マインスイーパーを作ろう

秋山 知寛

目次

1　初めに

このテーマに決めた理由は、最近 14 minesweeper variants というゲームを遊んでいて、2 年後期に Java プログラミングの授業も控えていたので、その予習も兼ねて自分でもマインスイーパーを作ってみることにしました。

2　マインスイーパーのルール

- プレイヤーはゲームボード上のマスを選択してオープンする。マスをオープンすると、そのマスには数字、地雷、または何もないことが表示される。
- 数字のマスは、周囲の 8 つのマスにある地雷の数を示す。この情報を利用して地雷に位置を推測する。
- プレイヤーは地雷だと思われるマスにフラグを設置することができる。これにより、地雷の位置を誤ってオープンすることがないようにできる。
- 地雷に触れるとゲームオーバーになる。地雷の位置を正確に推測し、地雷の内マスをすべてオープンするとゲームクリアです。

3　全体のコード

```
import java.awt.event.MouseListener;
import java.util.Random;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Color;
import java.awt.Font;
import java.util.Random;

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

public class Main {
    public static void main (String args[]){

        JFrame window = new MyFrame();
        window.setVisible(true);

    }
}

class MyFrame extends JFrame{

    MyPanel myPanel ;
    MyFrame(){
```

```
        setBounds(0 ,0 ,1000 ,600);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        myPanel = new MyPanel();
        add(myPanel);
        setVisible(true);
    }

}

class MyPanel extends JPanel implements MouseListener{
    static final int CELL_STATE_UNREVEALED = 0;
    static final int CELL_STATE_REVEALED = 1;
    static final int CELL_STATE_FLAG = 2;

    static final int GAME_STATE_NANIMONAI = 0;
    static final int GAME_STATE_PLAY_NOW =1;
    static final int GAME_STATE_CLEAR = 2;
    static final int GAME_STATE_OVER = 3;

    static final int HEIGHT = 10;
    static final int WIDTH = 15;

    Random random;
    int[][] aroundMineNum;
    boolean[][] isMine;
    int[][] cellState;
    int gameState;

    int mineCount = 15;
    int cellSize = 30;

    MyPanel(){
        random = new Random();
        addMouseListener(this);
        setBackground(new Color(0, 255, 255));


        aroundMineNum = new int[HEIGHT][WIDTH];
        isMine = new boolean[HEIGHT+2][WIDTH+2];
        cellState = new int[HEIGHT][WIDTH];

        gameState = GAME_STATE_NANIMONAI;

        for(int i = 0; i < HEIGHT; i++){
            for(int j = 0; j < WIDTH; j++){
                cellState[i][j] = CELL_STATE_UNREVEALED;
            }
        }
        for(int i = 0; i < HEIGHT+2; i++){
            for(int j = 0; j <WIDTH +2; j++){
                isMine[i][j] = false;
```

```
        }
    }
}

void saisyo(int safeX, int safeY){
            for(int i = 0; i < mineCount; i++){
        while(true){
            int y = random.nextInt(HEIGHT);
            int x = random.nextInt(WIDTH);
            if(!isMine[y+1][x+1]){
                if((safeY != y || safeX != x) && !(Math.abs(safeX-x) <= 1 && Math.abs(safeY-y) <= 1)){
                isMine[y+1][x+1] = true;
                break;
                }
            }
        }
    }
}

private void calcAroundMineNum(){
    for(int i = 0; i < HEIGHT; i++){
        for(int j = 0; j < WIDTH; j++){
            aroundMineNum[i][j] = 0;
        }
    }
    for(int i = 0; i < HEIGHT; i++){
        for(int j = 0; j < WIDTH; j++){
            for(int k = -1; k <= 1; k++){
                for(int l = -1; l <= 1; l++){
                    if(k == 0 && l == 0){
                        continue;
                    }
                    if(isMine[i+k+1][j+l+1]){
                        aroundMineNum[i][j]++;
                    }
                }
            }
        }
    }
    for(int i = 0; i < HEIGHT; i++){
        for(int j = 0; j < WIDTH; j++){
            System.out.print(aroundMineNum[i][j] + " ");
        }
        System.out.println();
    }
}

@Override
protected void paintComponent(Graphics g){
    Font lergeFont = new Font("MS ゴシック", Font.BOLD, 72);

    for (int i=0;i<HEIGHT;i++){
        for (int j=0;j<WIDTH;j++){
```

```
            if(cellState[i][j] == CELL_STATE_UNREVEALED){
