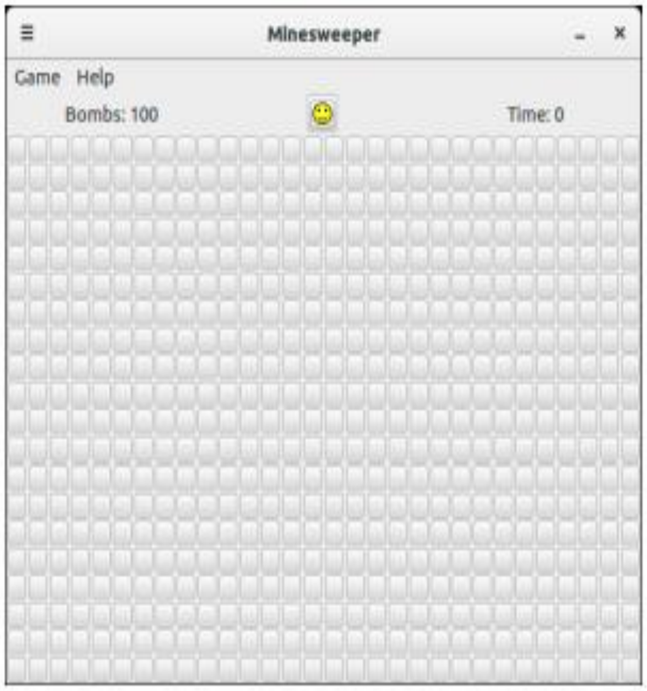
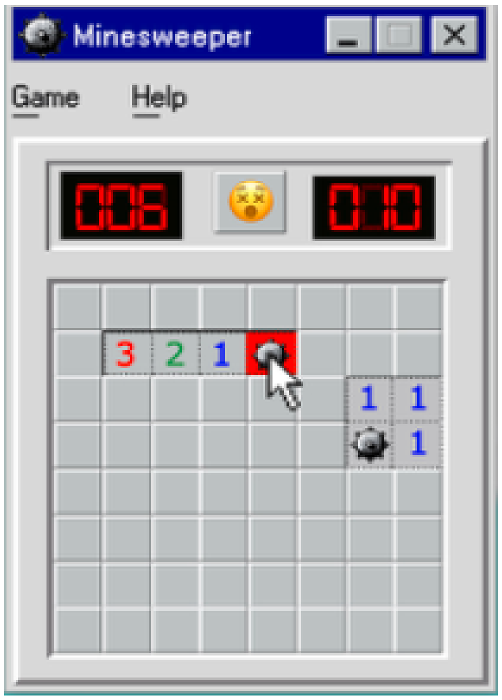
Final Project: Minesweeper

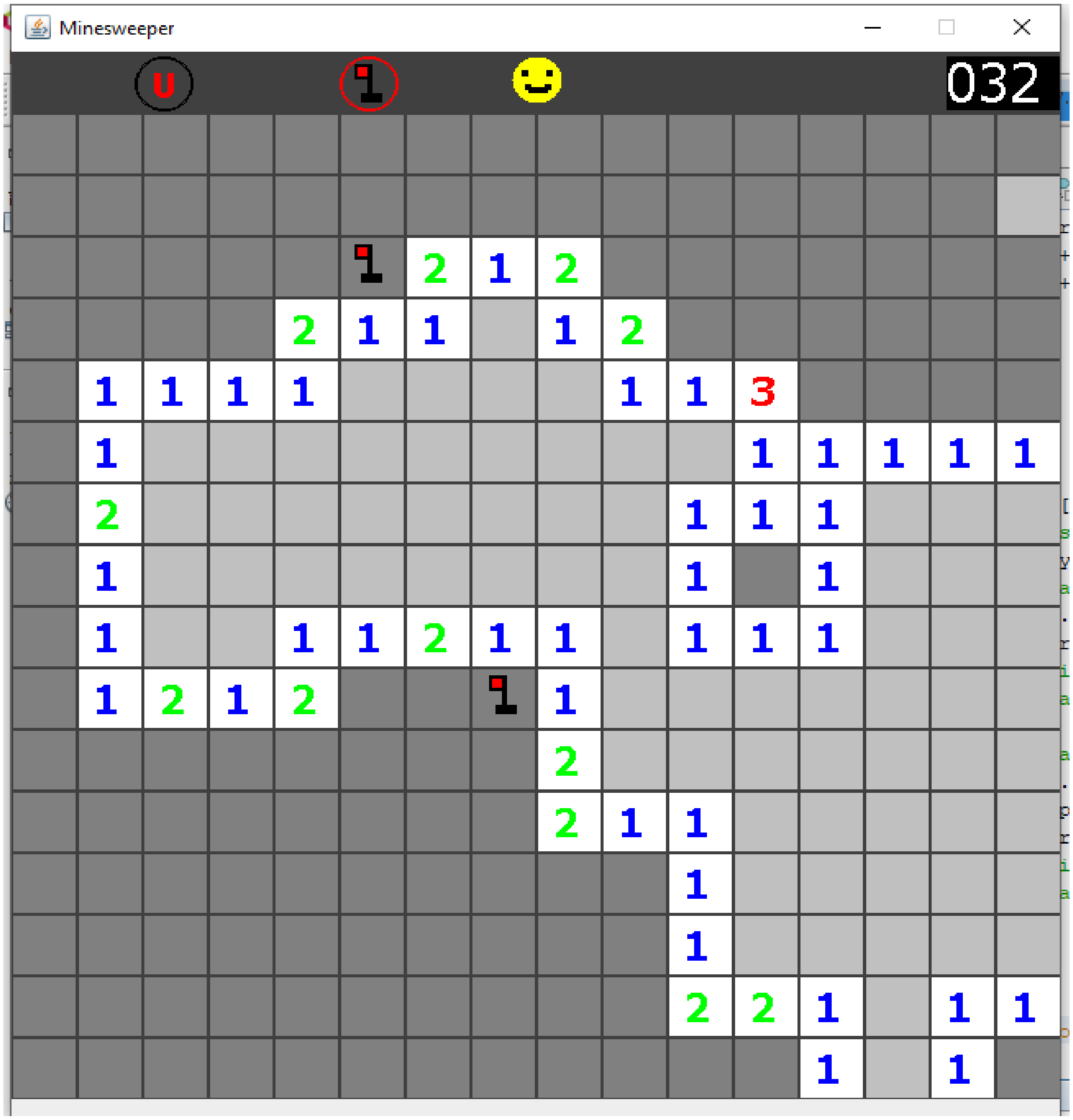
Minesweeper is a popular one- player computer game written by Robert Donner and Curt Johnson which was included in Microsoft Windows in 1991. Playing the game requires logical, arithmetic and probabilistic reasoning based on spatial relationships on the board.

