Name: Ngô Triệu Gia Gia _ Student's ID: ITDSIU20034

Project Process Report

How I Create This Game

<u>Class:</u> Object-Oriented Programming

<u>Project Topic:</u> Pokemon Classic Game

I. <u>Introduction:</u>

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.

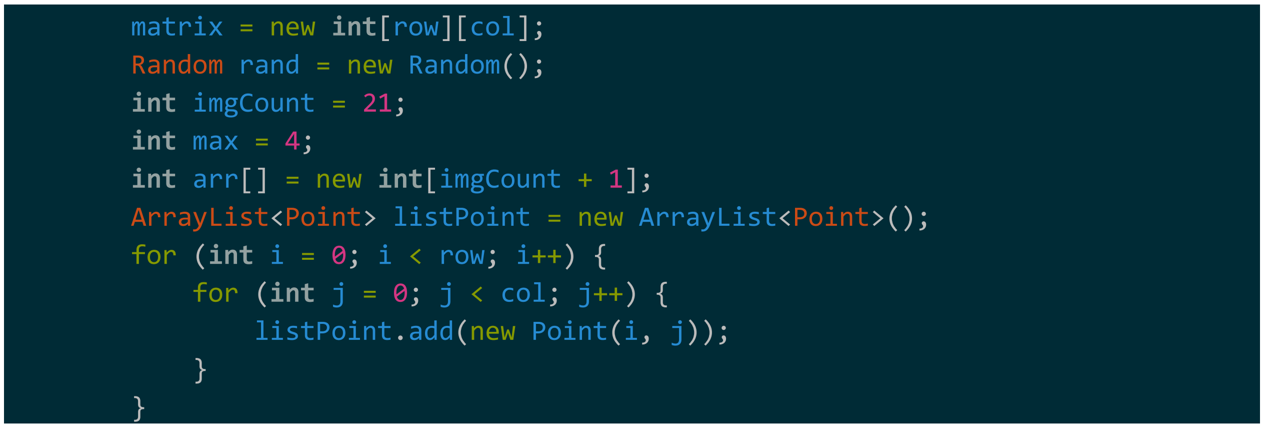
II. <u>Body:</u> Create the game

1. <u>Create basic function:</u>

First and foremost, I will create a project in VSCode called 'POKEMONGAME'. Inside the project folder, I will create two diffirent directories named 'Controller' to store the code of the game and another called 'Icon' to store the pictures in the size of a small icon I used in the game. There are total 21 icons in the 'Icon' folder. Then, let move to the next step.

1.1/ Create simple matrix to store the icons in the game scene:

Let consider all of the icons that appear in the game scene like the numbers that appear in a matrix. In that matrix, all numbers will appear randomly so I will use Random function to create the matrix. Since there are total 21 icons that may appear in the game, the Random function will set to generate random integer number from 1 to 21 (code: **int index = rand.nextInt(imgCount) + 1**). The **imgCount** in this case is equal 21 in default. Next, to set the location to the icons in the created matrix. I will create a **ArrayList<Point> listPoint** and add **Point** to the **listPoint** by using a two for loop (code below):



```
        matrix = new int[row][col];
        Random rand = new Random();
        int imgCount = 21;
        int max = 4;
        int arr[] = new int[imgCount + 1];
        ArrayList<Point> listPoint = new ArrayList<Point>();
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                listPoint.add(new Point(i, j));
            }
        }
```

In addition, as we want the number of similar icons in the game always even, I will create an additional for loop function that run two times to store the random numbers created in the matrix. Code in createMatrix() function:

```
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);
}
```

Note: The above code will be changed in the following sections in order to complete the game function and algorithm.

1.2/ Add icons to the matrix and show up in the game scene:

In this section, I will create and add the icons to the matrix. In this report, I will create a 8x8 matrix. First, I will create a class named ‘ButtonEvent’ that extends ‘Jpanel’ and implements ‘ActionListener’. In order to add icons to the matrix, I will use the code:

```
Image image = new ImageIcon(getClass().getResource(
                "/Icon/" + index + ".png")).getImage();
```

This code allow us to get the icon with name ‘index’ in package Icon (the second directory that I have created earlier).