                g.setColor(Color.GRAY);
            }else if(cellState[i][j] == CELL_STATE_REVEALED){
                if(isMine[i+1][j+1]){
                    g.setColor(Color.RED);
                }else{
                    g.setColor(Color.WHITE);
                }
            }else{
                g.setColor(Color.GREEN);
            }
            g.fillRect(cellSize*j, cellSize*i, cellSize, cellSize);
            g.setColor(Color.BLACK);
            g.drawRect(cellSize*j, cellSize*i, cellSize, cellSize);
            if(cellState[i][j] == CELL_STATE_REVEALED && aroundMineNum[i][j] >= 1 && !isMine[i+1][j
+1]){
                g.setFont(new Font(Font.DIALOG_INPUT,Font.BOLD,24));
                drawStringCenter(g, "" + aroundMineNum[i][j], j * cellSize + cellSize /2, i * cellSize
 + cellSize / 2);
            }
        }
    }

    if(gameState == GAME_STATE_CLEAR)
    {
        g.setColor(new Color(255, 0, 255));
        g.setFont(lergeFont);
        g.drawStringクリアおめでとう("", 50, 400);
        repaint();
    }
}

private static void drawStringCenter(Graphics g, String text, int x , int y){
    FontMetrics fm = g.getFontMetrics();
    Rectangle rectText = fm.getStringBounds(text, g).getBounds();
    x = x - rectText.width / 2;
    y = y - rectText.height / 2 + fm.getMaxAscent();
    g.drawString(text, x, y);
}


@Override
public void mouseExited(MouseEvent e){

}

@Override
public void mouseEntered(MouseEvent e){

}

@Override
public void mouseReleased(MouseEvent e){
```

```
}

@Override
public void mouseClicked(MouseEvent e){

}

@Override
public void mousePressed(MouseEvent e){
    int button = e.getButton();

    if (gameState == GAME_STATE_CLEAR) {
        return;
    }

    if(button == MouseEvent.BUTTON1){
        int x = e.getX() / cellSize;
        int y = e.getY() / cellSize;
        if(gameState == GAME_STATE_NANIMONAI){
            saisyo(x,y);
            calcAroundMineNum();
            gameState = GAME_STATE_PLAY_NOW;
        }
        if (isCleared()) {
            gameState = GAME_STATE_CLEAR;
        }
        ippaiakeru(x, y);
        if(isMine[y+1][x+1] == true){
            gameState = GAME_STATE_OVER;
            System.exit(0);
        }
    }
    else if(button == MouseEvent.BUTTON3){
        int x = e.getX() / cellSize;
        int y = e.getY() / cellSize;
        if(cellState[y][x] != CELL_STATE_UNREVEALED){
            return;
        }
        cellState[e.getY()/cellSize][e.getX()/cellSize] = CELL_STATE_FLAG;
        repaint();

    }
}

boolean isValidCell(int x, int y){
    if(x < 0 || y < 0 || x >= WIDTH ||y >= HEIGHT){
        return false;
    }
    return true;
}

private boolean isCleared(){
```

```
        for(int y = 0; y < HEIGHT; y++){
            for(int x = 0; x < WIDTH; x++){
                if(!isMine[y+1][x+1] && cellState[y][x] != CELL_STATE_REVEALED){
                    return false;
                }
            }
        }
        return true;
    }

    void ippaiakeru(int x, int y){
        cellState[y][x] = CELL_STATE_REVEALED;
        repaint();
            if(isMine[y][x] == false)
            {
                if (aroundMineNum[y][x]==0)
                {
                    for(int k = -1; k <= 1; k++){
                        for(int l = -1; l <= 1; l++){
                            if(k == 0 && l == 0){
                                continue;
                            }
                            if(isValidCell(x+l, y+k) && cellState[y+k][x+l] == CELL_STATE_UNREVEALED)
                            {
                                ippaiakeru(x+l,y+k);

                                repaint();
                            }
                        }
                    }
                }
            }

    }

}
```

