International University - VNU

School : Information and Technology

DSA PROJECT REPORT
Minesweeper Game

Submitted by:
Group 8

Lecturer:
Tran Thanh Tung

Members:
Nguyễn Đông Hải (ITITWE19011)
Nguyễn Vỹ Bình Nguyên (ITITIU19038)
Đặng Quang Vinh (ITITIU19247)
Ngô Thanh Thế (ITITIU19211)
Nguyễn Trí Nhân (ITITIU19170)

APPENDIX:

REPORT

I/ OVERVIEW OF THE PROJECT AND GITHUB REPOSITORY

In this report, we introduce our remake version of the Minesweeper game. This is a well-known game that we all have on our computer from 1990. We developed this game with the help of Java . We will give you a brief and concise overview of our game.

II/ PRESENTS THE GAME RULES AND THE EXTRA FEATURES

The goal of *Minesweeper* is to uncover all the squares on a grid that do not contain mines without being "blown up" by clicking on a square with a mine underneath. The location of most mines is discovered through a logical process, but some require guessing, usually with a 50-50 chance of being correct. Clicking on the game board will reveal what is hidden underneath the chosen square or squares (a large number of blank squares [bordering 0 mines] may be revealed in one go if they are adjacent to each other). Some squares are blank while others contain numbers (from 1 to 8), with each number being the number of mines adjacent to the uncovered square.

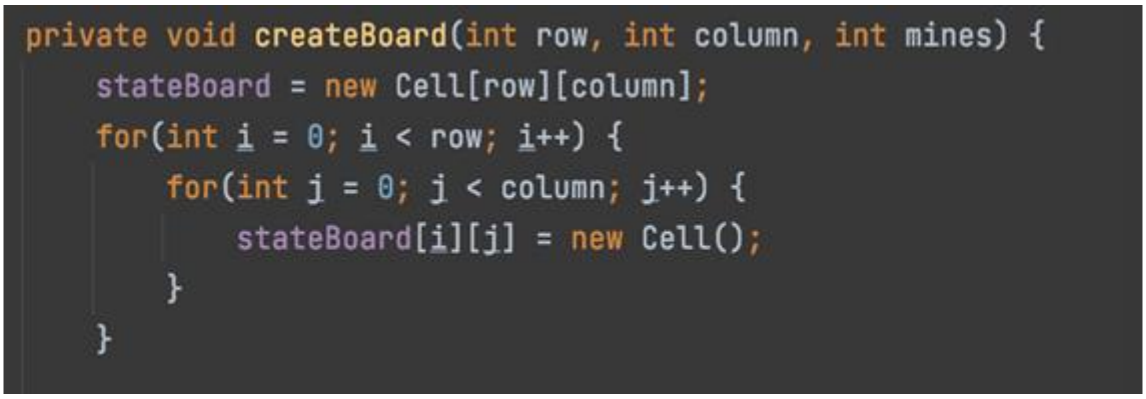
To help the player avoid hitting a mine, the location of a suspected mine can be marked by flagging it with the right mouse button. The game is won once all blank or numbered squares have been uncovered by the player without hitting a mine; any remaining mines not identified by flags are automatically flagged by the computer. However, in the event that a game is lost and the player had mistakenly flagged a safe square, that square will either appear with a red X, or else a red X covering the mine (both denoting the square as safe). The game board comes in three set sizes with a predetermined number of mines: "beginner", "intermediate", and "expert", although a "custom" option is available as well.

Github Link : DSA-project-Minesweeper/DSA (github.com)

