COMS3005
Advanced Analysis of Algorithms
Peg Solitaire Assignment

WITS
UNIVERSITY

Abdulkadir Dere - 752817
Shaneel James - 718840
Sabeehah Ismail - 797621

School of Computer Science
University of Witwatersrand
31 October 2017

Contents

1 Introduction

We will conduct an experiment to analyse the performance of execution time of Peg Solitaire using backtracking algorithm.

2 Objective

From our experiment we aim to determine if a threshold exists where a certain number of pegs in the starting configuration can drastically affect the empirical analysis of the algorithm. Our experiment measures the execution time by increasing the number of pegs on the initial configuration board and recording the running time of each addition.

3 Summary of Theory

3.1 Peg Solitaire

Peg Solitaire is a board game where a single player moves a series of pegs on a board that contains holes. Peg Solitaire can be played on various sizes and types of boards, but for our experiments we will be using an English board which contains 33 holes.

3.1.1 Configuration

Traditionally the initial configuration of the English board is arranged so that 32 pegs are placed in the board and an empty space is left in the centre of the board. Peg Solitaire includes a variety of alternate initial configurations of pegs, which can include either the positioning or the number of pegs on the board.

3.1.2 Rules

Peg Solitaire is played by moving a peg either horizontally or vertically, not diagonally. A peg is moved by jumping over another peg into an empty space, this empty space must be located next to the peg being jumped over and in the direction of the jump.The peg that is jumped over is removed from the board which leads to an additional empty space on the board. You can jump over only one peg at a time.

3.1.3 Goal

The goal of peg solitaire is to arrive to a board whereby all the pegs have been removed except for one peg. This last peg must be located in the centre of the board for the English board configuration, where we had an empty space at the beginning of the game.

3.2 Backtracking

Backtracking is a recursive search algorithm which aims to build possible paths until the algorithm reaches a solution. While building paths, if the algorithm determines that a given step will not lead to a desired solution, the algorithm will then 'backtrack' to its previous step and consider a new step that could lead to a solution. If all paths have been explored and the algorithm has not reached a solution, then it concludes that there is no solution to the given problem.

4 Experimental Methodology

We shall measure the execution time for different initial configurations whereby each iteration will have one additional peg, it is expected that as the number of pegs increase so too does the execution time. The measurements will be analysed to determine if there is a threshold between a certain number of pegs where the threshold drastically increases.
Our test sample database will be filled with initial configurations that all lead up to a solution. The test sample database will consist of configurations which start with 2 pegs and end with 32 peg configuration. Addition of each peg is determined randomly.
The execution times for each iteration will be plotted on a graph where the independent variable will be the number of pegs and the dependent variable will be the execution time. The execution time will be measured by nanoseconds method within the Java System class to time how long the algorithm took to find a solution.
We will stop the execution of the algorithm after 15 minutes if the algorithm does not complete within 15 minutes. We will take this approach as we are constrained by time.
Order of peg movements will remain constant throughout each test for consistency of the results.

5 Theoretical Analysis

The final state of Peg Solitaire is when there is only one peg left and this peg must be situated in the centre of the board. The backtracking algorithm will consider all the possible moves that a peg can make and it will disregard all the paths which do not lead to a solution. The algorithm will stop when it reaches a solution or after it explores all possible paths. At the end of the execution of the algorithm, the program will display the number of pegs for the initial board, execution time, number of moves and the final board.
The number of executions of the algorithm is the basic operation.
N.B: Following cases are with regards to solvable configurations.

5.1 Best Case

The best case of Peg Solitaire will occur when the starting configuration has 2 pegs, placed in such a way that it would lead directly to a solution hence it will lead to the shortest execution time. The algorithm will reach a solution in its first execution.
The complexity of the best case would be O(1).

5.2 Worst Case

The worst case of Peg Solitaire will occur when the starting configuration has 32 pegs, hence complete board with one hole in the centre of the board. This will lead to the largest execution time because our backtracking algorithm runs recursively to find a desired solution.
The complexity of the worst case would be $O(4^{32})$. A full board contains 32 pegs and each peg can make 4 possible moves (up,right,down and left) which leads to 4^{32} possible paths, hence the worst case.
N.B: Worst case of the algorithm will not be when there is no solution for a certain configuration but rather when there are 32 pegs because our test sample database contains only solvable configurations.

