Academic Year 2021 – 2022

Semester 1

Project Report

Course: Object-Oriented Programming (IT069IU)

Lecturers: T.T.Tung (Theory) _ P.Q.S.Lam (Lab)

Name: Ngo Trieu Gia Gia _ ID: ITDSIU20034

Topic: Pokemon Basic Game

Table of Contents

I. Introduction:

In this report, I will show you the way to create a 'Pokemon Classic Game' through diffirent number of steps using Java. The body of this report will also explain about the function I use in the game and also some pictures about this game. To be convenient, I will use 'Visual Studio Code' to create and do the project.

II. Game Features:

This game is a basic matching icons games. When playing this game, it can help you improve concentration, train visual memory, increase short term memory, increase attention to detail, improve the ability to find similarities and differences in objects, help to classify objects that are grouped by similar traits, and so on.

The rule of the game is very simple. You need to find the two icons that are similar to each other. After you found the icons, you need to check whether there is a space between two icons. There are three kinds of space in this game that have been separated into three different algorithms applied to check the line between two icons. I will talk more about the algorithms on later sections. If there is a space between two icons you have found before, you can connect two icons and it will disappear. If there no icons in the game, you win the game.

III. Game Design:

In this game, I have created two directories named 'Controller' and 'Icon'. The 'Controller' directory or we could call package Controller when writing Java code is contained five different Java files. Each file contain a code of a class I built for the game. The 'Icon' directory stores the images (icons) used for the game. There are total 21 image in 'Icon' folder.

1) Classes in the game:

There are total five classes in the game called 'Controller', 'ButtonEvent', 'PointLine', 'MainFrame', and 'Main' respectively. In this section, I will show you how these five classes are linked together.

1.1/ The 'PointLine' class:

First and foremost, lets talk about the most simple class called 'PointLine'. The main purpose of this class is to define the connection of two Icons together. Each PointLine contains two Point p1, p2 which are two icons in the game. The code of this class will be displayed below:

File: **PointLine.java**

```
package Controller;

import java.awt.Point;

public class PointLine {
    public Point p1;
    public Point p2;

    public PointLine(Point p1, Point p2) {
        super();
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

The code of this class is very simple since this class is only used for checking the connection between two icons.

1.2/ The 'Controller' class:

Now, I will introduce you about one of the most important class of the game named 'Controller'. The 'Controller' class contains some important function of the game. It contains a function to create the matrix that stores the icons of the game, and also, inside Controller class, there are algorithms use for the game.

In order to store the Icons of the game, I need to create a nxn matrix. Each point in the matrix will store a random icon form 21 image in 'Icon' directory. The code of createMatrix() function will be shown below:

File: **Controller.java _ createMatrix() function:**

```
    private void createMatrix() {
        matrix = new int[row][col];
        int sizeOfMatrix = (row - 2) * (col - 2);
        Random rand = new Random();
        int imgCount = 21; // 21 icons

        int max = 0;
        if ((int)((sizeOfMatrix / imgCount) + 1) % 2 == 0)
            max = (int)((sizeOfMatrix / imgCount) + 1);
        else
            max = (int)((sizeOfMatrix / imgCount) + 1) + 1;
        int arr[] = new int[imgCount + 1];
```

```
        ArrayList<Point> listPoint = new ArrayList<Point>();

        for (int i = 0; i < col; i++) {
            matrix[0][i] = matrix[row - 1][i] = 0;
        }
        for (int i = 0; i < row; i++) {
            matrix[i][0] = matrix[i][col - 1] = 0;
        }

        for(int i = 1; i < row - 1; i++) {
            for(int j = 1; j < col - 1; j++) {
                listPoint.add(new Point(i, j));
            }
        }

        int i = 0;
        do {
            int index = rand.nextInt(imgCount) + 1;
            if(arr[index] < max) {
                arr[index] += 2;
                for(int j = 0; j < 2; j++) {
                    int size = listPoint.size();
                    int pointIndex = rand.nextInt(size);
                    matrix[listPoint.get(pointIndex).x][listPoint.get(pointIndex)
.y] = index;
                    listPoint.remove(pointIndex); // Remove field that already
has value
                }
                i++;
            }
        } while(i < (row-2) * (col-2) / 2);
    }

    public PointLine checkTwoPoint(Point p1, Point p2) {
        if (!p1.equals(p2) && matrix[p1.x][p1.y] == matrix[p2.x][p2.y]) {
            // check line with x
            if (p1.x == p2.x) {
                System.out.println("line x");
                if (checkLineX(p1.y, p2.y, p1.x)) {
                    return new PointLine(p1, p2);
                }
            }
            // check line with y
            if (p1.y == p2.y) {
```

