INTERNATIONAL UNIVERSITY
VIETNAM NATIONAL UNIVERSITY

ALGORITHMS & DATA STRUCTURE

MINESWEEPER GAME

ĐÀO HUỲNH THIÊN LONG	ITITIU19026
LA TRÍ NGUYÊN	ITITIU19036
ĐỖ ĐÔNG QUÂN	ITITIU19043
VÕ HUỲNH HUY THỊNH	ITITIU19043

Table of Contents

I. INTRODUCTION

- This project was designed based on Algorithms & Data Structure method by Java language. This method has solved some problems occurred in the building process using algorithms and structure method:

  - The code is clear, concise and easy to understand.
  - The project is a unified logic system of related classes.
  - Each class has methods which take on different behaviors.
  - The ability of reusing the resources.

- The purpose of this project is to design a basic game that was built on the foundation of data and algorithms structure. Our group decided to modify a game named “MineSweeper – DSA_Project”, following the principle of this method. Minesweeper is a game which requires the user to open all cells on the board in order that you won’t touch unrevealed bombs under cells. This game will clearly show the course properties, the combinations of classes and the relationships between classes and objects.

- Being inspired by this famous game, our group choose to create a special version of MineSweeper by applying the knowledge that we have learnt from the course ADS. Furthermore, we want to challenge and improve ourselves on building a software that runs smoothly. In the process of creating, we have met many difficulties that force us to practice our soft skills, such as teamwork, problem solving, information finding and selecting. Through this project, we have found ourselves getting better and having a deeper understanding of the algorithms and data theories.

II. PROPERTIES OF MINESWEEPER

1. **Rules**

   - The board is divided into cells, with mines randomly distributed. To win, you need to open all the cells. The number on a cell shows the number of mines adjacent to it. Using this information, you can determine unrevealed cells that are safe, as well as cells that contain mines. Cells suspected of being mines can be marked with a flag using the right mouse button.

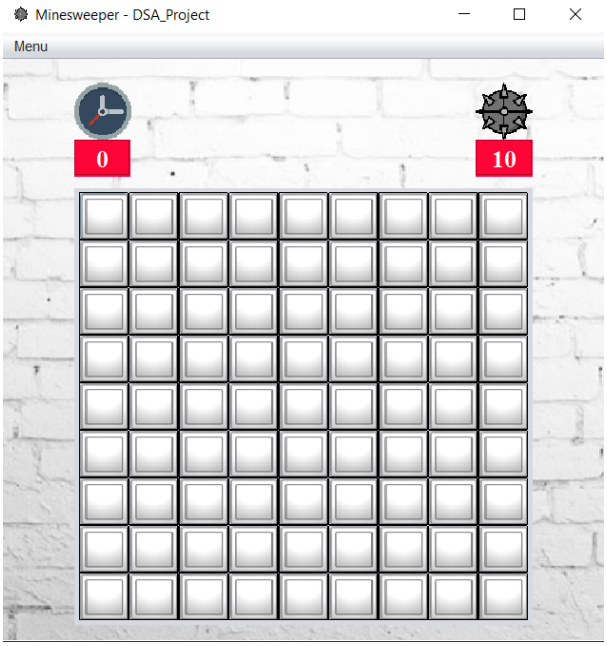
2. **Functions**

   - Our minesweeper game will be played according to the basic rules but there will be some more open functions that attract players:

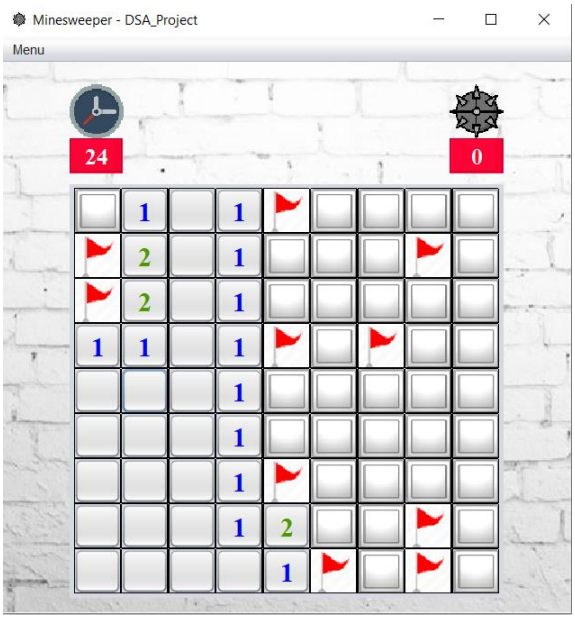
   - You can mark the empty box that you think an unrevealed bomb by right-clicking and “Flag” will be displayed.
   - If you click on any cell it will have 2 directions displayed:
   - You click on the cell with the number, it will show the number of bombs around with the number in that cell.
   - Unfortunately, you click on the bomb box, the game will show you 10 red cells which have bombs under and will automatically save the result including time and your performance on the leaderboard.
   - If you click an empty cell, the system will display a wide hint until there are cells with numbers for you to guess.
   - A special thing in our game project is that the system will total up your performance parameters. If you click on the "Menu" → "Detail" the system will give a table of data of players who have been saved so far such as:
     - Winning streak
     - Losing streak
     - Number of games played
     - Number of wins
     - Win Percentage
     - Current Streak
   - In this section if you click "Reset" it will delete all your material to the beginning.

III. IMPLEMENTATIONS

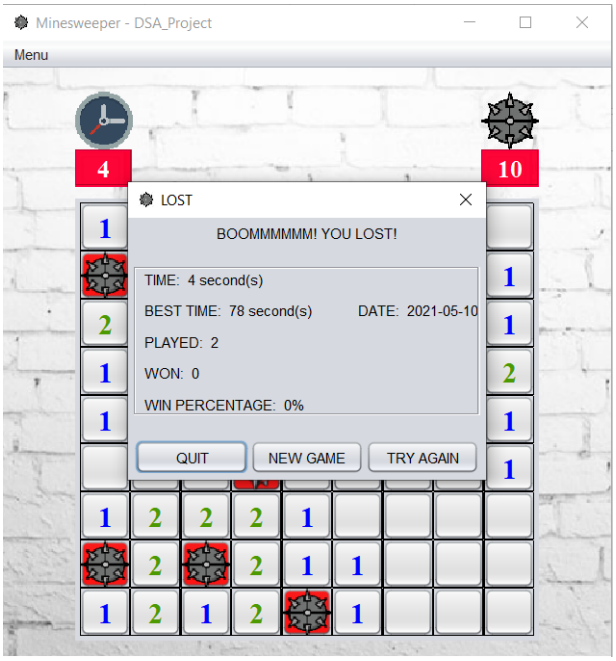
- On the game screen, system will display the number of bombs to guess and will decrease if you mark by flag on the game board.
- The time it takes to count the playing time of 1 game is saved in the system.
- Board game is a 9x9 two-dimensional array.



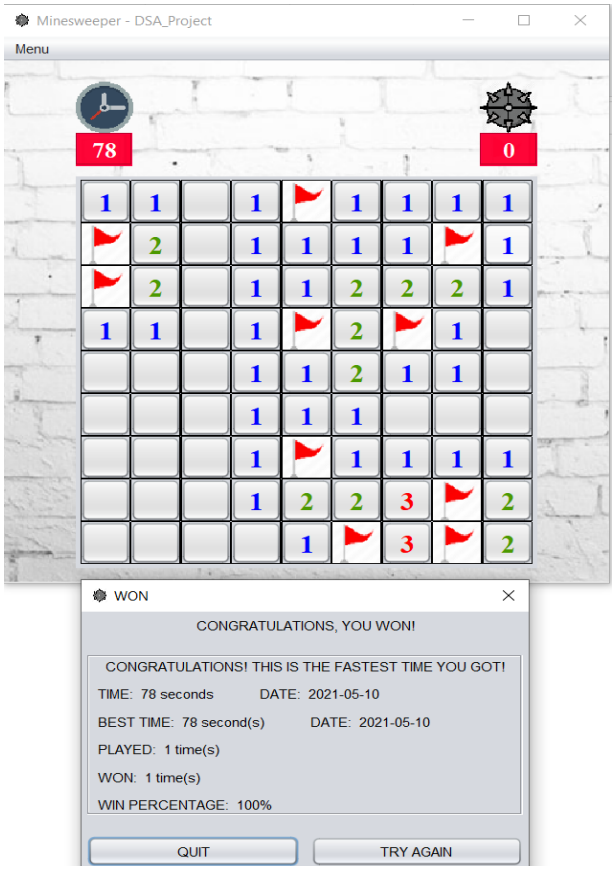
- When you want to mark a bomb, right-click on the cell then the system will display a flag on the selected cell and reduce the number of bombs to guess by 1.