6 Presentation of Results

6.1 Test Sample Database of Configurations

Each initial configuration in the following test sample database contains an incrementing number of pegs. All configurations are solvable. Each configuration is chosen randomly (i.e. addition of new peg

in each case is random).

2 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

3 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

4 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

5 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

6 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 1, 1, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

7 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 0, 0, 0, 0]
[0, 1, 1, 1, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

8 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 0, 0, 0, 0]
[0, 1, 1, 1, 1, 0, 0]
[0, 0, 1, 1, 0, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

9 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 0, 0, 0, 0]
[0, 1, 1, 1, 1, 0, 0]
[0, 0, 1, 0, 1, 1, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

10 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 0, 0, 1, 0]
[0, 1, 1, 1, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

11 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 1, 1, 0, 0]
[0, 1, 1, 1, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

12 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 1, 1, 0, 0]
[0, 1, 1, 1, 1, 1, 0]
[1, 1, 0, 0, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

13 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[0, 0, 1, 1, 1, 0, 0]
[0, 1, 1, 1, 1, 1, 0]
[1, 0, 1, 1, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

14 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[1, 0, 1, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 0]
[0, 0, 1, 1, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

15 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[1, 0, 1, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 0, 1, 1, 0, 0]
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

16 Peg Board

```
[ ,  , 0, 0, 0,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[1, 0, 1, 1, 1, 0, 0]
[1, 1, 0, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

17 Peg Board

```
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

18 Peg Board

```
[ ,  , 0, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

19 Peg Board

```
[ ,  , 0, 1, 1,  ,  ]
[ ,  , 0, 1, 0,  ,  ]
[1, 0, 1, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

20 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 0, 0, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

21 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 1, 1, 0, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

22 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 1, 1, 0, 0, 1, 1]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

23 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 1, 1, 1, 1, 0, 1]
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 0, 0]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

24 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 1, 1, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

25 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
[1, 1, 1, 1, 0, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

26 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1]
[1, 1, 1, 1, 1, 0, 1]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 0,  ,  ]
```

27 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1]
[1, 1, 1, 1, 0, 0, 1]
[ ,  , 1, 0, 1,  ,  ]
[ ,  , 0, 0, 1,  ,  ]
```

28 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 0, 0, 1,  ,  ]
```

29 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1]
[ ,  , 1, 0, 0,  ,  ]
[ ,  , 1, 1, 0,  ,  ]
```

30 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 0, 1]
[ ,  , 1, 0, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
```

31 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 0, 1, 1, 1]
[ ,  , 1, 0, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
```

32 Peg Board

```
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
[1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 0, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1]
[ ,  , 1, 1, 1,  ,  ]
[ ,  , 1, 1, 1,  ,  ]
```

Number of Pegs	Number of Executions	Execution Time (Nanoseconds)
2	3	28201
3	4	39738
4	5	26919
5	18	160659
6	19	98702
7	22	154676
8	1029	2813238
9	3912	7785968
10	3913	6610510
11	3914	7383040
12	2297	3946822
13	558960	134765788
14	558961	133485219
15	3500202	687694734
16	21386639	900000000000
17	19210234	900000000000
18	20696415	900000000000
19	21105492	900000000000
20	20645441	900000000000
21	21869533	900000000000
22	21020631	900000000000
23	21066551	900000000000
24	21235482	900000000000
25	21771093	900000000000
26	21153033	900000000000
27	21061481	900000000000
28	21342145	900000000000
29	21426247	900000000000
30	21047973	900000000000
31	21361083	900000000000
32	5606795	991749150

Figure 1: The table above shows the Number of Pegs versus Number of Executions versus Time of Execution.

When the number of pegs is between 16 and 31, execution time is 15 minutes ($9x10^{11}$ nanoseconds) because we limited the running time of these configurations to 15 minutes due to the fact that these configurations took over 15 minutes to reach their solution.