III/ EXPLAINS THE ALGORITHMS

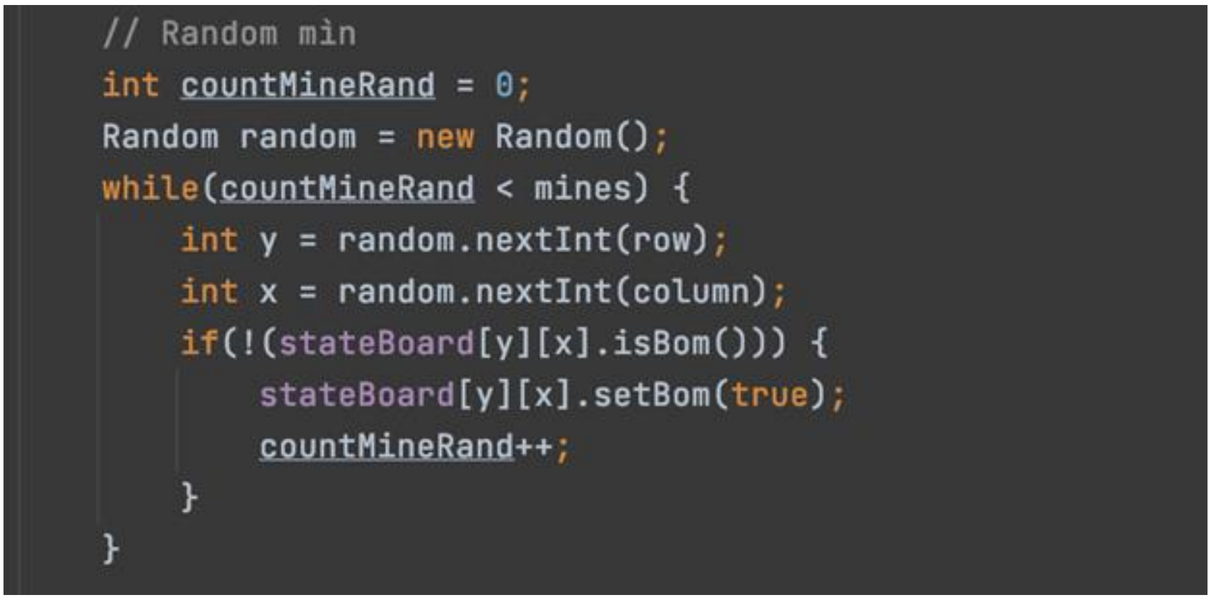
We will create tables and cells, the number of mines required by the user to choose: there are 3 levels (8x8, 16x16, 30x16)
The number of mines and the number of flags are equal.
//function to create the table:
Create a table by creating a two-dimensional array of columns and rows.

Create a table by creating a two-dimensional array of columns and rows.



```java
private void createBoard(int row, int column, int mines) {
    stateBoard = new Cell[row][column];
    for(int i = 0; i < row; i++) {
        for(int j = 0; j < column; j++) {
            stateBoard[i][j] = new Cell();
        }
    }
```

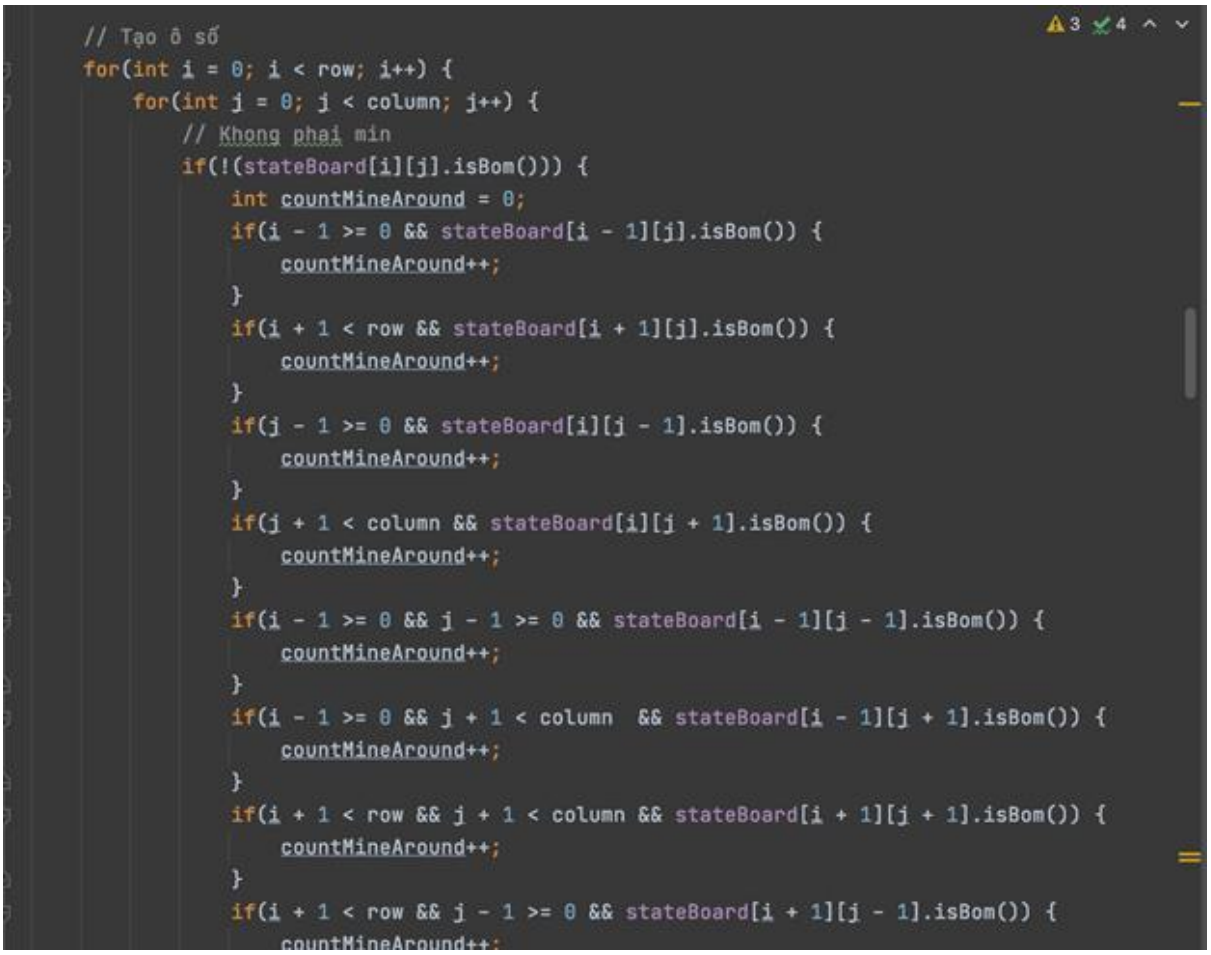
Generate mines by randomizing the x and y coordinates respectively and check that the coordinates are not mines to avoid overlapping, then place mines, reducing the number of mines by one, until the given number of mines is zero .



```java
    // Random min
    int countMineRand = 0;
    Random random = new Random();
    while(countMineRand < mines) {
        int y = random.nextInt(row);
        int x = random.nextInt(column);
        if(!(stateBoard[y][x].isBom())) {
            stateBoard[y][x].setBom(true);
            countMineRand++;
        }
    }
```