1. The Basic Game

The minesweeper is single-player puzzle video game. At the beginning of the game, a 2D grid of identically looking tiles is presented to the player. The number of aimed tiles is known to the player and it determines the difficulty of the games. The goal of the game is to uncover all the tiles which do not contain a mine. Each turn the player can select one of three actions: to mark a tile as a mine, to unmark a tile, and to uncover the tile If a tile hiding a mine is revealed , the player will lose the game. If all the tiles not containing a mine are revealed, the player will win.







2. Game Console

GAME	MINESWEEPER
Basic Features	- Ability to play and replay game. - Randomly generated mines.

	- Notifices player the number of mines adjacent to a block once it has been selected. - Time counter.
Additional Features	- Buzzer notification when the player wins or loses by pressing the block containing mine. - Create flags to mark doubtful blocks. - Undo features.

3. Variation and Offshort

The game itself opens to many possibilities and, indeed, many variations of it have been already implemented. Here below a list of different versions of the game.

(1) A first alternative is to play not on a rectangular field. Any shape can be used, but tiles are always square.

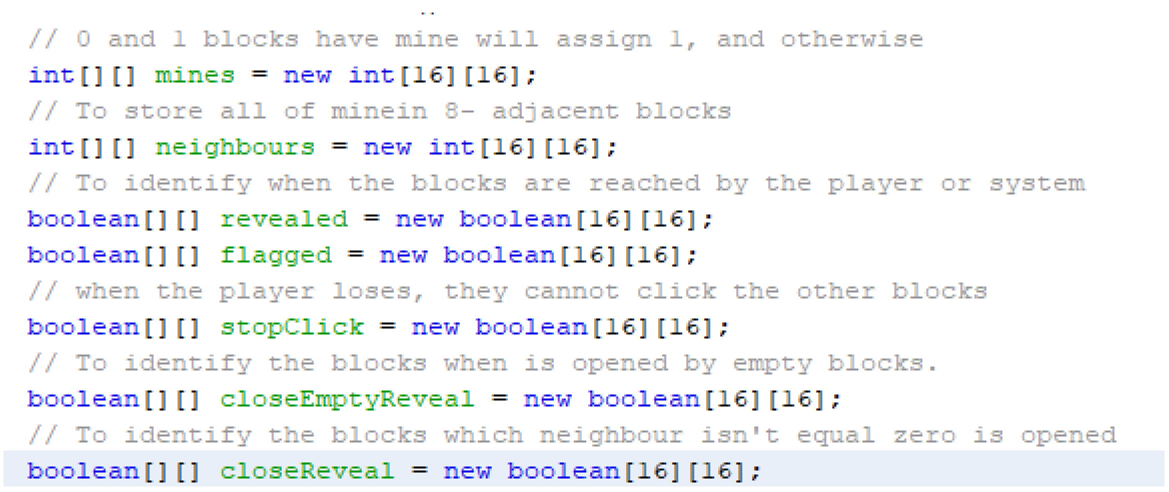
(2) A similar idea is to change the shape of the tile. Triangles or hexagons are the most common alternative shapes. This reduces the maximum number of adjacent tiles.

(3) A more challenging option is to allow more mines to be under the same tile. Even if in principle not needed, usually a maximum number of mines allowed to stay together is fixed. Moreover, the player is commonly told about how many tiles contain one mine, how many contain two, and so on.

(4) Another rule that can be changed is the number of dimensions of the grid. It exists a 3D version of Minesweeper, which implies the possibility to have up to 26 adjacent mines per tile. Of course, the idea to have to safely walk on a mined field is a bit lost, but you could have to swim underwater in a dangerous sea with invisible poison localised here and there. . .