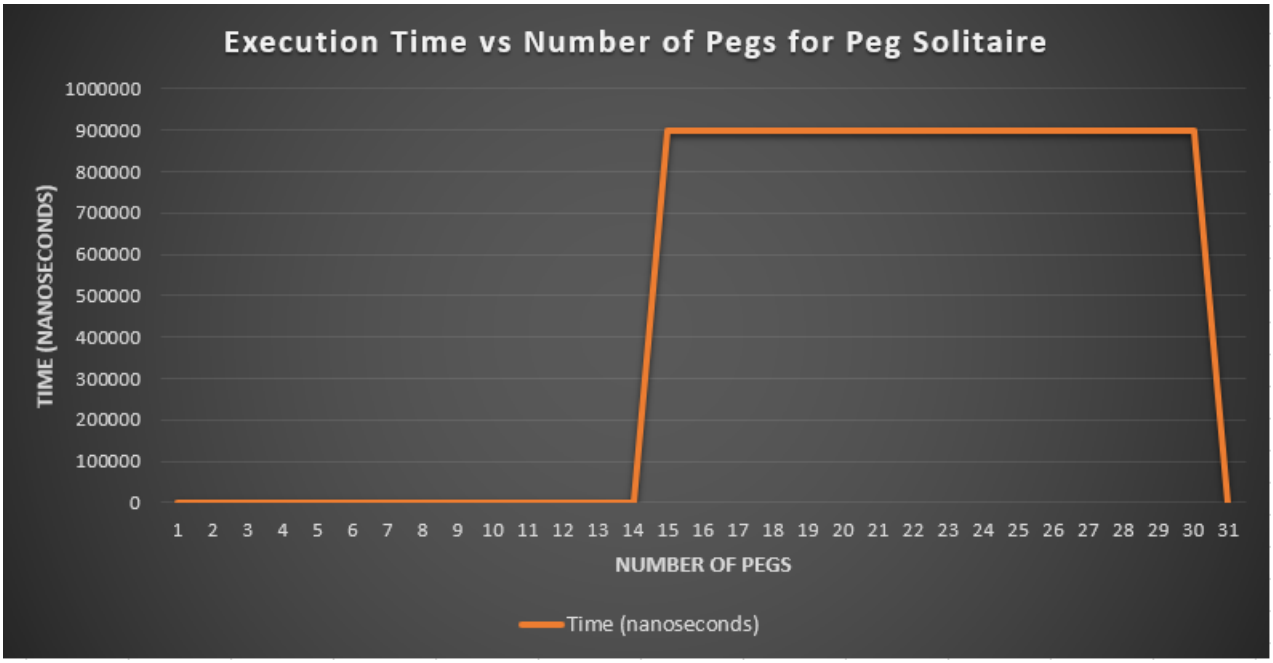

Figure 2: The graph above presents the execution time versus number of pegs.

