**Java Programming (CSE220): Mini Project Report**

**Mine Sweeper**

**Index(**Use hyperlinks**)**

**Introduction 2**

Mini project purpose [2](#puropose)

Development Environment[3](#environment)

**Body 3**

Game Description [3](#game_description)

Additional functions [4](#additional_functions)

Design (Class Diagram) [5](#Design)

Major functions Description[5](#major_functions_description)

Cell class [5](#major_functions_description)

MineMap class [7](#minemap_class)

Gameboardpanel class [8](#gameboardpanel_class)

View class [13](#view_class)

MinewSweeperMain class [14](#minewSweeperMain_class)

Results/Output [16](#results_output)

**Conclusion** [20](#conclusion)

**Team(01)**

1. ID: 2019136048 Name: 김형진(Kim Hyeongjin) Role: PPT
2. ID: 2020136025 Name: 김예원(Kim Yewon) Role: presentation
3. ID: 2020136149 Name: 김태섭(Kim Taeseob) Role: GUI Code, Report
4. ID: 2020136135 Name: 한송이(Han Songyi) Role: Additional functions Code

**Project Title**

Mine Sweeper game

**Introduction**

* Mini project purpose

1. We can create a simple GUI program by collaborating with team members using Java.

2. All types of Java's inheritance concept and Java grammar learned so far are utilized.

3. Use existing Java example code to add new features.

4. Understand the grammar and library functions of Java FX.

5. Make and present simple data about the completed GUI program.

6. We can fully understand the mine sweeper game and implement additional functions.

* Development Environment

Development Language: Java FX

Development OS: Windows 11 64bit

Execution Environment: JavaFX Application

Development Tools: eclipse workspace

**Body**

* Game Description

Mine Sweeper is a simple and popular game that everyone can enjoy. The goal of play is to click all the spaces without mines.

1. You can choose from a total of three difficulties, and there are 'Beginner', 'Intermediate', and 'Advanced' difficulties in easy order.
2. You can select cells one by one with your mouse to avoid mines before the timer runs out.
3. When you select a cell, the number appears in 8 directions relative to the selected location, which allows you to predict the location of the mine.
4. Additionally, you can plant the flag by right-clicking on the expected mine location.
5. The flag is a kind of ‘Chance’ function, there are three flags in total, and if a mine is found, the cell is revealed.

