, columns, and blocks with a specific number (example 1 and 2 stars).

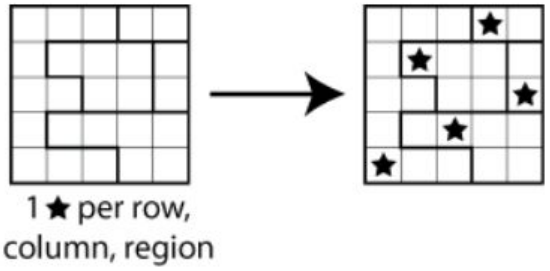

Figure 8 - Star Battle Puzzle 5x5 grid and its solution

Source: https://www.wired.com/2010/12/dr-sudoku-prescribes-star-battle/

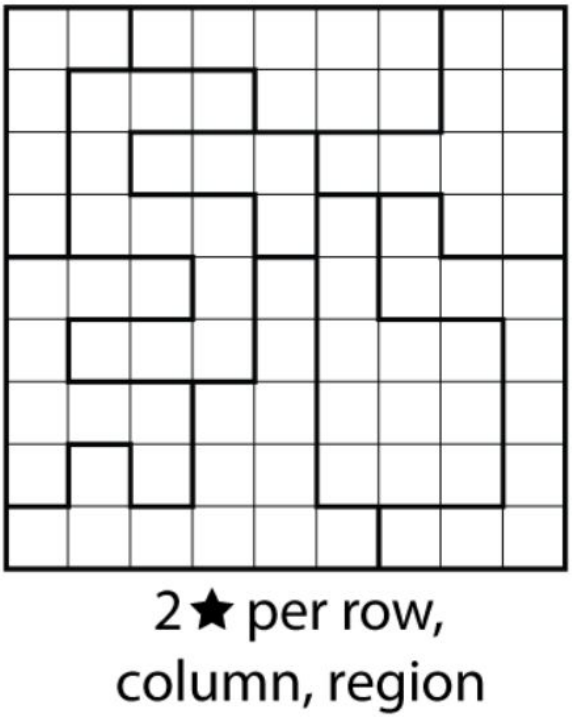

Figure 9 - Star Battle Puzzle 9x9 grid

Source: https://www.wired.com/2010/12/dr-sudoku-prescribes-star-battle/

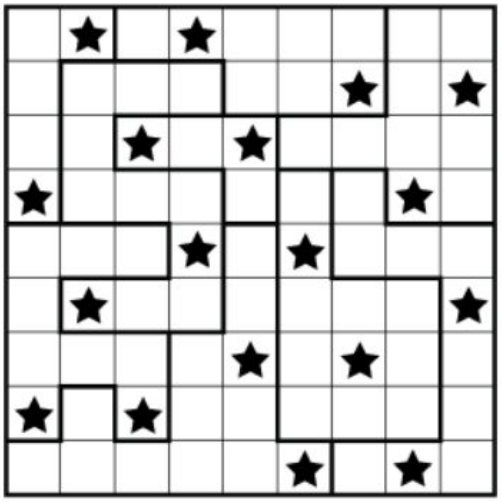

Figure 10 - Solution for figure 8

Source:
https://www.wired.com/2010/12/dr-sudoku-prescribes-star-battle/

Star Battle is created by Hans Eendebak. First appeared in the 2003 World Puzzle Championship in the Netherlands.

The rules are:

- place 1 star (or 2 stars, depend on the puzzle) on each row, column and block.
- 2 stars cannot be adjacent horizontally, vertically or diagonally.

III. USING BACKTRACK TO SOLVE STAR BATTLE PUZZLE

As we can see, star battle 1 star puzzle is similar to the N-queens problem. Where our target is to put every object in every row and column, with addition of every block or region containing 1 star. Thanks to this, we can solve the puzzle with backtracking algorithm similar to N-queens'. in this paper, the author will use a 5x5 puzzle like figure 11. We can apply the algorithm from the left upper side and go to the right lower side.