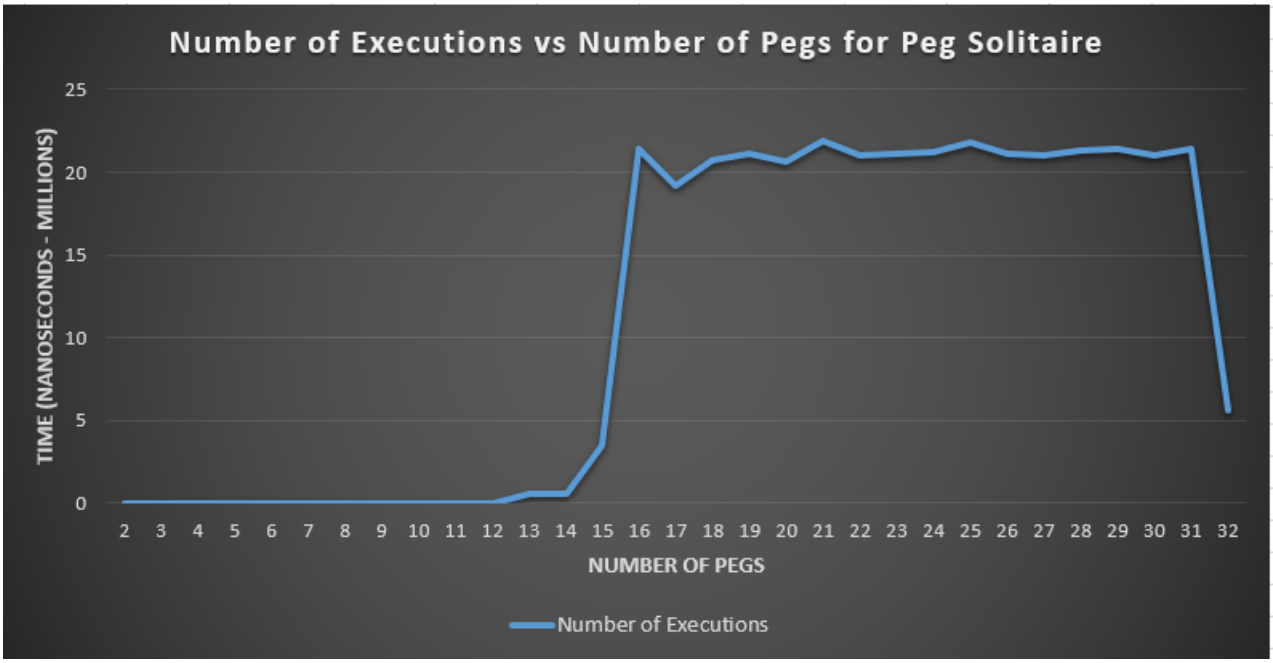

Figure 3: The graph above presents the number of executions versus number of pegs.

7 Interpretation of Results

When analysing the running time of the algorithm, we noticed two thresholds. These two thresholds indicate the rapid increase and decrease in time. At 16 pegs we noticed a rapid increase in the running time of the peg solitaire solution. This indicates that there exists a threshold at 16 pegs such that n < 16 produces a short running time and n >= 16 produces a very long running time, n being the number of pegs on the board. Where n<16 our algorithm produces a running time less than 1 second. Whereas when is n >= 16 our algorithm produces a running time greater than 15 minutes. We stopped the execution of the algorithm after 15 minutes because this resulted in a long waiting although we know that will find a solution (test sample database contains only solvable configurations). At 32 pegs (full board with an empty centre) running time decreases rapidly and gives running time of almost 1 second, this is where the second threshold exists. This rapid decrease is the result of the current movement order of pegs (right,left,up and down). So it is in an adequate order to produce a desired solution.
We have also analysed the number of recursive runs in order to identify why there are thresholds that exist at n = 16 and n = 32. At 16 <= n <= 31, the number of recursive runs is extensively greater than the number of recursive runs at n < 16 and n = 32. From these graph (Figure 2 and Figure 3) we can conclude that greater the number of recursive runs, the greater the running time.

8 Relationship Between Results and Theory

From our theoretical analysis, we expected the running time of the algorithm to increase as the number of pegs increase. We also expected to find a threshold where the running time drastically increases at a certain number of pegs. From Figure 2, we can verify the existence of a threshold that occurs when there are 16 or more pegs in the initial configuration board.
From analysing Figures 2 and 3, we notice that a drastic increase occurs at 16 pegs because there exists a higher number of possible moves that a peg can make before reaching a solution. This is caused by our chosen order of movements that a peg can make which is: right, left, up and then down. This running time could be improved by changing the order of the movements. For example: up, right, down and then left.
We also noticed that at 32 pegs running time drastically decreases which serves as a second threshold. We did not expect the second threshold in our theoretical analysis. Upon analysing Figure 1 and Figure 3, we notice that the number of executions at 32 pegs is lower than the number of executions at 31 pegs. This is the reason as to why the running time at 32 pegs drastically decreases as there are less possible moves to make compared to 31 pegs, since the number of holes is less in 32 pegs.

9 Conclusion

From our results, observations and analysis, we can conclude that:

- As the number of holes increase on a board so does the number of possible moves that a peg can make in the given configurations.
- There exists a high threshold at 16 pegs where the running time drastically increases and this is caused by an increased amount of possible moves that a peg can make. Hence the number of executions increase with respect to the increased amount of possible moves.
- There exists a low threshold at 32 pegs where the running time drastically decreases and this is because there is only one hole in the board and therefore less initial moves that a peg can possibly make.
- By changing the order of peg movements, running times vary and could be improved.

10 Contribution

Abdulkadir Dere - 752817	Shaneel James - 718840	Sabeehah Ismail - 797621
Implementation Report Analysis	Report Analysis Database	Report Analysis Database

The work done was broken down in such a way that each member contributed 33% each.