```
                System.out.println("line y");
                if (checkLineY(p1.x, p2.x, p1.y)) {
                    System.out.println("ok line y");
                    return new PointLine(p1, p2);
                }
            }
            // check in rectangle with x
            if (checkRectX(p1, p2)) {
                System.out.println("rect x");
                return new PointLine(p1, p2);
            }
            // check in rectangle with y
            if (checkRectY(p1, p2)) {
                System.out.println("rect y");
                return new PointLine(p1, p2);
            }
            // check more right
            if (checkMoreLineX(p1, p2, 1)) {
                System.out.println("more right");
                return new PointLine(p1, p2);
            }
            // check more left
            if (checkMoreLineX(p1, p2, -1)) {
                System.out.println("more left");
                return new PointLine(p1, p2);
            }
            // check more down
            if (checkMoreLineY(p1, p2, 1)) {
                System.out.println("more down");
                return new PointLine(p1, p2);
            }
            // check more up
            if (checkMoreLineY(p1, p2, -1)) {
                System.out.println("more up");
                return new PointLine(p1, p2);
            }
        }
        return null;
    }
```

Inside this function, there is also a small function that use the class ‘PointLine’ mentioned before to check the line (or space) between two similar Icons. To check the line between two icons, I need to write algorithms for that function. Later, I will show the algorithms that I applied for the game.

1.3/ The 'ButtonEvent' class and 'MainFrame' class:

The main purpose of these classes is to create the game scene and some functions that help people interact when playing game. Class MainFrame create the game scene and some button in that scene such as 'New Game', 'Reset Level', or time bar. Class ButtonEvent use to get and display the Icons from it directory to the game scene. Class ButtonEvent also used for showing the Dialog when player win the game.

1.4/ The 'Main' class:

Main class is the place where I write code to run and test the game. Inside 'Main' class, I also write a Time function in order to decrease time when the game is running. When the time reach 0, it mean that you lose the game and a Dialog will appear to ask players whether they want to play again.

2) Algorithm to find the path between two icons:

In order to match two icons together, there are two conditions. First, the two icons are similar to each other. Second, the path between two icons is not blocked. This mean that when it needs more than 3 lines to draw the shortest distance from two icons, the path between them is blocked. In addition, I need to write the algorithms to find the shortest path between two similar icons. There are three cases in total.

2.1/ Two icons are on the same edge:

This is the simplest and easiest case to consider. I divided it into 2 cases: lying on the same horizontal row (x1 = x2) or lying on the same vertical row (y1 = y2). Considering the case of 2 icons lying on the same horizontal row (x1 = x2), I consider the remaining 2 coordinates y1 and y2 to find the point with smaller coordinates (assuming y1 < y2), then consider the relationship. Continue the cells in a horizontal row x1 from position y1 to y2, if the cells considered are all blank (with a value of 0), then these are 2 icons located in a satisfactory position. Similar to the case of 2 icons located on the same vertical row. And to do this, I wrote 2 new functions, checkLineX() and checkLineY() in the Controller class, corresponding to 2 cases where 2 icons are on the same horizontal row, and on the same vertical row. The code of these two functions will be displayed below:

File: Controller.java _ checkLineX() and checkLineY() functions:

```
    private boolean checkLineX(int y1, int y2, int x) {
        System.out.println("Check line x");

        int min = Math.min(y1, y2);
        int max = Math.max(y1, y2);

        for(int y = min + 1; y < max; y++) {
```

```
            if(matrix[x][y] != 0) {
                System.out.println("Die: " + x + "" + y);
                return false;
            }
            System.out.println("OK: " + x + "" + y);
        }

        return true;
    }

    private boolean checkLineY(int x1, int x2, int y) {
        System.out.println("Check line y");

        int min = Math.min(x1, x2);
        int max = Math.max(x1, x2);

        for (int x = min + 1; x < max; x++) {
            if (matrix[x][y] != 0) {
                System.out.println("Die: " + x + "" + y);
                return false;
            }
            System.out.println("OK: " + x + "" + y);
        }

