
International University, HCMC National University

School of Computer Science and Engineering

# PROJECT REPORT

## Algorithms & Data Structures

# Game: MINE SWEEPER

| No. | Full Name | Student ID | Evaluation (%) |
|---|---|---|---|
| 1 | Nguyễn Duy Phúc | ITDSIU21030 | 25% |
| 2 | Vũ Vỹ Khang | ITDSIU21090 | 25% |
| 3 | Phan Quốc Anh | ITDSIU21001 | 25% |
| 4 | Nguyễn Quốc Huy | ITDSIU21067 | 25% |

COURSE: Algorithms & Data Structures_S2_2022-23_G01

INSTRUCTOR: Teacher Lam (Lab), Teacher Tung (Theory)

**TABLE OF CONTENTS**



---

# 1.Introduction:

Minesweeper is a classic computer game that has been enjoyed by millions of players around the world for decades. The game is simple yet challenging, requiring players to use strategy and deduction to clear a minefield without detonating any hidden mines. In this report, we will explore the history of Minesweeper, its basic rules and gameplay mechanics, and the strategies that players can use to improve their performance.

Whether you are a seasoned Minesweeper player or a newcomer to the game, this report will provide valuable insights and information about one of the most beloved computer games of all time.

# 2. Ideas & Algorithms

## 1. Ideas:

Minesweeper is a game that involves a map, bombs, flags, and other features that make it both challenging and exciting. The map is a grid of squares that can vary in size, and some of these squares contain hidden bombs. The objective of the game is to uncover all the squares that do not contain bombs while avoiding the ones that do. To do this, players must use logic and deduction to determine which squares are safe to uncover and which ones are not.

In addition to uncovering squares, players can also mark squares that they suspect contain bombs by placing a flag on them. This helps players keep track of which squares they believe are dangerous and avoid them. However, players must be careful not to use too many flags, as they have a limited number available.

To make the game even more challenging, there are different difficulty levels that players can choose from. These levels determine the size of the map and the number of bombs hidden within it. The higher the difficulty level, the more challenging the game becomes.

To help players navigate the game, there are also additional features such as undo, save game, and choose difficulty. The undo feature allows players to undo their last move if they make a mistake, while the save game feature allows players to save their progress and come back to the game later. The choose difficulty feature allows players to select the level of difficulty that they feel comfortable with.

## 2. Rule:

Players must clear a minefield without detonating any hidden mines. The game is played on a grid of **squares**, and the objective is to uncover all the squares that do not contain mines while avoiding the ones that do. To play, the player clicks on a square to uncover it. If the square contains a mine, the game is over, and the player loses. If the square does not contain a mine, a number is displayed indicating how many mines are adjacent to that square. Using the numbers as clues, the player can deduce which squares are safe to uncover and which ones contain mines. The player can mark squares that they suspect contain mines by right-clicking on them with the mouse. The game is won when all the squares that do not contain mines have been uncovered. The game can be customized with different difficulty levels.

# 3. Code and Algorithms

## 3.1 Create Board

```
public class Board extends JPanel implements ActionListener {
    //DEFINE FIELDS
        //Fields related to board and cells
        private final int CELL_SIZE = 15;
        private final int N_MINES = 10;
        private static int N_ROWS = 16;
        private static int N_COLS = 16;
        private static String CELL_SPLITTER = " - ";
        private static String OBJECT_SPLITTER = "R";
        private final int BOARD_WIDTH = N_ROWS * CELL_SIZE + 1;
        private final int BOARD_HEIGHT = N_COLS * CELL_SIZE + 1;
        private int minesLeft;//keeps track of how many mines are left based on what user has flagged
        //2D array to represent game board
        protected static Cell[][] gameBoard;
        //total number of cells
        private int allCells;
        //Fields related to images used in our game to represent cells and bombs
        private final int NUM_IMAGES = 13;
        //Using map as collection to store images and their names, which can make it more easily retrievable
        private java.util.Map<String, Image> images;
        //Fields related to game status
        private boolean inGame;
        private static JLabel statusbar;
        private static JButton bUndo;
        private static JButton bRule;
        private static JTextArea textArea;
        private static String STATUS_FILE = "Status.txt";
        private Stack gameSteps = new Stack();
```

```java
public Board(JLabel statusbar, JButton bUndo, JButton bRule, JTextArea textArea) throws IOException {
    this.statusbar = statusbar;
    this.bUndo = bUndo;
    this.bUndo.addActionListener(this);
    this.bRule = bRule;
    this.bRule.addActionListener(this);
    this.textArea = textArea;
    initBoard();
}
```