11 References

References

[1] How to Win the Peg Solitaire Game (English Board)
`https://www.wikihow.com/Win-the-Peg-Solitaire-Game-(English-Board)`

[2] Peg Solitaire Rules
`https://ourpastimes.com/peg-solitaire-rules-6181682.html`

[3] Introduction To Backtracking Programming
`http://algorithms.tutorialhorizon.com/author/sumitjain/`

[4] Peg Solitaire / Hi-Q
`https://www.jaapsch.net/puzzles/solitaire.htm`

[5] Durango Bill's 33 Hole Peg Solitaire
`http://www.durangobill.com/Peg33.html`

12 Source Code

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import java.util.Arrays;
import java.util.Scanner;

/**
 *
 * @author Kadir
 */
public class pegSolitaire {

    static int countMoves;

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        long dur = 0;
```

```
        pegSolitaire.countMoves = 0;
        int[][] initialBoard = null;

        System.out.print("Enter number of pegs: ");
        //int pegCount = in.nextInt();
        for(int tempi=0;tempi<33;tempi++){
            //initialBoard = setupGameboard(pegCount);
            initialBoard = setupGameboard(tempi);
            System.out.println(tempi+" Peg Board\n");
            printBoard(initialBoard);
        }
        final long startTime = System.nanoTime(); //start of timer
        //possibleMoves(initialBoard);
        final long endTime = System.nanoTime(); //end of timer

        dur = (endTime - startTime); // /1000000000; //for seconds
        //System.out.println("Number of Pegs: " + pegCount);
        System.out.println("Duration " + dur); // +" seconds");
        System.out.println("Counter: " + countMoves);
        countMoves = 0;

        if (possibleMoves(initialBoard) == false) {
            System.out.println("No solution"); // if there is no solution
        } else {
            System.out.println("Solution reached\n");
            //printBoard(initialBoard);
        }
    }

    public static boolean possibleMoves(int[][] gameboard) {
        int height = 7;
        int width = 7;
        int[][] newboard = gameboard;

        if (checkResult(gameboard) == true && gameboard[3][3] == 1) { //if we have reache
            return true;
        } else {
            countMoves++;
            //System.out.println(countMoves);
            for (int row = 0; row < gameboard.length; row++) {
                for (int col = 0; col < gameboard.length; col++) {
                    //move West
                    if (col < width - 2 && gameboard[row][col] == 1 && gameboard[row][co
                        // printBoard(newboard);
                        newboard[row][col] = 0;
                        newboard[row][col + 2] = 1;
                        newboard[row][col + 1] = 0;
                        //System.out.println("Iteration right");
                        //countMoves++;
                        if (possibleMoves(newboard) == true) {
                            return true;
                        } else {
```

```
            possibleMoves(newboard);
            newboard[row][col] = 1;
            newboard[row][col + 2] = 0;
            newboard[row][col + 1] = 1;
        }
    }
    //move East
    if (col > 1 && gameboard[row][col] == 1 && gameboard[row][col - 2] =
        newboard[row][col] = 0;
        newboard[row][col - 2] = 1;
        newboard[row][col - 1] = 0;
        //countMoves++;
        //System.out.println("Iteration left");
        if (possibleMoves(newboard) == true) {
            return true;
        } else {
            possibleMoves(newboard);
            newboard[row][col] = 1;
            newboard[row][col - 2] = 0;
            newboard[row][col - 1] = 1;
        }
    }
    //move North
    if (row < height - 2 && gameboard[row][col] == 1 && gameboard[row +
        newboard[row][col] = 0;
        newboard[row + 2][col] = 1;
        newboard[row + 1][col] = 0;
        //System.out.println("Iteration up");
        //countMoves++;
        if (possibleMoves(newboard) == true) {
            return true;
        } else {
            possibleMoves(newboard);

            newboard[row][col] = 1;
            newboard[row + 2][col] = 0;
            newboard[row + 1][col] = 1;
        }
    }
    //move South
    if (row > 1 && gameboard[row][col] == 1 && gameboard[row - 2][col] =
        newboard[row][col] = 0;
        newboard[row - 2][col] = 1;
        newboard[row - 1][col] = 0;
        //countMoves++;
        //System.out.println("Iteration down");
        if (possibleMoves(newboard) == true) {
            return true;
        } else {
            possibleMoves(newboard);