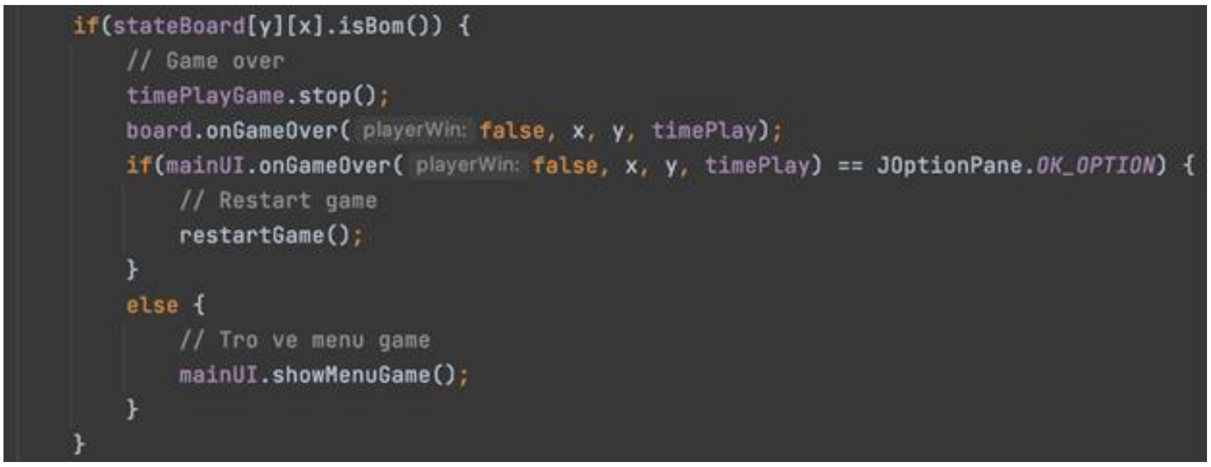
create number boxes:
- 2 for loops to run each cell in the previously created table in turn, then use if to check the bomb 8 surrounding cells of that cell and the cell being considered is not a bomb, the cell being checked is not close to wall (x and y = 0 or both x and y = 0) if there are bombs in 8 around the cell in question, it will automatically count the number of bombs and assign the value

to the cell being checked, and so on until the end of the table.