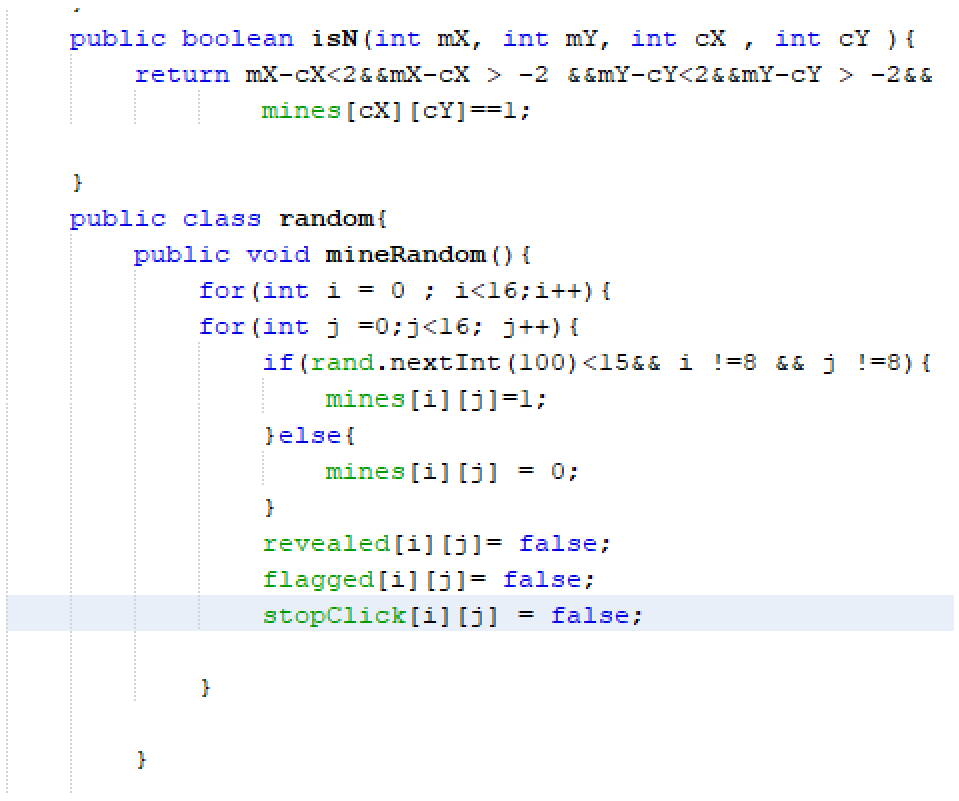
4. Implementing the game.

- The first part of this project is to implement a basic version of the basic game, which is played within the terminal. The idea is to get the grid displayed and to enter the coordinate of the tile we want to reveal. If we neither win nor loose, we get the grid displayed again and we are asked again for new coordinates.
- Next, we built GUI class to handle functions of game. In GUI, we drew an user interface with 16x16 blocks and used two-dimension arrays for storing information of a blocks about mine, neighbours, flagged and revealed.

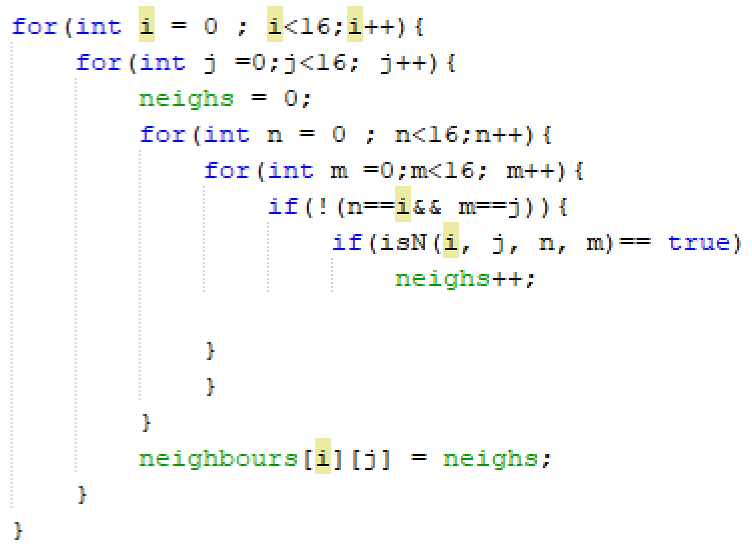


```
// 0 and 1 blocks have mine will assign 1, and otherwise
int[][] mines = new int[16][16];
// To store all of minein 8- adjacent blocks
int[][] neighbours = new int[16][16];
// To identify when the blocks are reached by the player or system
boolean[][] revealed = new boolean[16][16];
boolean[][] flagged = new boolean[16][16];
// when the player loses, they cannot click the other blocks
boolean[][] stopClick = new boolean[16][16];
// To identify the blocks when is opened by empty blocks.
boolean[][] closeEmptyReveal = new boolean[16][16];
// To identify the blocks which neighbour isn't equal zero is opened
boolean[][] closeReveal = new boolean[16][16];
```

- We creates a class Random to genearate the mines randomly. In this class, two for loop were used to check all of blocks. Each block have i and j which represent the position of block. When we have i and j, we use a random function to take a number from 0-100. If this number is less than 15, the mines will be placed in this block. Hence, The number of mines is not the same for each game. And, in this class, we will count the number of mines in 8 adjacent-blocks.