## 3.2 Create Mines

```java
//2D Array of cells in gameboard
gameBoard = new Cell[N_ROWS][N_COLS];

for (int x = 0; x < N_ROWS; x++) {//initially have everything be empty
    for(int y=0; y < N_COLS; y++) {
        gameBoard[x][y] = new EmptyCell();
    }
}

statusbar.setText("Flags Left: " + Integer.toString(minesLeft));

int i = 0;

//set up the grid
while (i < N_MINES) {
    Random random = new Random();
    int positionX = (int) (random.nextInt(bound:16));
    int positionY = (int) (random.nextInt(bound:16));

    //randomly place the bomb cell
    if(gameBoard[positionX][positionY].getCellType() != CellType.Bomb) {
        gameBoard[positionX][positionY] = new BombCell();


        //sets up neighbor cells
        for(int dx = -1; dx <= 1; dx++) {
            for(int dy = -1; dy <= 1; dy++) {
                if((dx != 0 || dy != 0) && positionX + dx < N_COLS && positionY + dy < N_ROWS
                        && positionX + dx >= 0 && positionY + dy >=0) {
                    CellType typeOfCell = gameBoard[positionX + dx][positionY + dy].getCellType();
```

```
                    if(typeOfCell != CellType.Bomb) {//not already a neighbor cell
                        if (typeOfCell != CellType.BombNeighbor) {
                            NeighborOfBombCell neighbor = new NeighborOfBombCell();
                            neighbor.cellCount();
                            gameBoard[positionX + dx][positionY + dy] = neighbor;
                        }
                        else {//already a neighbor cell, just need to update the neighbor count

                        gameBoard[positionX + dx][positionY + dy].cellCount();
                    }
                    }

                }
            }
        }
        i++;

    }

}
}
```

## 3.3 Add image into Board

```
private void initBoard() throws IOException {

    setPreferredSize(new Dimension(BOARD_WIDTH, BOARD_HEIGHT));
    images = new java.util.HashMap<>();


  //Put all relevant images in the map, some images named with integers, others named with descriptors
    for (int i = 1; i < 9; i++) {
        String path = "D:/Game/Project/src/resources/" + i + ".png";
        images.put(Integer.toString(i), (new ImageIcon(path)).getImage());
    }
    images.put(key:"Bomb", (new ImageIcon(filename:"D:/Game/Project/src/resources/Bomb.png")).getImage());
    images.put(key:"Covered", (new ImageIcon(filename:"D:/Game/Project/src/resources/Covered.png")).getImage());
    images.put(key:"Empty", (new ImageIcon(filename:"D:/Game/Project/src/resources/Empty.png")).getImage());
    images.put(key:"Marked", (new ImageIcon(filename:"D:/Game/Project/src/resources/Marked.png")).getImage());
    images.put(key:"Wrongmarked", (new ImageIcon(filename:"D:/Game/Project/src/resources/Wrongmarked.png")).getImage());
```

