CS4306 Algorithm Analysis
Fall 2021
Department of Computer Science
Kennesaw State University

Programming Assignment 1: Game of Life
Due Date: Thursday, September 2, 2021 (by 11:59pm)

Report

My Anh Huynh

a) Description of the analysis of the problems:
a) The game of life rule:

- Birth: a dead cell with exactly three live neighbours becomes a live cell

- Survival: a live cell with two or three neighbours stays alive

- Overcrowding/ Loneliness: in all other cases a cell dies or remains dead:

b) IDE tool and operating system:
This analysis use Java programming language with IDE that is Intelli J Idea in a MacOS version 10.14.6. Also use execel to generate graph of the function of time.
c) Data structure, time and complexity:

Generally, with different types of representation of matrix or two dimensional array have different performance in time and space cost. This assignment I use a two dimensional array 10x10 that generated randomly.
The problem uses nested loops contain sizes M and N so the cost is O(M*N) to generate the original array and update cells.
For example:

```
// populating the grid
for (int i = 0; i < M; i++) {

        for (int j = 0; j < N; j++) {
            int randomNum =
randomNumberGenerator.ints(0,2).findFirst().getAsInt();
            grid[i][j] = randomNum;
            System.out.print(grid[i][j] + " ");
            if (j == M - 1) {
                System.out.println();
            }
        }
    }
```

d) System time:
Also, we use the function System.nanoTime() to calculate the time that the program run for each generation execute. For the size, I also run both size 10*10 and 5*5.

```
long startTime = System.nanoTime();
        long endTime = System.nanoTime();
        double duration = (double)(endTime- startTime)/1000000;// divide by 1000000 to get millisecond
        System.out.println("duration is:"+duration);
    }
}
```

From every execute time, we collect the time and draw the figure below:

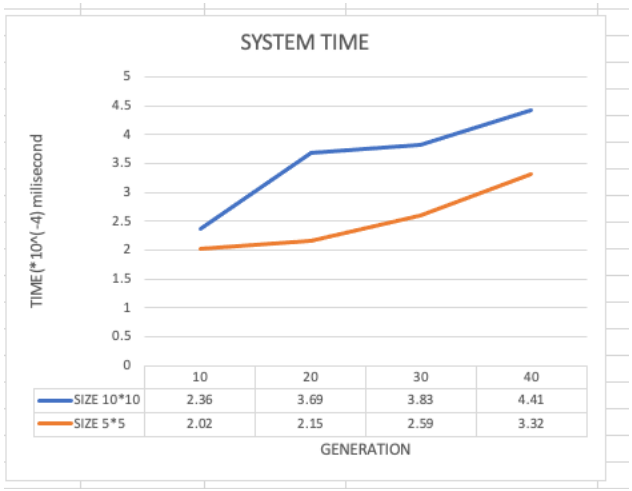
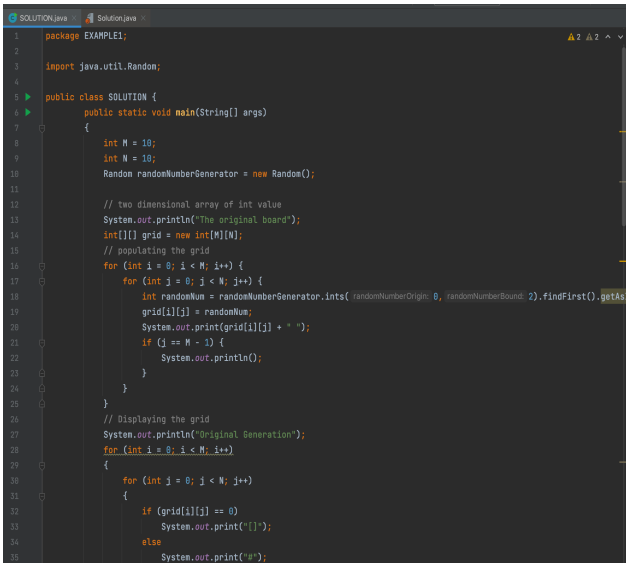