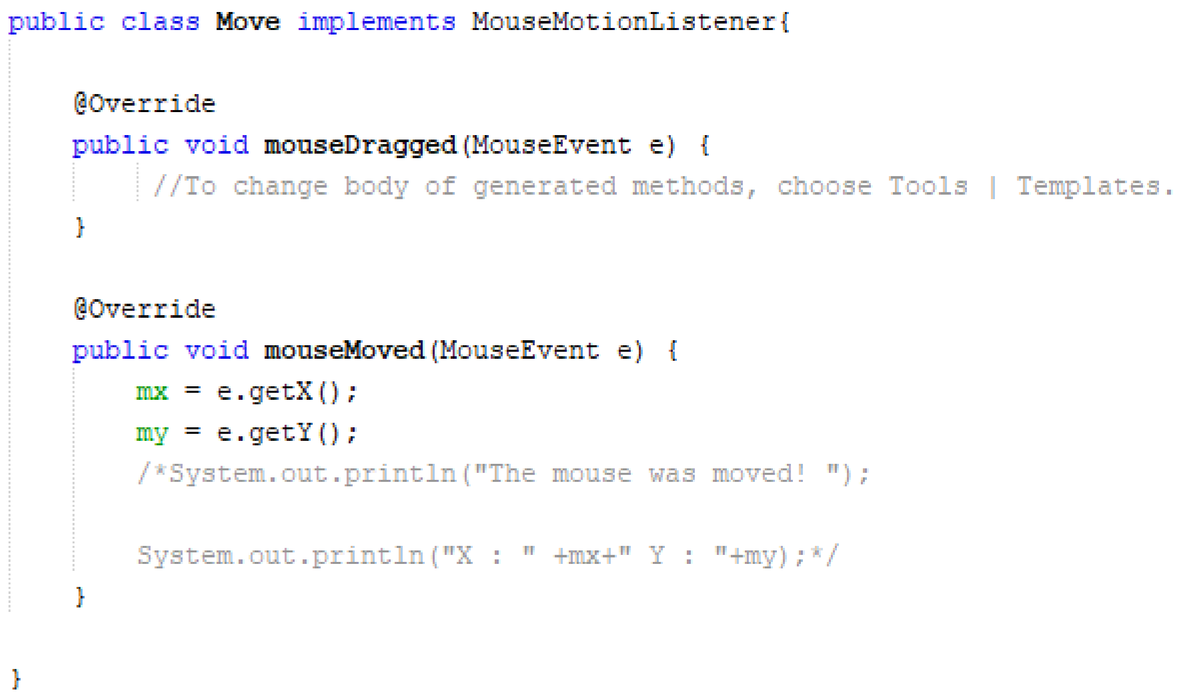


```
public boolean isN(int mX, int mY, int cX , int cY ){
    return mX-cX<2&&mX-cX > -2 &&mY-cY<2&&mY-cY > -2&&
            mines[cX][cY]==1;

}
public class random{
    public void mineRandom(){
        for(int i = 0 ; i<16;i++){
        for(int j =0;j<16; j++){
            if(rand.nextInt(100)<15&& i !=8 && j !=8){
                mines[i][j]=1;
            }else{
                mines[i][j] = 0;
            }
            revealed[i][j]= false;
            flagged[i][j]= false;
            stopClick[i][j] = false;

        }

    }
```