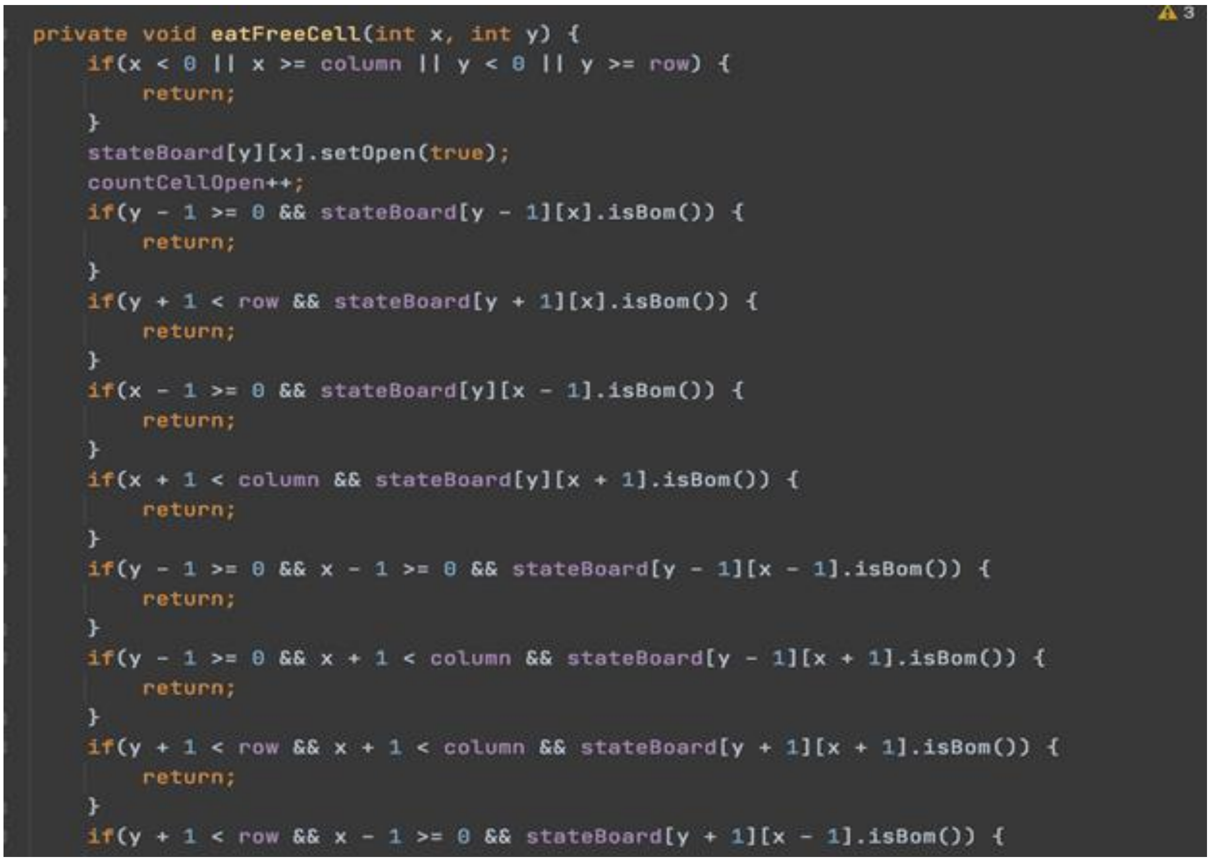
```java
// Tạo ô số
for(int i = 0; i < row; i++) {
    for(int j = 0; j < column; j++) {
        // Khong phai min
        if(!(stateBoard[i][j].isBom())) {
            int countMineAround = 0;
            if(i - 1 >= 0 && stateBoard[i - 1][j].isBom()) {
                countMineAround++;
            }
            if(i + 1 < row && stateBoard[i + 1][j].isBom()) {
                countMineAround++;
            }
            if(j - 1 >= 0 && stateBoard[i][j - 1].isBom()) {
                countMineAround++;
            }
            if(j + 1 < column && stateBoard[i][j + 1].isBom()) {
                countMineAround++;
            }
            if(i - 1 >= 0 && j - 1 >= 0 && stateBoard[i - 1][j - 1].isBom()) {
                countMineAround++;
            }
            if(i - 1 >= 0 && j + 1 < column  && stateBoard[i - 1][j + 1].isBom()) {
                countMineAround++;
            }
            if(i + 1 < row && j + 1 < column && stateBoard[i + 1][j + 1].isBom()) {
                countMineAround++;
            }
            if(i + 1 < row && j - 1 >= 0 && stateBoard[i + 1][j - 1].isBom()) {
                countMineAround++;
```