            newboard[row][col] = 1;
```

```
                        newboard[row - 2][col] = 0;
                        newboard[row - 1][col] = 1;
                    }
                }
            }
        }
    }
    //printBoard(newboard);
    return false;
}

public static boolean checkResult(int[][] board) {
    int count = 0;
    for (int x = 0; x < board.length; x++) {
        for (int y = 0; y < board.length; y++) {
            if (board[x][y] == 1) {
                count++;
            }
        }
    }
    if (count > 1) {
        return false;
    } else {
        return true;
    }
}

public static int[][] setupGameboard(int pegCount) {
    int[][] gameboard = new int[7][7];

    for (int i = 0; i < gameboard.length; i++) {
        for (int j = 0; j < gameboard.length; j++) {
            gameboard[i][j] = -1;
        }
    }

    gameboard[3][3] = 0; //0 means there is no peg
    gameboard[0][2] = 1; //1 means there is a peg
    gameboard[0][3] = 1;
    gameboard[0][4] = 1;
    gameboard[1][2] = 1;
    gameboard[1][3] = 1;
    gameboard[1][4] = 1;
    gameboard[2][0] = 1;
    gameboard[2][1] = 1;
    gameboard[2][2] = 1;
    gameboard[2][3] = 1;
    gameboard[2][4] = 1;
    gameboard[2][5] = 1;
    gameboard[2][6] = 1;
    gameboard[3][0] = 1;
    gameboard[3][1] = 1;
```

```
gameboard[3][2] = 1;
gameboard[3][4] = 1;
gameboard[3][5] = 1;
gameboard[3][6] = 1;
gameboard[4][0] = 1;
gameboard[4][1] = 1;
gameboard[4][2] = 1;
gameboard[4][3] = 1;
gameboard[4][4] = 1;
gameboard[4][5] = 1;
gameboard[4][6] = 1;
gameboard[5][2] = 1;
gameboard[5][3] = 1;
gameboard[5][4] = 1;
gameboard[6][2] = 1;
gameboard[6][3] = 1;
gameboard[6][4] = 1;

//Sample Test Database
switch (pegCount){
    case 32:
        gameboard[3][3] = 0; //0 means there is no peg
        break;
    case 31:
        gameboard[3][3] = 1;
        gameboard[4][3] = 0;
        gameboard[5][3] = 0;
        break;

    case 30:
        gameboard[3][3] = 1;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[5][3] = 0;
        break;

    case 29:
        gameboard[3][3] = 1;
        gameboard[4][5] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][4] = 0;
        break;

    case 28:
        gameboard[3][3] = 1;
        gameboard[4][5] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        break;
```

```
case 27:
    gameboard[3][3] = 1;
    gameboard[3][4] = 0;
    gameboard[4][4] = 0;
    gameboard[4][5] = 0;
    gameboard[5][3] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    break;

case 26:
    gameboard[3][3] = 1;
    gameboard[3][4] = 0;
    gameboard[4][5] = 0;
    gameboard[5][3] = 0;
    gameboard[5][4] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    gameboard[6][4] = 0;
    break;

case 25:
    gameboard[3][3] = 1;
    gameboard[1][4] = 0;
    gameboard[2][4] = 0;
    gameboard[4][5] = 0;
    gameboard[5][3] = 0;
    gameboard[5][4] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    gameboard[6][4] = 0;
    break;

case 24:
    gameboard[3][3] = 1;
    gameboard[1][4] = 0;
    gameboard[2][5] = 0;
    gameboard[2][6] = 0;
    gameboard[4][5] = 0;
    gameboard[5][3] = 0;
    gameboard[5][4] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    gameboard[6][4] = 0;
    break;

case 23:
    gameboard[3][3] = 1;
    gameboard[1][4] = 0;
    gameboard[2][5] = 0;
    gameboard[3][6] = 0;
```

```
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 22:
        gameboard[3][3] = 1;
        gameboard[1][4] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 21:
        gameboard[3][3] = 1;
        gameboard[1][4] = 0;
        gameboard[2][3] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 20:
        gameboard[3][3] = 1;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
```

```
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 19:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 18:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][2] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 17:
        gameboard[3][3] = 1;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][2] = 0;
```