After getting the icons, I wrote the addArrayButton() function to create the buttons in the table, and setIcon for the created buttons. We need to create enough buttons corresponding to the size of the table, and the icons for each button are retrieved by the getIcon(int index) function with the index value being the corresponding value of the matrix at the button's position in the matrix. In this way, our matrix is now officially a miniature of the Pikachu game, allowing us to handle a lot of things about the game's algorithm in this matrix. The full code of this section will be displayed below:

File: ButtonEvent.java

```
package controller;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```java
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class ButtonEvent extends JPanel implements ActionListener {
    private int row;
    private int col;
    private int bound = 2;
    private int size = 50;
    private JButton[][] btn;
    private Controller controller;
    private Color backGroundColor = Color.lightGray;
    private MainFrame frame;

    public ButtonEvent(MainFrame frame, int row, int col) {
        this.frame = frame;
        this.row = row;
        this.col = col;

        setLayout(new GridLayout(row, col, bound, bound));
        setBackground(backGroundColor);
        setPreferredSize(new Dimension((size + bound) * col, (size + bound)
                * row));
        setBorder(new EmptyBorder(10, 10, 10, 10));
        setAlignmentY(JPanel.CENTER_ALIGNMENT);

        newGame();

    }

    public void newGame() {
        controller = new Controller(this.row, this.col);
        addArrayButton();

    }

    private void addArrayButton() {
        btn = new JButton[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                btn[i][j] = createButton(i + "," + j);
                Icon icon = getIcon(controller.getMatrix()[i][j]);
                btn[i][j].setIcon(icon);
                add(btn[i][j]);
            }
        }
    }

    private Icon getIcon(int index) {
        int width = 48, height = 48;
        Image image = new ImageIcon(getClass().getResource(
                "/icon/" + index + ".png")).getImage();
        Icon icon = new ImageIcon(image.getScaledInstance(width, height,
                image.SCALE_SMOOTH));
        return icon;

    }

    private JButton createButton(String action) {
        JButton btn = new JButton();
        btn.setActionCommand(action);
        btn.setBorder(null);
        btn.addActionListener(this);
        return btn;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
    }
}
```

The createButton(String action) function is using to create events for the icons that being created in the early section.

1.3/ Run the program and display the icons:

First, I will create a new class called “MainFrame” extends ‘JFrame’ implements ‘ActionListener’, ‘Runnable’. Inside the class, I will create two new functions called “createMainPanel()”, “createGraphicsPanel()” respectively. The full code of this file will be reveal below:

<u>File:</u> MainFrame.java

```
package controller;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
```

```java
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class MainFrame extends JFrame implements ActionListener, Runnable {
    private int row = 8;
    private int col = 8;
    private int width = 700;
    private int height = 500;
    private ButtonEvent graphicsPanel;
    private JPanel mainPanel;

    public MainFrame() {
        add(mainPanel = createMainPanel());
        setTitle("Pokemon Game");
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(width, height);
        setLocationRelativeTo(null);
        setVisible(true);

    }

    private JPanel createMainPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(createGraphicsPanel(), BorderLayout.CENTER);
        return panel;
    }

    private JPanel createGraphicsPanel() {
        graphicsPanel = new ButtonEvent(this, row, col);
        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBackground(Color.gray);
        panel.add(graphicsPanel);
        return panel;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
    }


    @Override
    public void run() {
    }
}
```