After creating the cells, bombs, and assigning the value of the number of adjacent bombs, it will be the player's turn, and the time will start to count (unit: seconds). The player uses the left mouse button to open a box, any, and the right mouse button to place the flag, if opening a box is a bomb, the game will end and there will be 2 options to return to the menu or start the game again.



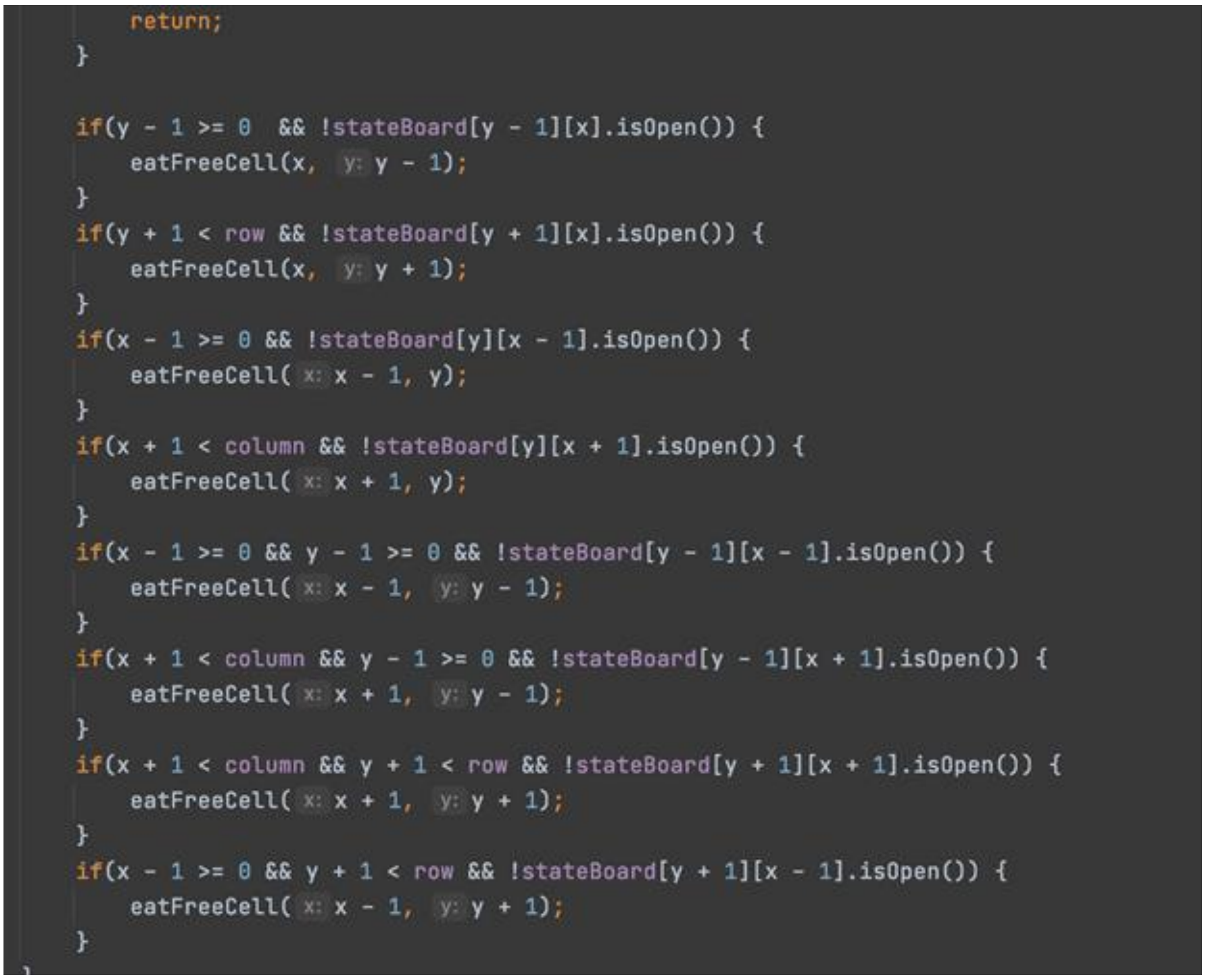
```java
if(stateBoard[y][x].isBom()) {
    // Game over
    timePlayGame.stop();
    board.onGameOver( playerWin: false, x, y, timePlay);
    if(mainUI.onGameOver( playerWin: false, x, y, timePlay) == JOptionPane.OK_OPTION) {
        // Restart game
        restartGame();
    }
    else {
        // Tro ve menu game
        mainUI.showMenuGame();
    }
}
```