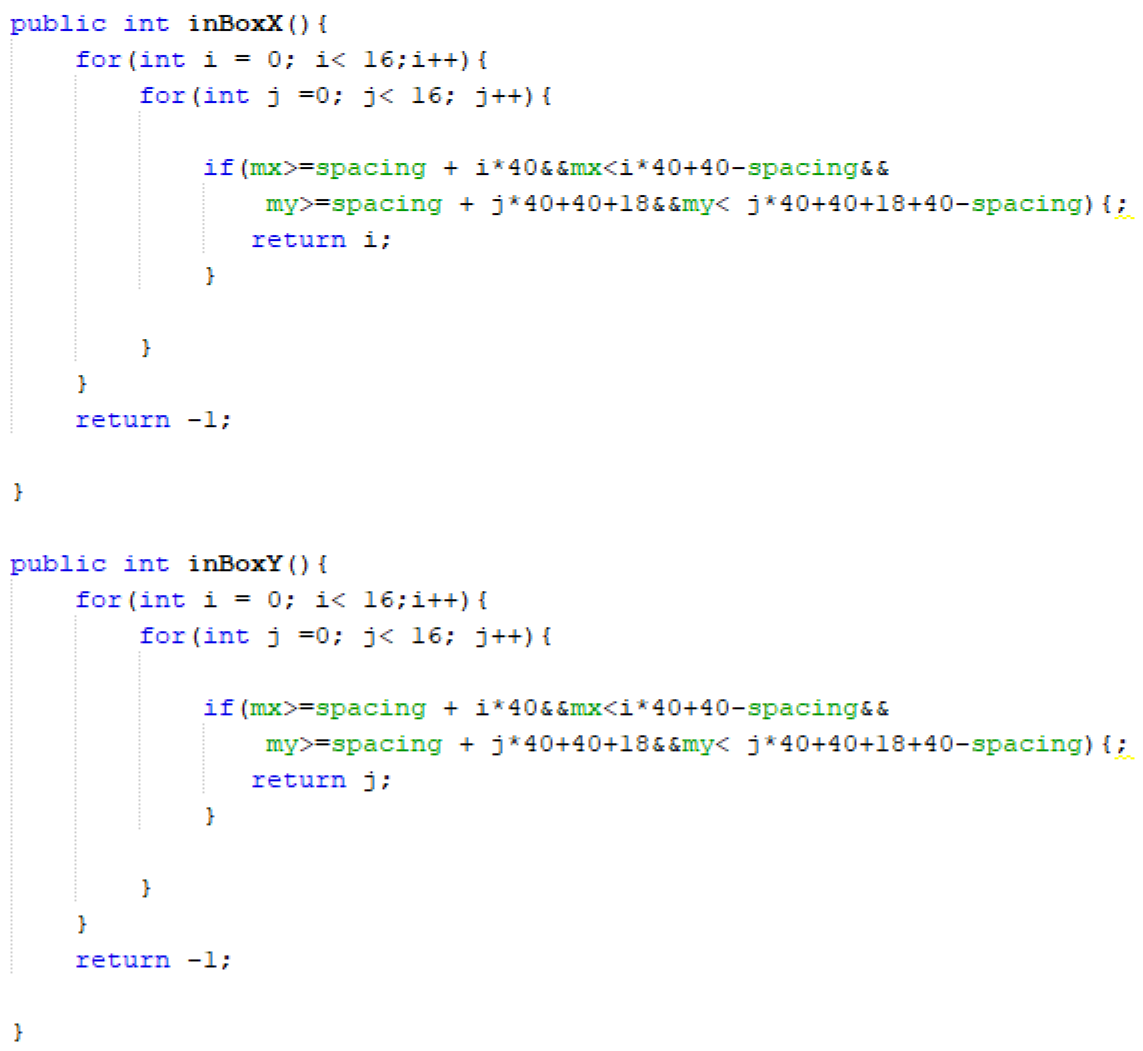


```
for(int i = 0 ; i<16;i++){
    for(int j =0;j<16; j++){
        neighs = 0;
        for(int n = 0 ; n<16;n++){
            for(int m =0;m<16; m++){
                if(!(n==i&& m==j)){
                    if(isN(i, j, n, m)== true)
                        neighs++;

                }
            }
        }
        neighbours[i][j] = neighs;
    }
}
```

- Next, when player clicks a blocks on the board, the coordinate of it will be return. It will be difficult for our to handle some feature and store information. Hence our task is converting the coordinate to [i][j] which represent the position of blocks in two -demension arrays. We created class “Move” implements the mouseMotionListener interface to get the coordinates when we move the mouse in the board.

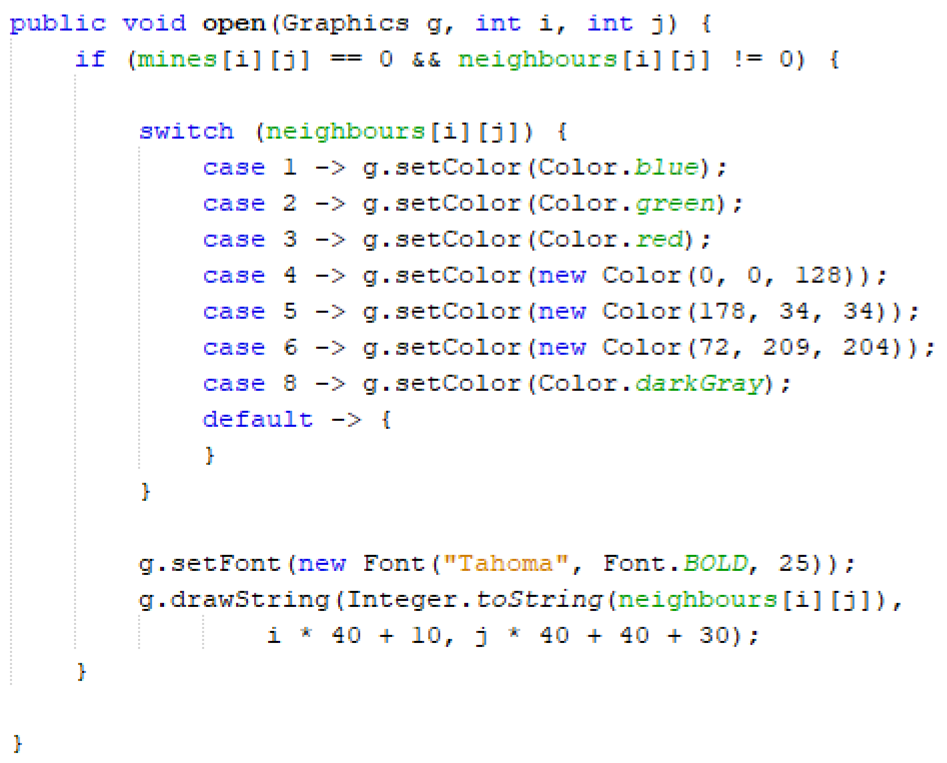


```
public class Move implements MouseMotionListener{

    @Override
    public void mouseDragged(MouseEvent e) {
        //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        mx = e.getX();
        my = e.getY();
        /*System.out.println("The mouse was moved! ");

        System.out.println("X : " +mx+" Y : "+my);*/
    }

}
```