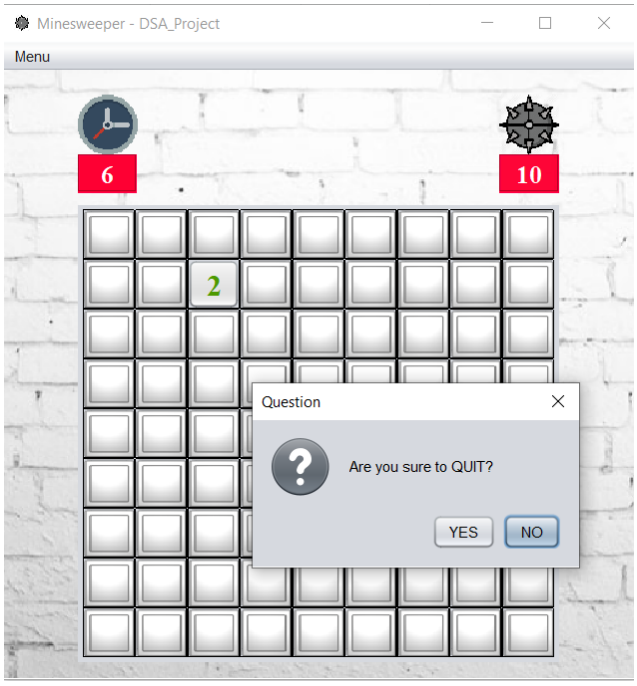
- Unluckily, when you touch an unrevealed bomb, you will lose and the system will display a message board of parameters you have played and ask if you want to play a new game. It will also automatically count the increased number of games played if you click “Try again” and will reset all your previous details when you click “New game”.



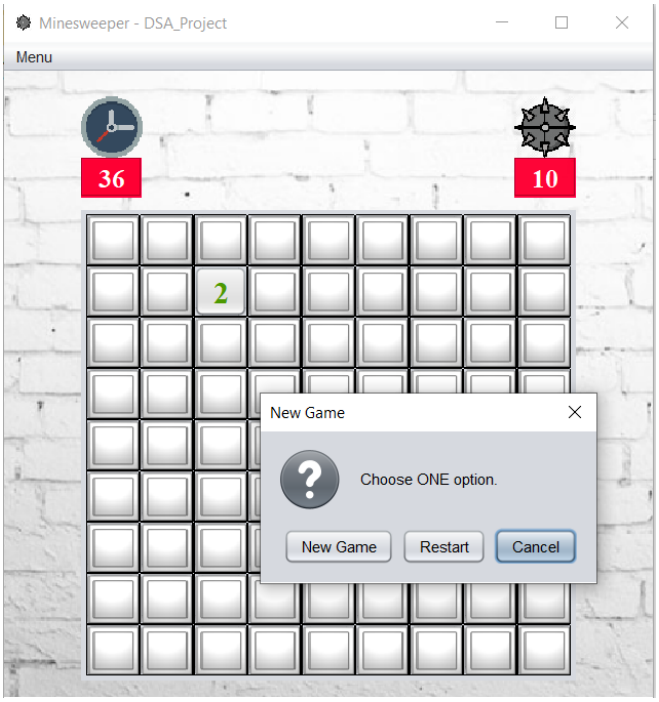
- Congratulations message will be displayed when you win along with invitation asking you to continue playing with "Try again" or turn off the game with "Quit". In addition, your victory information will include "Best Time", "Date", and "Win Percentage".