If the opened cell is not a bomb, that cell will show the value of the number of bombs in the 8 surrounding cells, and will continue to check the surrounding 8 cells of the 8 surrounding cells (not a bomb) and continue to open the box. output (using recursion - function eatFreeCell(x,y)) and only stops the recursion when it encounters a bomb, and the cell is not close to the wall.



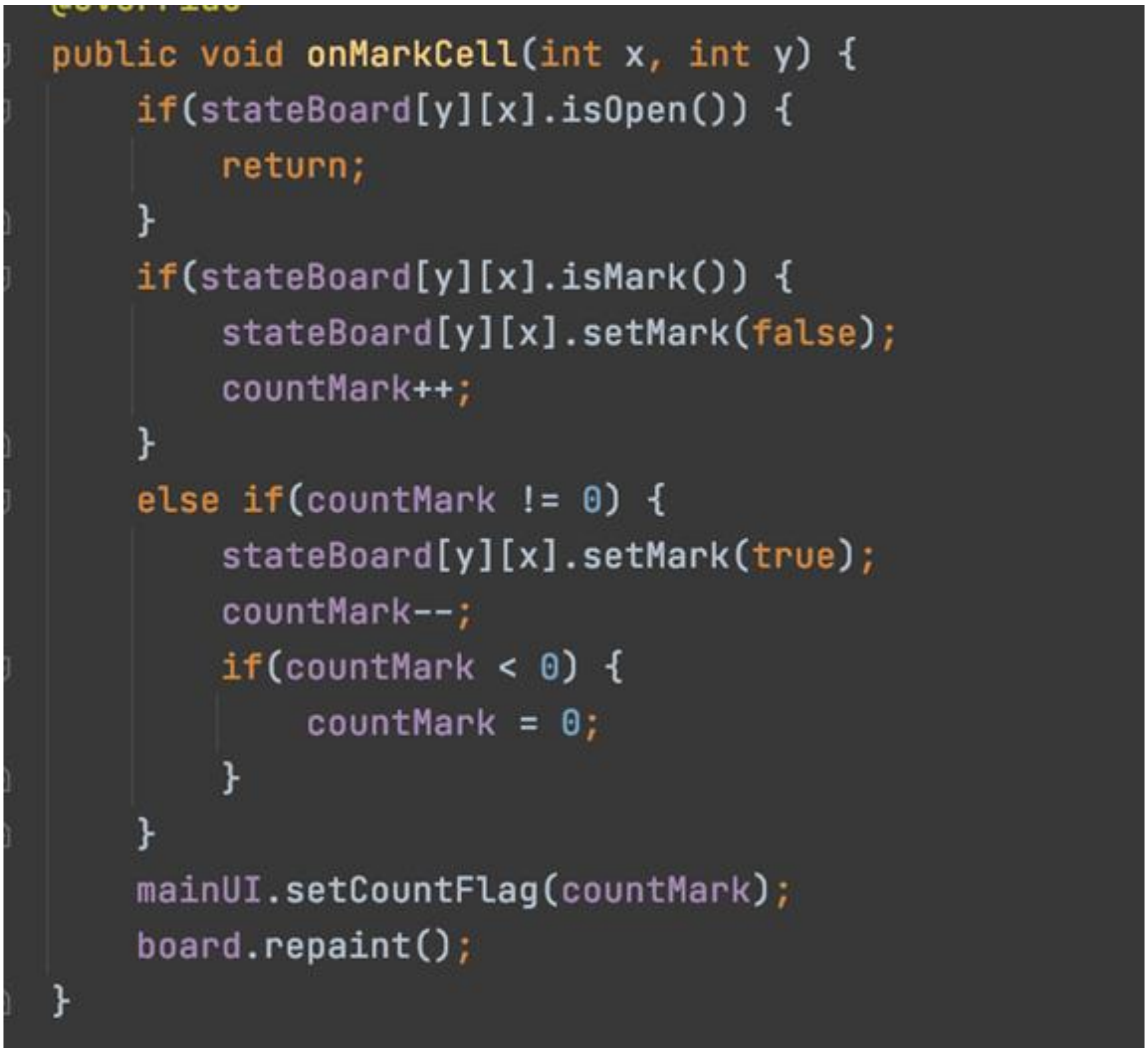
```
private void eatFreeCell(int x, int y) {
    if(x < 0 || x >= column || y < 0 || y >= row) {
        return;
    }
    stateBoard[y][x].setOpen(true);
    countCellOpen++;
    if(y - 1 >= 0 && stateBoard[y - 1][x].isBom()) {
        return;
    }
    if(y + 1 < row && stateBoard[y + 1][x].isBom()) {
        return;
    }
    if(x - 1 >= 0 && stateBoard[y][x - 1].isBom()) {
        return;
    }
    if(x + 1 < column && stateBoard[y][x + 1].isBom()) {
        return;
    }
    if(y - 1 >= 0 && x - 1 >= 0 && stateBoard[y - 1][x - 1].isBom()) {
        return;
    }
    if(y - 1 >= 0 && x + 1 < column && stateBoard[y - 1][x + 1].isBom()) {
        return;
    }
    if(y + 1 < row && x + 1 < column && stateBoard[y + 1][x + 1].isBom()) {
        return;
    }
    if(y + 1 < row && x - 1 >= 0 && stateBoard[y + 1][x - 1].isBom()) {
```

