

# Game Battleship Board Checking

Hartono Sulaiman Wijaya 13509046  
Informatics Engineering StudyProgram  
School of Electrical Engineering and Informatics  
Bandung Institute of Technology, Jl. Ganessa 10 Bandung 40132, Indonesia  
hartono.sulaiman@gmail.com

Game Battleship is a kind of guessing game played by two people. This game uses chequered board that consists of 10x10 squares. This paper will discuss about algorithm used to validate the board.

*Index Terms*—Battleship, brute force, exhaustive search, validation

## I. INTRODUCTION

Game Battleship is a kind of guessing game using minimum two boards ( each board 10 x 10 squares) and is played by two player. Each square will be given number to represent its coordinate vertically and horizontally. Each player get one board that represents the sea where the ships will be placed. Each player only get 10 ships with different size and player should arrages the ship secretly. The ships only can be arranged at the board vertically or horizontally. The ships cannot overlap, only one ship that can occupy some squares in board, once squares were occupied by a ship, other ships cannot be placed there. The ship also should not touch other ships which means a ship cannot be placed beside other ships exactly in any way.

In this game (especially for game which this paper discussed), there are four kinds of ships.

- One ship of size 4 squares (4 x 1 or 1 x 4 squares)
- Two ships of size 3 squares (3 x 1 or 1 x 3 squares)
- Three ships of size 2 squares (2 x 1 or 1 x 2 squares)
- Four ships of size 1 square (1 x 1 square).

After both of players put all the ships, they can start attacking the enemy's ships. For each turn, each player can choose coordinate (1 square) to attack. After player attacks, the enemy should tells whether a ship has been attacked or not. The ship will sink if the ship is attacked on its full body (which means if four sized-ship will sink when all four squares at ship's position are attacked). When all of enemy's ships have been sunk, the game will end and the player wins.

	A	B	C	D	E	F	G	H	I	L
1										
2										
3										
4			X							
5						X	X			
6		X						X		X
7				X						X
8	X	X						X		
9										
10										

Fig. 1 Pencil and paper game version

While playing this game traditionally, we only use pencil and paper. You can see the Fig.1. There is a board that consists of squares. You can refer to the upper left square by just saying A1 square (coordinate system). You also see some rectangles that represent the ships. For marking the attacked square, we can use cross mark at the square.

The problem when playing this kind of game, you have to put the ships correctly, but no one can exactly validate the board because the arrangement of the ships is secret. Nowadays, we can use computer to play the game and the program always do validation about the board and prevent cheating.

For this purpose, this paper will discuss about how do program validate the board. For checking the board, we can use brute force approach. One of brute force technique is exhaustive search. To know about validation of the arrangement, we should check whether all kind of ships placed properly, there are no overlapping ships, there are no ships touched each other. We must check all squares in matrix.

## II. PROBLEM SPACE

For checking the board, we have to simulate the board. Given for each board checked, it is represented by a matrix of character. The 10x10 matrix will be filled with

two kind characters. For blank square, it will be represented by '0' and for square that occupied by ship will be '\*'. The first row of matrix represent the first row of the board, and so on. The example of this matrix can be seen in Fig.2.

```

****000000
0000000000
***00***00
0000000000
00000000**
000*000000
00000000**
000*000000
00000*00*0
0*00000000
  
```

Fig. 2 Matrix represents the board and ships

Example in Fig.2 is represent a valid arrangement for ships. There are a four-squares ship , two three-squares ships, three two-squares ships, and four one-square ships. The ships are placed correctly without overlapping each other.

```

****000000
0000000000
***00***00
0000000000
00000000**
000*000000
00000000**
0000*00000
00000*00*0
0*00000000
  
```

Fig. 3 Invalid arrangement of ships – touched diagonally

Example in Fig. 3 is represent the invalid arrangement of ships. Ship in row 8 and row 9 are “touched” diagonally, so it broke the rule and made it as invalid arrangement. For another rule, we can see in Fig. 4. The example in Fig. 4 broke the rules which are the number of one-square ships are 8, it should be 4 ships, and in row 3 and 4, there are ships that overlapping.

```

****000000
0000000000
***00*0*00
00*0000000
000000000*
000*000000
000000000*
000*000000
00000*00*0
0*00000000
  
```

Fig. 4 Invalid arrangement of ships – ship number wrong

### III. SOLUTION

As explained before, to check whether the arrangement is valid or not, we use brute force approach. With brute force approach, we can use exhaustive search technique to check all element in the matrix.

For the first step, we should identify the ships are placed without any ship overlapped each other diagonally. This can be done by checking each square one by one. For each square checked, let says the square at row i-th and column j-th. We will check from square at row 1st till 9th and column 1st till 9th. We check whether square at (i-th row , j-th column) has been occupied with a ship (we check for '\*' character at the square). If true, then we check square at (i+1 -th row , j+1 -th column) to see whether it has also been occupied by a ship. We also check square at (i-th row, j+1 -th column) and (i+1 -th row, j-th column) to see whether they have been occupied by a ship. If one of squares in matrix meet the conditions, it seems that the matrix will bring an invalid arrangement of ships. You can see the position of the diagonal we checked before at Fig.5 and 6.