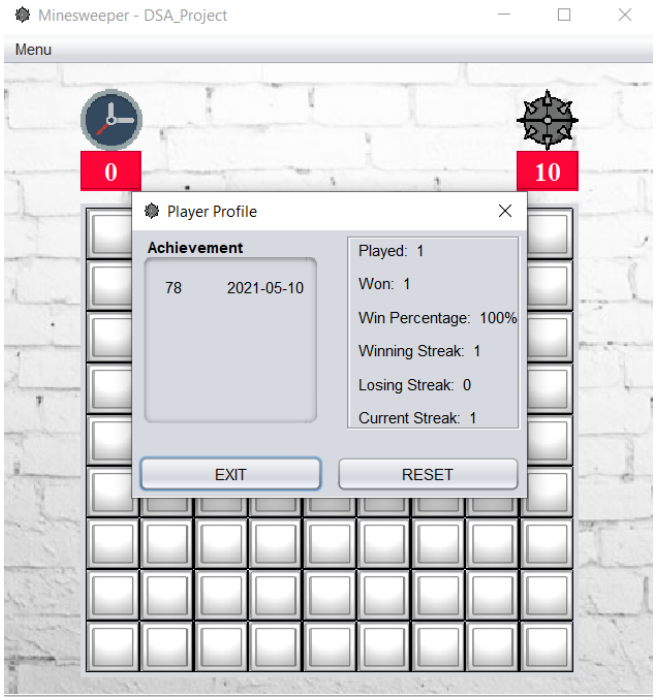
- You can exit the game at any time, the system will confirm again and will not save the game information you have played.



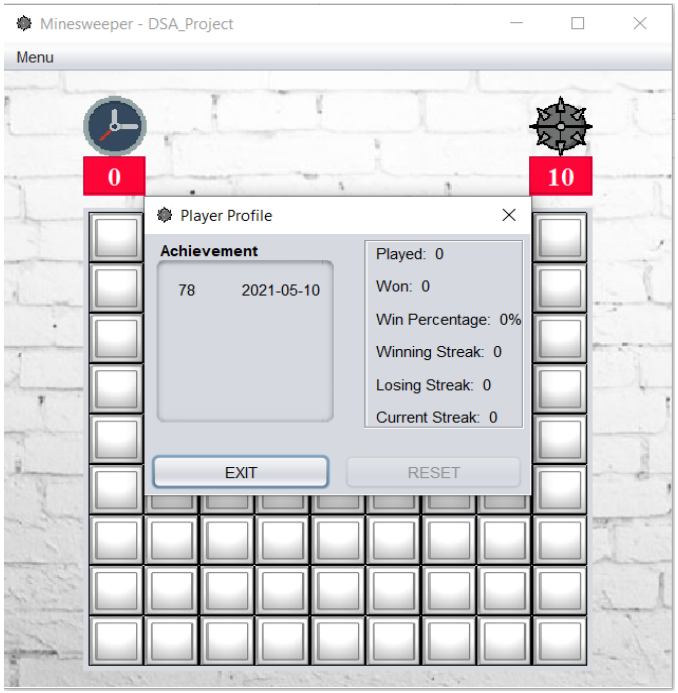
- You can have 2 options: "New Game" to play a new game or "Restart" to play again from the beginning if you choose "Menu" --> "New game".



- In “Menu” have another option “Player profile” to see statistics of you have played. System will show you statistics about:
    - Winning streak
    - Losing streak
    - Number of games played
    - Number of wins
    - Win Percentage
    - Current Streak



- And special thing that show you information about game you won with time and date. You also can clear all statistics with “Reset” to recount all previous information when clicking “Reset”.