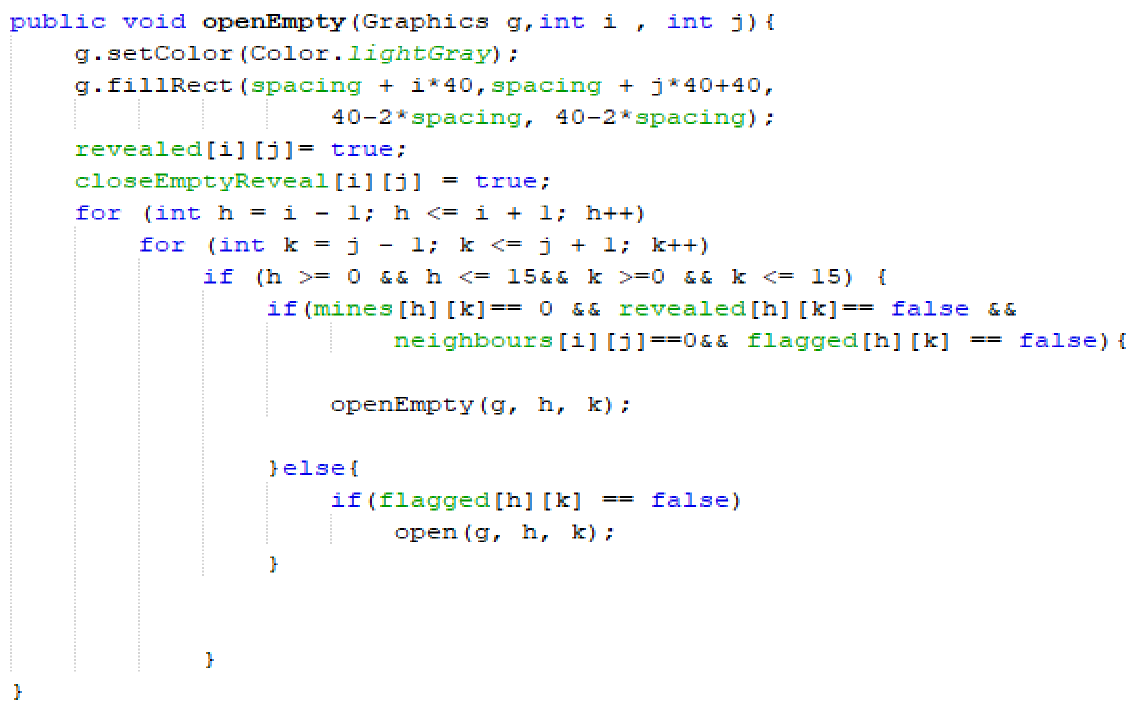
```
public int inBoxX(){
    for(int i = 0; i< 16;i++){
        for(int j =0; j< 16; j++){

            if(mx>=spacing + i*40&&mx<i*40+40-spacing&&
                my>=spacing + j*40+40+18&&my< j*40+40+18+40-spacing){;
              return i;
            }


        }
    }
    return -1;

}

public int inBoxY(){
    for(int i = 0; i< 16;i++){
        for(int j =0; j< 16; j++){

            if(mx>=spacing + i*40&&mx<i*40+40-spacing&&
                my>=spacing + j*40+40+18&&my< j*40+40+18+40-spacing){;
              return j;
            }


        }
    }
    return -1;

}
```

- Next task is opening the blocks when the player clicks it. There are three situiations:
  + A block contain mine: all of mines in this game will be displayed, and player cannot click the remain blocks in the board.
  + A block contain the number of mines in 8 adjacent-blocks: display the number of mines and set the value of revealed[i][j] = true (this block already reaches by the player). We cannot click this block.