        return true;
    }
```

2.2/ Two icons are connected by up to 3 lines within a rectangle formed from the coordinates of 2 icons:

Consider 2 points only within the bounded range from 2 coordinates of 2 icons corresponding to 2 ends of the rectangle. In this case, I have divided into 2 smaller cases: horizontally and vertically. These two cases are similar in terms of consideration, just different in direction. The main purpose of dividing into 2 cases is that we can cover all the lines with different directions. Take an intersection as a example, we have four directions, and we need to know which direction leads to our destination. Therefore, we need to cover the road from all 4 directions so as to find the shortest way to reach of goal.

To apply this algorithm, let's consider 2 points p1, p2 together with the horizontal case first. By starting to find the horizontal direction, I took out 2 coordinates of 2 points, y1 and y2. Then I compared these 2 coordinates in order to find the smaller coordinate, and start translating from the cell with the smaller coordinates to the cell with the larger coordinates with the for loop and the increment value of 1. For each time translating y by

1 unit, I constantly check whether the cell located at the test position is an empty cell. If it is an empty cell, I will check the remaining 2 lines to connect those 2 points with the 2 functions checkLineX() and checkLineY() with the corresponding input parameters being the last point in each segment. I have write two functions checkRectX() and checkRectY() so as to check this case. The code of these two functions is given as below:

File: Controller.java _ checkRectX() and checkRectY() functions:

```
    private boolean checkRectX(Point p1, Point p2) {
        System.out.println("Check rect x");

        // find point have y min and max
        Point pMinY = p1, pMaxY = p2;

        if (p1.y > p2.y) {
            pMinY = p2;
            pMaxY = p1;
        }

        for (int y = pMinY.y; y <= pMaxY.y; y++) {
            if (y > pMinY.y && matrix[pMinY.x][y] != 0) {
                return false;
            }
            // check two line
            if ((matrix[pMaxY.x][y] == 0) && checkLineY(pMinY.x, pMaxY.x, y)
                    && checkLineX(y, pMaxY.y, pMaxY.x)) {

                System.out.println("Rect x");
                System.out.println("(" + pMinY.x + "," + pMinY.y + ") -> ("
                        + pMinY.x + "," + y + ") -> (" + pMaxY.x + "," + y
                        + ") -> (" + pMaxY.x + "," + pMaxY.y + ")");

                // if three line is true return column y
                return true;
            }
        }

        // have a line in three line not true then return -1
        return false;
    }

    private boolean checkRectY(Point p1, Point p2) {
        System.out.println("Check rect y");

        // find point have y min
```

```
        Point pMinX = p1, pMaxX = p2;

        if (p1.x > p2.x) {
            pMinX = p2;
            pMaxX = p1;
        }

        // find line and y begin
        for (int x = pMinX.x; x <= pMaxX.x; x++) {
            if (x > pMinX.x && matrix[x][pMinX.y] != 0) {
                return false;
            }
            if ((matrix[x][pMaxX.y] == 0) && checkLineX(pMinX.y, pMaxX.y, x)
                    && checkLineY(x, pMaxX.x, pMaxX.y)) {

                System.out.println("Rect y");
                System.out.println("(" + pMinX.x + "," + pMinX.y + ") -> (" + x
                        + "," + pMinX.y + ") -> (" + x + "," + pMaxX.y
                        + ") -> (" + pMaxX.x + "," + pMaxX.y + ")");