IV. DESIGN OF PROJECT

1. Creation board game, set bomb and hint numbers:

- To create a game board, we use a 9x9 two-dimension array and set 81 empty cells for that array.
- We put up 10 bombs randomly within the boundaries of the board and set up the number cells around the bombs that were placed.

```
public void createBoard() {
    // Create a new board
    int mines = 10;

    int r = 9;
    int c = 9;

    this.board = new Board(mines, r, c);

}
```

```
public Board(int numberOfMines, int r, int c)
{
    this.rows = r;
    this.cols = c;
    this.numberOfMines = numberOfMines;

    cells = new Cell[rows][cols];

    //Step 1: First create a board with empty Cells
    createEmptyCells();

    //Step 2: Then set mines randomly at cells
    setMines();

    //Step 3: Then set the number of surrounding mines("neighbours") at each cell
    setSurroundingMinesNumber();
}
```

```java
//STEP 1//
public void createEmptyCells()
{
    for (int x = 0; x < cols; x++)
    {
        for (int y = 0; y < rows; y++)
        {
            cells[x][y] = new Cell();
        }
    }
}
```

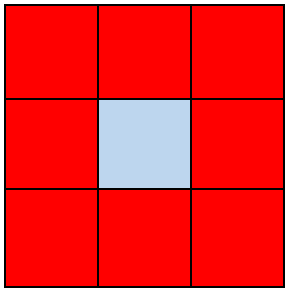
```java
//STEP 2//
public void setMines()
{
    int x,y;
    boolean hasMine;
    int currentMines = 0;

    while (currentMines != numberOfMines)
    {
        // Generate a random x coordinate (between 0 and cols)
        x = (int)Math.floor(Math.random() * cols);

        // Generate a random y coordinate (between 0 and rows)
        y = (int)Math.floor(Math.random() * rows);

        hasMine = cells[x][y].getMine();

        if(!hasMine)
        {
            cells[x][y].setMine(true);
            currentMines++;
        }
    }
}
```

```
//STEP 3//
public void setSurroundingMinesNumber()
{
    for(int x = 0 ; x < cols ; x++)
    {
        for(int y = 0 ; y < rows ; y++)
        {
            cells[x][y].setSurroundingMines(calculateNeighbours(x,y));
        }
    }
}
//------------------------------------------------------------------------//
```

2. **Set cell icons**

```
private static Icon resizeIcon(ImageIcon icon, int resizedWidth, int resizedHeight)
{
    Image img = icon.getImage();
    Image resizedImage = img.getScaledInstance(resizedWidth, resizedHeight,  java.awt.Image.SCALE_SMOOTH);
    return new ImageIcon(resizedImage);
}

public void setIcons()
{
    //--------------------Set Icons----------------------------//

    int bOffset = buttons[0][1].getInsets().left;
    int bWidth = buttons[0][1].getWidth();
    int bHeight = buttons[0][1].getHeight();

    ImageIcon d;

        URL url;
        d = new ImageIcon(getClass().getResource( name: "/resources/images/redmine.png"));
        redMine = resizeIcon(d,  resizedWidth: bWidth - bOffset,  resizedHeight: bHeight - bOffset);

        d = new ImageIcon(getClass().getResource( name: "/resources/images/mine1.png"));
        mine = resizeIcon(d,  resizedWidth: bWidth - bOffset,  resizedHeight: bHeight - bOffset);

        d = new ImageIcon(getClass().getResource( name: "/resources/images/flag1.png"));
        flag = resizeIcon(d,  resizedWidth: bWidth - bOffset,  resizedHeight: bHeight - bOffset);

        d = new ImageIcon(getClass().getResource( name: "/resources/images/tile1.png"));
        tile = resizeIcon(d,  resizedWidth: bWidth - bOffset,  resizedHeight: bHeight - bOffset);

    //-----------------------------------------------------//
```

3. **Set hint numbers around bomb cells**

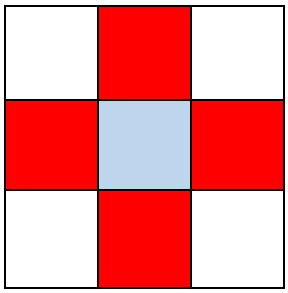


- To set the number cells, the system considers from the cells with bombs (blue cells) and 8 surrounding cells (red cells), when there is a bomb the red cells will automatically count from