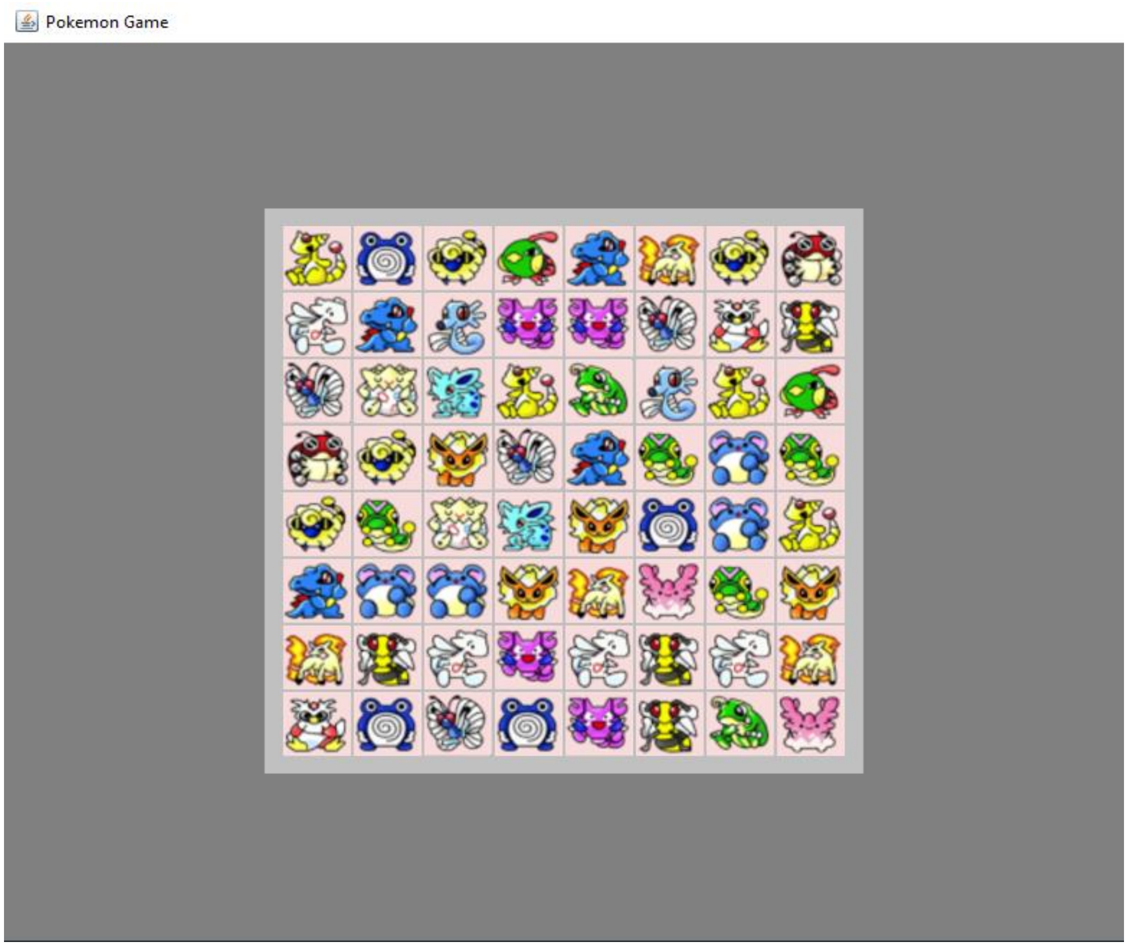
Notes: Since I will create a 8x8 matrix, the row and column will be set equal 8, and scene that use to display the icon will be set as 700x500 (width x height). In addition, the code above will be modified in the next section.

Second, in class "Main", I will write the code to run the program.