                return true;
            }
        }

        return false;
    }
```

2.3/ Two icons are connected by up to 3 lines beyond the rectangular area formed from the coordinates of the 2 icons:

This is the last part of the algorithm that I applied for the game. This section is a quite similar to the previous section. This part is finding the path between two in a area that is bigger than the part above.

For this case, I created the checkMoreLineX() function in the Controller class. To solve the problem of the line extending beyond the rectangle formed from the coordinates of 2 icons, I created the type variable in the parameter passed to the checkMoreLineX() function. The type variable only takes 2 values: 1 and -1, respectively, 2 directions forward, or reverse. Compare 2 icons and then find the icon with smaller coordinates, respectively pMinY and pMaxY. Corresponding to the value of type, if type = 1, create a variable y that will receive a value equal to pMaxY.y + type , otherwise, if type = -1, your variable y will receive a value equal to pMinY.y + type. For this horizontal consideration, with 2 variables pMaxY.y and pMinY.y is the horizontal limit of the rectangle formed by the 2 icons under

consideration. Adding type to the two variables expands the search path beyond the scope of that rectangle. The code of the last two functions is:

File: Controller.java _ checkMoreLineX() and checkMoreLineY() functions:

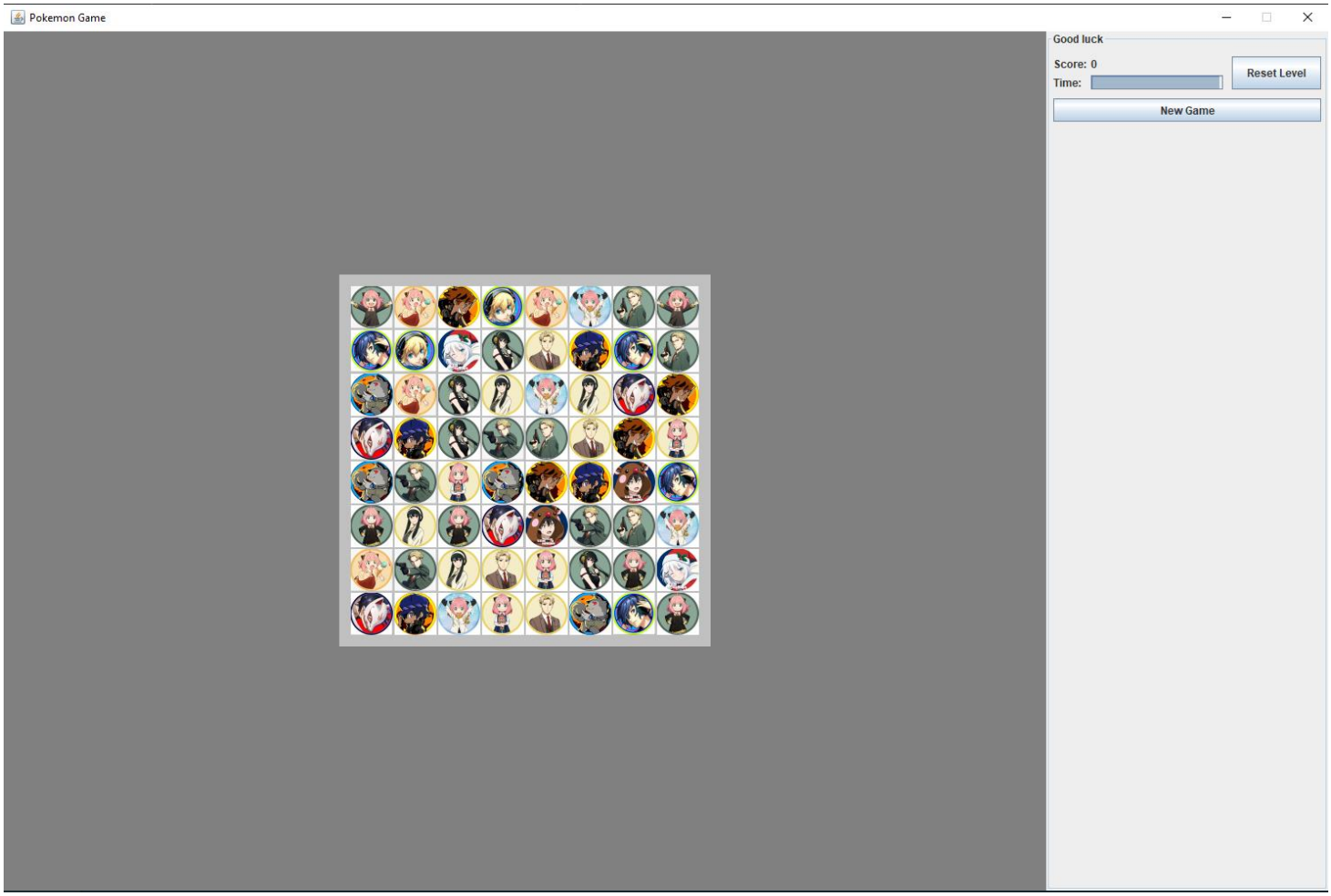
```
    private boolean checkMoreLineX(Point p1, Point p2, int type) {
        System.out.println("check more line x");
        // find point have y min
        Point pMinY = p1, pMaxY = p2;
        if (p1.y > p2.y) {
            pMinY = p2;
            pMaxY = p1;
        }
        // find line and y begin
        int y = pMaxY.y + type;
        int row = pMinY.x;
        int colFinish = pMaxY.y;
        if (type == -1) {
            colFinish = pMinY.y;
            y = pMinY.y + type;
            row = pMaxY.x;
            System.out.println("colFinish = " + colFinish);
        }