above is to stop the recursion when it encounters a cell with a bomb in the 8 surrounding cells



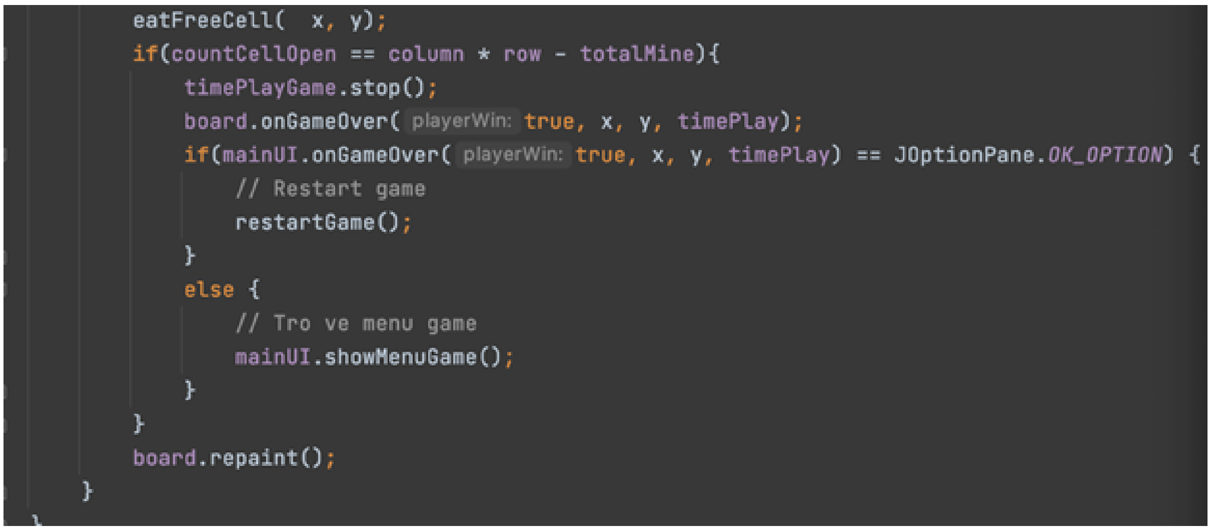
```
        return;
    }

    if(y - 1 >= 0  && !stateBoard[y - 1][x].isOpen()) {
        eatFreeCell(x,  y: y - 1);
    }
    if(y + 1 < row && !stateBoard[y + 1][x].isOpen()) {
        eatFreeCell(x,  y: y + 1);
    }
    if(x - 1 >= 0 && !stateBoard[y][x - 1].isOpen()) {
        eatFreeCell( x: x - 1, y);
    }
    if(x + 1 < column && !stateBoard[y][x + 1].isOpen()) {
        eatFreeCell( x: x + 1, y);
    }
    if(x - 1 >= 0 && y - 1 >= 0 && !stateBoard[y - 1][x - 1].isOpen()) {
        eatFreeCell( x: x - 1,  y: y - 1);
    }
    if(x + 1 < column && y - 1 >= 0 && !stateBoard[y - 1][x + 1].isOpen()) {
        eatFreeCell( x: x + 1,  y: y - 1);
    }
    if(x + 1 < column && y + 1 < row && !stateBoard[y + 1][x + 1].isOpen()) {
        eatFreeCell( x: x + 1,  y: y + 1);
    }
    if(x - 1 >= 0 && y + 1 < row && !stateBoard[y + 1][x - 1].isOpen()) {
        eatFreeCell( x: x - 1,  y: y + 1);
    }
```