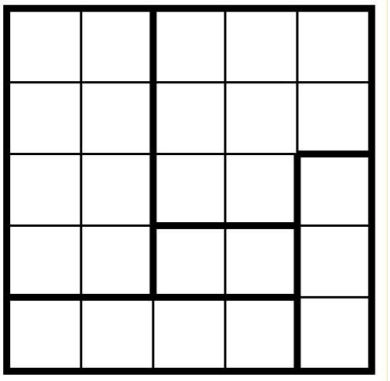

Figure 11 - Star Battle Puzzle 5x5

Before Applying the algorithm we must know the property of backtrack:

1. Solution of the problem :8-tuple vector (X = (x1 , x2 , ... , x8), i represent every row and X_i represent which column the star is placed.
2. Generating Function : loop for every column in a row
3. Bounding Function : if to check whether there's a star in this column, this block or the neighbour cells (right and left) in the previous cell. return true if available, otherwise false

So the idea to solve this problem is, first check a cell to put the star, for the first one we will put it at the left upper corner (1,1). I'll use yellow as the star and red as the not possible cell.

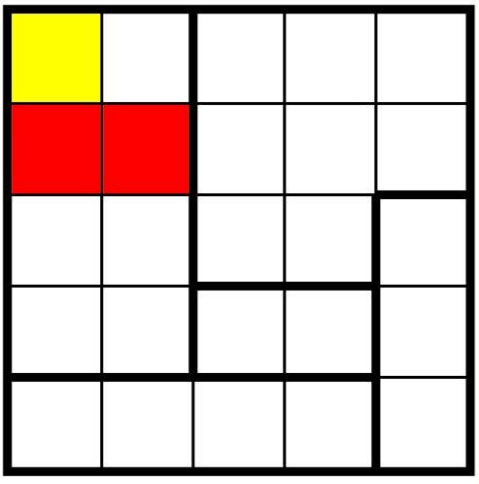

Figure 12

After putting the first star, move to the second row and pick a cell to put the second star. Before putting it, check if the cell is available using the bounding function. If the cell is available, then put the star and continue to the next row, otherwise check the next column. If there's no possible place to put the star (figure 13). Do the backtrack and search for other cell in previous row.

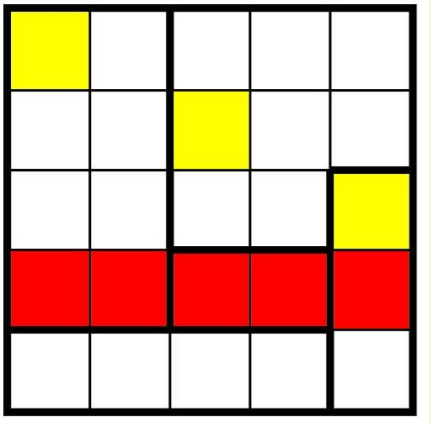

Figure 13

The process is continued until it reach the goal state (figure 14) or there is no possible answer. Figure 14 is the solution with X=(4,2,5,3,1).

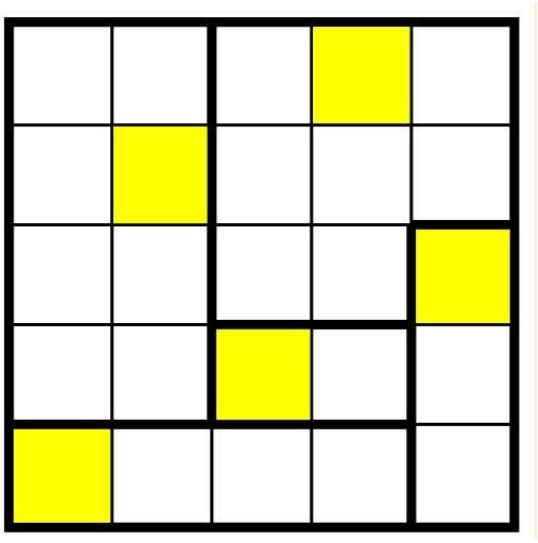

Figure 14 - Solution

The State space tree that the backtrack algorithm make for this puzzle is in the figure 15. Read it from upper-left to lower -right order.