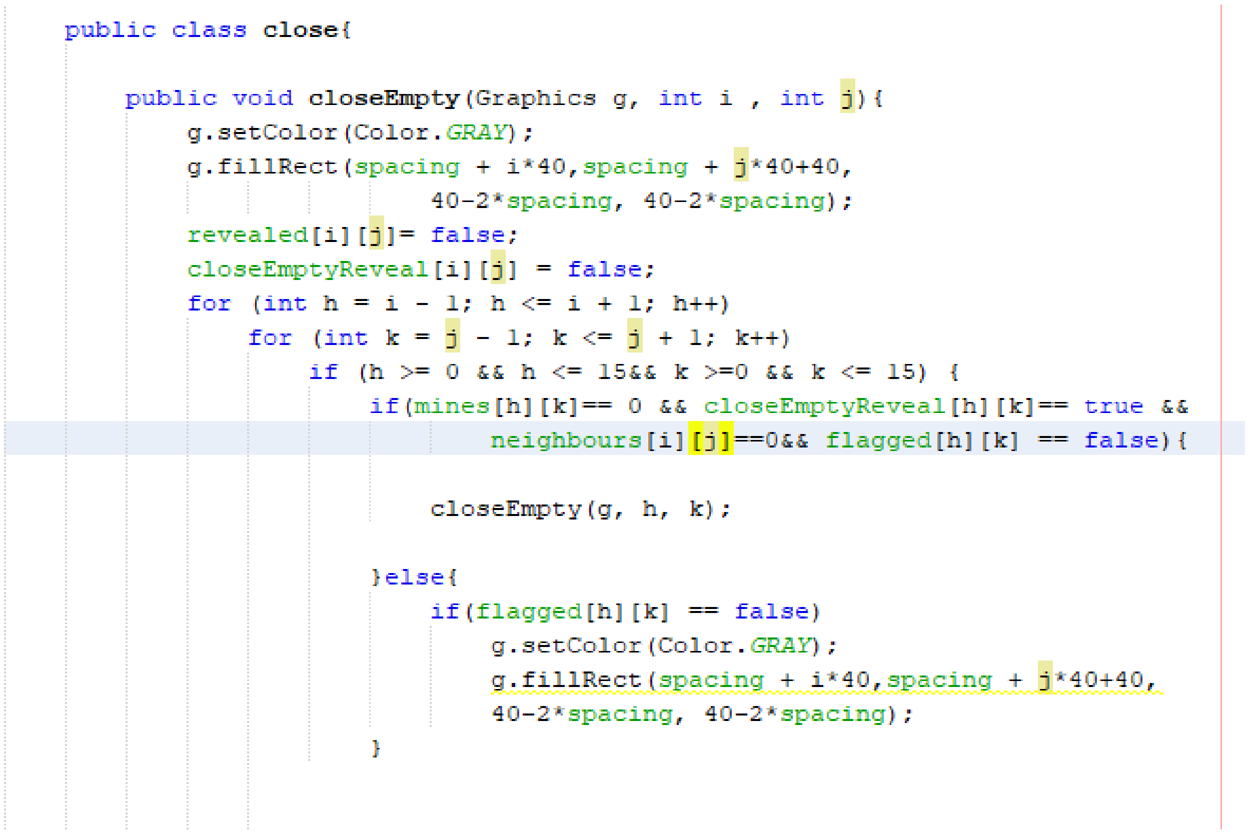
```
public void open(Graphics g, int i, int j) {
    if (mines[i][j] == 0 && neighbours[i][j] != 0) {

        switch (neighbours[i][j]) {
            case 1 -> g.setColor(Color.blue);
            case 2 -> g.setColor(Color.green);
            case 3 -> g.setColor(Color.red);
            case 4 -> g.setColor(new Color(0, 0, 128));
            case 5 -> g.setColor(new Color(178, 34, 34));
            case 6 -> g.setColor(new Color(72, 209, 204));
            case 8 -> g.setColor(Color.darkGray);
            default -> {
            }
        }

        g.setFont(new Font("Tahoma", Font.BOLD, 25));
        g.drawString(Integer.toString(neighbours[i][j]),
                i * 40 + 10, j * 40 + 40 + 30);
    }

}
```

+ An empty block: When player clicks an empty block, we will check 8 adjacent- block of this to find the other empty block, call the recursive function for this and set the value of revealed[i][j] (because a blocks can be checked many times). Open an emty blocks when all of empty blocks are covered by the non-empty blocks.



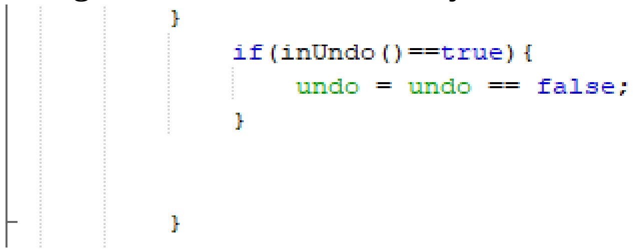
```
public void openEmpty(Graphics g,int i , int j){
    g.setColor(Color.lightGray);
    g.fillRect(spacing + i*40,spacing + j*40+40,
                    40-2*spacing, 40-2*spacing);
    revealed[i][j]= true;
    closeEmptyReveal[i][j] = true;
    for (int h = i - 1; h <= i + 1; h++)
        for (int k = j - 1; k <= j + 1; k++)
            if (h >= 0 && h <= 15&& k >=0 && k <= 15) {
                if(mines[h][k]== 0 && revealed[h][k]== false &&
                        neighbours[i][j]==0&& flagged[h][k] == false){

                    openEmpty(g, h, k);

                }else{
                    if(flagged[h][k] == false)
                        open(g, h, k);
                }


            }
}
```

- One of the additional features is undo features. To handle this feature, we used two stacks to store position of the block when the player clicks. This feature is depend on Open class, we called Close class. We used two dimension arrays to identify the blocks which is opened from the empty block (closeEmptyRevealed[][]) and non-empty blocks (closeRevealed[][]). Depend on the Open method, when a player clicks the blocks, this blocks will be marked to know what process this block is opened. Similar to open method, recursive function is used to find the blocks which are opened from empty-block.