1, if the red cells and bombs will not count but count the surrounding cells, the number cells will increase if more bombs are detected than the number of bombs checked.

```
//Calculates the number of surrounding mines ("neighbours")
public int calculateNeighbours(int xCo, int yCo)
{
    int neighbours = 0;

    // Check the neighbours (the columns xCo - 1, xCo, xCo + 1)
    for(int x=makeValidCoordinateX( i: xCo - 1); x<=makeValidCoordinateX( i: xCo + 1); x++)
    {
        // Check the neighbours (the rows yCo - 1, yCo, yCo + 1).
        for(int y=makeValidCoordinateY( i: yCo - 1); y<=makeValidCoordinateY( i: yCo + 1); y++)
        {
            // Skip (xCo, yCo), since that's no neighbour.
            if(x != xCo || y != yCo)
                if(cells[x][y].getMine())   // If the neighbour contains a mine, neighbours++.
                    neighbours++;
        }
    }

    return neighbours;
}
```

4. Calculate when player choose empty cell



- When the player selects an empty cell (for example, a blue cell), the system checks for 4 cells on the edges of the empty cell (red cell). If these red cells is blank cells, the system will continue to consider the rules with that cell, and if the red cell is not the empty cell, it will be considered if the cell is a number cell, it will stop.

```
    * If a player clicks on a zero, all surrounding cells ("neighbours") must revealed.
    * This method is recursive: if a neighbour is also a zero, his neighbours must also be revealed.
    */
    public void findZeroes(int xCo, int yCo) {
        int neighbours;

        Cell cells[][] = board.getCells();
        JButton buttons[][] = gui.getButtons();

        // Columns
        for (int x = board.makeValidCoordinateX(xCo - 1); x <= board.makeValidCoordinateX(xCo + 1); x++) {
            // Rows
            for (int y = board.makeValidCoordinateY(yCo - 1); y <= board.makeValidCoordinateY(yCo + 1); y++) {
                // Only unrevealed cells need to be revealed.
                if (cells[x][y].getContent().equals("")) {
                    // Get the neighbours of the current (neighbouring) cell.
                    neighbours = cells[x][y].getSurroundingMines();

                    // Reveal the neighbours of the current (neighbouring) cell
                    cells[x][y].setContent(Integer.toString(neighbours));

                    if (!cells[x][y].getMine())
                        buttons[x][y].setIcon(null);

                    // Is this (neighbouring) cell a "zero" cell itself?
```

```
                    // Is this (neighbouring) cell a "zero" cell itself?
                    if (neighbours == 0) {
                        // Yes
                        buttons[x][y].setBackground(Color.lightGray);
                        buttons[x][y].setText("");
                        findZeroes(x, y);
                    } else {
                        // No
                        buttons[x][y].setBackground(Color.lightGray);
                        buttons[x][y].setText(Integer.toString(neighbours));
                        gui.setTextColor(buttons[x][y]);
                    }
                }
            }
        }
    }
```

5. Calculate the statistics of player

```
public boolean populate()
{
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;

    try {
        String dbURL = Game.dbPath;

        connection = DriverManager.getConnection(dbURL);
        statement = connection.createStatement();
        resultSet = statement.executeQuery( sql: "SELECT * FROM SCORE");

        while(resultSet.next())
        {
            gamesPlayed = resultSet.getInt( columnLabel: "GAMES_PLAYED");
            gamesWon = resultSet.getInt( columnLabel: "GAMES_WON");

            longestWinningStreak = resultSet.getInt( columnLabel: "LWSTREAK");
            longestLosingStreak = resultSet.getInt( columnLabel: "LLSTREAK");

            currentStreak = resultSet.getInt( columnLabel: "CSTREAK");

            currentWinningStreak = resultSet.getInt( columnLabel: "CWSTREAK");
            currentLosingStreak = resultSet.getInt( columnLabel: "CLSTREAK");
        }

        // cleanup resources, once after processing
        resultSet.close();
        statement.close();
```

```
        //-------------------------LOAD TIMES-------------------//

        statement = connection.createStatement();
        resultSet = statement.executeQuery( sql: "SELECT * FROM TIME");


        while(resultSet.next())
        {
            int time = resultSet.getInt( columnLabel: "TIME_VALUE");
            Date date = resultSet.getDate( columnLabel: "DATE_VALUE");

            bestTimes.add(new Time(time,date));
        }


        // cleanup resources, once after processing
        resultSet.close();
        statement.close();


        // and then finally close connection
        connection.close();

        return true;
    }
    catch(SQLException sqlex)
    {
        sqlex.printStackTrace();
        return false;
    }
}
```