File: Main.java

```
package controller;

public class Main {
        MainFrame frame;
        public Main() {
        frame = new MainFrame();
    }
        public static void main(String[] args) {
        new Main();
    }
}
```

Result after running the program like the picture below:



2. Complete interface and basic operations with icons:

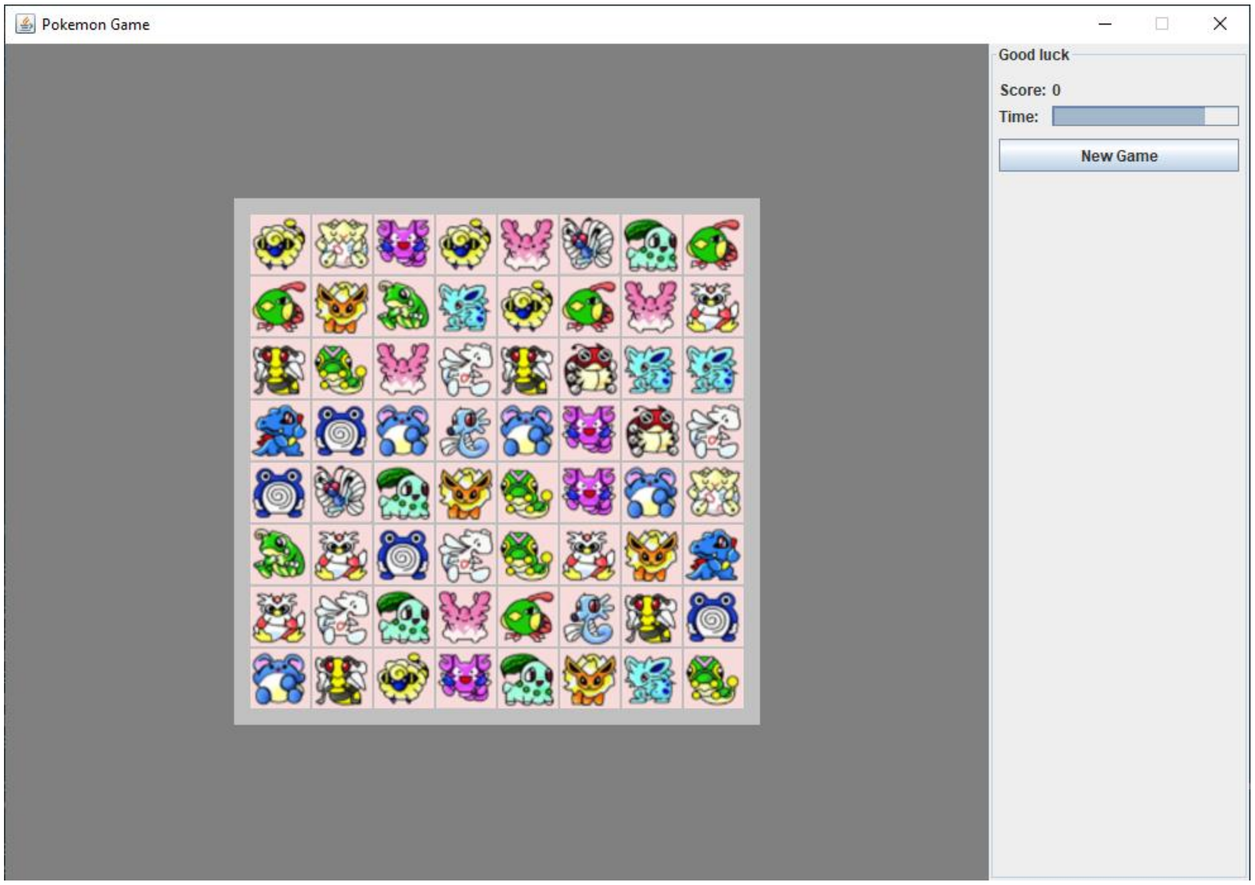
2.1.1/ Complete interface:

In class "MainFrame", I will create additional function called 'createControlPanel()' so as to set Score and Time, and a New Game button to the game. The code of the function will be displayed below:

File: MainFrame.java _ Function: createControlPanel()

```
    private JPanel createControlPanel() {
        // Create JLabel and set lbScore equal 0
        lbScore = new JLabel("0");
        progressTime = new JProgressBar(0, 100);
        progressTime.setValue(100);

        // Create Panel to store Score and Time

        JPanel panelLeft = new JPanel(new GridLayout(2, 1, 5, 5));
        panelLeft.add(new JLabel("Score:"));
        panelLeft.add(new JLabel("Time:"));

        JPanel panelCenter = new JPanel(new GridLayout(2, 1, 5, 5));
        panelCenter.add(lbScore);
        panelCenter.add(progressTime);

        JPanel panelScoreAndTime = new JPanel(new BorderLayout(5, 0));
        panelScoreAndTime.add(panelLeft, BorderLayout.WEST);
        panelScoreAndTime.add(panelCenter, BorderLayout.CENTER);

        // Create Panel include panelScoreAndTime and New Game button
        JPanel panelControl = new JPanel(new BorderLayout(10, 10));
        panelControl.setBorder(new EmptyBorder(10, 3, 5, 3));
        panelControl.add(panelScoreAndTime, BorderLayout.CENTER);
        panelControl.add(btnNewGame = createButton("New Game"),
                BorderLayout.PAGE_END);


        // Set BorderLayout to display panelControl
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(new TitledBorder("Good luck"));
        panel.add(panelControl, BorderLayout.PAGE_START);

        return panel;
    }
```

After, I will generate this function in createMainPanel(), then get the result:



2.1.2/ Complete interface's functions:

In this section, I will write a function for 'New Game' button called "newGame()" and another function called "showDialogNewGame()" to apply the "newGame()" function when doing some actions (Pause, Resume, Start a New Game). To begin with, lets write these two functions. The code will be shown below:

<u>File:</u> MainFrame.java _ Function: newGame() and showDialogNewGame()

```
    public void newGame() {
        time = MAX_TIME;
        graphicsPanel.removeAll();
        mainPanel.add(createGraphicsPanel(), BorderLayout.CENTER);
        mainPanel.validate();
        mainPanel.setVisible(true);
        lbScore.setText("0");
    }

    public void showDialogNewGame(String message, String title, int t) {
        pause = true;
        resume = false;

        int select = JOptionPane.showOptionDialog(null, message, title,
        JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null,
                    null, null);
        if (select == 0) {
            pause = false;
            newGame();
        } else {
            if(t==1) {
                System.exit(0);
            } else {
                resume = true;
            }
        }
    }
```

When we click the ‘New Game’ button the Time will stop (pause) and if we choose to start a new game the time and the scene will be reset. Else we will resume to the game. Next, I will complete this section by Override the “actionPerformed(ActionEvent e)” and “run()”. After that, in order to count the time of the game, I will create a new class called “Time” in the Main.java file.

<u>File:</u> MainFrame.java _ Function: actionPerfomed(ActionEvent e) and run()

```
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == btnNewGame) {
            showDialogNewGame("Your game hasn't done yet. Do you want to start a
new game ?", "Warning", 0);
        }
    }


    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            progressTime.setValue((int) ((double) time / MAX_TIME * 100));
        }
    }
```

<u>File:</u> Main.java _ Class: Time _ Function: Main()

```
    MainFrame frame;
    class Time extends Thread {
        public void run() {
            while (true) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                if(frame.isPause()) {
                    if(frame.isResume()) {
                        frame.time--;
                    }
                } else {
                    frame.time--;
                }
                if (frame.time == 0) {
                    frame.showDialogNewGame("Time up !\nDo you want to play again
?", "Lose", 1);
                }
            }
        }
    }

    public Main() {
        frame = new MainFrame();
        Time time = new Time();
        time.start();
        new Thread(frame).start();
    }
```

2.2/ Basic operations with icons:

In this part, I will do some basic function to handling the icons which is find two similar icons and delete them. Since I have not write the algorithm for the game yet, these functions will just delete the icons regardless of their position.

To begin with, I will created an additional class called "PointLine" in the package controller. As in part 1, I consider a Point with 2 coordinates (x, y) for each icon, and this class PointLine is created to define the connection of 2 Icons together. Specifically, a PointLine will include two Points p1 and p2, respectively.