4　コードの説明

```
void saisyo(int safeX, int safeY){
            for(int i = 0; i < mineCount; i++){
        while(true){
            int y = random.nextInt(HEIGHT);
            int x = random.nextInt(WIDTH);
            if(!isMine[y+1][x+1]){
                if((safeY != y || safeX != x) && !(Math.abs(safeX-x) <= 1 && Math.abs(safeY-y) <= 1)){
                isMine[y+1][x+1] = true;
                break;
                }
            }
        }
    }
}
```

この関数では、カーソルの x,y 座標を safeX,safeY として取得して、最初にクリックしたマスとその周り 8 マスが地雷でなくなるように地雷を生成しています。

```
void ippaiakeru(int x, int y){
    cellState[y][x] = CELL_STATE_REVEALED;
    repaint();
        if(isMine[y][x] == false)
        {
            if (aroundMineNum[y][x]==0)
            {
                for(int k = -1; k <= 1; k++){
                    for(int l = -1; l <= 1; l++){
                        if(k == 0 && l == 0){
                            continue;
                        }
                        if(isValidCell(x+l, y+k) && cellState[y+k][x+l] == CELL_STATE_UNREVEALED)
                        {
                            ippaiakeru(x+l,y+k);

                            repaint();
                        }
                    }
                }
            }
        }

}
```

この関数では、クリックしたマスとその周囲に地雷がない時にそこから繋がっている同様のマスを一気に開けるための関数になっています。
実行してみると以下のようになります。

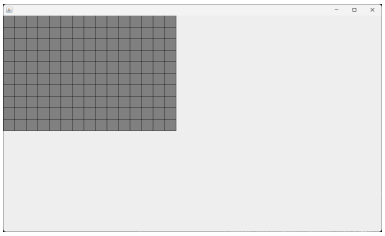

図 1　クリックする前

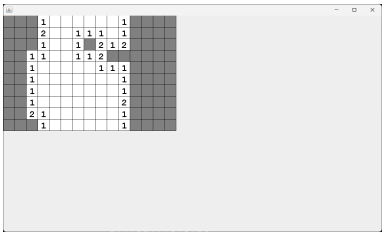

図 2　クリックした後

5　課題

- 盤面の外側をクリックしても生成されてしまう。
- ランダムに地雷を設置しているだけなので、確実に解けるようになっていない。

以上の課題を解決したい。

5.1　解決策

1 つ目の問題は、

```
boolean isValidCell(int x, int y){
  if(x < 0 || y < 0 || x >= WIDTH ||y >= HEIGHT){
      return false;
  }
  return true;
}
```

この関数は画面外を参照した時に `false` を返すようになっているので、この関数を地雷の生成の部分で応用すればできるのではないかと考えている。

2 つ目の問題は、SAT ソルバーを利用すれば解決できるのではないかと考えている。

SAT とは、命題論理式の充足可能判定のことで、CNF を扱うのが一般的である。

5.2 CNF 式とは

CNF 式とは、節を論理積 $\wedge$ で結合した論理式の事です。
節とは、リテラルを論理和 $\vee$ で結合した論理式のことで、
リテラルは変数 x, またはその否定 $\neg x$ の事です。

例 1. $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3)$

以上を満たす解は $x_1 = 1, x_2 = 0, x_3 = 0$ があります。
SAT ソルバーはこれを解くソルバーのことです。