The image above is to continue using the function eatFreeCell recursively for the 8 surrounding cells and consider the cells lying on the wall (x or y = 0 and x and y = 0)
The player can use the flag to mark the cells they suspect are bombs, and the number of flags will also be reduced by 1 after each tick.



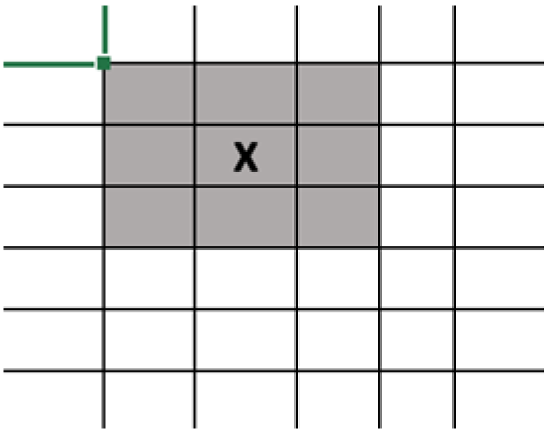
```java
@Override
public void onMarkCell(int x, int y) {
    if(stateBoard[y][x].isOpen()) {
        return;
    }
    if(stateBoard[y][x].isMark()) {
        stateBoard[y][x].setMark(false);
        countMark++;
    }
    else if(countMark != 0) {
        stateBoard[y][x].setMark(true);
        countMark--;
        if(countMark < 0) {
            countMark = 0;
        }
    }
    mainUI.setCountFlag(countMark);
    board.repaint();
}
```

The image above is the function onMarkCell : the function uses if to check a cell with x, y coordinates (unchecked on open cells, if that cell is checked, ignore it, if that cell is not checked, countMark will be checked) decrement by 1 and mark the flag.



```java
            eatFreeCell(  x, y);
            if(countCellOpen == column * row - totalMine){
                timePlayGame.stop();
                board.onGameOver( playerWin: true, x, y, timePlay);
                if(mainUI.onGameOver( playerWin: true, x, y, timePlay) == JOptionPane.OK_OPTION) {
                    // Restart game
                    restartGame();
                }
                else {
                    // Tro ve menu game
                    mainUI.showMenuGame();
                }
            }
            board.repaint();
        }
}
```

The condition to check the winning game is that the number of open cells is equal to the number of cells originally created and minus the number of bombs ===> shows win message, the counter stops, mainUI shows 2 options restart and back to game menu



NOTE: gray is the area of 8 cells around when considering

IV/ EVALUATION

1.Nguyễn Đông Hải (ITITWE19011)

-This guy can be described as Nick Fury. He assembled a team which has many attributes to contribute to this project.

2.Nguyễn Vỹ Bình Nguyên (ITITIU19038)

-The energetic one who always says yes to everything-which is good. Always try his best,that's good enough.

3.Đặng Quang Vinh (ITITIU19247)

-The energetic one who always help everyone to understand the code and algorithm everything-which is good. Always try his best,that's good enough.

4.Ngô Thanh Thế (ITITIU19211)

-A hard-working person, despite not being interested in algorithm and data structure or even building one, he's a big game player. Indeed, he is what he is.

5.Nguyễn Trí Nhân (ITITIU19170)

-The silencer. Honestly, do not ever underestimate the quiet kid, since his idea can be critical and unpredictable. Also, he’s a supportive one too.

Name	**Contribution**
Nguyễn Đông Hải	20%
Nguyễn Vỹ Bình Nguyên	20%
Trần Quang Vinh	20%
Ngô Thanh Thế	20%
Nguyễn Trí Nhân	20%