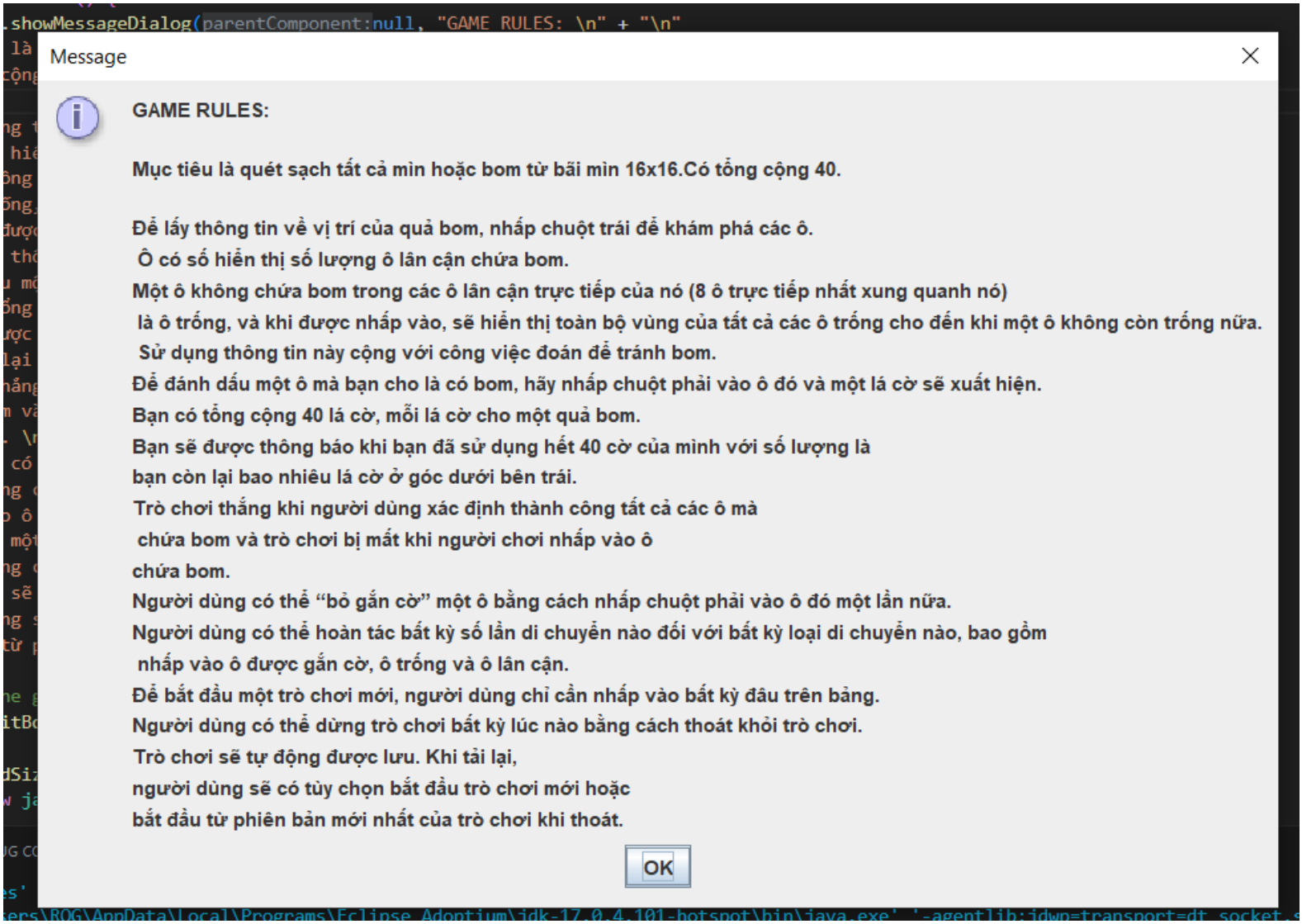
## 3.4 Create Mark with right clicked on mouse

```
//RIGHT MOUSE CLICK
if (e.getButton() == MouseEvent.BUTTON3) {

    if (gameBoard[cRow][cCol].isCoveredCell()) {
        doRepaint = true;//implies that we do nothing if user right clicks on a number cell
        //right click on an unmarked cell
                if (!gameBoard[cRow][cCol].isMarkedCell() && minesLeft > 0) {
                    Cell cell = gameBoard[cRow][cCol];
                    cell.changeWhetherMarked();//changed to marked cell
                    minesLeft--;
                    if (minesLeft > 0) {
                        String msg = Integer.toString(minesLeft);
                        statusbar.setText("Flags Left: " + msg);
                    } else {
                       statusbar.setText(text:"No marks left");
                    }
                    //add steps to gameSteps stack
                    gameSteps.push(cRow * N_COLS + cCol);
                }
                else if (gameBoard[cRow][cCol].isMarkedCell()) {//right click on already marked cell, removes marks and increase number of cells to be marked
                    gameBoard[cRow][cCol].changeWhetherMarked();//changes it to not marked
                    minesLeft++;
                    String msg = Integer.toString(minesLeft);
                    statusbar.setText("Flags Left: " + msg);
                }


    }
```