6. **Save the player's statistics**

```
public void save()
{
    Connection connection = null;
    PreparedStatement statement = null;


    try {
        String dbURL = Game.dbPath;

        connection = DriverManager.getConnection(dbURL);


        //----------EMPTY SCORE TABLE------//
        String template = "DELETE FROM SCORE";
        statement = connection.prepareStatement(template);
        statement.executeUpdate();

        //----------EMPTY TIME TABLE------//
        template = "DELETE FROM TIME";
        statement = connection.prepareStatement(template);
        statement.executeUpdate();

        //--------------INSERT DATA INTO SCORE TABLE----------//
        template = "INSERT INTO SCORE (GAMES_PLAYED,GAMES_WON, LWSTREAK, LLSTREAK, CSTREAK, CWSTREAK, CLSTREAK) values (?,?,?,?,?,?,?)";
        statement = connection.prepareStatement(template);

        statement.setInt( parameterIndex: 1, gamesPlayed);
        statement.setInt( parameterIndex: 2, gamesWon);
        statement.setInt( parameterIndex: 3, longestWinningStreak);
        statement.setInt( parameterIndex: 4, longestLosingStreak);
        statement.setInt( parameterIndex: 5, currentStreak);
        statement.setInt( parameterIndex: 6, currentWinningStreak);
        statement.setInt( parameterIndex: 7, currentLosingStreak);
```

```
        statement.executeUpdate();

        //-------------------INSERT DATA INTO TIME TABLE-----------//
        template = "INSERT INTO TIME (TIME_VALUE, DATE_VALUE) values (?,?)";
        statement = connection.prepareStatement(template);


        for (int i = 0; i < bestTimes.size(); i++)
        {
            statement.setInt( parameterIndex: 1, bestTimes.get(i).getTimeValue());
            statement.setDate( parameterIndex: 2, bestTimes.get(i).getDateValue());

            statement.executeUpdate();
        }

        //------------------------------------------------------------//

        statement.close();

        // and then finally close connection
        connection.close();
    }
    catch(SQLException sqlex)
    {
        sqlex.printStackTrace();
    }

}
```

7. Check each clicked cell

```
private void showAll() {
    String cellSolution;

    Cell cells[][] = board.getCells();
    JButton buttons[][] = gui.getButtons();

    for (int x = 0; x < board.getCols(); x++) {
        for (int y = 0; y < board.getRows(); y++) {
            cellSolution = cells[x][y].getContent();

            // Check unrevealed cells
            if (cellSolution.equals("")) {
                buttons[x][y].setIcon(null);

                // Get Neighbours
                cellSolution = Integer.toString(cells[x][y].getSurroundingMines());

                // Check mines
                if (cells[x][y].getMine()) {
                      cellSolution = "M";
                    cellSolution.equals("M");

                    //mine
                    buttons[x][y].setIcon(gui.getIconMine());
                    buttons[x][y].setBackground(Color.RED);
                    gui.setTextColor(buttons[x][y]);
                } else {
                    if (cellSolution.equals("0")) {
                        buttons[x][y].setText("");
                        buttons[x][y].setBackground(Color.LIGHT_GRAY);
                    } else {
                        buttons[x][y].setBackground(Color.LIGHT_GRAY);
                            buttons[x][y].setText(cellSolution);
                            gui.setTextColor(buttons[x][y]);
                        }
                    }
                }

                // This cell is already flagged!
                else if (cellSolution.equals("F")) {
                    // Is it correctly flagged?
                    if (!cells[x][y].getMine()) {
                        buttons[x][y].setBackground(Color.blue);
                    } else
                        buttons[x][y].setBackground(Color.green);
                }

            }
        }
    }
```