```
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 16:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][2] = 0;
        gameboard[3][6] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 15:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][1] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][6] = 0;
        gameboard[4][2] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;
```

```
case 14:
    gameboard[3][3] = 1;
    gameboard[0][2] = 0;
    gameboard[0][3] = 0;
    gameboard[0][4] = 0;
    gameboard[1][2] = 0;
    gameboard[1][4] = 0;
    gameboard[2][1] = 0;
    gameboard[2][5] = 0;
    gameboard[2][6] = 0;
    gameboard[3][6] = 0;
    gameboard[4][0] = 0;
    gameboard[4][1] = 0;
    gameboard[4][5] = 0;
    gameboard[4][5] = 0;
    gameboard[4][6] = 0;
    gameboard[5][2] = 0;
    gameboard[5][3] = 0;
    gameboard[5][4] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    gameboard[6][4] = 0;
    break;

case 13:
    gameboard[3][3] = 1;
    gameboard[0][2] = 0;
    gameboard[0][3] = 0;
    gameboard[0][4] = 0;
    gameboard[1][2] = 0;
    gameboard[1][4] = 0;
    gameboard[2][0] = 0;
    gameboard[2][1] = 0;
    gameboard[2][5] = 0;
    gameboard[2][6] = 0;
    gameboard[3][0] = 0;
    gameboard[3][6] = 0;
    gameboard[4][1] = 0;
    gameboard[4][5] = 0;
    gameboard[4][6] = 0;
    gameboard[5][2] = 0;
    gameboard[5][3] = 0;
    gameboard[5][4] = 0;
    gameboard[6][2] = 0;
    gameboard[6][3] = 0;
    gameboard[6][4] = 0;
    break;

case 12:
    gameboard[3][3] = 1;
    gameboard[0][2] = 0;
```

```
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][6] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 11:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][3] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 10:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
```

```
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][3] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 9:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][5] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][3] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 8:
```

```
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][5] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 7:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][5] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
```

```
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 6:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][1] = 0;
        gameboard[3][5] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 5:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][1] = 0;
```

```
        gameboard[3][5] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 4:
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][1] = 0;
        gameboard[3][3] = 0;
        gameboard[3][4] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 3:
        gameboard[3][3] = 1;
        gameboard[0][2] = 0;
```

```
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][3] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][1] = 0;
        gameboard[3][4] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
        gameboard[4][2] = 0;
        gameboard[4][3] = 0;
        gameboard[4][4] = 0;
        gameboard[4][5] = 0;
        gameboard[4][6] = 0;
        gameboard[5][2] = 0;
        gameboard[5][3] = 0;
        gameboard[5][4] = 0;
        gameboard[6][2] = 0;
        gameboard[6][3] = 0;
        gameboard[6][4] = 0;
        break;

    case 2:
        gameboard[0][2] = 0;
        gameboard[0][3] = 0;
        gameboard[0][4] = 0;
        gameboard[1][2] = 0;
        gameboard[1][3] = 0;
        gameboard[1][4] = 0;
        gameboard[2][0] = 0;
        gameboard[2][1] = 0;
        gameboard[2][2] = 0;
        gameboard[2][3] = 0;
        gameboard[2][4] = 0;
        gameboard[2][5] = 0;
        gameboard[2][6] = 0;
        gameboard[3][0] = 0;
        gameboard[3][1] = 0;
        gameboard[3][2] = 0;
        gameboard[3][3] = 0;
        gameboard[3][6] = 0;
        gameboard[4][0] = 0;
        gameboard[4][1] = 0;
```

```
                gameboard[4][2] = 0;
                gameboard[4][3] = 0;
                gameboard[4][4] = 0;
                gameboard[4][5] = 0;
                gameboard[4][6] = 0;
                gameboard[5][2] = 0;
                gameboard[5][3] = 0;
                gameboard[5][4] = 0;
                gameboard[6][2] = 0;
                gameboard[6][3] = 0;
                gameboard[6][4] = 0;
                break;
            }

            //printBoard(gameboard);
            return gameboard;
        }

        public static void printBoard(int[][] board) {
            System.out.println(Arrays.deepToString(board).replace("-1", "  ").replace("], ",
        }
    }
```