## 3.5 Left mouse click and repaint when click on boom

```
    //LEFT MOUSE CLICK
    } else {

        //nothing happens if user left clicks on covered and marked cell
        if (gameBoard[cRow][cCol].isMarkedCell()) {
            return;
        }

        //user left clicks to remove a cover from cell
        if (gameBoard[cRow][cCol].isCoveredCell()
                //&& (gameBoard[cRow][cCol].getCellType() == CellType.Bomb )
                ) {

            gameBoard[cRow][cCol].flipUp();
            doRepaint = true;
            gameSteps.push(cRow * N_COLS + cCol);

            //if user clicks on mine, game is over
            if (gameBoard[cRow][cCol].getCellType() == CellType.Bomb
                    && !gameBoard[cRow][cCol].isCoveredCell()) {
                inGame = false;
            }

            //if user clicks on empty cell, call empty cell function which will handle the situation
            if (gameBoard[cRow][cCol].getCellType() == CellType.Empty) {
                find_empty_cells(cRow, cCol);
            }
        }
    }
    if (doRepaint) {
        repaint();
```

3.6 Create button Rules to notice Rules with player


showMessageDialog(parentComponent:null, "GAME RULES: \n" + "\n"
Message
GAME RULES:
Mục tiêu là quét sạch tất cả mìn hoặc bom từ bãi mìn 16x16.Có tổng cộng 40.
Để lấy thông tin về vị trí của quả bom, nhấp chuột trái để khám phá các ô.
Ô có số hiển thị số lượng ô lân cận chứa bom.
Một ô không chứa bom trong các ô lân cận trực tiếp của nó (8 ô trực tiếp nhất xung quanh nó)
là ô trống, và khi được nhấp vào, sẽ hiển thị toàn bộ vùng của tất cả các ô trống cho đến khi một ô không còn trống nữa.
Sử dụng thông tin này cộng với công việc đoán để tránh bom.
Để đánh dấu một ô mà bạn cho là có bom, hãy nhấp chuột phải vào ô đó và một lá cờ sẽ xuất hiện.
Bạn có tổng cộng 40 lá cờ, mỗi lá cờ cho một quả bom.
Bạn sẽ được thông báo khi bạn đã sử dụng hết 40 cờ của mình với số lượng là
bạn còn lại bao nhiêu lá cờ ở góc dưới bên trái.
Trò chơi thắng khi người dùng xác định thành công tất cả các ô mà
chứa bom và trò chơi bị mất khi người chơi nhấp vào ô
chứa bom.
Người dùng có thể "bỏ gắn cờ" một ô bằng cách nhấp chuột phải vào ô đó một lần nữa.
Người dùng có thể hoàn tác bất kỳ số lần di chuyển nào đối với bất kỳ loại di chuyển nào, bao gồm
nhấp vào ô được gắn cờ, ô trống và ô lân cận.
Để bắt đầu một trò chơi mới, người dùng chỉ cần nhấp vào bất kỳ đâu trên bảng.
Người dùng có thể dừng trò chơi bất kỳ lúc nào bằng cách thoát khỏi trò chơi.
Trò chơi sẽ tự động được lưu. Khi tải lại,
người dùng sẽ có tùy chọn bắt đầu trò chơi mới hoặc
bắt đầu từ phiên bản mới nhất của trò chơi khi thoát.
OK

```
private void showRules() {
    JOptionPane.showMessageDialog(parentComponent:null, "GAME RULES: \n" + "\n"
    + "Mục tiêu là quét sạch tất cả mìn hoặc bom từ bãi mìn 16x16."
    + "Có tổng cộng 40. \n" +
    "\n" +
    "Để lấy thông tin về vị trí của quả bom, nhấp chuột trái để khám phá các ô." +"\n"
    + " Ô có số hiển thị số lượng ô lân cận chứa bom." +"\n"
    + "Một ô không chứa bom trong các ô lân cận trực tiếp của nó (8 ô trực tiếp nhất xung quanh nó) sẽ là ô trống"+
    " và khi được nhấp vào, sẽ hiển thị toàn bộ vùng của tất cả các ô trống cho đến khi một ô không còn trống nữa."+"\n"
    + " Sử dụng thông tin này cộng với công việc đoán để tránh bom. \n" +
    "Để đánh dấu một ô mà bạn cho là có bom, hãy nhấp chuột phải vào ô đó và một lá cờ sẽ xuất hiện."+"\n"
    + "Bạn có tổng cộng 40 lá cờ, mỗi lá cờ cho một quả bom." +"\n"
    + "Bạn sẽ được thông báo khi bạn đã sử dụng hết 40 cờ của mình với số lượng là "+"\n"
    + "bạn còn lại bao nhiêu lá cờ ở góc dưới bên trái. \n" +
    "Trò chơi thắng khi người dùng xác định thành công tất cả các ô mà chứa bom và thua khi người chơi nhấp vào ô có bom" +
    "Người dùng có thể “bỏ gắn cờ” một ô bằng cách nhấp chuột phải vào ô đó một lần nữa."+"\n"
    + "Người dùng có thể hoàn tác bất kỳ số lần di chuyển nào đối với bất kỳ loại di chuyển nào, bao gồm"+"\n"
    + " nhấp vào ô được gắn cờ, ô trống và ô lân cận."+"\n" +
    "Để bắt đầu một trò chơi mới, người dùng chỉ cần nhấp vào bất kỳ đâu trên bảng." +"\n"
    + "Người dùng có thể dừng trò chơi bất kỳ lúc nào bằng cách thoát khỏi trò chơi." +"\n"
    + "Trò chơi sẽ tự động được lưu. Khi vào lại người dùng sẽ có tùy chọn bắt đầu trò chơi mới hoặc tiếp tục chơi ở màn đã lưu "
    );
}
```