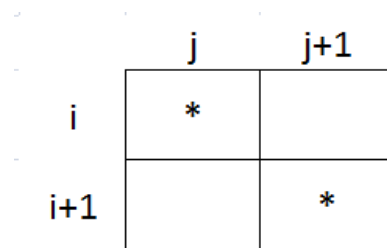


Fig. 5 diagonal -1

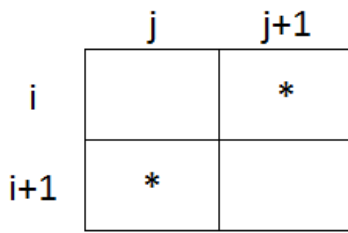


Fig. 6 diagonal -2

Then we must check the number of ships. First we check for ship whose size is bigger than 4. If we found it vertically or horizontally, the arrangement is invalid. We can choose to check if there are ships which size is 5. We use 5 squares – ship as lower bound to check.

After we check for false-size ships, we can continue to check other kind ships. We count the number of ship and then compare the result with the rules. We can see the relationship between ship’s size and ships count in table 1. we can compare the size + count with 5. If they equals, then the ship count of that size is correct and we can continue with counting the other size ship until all kind of ship are counted properly.

Ship's size	Number
4	1
3	2
2	3
1	4

Table 1 Relationship between size and count

When counting then number of ships with length k, we create counter to save the number of ships. Then we iterate each square which is the neighbour of square checked horizontally. Let see Fig. 7, the red one is the square we checked. First checking is to see the content of square beside the red square till column i+k (the green rectangle). if ‘\*’ count matches the size of ship (k) then the counter increase.

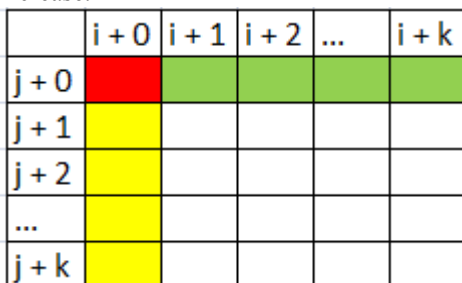


Fig. 7 Checking square vertically and horizontally

The way we check the vertical position of the ship is

same as horizontal checking. See the Fig. 7. Examine the yellow rectangle. it will count the ‘\*’ character and validate the count with ship’s size. At the end of checking, program will try to match the counter with number in table 1.

The method that explained before was the brute force way. We also will try finding another way that more effective than brute force way.

The brute force way check all square from row 1 till 10 and from column 1 to 10, so it will check 10 x 10 x 5 squares to get the result. The size of this problem is small enough to solve with brute force. But is there any method?

We can add variety to the brute force way. Before check the square, we check whether it square has been checked before or not. We will list all the squares that have been checked. When we find a ship, we list all the squares occupied by that ship. with this method, we can minimize the number of squares that will be checked. We can call this skip method.

```

check if there are some ships that touch diagonally with other ships
for length = 5 to 1
  count the ships that have length same with length
  counter = 0
  for i = 1 to 10
    for j = 1 to 10
      check if the square at (i,j) has been occupied by ship or not
      count '*' horizontally
      if count = length
        counter ++
      count '*' vertically
      if count = length
        counter ++
    if length = 5 and counter > 0 -> return false
  if length + counter != 5 -> return false

```

Fig. 8 Pseudocode for this method

#### IV. PERFORMANCE ANALYSIS

At table 1, we can see the result of test for these methods. These methods are applied on 9 variations of ship arrangements. Then runtime of the program is calculated in millisecond. For fifteen iteration of the program, it gives the result as written in table below.

NO	BRUTE FORCE METHOD	SKIP METHOD
1	48	44
2	38	38
3	45	41
4	40	42
5	39	40
6	38	38
7	39	41

8	39	39
9	39	40
10	42	41
11	41	40
12	40	43
13	41	40
14	38	40
15	38	39
AVERAGE	40,3	40,4

*Table 2 Runtime result for 9 boards in milliseconds*

With this result, we can see that with small domain of problem, the brute force approach gives a good solution. The brute force average runtime is almost same as the skip method average runtime.

Given  $n$  as number of ship types,  $m$  as board size. This brute force algorithm has complexity  $O(n \times m^2)$ .

#### V. CONCLUSION

For this experiment, we can see that brute force method is also effective for problem with small domain.

#### REFERENCES

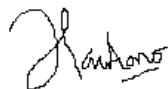
- [1] Wikipedia. "Battleship (game)." Internet: [http://en.wikipedia.org/wiki/Battleship\\_\(game\)](http://en.wikipedia.org/wiki/Battleship_(game)), Jun, 2011 [Dec 08,2011].

#### 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.

Bandung, 8 Desember 2011

ttd



Hartono Sulaiman Wijaya 13509046