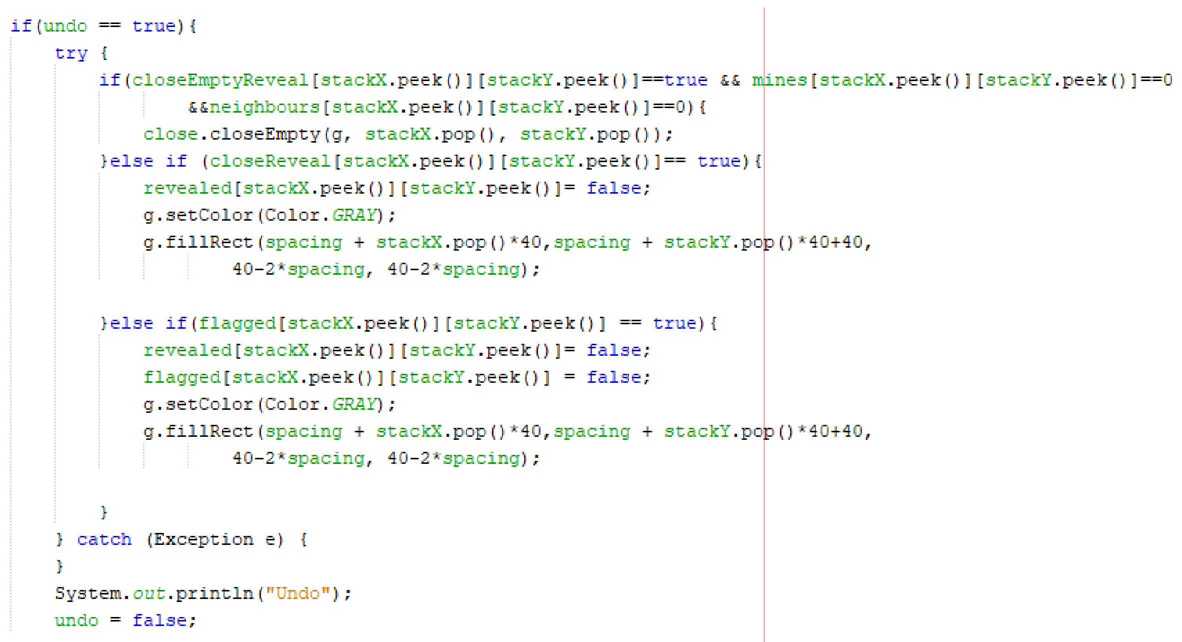


```
public class close{

    public void closeEmpty(Graphics g, int i , int j){
        g.setColor(Color.GRAY);
        g.fillRect(spacing + i*40,spacing + j*40+40,
                        40-2*spacing, 40-2*spacing);
        revealed[i][j]= false;
        closeEmptyReveal[i][j] = false;
        for (int h = i - 1; h <= i + 1; h++)
            for (int k = j - 1; k <= j + 1; k++)
                if (h >= 0 && h <= 15&& k >=0 && k <= 15) {
                    if(mines[h][k]== 0 && closeEmptyReveal[h][k]== true &&
                            neighbours[i][j]==0&& flagged[h][k] == false){

                        closeEmpty(g, h, k);

                    }else{
                        if(flagged[h][k] == false)
                            g.setColor(Color.GRAY);
                            g.fillRect(spacing + i*40,spacing + j*40+40,
                            40-2*spacing, 40-2*spacing);
                    }
```

- Finally, we will add and set-up some button for game. There are three button for this game such as flagger, undo, reset game. Each button will have a boolean variable. If the player clicks on the button, the value of button will be true. Then, some of processes will be executed, and the value of button will change into false automatically.



```
        }
            if(inUndo()==true){
                undo = undo == false;
            }


        }
```



```
if(undo == true){
    try {
        if(closeEmptyReveal[stackX.peek()][stackY.peek()]==true && mines[stackX.peek()][stackY.peek()]==0
                &&neighbours[stackX.peek()][stackY.peek()]==0){
            close.closeEmpty(g, stackX.pop(), stackY.pop());
        }else if (closeReveal[stackX.peek()][stackY.peek()]== true){
            revealed[stackX.peek()][stackY.peek()]= false;
            g.setColor(Color.GRAY);
            g.fillRect(spacing + stackX.pop()*40,spacing + stackY.pop()*40+40,
                    40-2*spacing, 40-2*spacing);

        }else if(flagged[stackX.peek()][stackY.peek()] == true){
            revealed[stackX.peek()][stackY.peek()]= false;
            flagged[stackX.peek()][stackY.peek()] = false;
            g.setColor(Color.GRAY);
            g.fillRect(spacing + stackX.pop()*40,spacing + stackY.pop()*40+40,
                    40-2*spacing, 40-2*spacing);

        }
    } catch (Exception e) {
    }
    System.out.println("Undo");
    undo = false;
```

In conclusion, The minesweeper is single-player puzzle video game. This game is built by using 2D- painting, knowledge of loops, stacks, and two dimension arrays, and some turtorials on the Internet. Our group's has basic function of this game and some additional features such as display win/lose, reset game or undo. But, this game have some ineffective solutions (ex: using 2D- array of buttons instead of painting each block) which we will deal with in the future.

(Git Link: https://github.com/ngphuc1110/Minesweeper?fbclid=IwAR1vrTMnoUs_qHjeznphlCUhi8i4m_bUkQXBJj05qs0emJfO7v_6GafFHbk)