## 3.7 Define username and method saving game

```
protected static void saveGameStatus2File() throws IOException {
    String userName = "";
    //lets user know that
    if ("".equals(textArea.getText()) || textArea.getText().equals(anObject:"Input your name here...")) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"We gave you a default user name, you may input your name next time.");
        userName = "Default user";
    } else {
        userName = textArea.getText();
    }
    if (gameBoard.length == 0) {
        System.exit(status:0);
    }
    //Writes user name
    //Goes through the entire game board, records the state of each cell, writes in a text file
    FileWriter writer = new FileWriter(STATUS_FILE, append:false);

    try (PrintWriter printLine = new PrintWriter(writer)) {
        printLine.println(OBJECT_SPLITTER + "User Name" + OBJECT_SPLITTER);
        printLine.println(userName);
        printLine.println(OBJECT_SPLITTER + "Cells" + OBJECT_SPLITTER);
        for (int i = 0; i < N_ROWS; i++) {
            for (int j = 0; j < N_COLS; j++) {
                if (null != gameBoard[i][j].getCellType()) switch (gameBoard[i][j].getCellType()) {
                //if cell is empty
                case Empty:
                        printLine.println(CellType.Empty.toString() + CELL_SPLITTER +
                                Boolean.toString(gameBoard[i][j].isCoveredCell()) + CELL_SPLITTER +
                                Boolean.toString(gameBoard[i][j].isMarkedCell()) + CELL_SPLITTER + "0");
                        break;
```

## Describe Algorithm use in project:

By default, the cells are closed, unexploded and unmarked.

When the new table is initialized, we perform 2 steps. Step 1, land mines in random cells. Step 2, browse each cell of the board and count the number of mines around that cell. If that cell already has mines, ignore it, otherwise write the count in the cell's mine count property (counting with 0 is still recorded).

When the left click event occurs, the open property is set to true, and the surrounding mines value is checked. If the value is 0, then use recursion to open the surrounding cells. The recursion will stop when that value is not 0. This is how when we click on an empty cell, a bunch of unimportant cells will open.

## 4. Conclusion:

After dealing with the project, we remember and take note about how recursion work, know how to implement UI with java swing, understand how to convert img with code. But we couldn't complete what we were supposed to add functionality like choosing the difficulty of the levels, creating more characters to make the game more vivid. We will try to implement that after we finish this presentation.

## 5. References


https://www.youtube.com/watch?v=jV1x6il0AzA&list=PLlD_ilI9JDbBVYP5uQUII81Q3w_pxAKhB

https://github.com/janbodnar/Java-Minesweeper-Game

https://github.com/luckytoilet/MSolver

https://zetcode.com/javagames/minesweeper/

https://codereview.stackexchange.com/questions/172338/beginner-java-minesweeper-game