        // find column finish of line
        // check more
        if ((matrix[row][colFinish] == 0 || pMinY.y == pMaxY.y)
                && checkLineX(pMinY.y, pMaxY.y, row)) {
            while (matrix[pMinY.x][y] == 0
                    && matrix[pMaxY.x][y] == 0) {
                if (checkLineY(pMinY.x, pMaxY.x, y)) {

                    System.out.println("TH X " + type);
                    System.out.println("(" + pMinY.x + "," + pMinY.y + ") -> ("
                            + pMinY.x + "," + y + ") -> (" + pMaxY.x + "," + y
                            + ") -> (" + pMaxY.x + "," + pMaxY.y + ")");
                    return true;
                }
                y += type;
            }
        }
        return false;
    }

    private boolean checkMoreLineY(Point p1, Point p2, int type) {
```

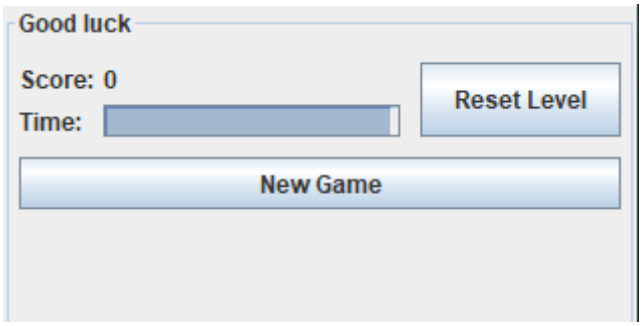
```
        System.out.println("check more line y");
        Point pMinX = p1, pMaxX = p2;
        if (p1.x > p2.x) {
            pMinX = p2;
            pMaxX = p1;
        }
        int x = pMaxX.x + type;
        int col = pMinX.y;
        int rowFinish = pMaxX.x;
        if (type == -1) {
            rowFinish = pMinX.x;
            x = pMinX.x + type;
            col = pMaxX.y;
        }
        if ((matrix[rowFinish][col] == 0|| pMinX.x == pMaxX.x)
                && checkLineY(pMinX.x, pMaxX.x, col)) {
            while (matrix[x][pMinX.y] == 0
                    && matrix[x][pMaxX.y] == 0) {
                if (checkLineX(pMinX.y, pMaxX.y, x)) {
                    System.out.println("TH Y " + type);
                    System.out.println("(" + pMinX.x + "," + pMinX.y + ") -> ("
                            + x + "," + pMinX.y + ") -> (" + x + "," + pMaxX.y
                            + ") -> (" + pMaxX.x + "," + pMaxX.y + ")");
                    return true;
                }
                x += type;
            }
        }
        return false;
    }
```

This is the end of the algorithm part. Next, I will show some visual images and some functions in the game scene after finishing all the sections.

IV. Game Scene:

This is the last section of the report before moving to conclusion part. In this section, I will illustrate some scene of the game. First, when running the game, you will see the scene:





The game scene include: A matrix that used for storing the icons, a score and time bar. There are also two addition buttion called “Reset Level” and “New Game”. The “Reset Level” button let player reset and play the current level again. When you win a level you will be asked whether you want to move to a harder level. This button let you to play the current level again and not have to play again from the beginning. The “New Game” button will reset everything, and let you play at the first level. At the first level, you will play with 8x8 matrix. After you finish a level the matrix will expanded to 10x10, 12x12 matrix respectively.

V. Conclusion:

In conclusion, I will summary all the parts of the report. At the beginning, I have introduced the features, the rule, and the benefits of playing this game. Then, I show some informations about the design of the game. This include the classes of the game, and how these classes are used in order to create the full game. Next, I shown about the algorithm that applied for the game. Last but not least, I have reveal the game scene and some addition button in that scene. Overall, this is a classic game that can help you relax when playing and improve some skills of players.

[*] Code Link: https://github.com/giagia2002123/OOP_Project_Pokemon_Classic_Game