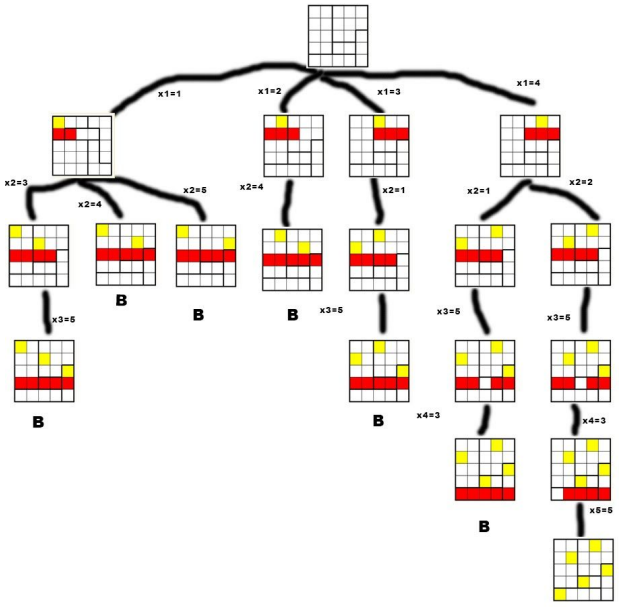

Figure 15 - State Space Tree for solving figure 11

Although this process takes a lot of effort and resources (memory and time), it is effective to solve most of the puzzles.

Here the result of the implementation of this algorithm to solve the given puzzle and other example.

```
0 0 1 1 1
0 0 1 1 1
0 0 1 1 2
0 0 3 3 2
4 4 4 4 2
The solution :
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
```

figure 16

```
Puzzle to be solved :
0 0 0 0 0 0 0 1 1 1
5 0 0 0 2 2 2 1 1 1
5 2 2 2 2 6 6 6 1 1
5 5 4 4 4 6 6 6 6 1
5 5 4 4 4 4 6 6 8 8
5 5 5 4 9 9 6 8 8 8
5 5 5 9 9 9 9 8 8 8
5 5 9 9 9 9 3 3 3 8
5 5 5 9 3 3 3 7 7 8
5 5 5 9 3 7 7 7 7 8
The solution :
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0
```

figure 17

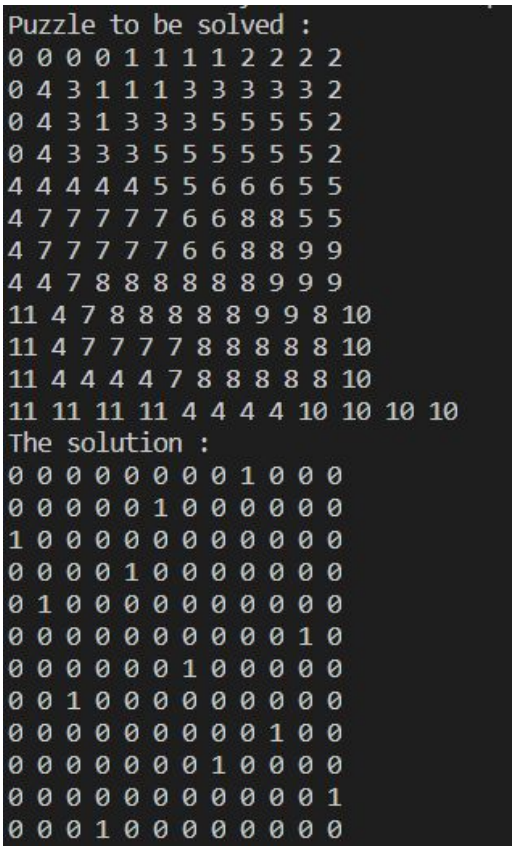

Figure 18

IV. Result

Backtracking algorithm used for solving Star Battle Puzzles could be efficient if the size of each Block is increasing from small to big as we explore the right bottom corner. The bigger the board, the more time is needed to backtracking which is really inefficient.

Video Link at Youtube

Here is the Youtube link for the explanation for this paper: https://youtu.be/immbtPIZZt4 .

Acknowledgment

The author would like to express his deepest gratitude to God for giving the author a chance to do and finish this paper. The author would also like to give special thanks for his teacher who gave him the opportunity to do this project and guidance given during this semester. Because of that the author can understand and can implement it on a real problem.

Last, the author would also like to thank his parents and friends who helped him by giving all the support that they can. Without this support the author wouldn't be able to finalize this project within the limited time frame.

.

References

[1] https://www.geeksforgeeks.org/recursion/ accessed at 26/04/2020

[2] https://www.geeksforgeeks.org/backtracking-introduction/ accessed at 26/04/2020

[3] Munir, Rinaldi. 2006. Strategi Algoritma. Bandung: Institut Teknologi Bandung.

[4] https://krazydad.com/starbattle/ accessed at 26/04/2020

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jambi, 26 April 2020

Ricky Fernando/13518062