8. Check neighbours

```java
public void findZeroes(int xCo, int yCo) {
    int neighbours;

    Cell cells[][] = board.getCells();
    JButton buttons[][] = gui.getButtons();

    // Columns
    for (int x = board.makeValidCoordinateX( xCo - 1); x <= board.makeValidCoordinateX( xCo + 1); x++) {
        // Rows
        for (int y = board.makeValidCoordinateY( yCo - 1); y <= board.makeValidCoordinateY( yCo + 1); y++) {
            // Only unrevealed cells need to be revealed.
            if (cells[x][y].getContent().equals("")) {
                // Get the neighbours of the current (neighbouring) cell.
                neighbours = cells[x][y].getSurroundingMines();

                // Reveal the neighbours of the current (neighbouring) cell
                cells[x][y].setContent(Integer.toString(neighbours));

                if (!cells[x][y].getMine())
                    buttons[x][y].setIcon(null);

                // Is this (neighbouring) cell a "zero" cell itself?
                if (neighbours == 0) {
                    // Yes
                    buttons[x][y].setBackground(Color.lightGray);
                    buttons[x][y].setText("");
                    findZeroes(x, y);
                } else {
                    // No
                    buttons[x][y].setBackground(Color.lightGray);
                    buttons[x][y].setText(Integer.toString(neighbours));
                    gui.setTextColor(buttons[x][y]);
                }
            }
```

9. Check game's solution

```java
public boolean isFinished() {
    boolean isFinished = true;

    String cellSolution;

    Cell cells[][] = board.getCells();

    for (int x = 0; x < board.getCols(); x++) {
        for (int y = 0; y < board.getRows(); y++) {
            // If a game is solved, the content of each Cell should match the value of its surrounding mines
            cellSolution = Integer.toString(cells[x][y].getSurroundingMines());

            if (cells[x][y].getMine())
                cellSolution = "F";

            // Compare player's board to the solution.
            if (!cells[x][y].getContent().equals(cellSolution)) {
                //If not
                isFinished = false;
                break;
            }
        }
    }

    return isFinished;
}


//Check the game to see if its finished or not
private void checkGame() {
    if (isFinished()) {
        gameWon();
    }
}
```

10. Result

```java
JPanel statistics = new JPanel();

statistics.setLayout(new GridLayout( rows: 6,  cols: 1,  hgap: 0,  vgap: 10));

JLabel gPlayed = new JLabel( text: "  Played:   " + score.getGamesPlayed());
JLabel gWon = new JLabel( text: "  Won:   " + score.getGamesWon());
JLabel gPercentage = new JLabel( text: "  Win Percentage:   " + score.getWinPercentage() + "%");
JLabel lWin = new JLabel( text: "  Winning Streak:   " + score.getLongestWinningStreak());
JLabel lLose = new JLabel( text: "  Losing Streak:   " + score.getLongestLosingStreak());
JLabel currentStreak = new JLabel( text: "  Current Streak:   " + score.getCurrentStreak());


statistics.add(gPlayed);
statistics.add(gWon);
statistics.add(gPercentage);
statistics.add(lWin);
statistics.add(lLose);
statistics.add(currentStreak);

Border loweredetched = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
statistics.setBorder(loweredetched);
```

V. RESULT

- Our project has followed completely the principle of Algorithms & Data Structure knowledge and the rule of MineSweeper. Classes and objects in the system are connected relatively and logically. We also success in manipulating the display to conect the user with program. This game can be control by mouse and keyboard.
- Morever, **some new features** out of the basic rule are added into our game such as the background while playing, some button to pause, stop, refresh in order to make the user interface more attractive.

VI. CONCLUSION

Through this project, we had a chance to study and understand deeper the application of Algorithms & Data Structure in building a program. We also show the systematical and logical relationships between the classes. However, due to the short time of the course, we cannot finish it as its best. It can be assured that our project will be completed in the future.