Figure 1: System Time

We could see that the bigger size of the array will take more time to execute in each generation.

e) Source code and screen shot:



```
package EXAMPLE1;

import java.util.Random;

public class SOLUTION {
        public static void main(String[] args)
        {
            int M = 10;
            int N = 10;
            Random randomNumberGenerator = new Random();

            // two dimensional array of int value
            System.out.println("The original board");
            int[][] grid = new int[M][N];
            // populating the grid
            for (int i = 0; i < M; i++) {
                for (int j = 0; j < N; j++) {
                    int randomNum = randomNumberGenerator.ints( randomNumberOrigin: 0, randomNumberBound: 2).findFirst().getAs
                    grid[i][j] = randomNum;
                    System.out.print(grid[i][j] + " ");
                    if (j == M - 1) {
                        System.out.println();
                    }
                }
            }
            // Displaying the grid
            System.out.println("Original Generation");
            for (int i = 0; i < M; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    if (grid[i][j] == 0)
                        System.out.print("[]");
                    else
                        System.out.print("#");
```

```
        }
        System.out.println();
    }
    System.out.println();
    nextGeneration(grid, M, N);
}

// Function to print next generation
static void nextGeneration(int grid[][], int M, int N)
{
    int[][] future = new int[M][N];

    // use for loop in each cell
    for (int l = 1; l < M - 1; l++)
    {
        for (int m = 1; m < N - 1; m++)
        {
            // number of neighbor is alive
            int aliveNeighbours = 0;
            for (int i = -1; i <= 1; i++)
                for (int j = -1; j <= 1; j++)
                    aliveNeighbours += grid[l + i][m + j];

            // The cell needs to be subtracted from
            // its neighbours as it was counted before
            aliveNeighbours -= grid[l][m];

            // the rule of life was implement

            // Cell is lonely and dies
            if ((grid[l][m] == 1) && (aliveNeighbours < 2))
                future[l][m] = 0;

                // Cell dies because of over population
            else if ((grid[l][m] == 1) && (aliveNeighbours > 3))
                future[l][m] = 0;
```

```
                // A new born cell
            else if ((grid[l][m] == 0) && (aliveNeighbours == 3))
                future[l][m] = 1;

                // Remains the same
            else
                future[l][m] = grid[l][m];
        }
    }

    System.out.println("Next Generation");
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (future[i][j] == 0)
                System.out.print("[]");
            else
                System.out.print("#");
        }
        System.out.println();
    }
    // calculate the time
    long startTime = System.nanoTime();
    long endTime = System.nanoTime();
    double duration = (double)(endTime- startTime)/1000000;// divide by 1000000 to get millisecond
    System.out.println("duration is:"+duration);
  }
}
```

```
The original board
1 1 1 1 1 0 0 1 1 0
0 0 0 0 1 0 1 0 0 0
0 0 0 1 0 0 1 0 0 1
0 0 0 1 1 1 1 1 1 1
1 0 1 0 1 1 1 0 0 0
1 0 0 1 1 0 0 0 0 0
0 1 0 0 0 1 1 0 0 1
1 0 1 0 0 0 0 1 1 0
1 0 0 0 0 1 0 0 1 0
0 0 0 0 1 0 1 1 0 1
```

```
Original Generation
###[]#[]#[]##
[]#[][]####[][]
##[][]######
[]#[][][]###[]#
[]#[][]#[][][]#[]
#[][][]#[]##[]#
##[][]#[][][]##
#####[][]##[]
#[][][]#[]####
[][][][][]##[]#[]
```

```
Next Generation
[][][][][][][][][][]
[][][][][][][][][][]
[]##[][][][][][][][]
[]##[][][][][][][][]
[]#[][]#[][][][][][]
[][][]##[][]#[][][]
[][][][]#[]#[][][]
[][]#[]#[]#[][][]
[][]#[]#[][][][][][]
[][][][][][][][][][]
duration is:1.83E-4
```