<u>File:</u> PointLine.java

```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;
    }
}
```

Next, in class Controller, I will create a new “checkTwoPoint(Point p1, Point p2)” function. This function is using to check whether the two icons the same.

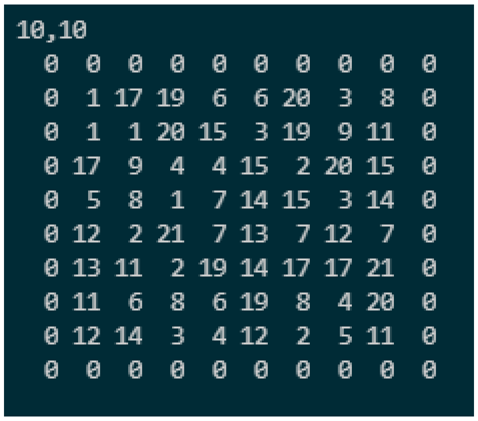
<u>File:</u> Controller.java _ Function: checkTwoPoint(Point p1, Point p2)

```java
    public PointLine checkTwoPoint(Point p1, Point p2) {
        if (!p1.equals(p2) && matrix[p1.x][p1.y] == matrix[p2.x][p2.y]) {
            return new PointLine(p1, p2);
        }
        return null;
    }
```

3. <u>Complete the game’s algorithm:</u>

3.1/ Modify the matrix:

First of all, in order to create the algorithm for the game, I need to created matrix that surronded by the zero number like the picture below:



```
10,10
  0  0  0  0  0  0  0  0  0  0
  0  1 17 19  6  6 20  3  8  0
  0  1  1 20 15  3 19  9 11  0
  0 17  9  4  4 15  2 20 15  0
  0  5  8  1  7 14 15  3 14  0
  0 12  2 21  7 13  7 12  7  0
  0 13 11  2 19 14 17 17 21  0
  0 11  6  8  6 19  8  4 20  0
  0 12 14  3  4 12  2  5 11  0
  0  0  0  0  0  0  0  0  0  0
```

The earlier 8x8 matrix will be surrounded by zero number, so it will become a 10x10 matrix. To do this, in ButtonEvent constructor, I will increase the ‘col’ and ‘row’ by a number of 2. That change leads to the next change in the assignment of values to icons as well as elements in the matrix, instead of starting from 0 and ending with row - 1 and col - 1, I only consider the cells in positions from 1 to row - 2 and from 1 to col - 2, and assign values to the elements at the border of the matrix with a value of 0. Then, the old matrix will automatically be wrap with a border of zeros. The changing functions would be createMatrix() in class Controller and addArrayButton() in class ButtonEvent. The code of the three fuctions will be displayed below:

<u>File:</u> ButtonEvent.java

```
    public ButtonEvent(MainFrame frame, int row, int col) {
        ...
        this.row = row + 2;
        this.col = col + 2;
        ...
    }
```

```
    private void addArrayButton() {
        btn = new JButton[row][col];
        for (int i = 1; i < row - 1; i++) {
            for (int j = 1; j < col - 1; j++) {
                ...
            }
        }
    }
```

<u>File:</u> Controller.java _ Function: createMatrix() (after changing)

```
    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);
    }
```

3.2/ Apply algorithm for the game:

In this part, I will apply some algorithms that find the path between two similar icons. There are three diffirent algorithms that I use in this game.

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.

3.2.1/ 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:

<u>File:</u> Controller.java _ checkLineX() and checkLineY() functions:

```java
    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;
    }
```

3.2.2/ 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:

<u>File:</u> 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 + ")");
```

```java
                // 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;
    }
```

3.2.3/ 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:

<u>File:</u> 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.

III. Conclusion:

In conclusion, I have shown all the sections that I done in order to create the game. After doing all the contents, now, you will have a relax game that you can play everytime that you like.

You can get the full code at the link:

[*]Github: Url: https://github.com/giagia2002123/OOP_Project_Pokemon_Classic_Game