* Additional functions

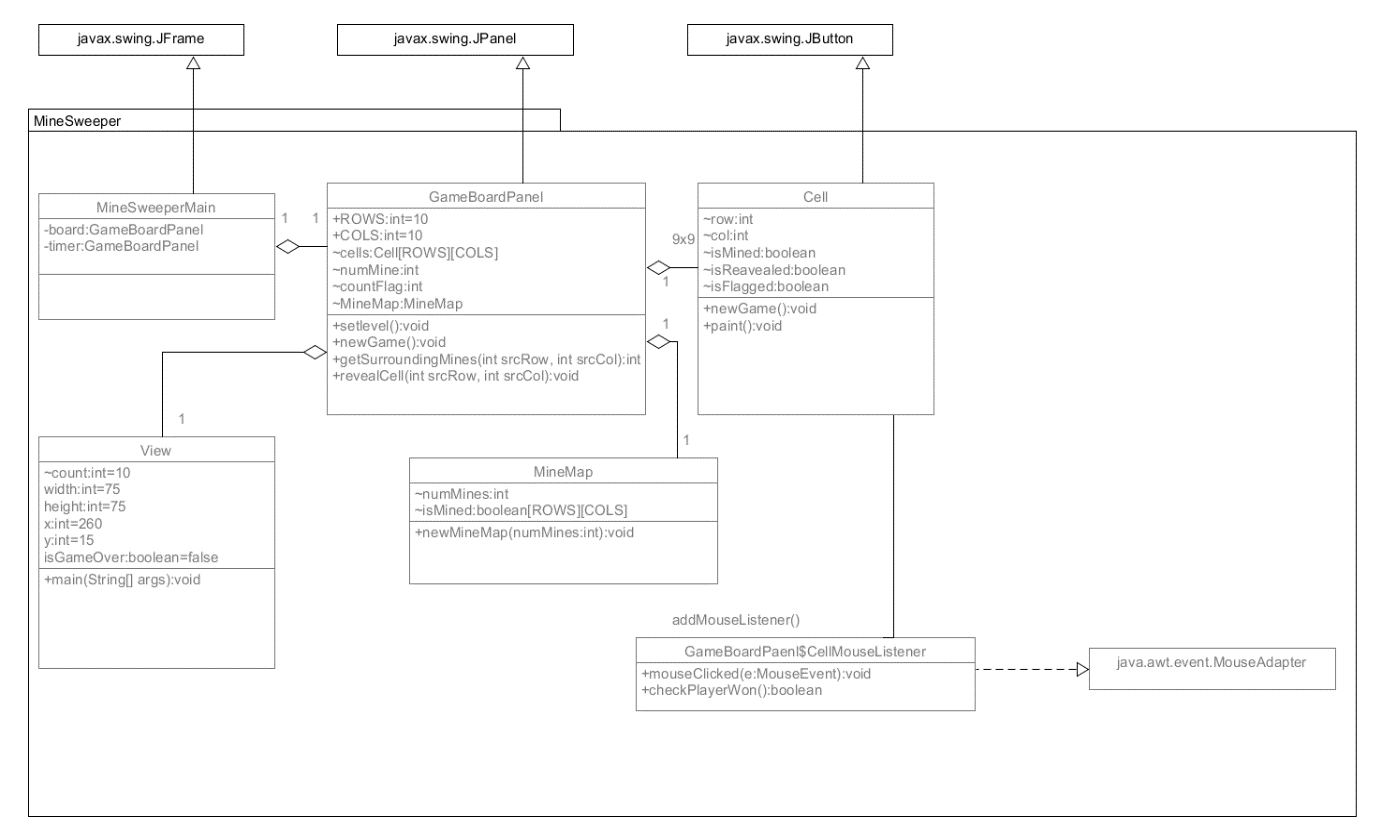
**Select Level**: You can choose from a total of three difficulties, and the difficulty levels of 'Beginner', 'Intermediate', and 'Advanced' exist in an easy order. The number of mins varies depending on the level selected. There are 5 Beginner's, 10 Intermediate's, and 15 Advanced's.

**Timer**: When the game starts, a timer flows, and if the time-over occurs, a dialog appears whether to start or end the game anew. Create a JLabel to indicate a timer.

**Flag**: You can plant a flag by right-clicking on the cell. There are only three flags in total, and you can erasing them. If you plant a flag on a mine, the cell is revealed. Flag is a kind of 'Chance' function.

**Mine Random Creation**: Every time a new game is played, the location of the mine is created in a random location, not in its original state. The number of mines is randomly arranged according to the selected level.

**Color**: You can check the number that appears in the cell in different colors depending on the number of mines around current mine. If there is one number of mines, it will appear white if there are two green ones and three yellow ones or more. Also, if it is Reveled, it appears dark gray, and if it is Not Reveled, it appears gray. In addition, the Flag and mines are marked in red.

* Design (Class Diagram)
* Major functions Description

1. **Cell class**

|  |
| --- |
| public class Cell extends JButton {  private static final long *serialVersionUID* = 1L; // to prevent serial warning  // Define named constants for JButton's colors and fonts  // to be chosen based on cell's state  public static final Color *BG\_NOT\_REVEALED* = Color.*GRAY*;  public static final Color *FG\_NOT\_REVEALED* = Color.*RED*; // flag, mines  public static final Color *BG\_REVEALED* = Color.*DARK\_GRAY*;  public static final Color *FG\_REVEALED\_1* = Color.*GREEN*;//mines 1  public static final Color *FG\_REVEALED\_2* = Color.*YELLOW*;//mines 2  public static final Color *FG\_REVEALED\_3* = Color.*WHITE*;//mines 3  public static final Font *FONT\_NUMBERS* = new Font("Monospaced", Font.*BOLD*, 20)  int row, col;  /\*\* Already revealed? \*/  boolean isRevealed;  /\*\* Is a mine? \*/  boolean isMined;  /\*\* Is Flagged by player? \*/  boolean isFlagged;  public Cell(int row, int col) {//constructor  super(); // JTextField  this.row = row;  this.col = col;// Set JButton's default display properties  super.setFont(*FONT\_NUMBERS*);  } |

Cell class is a class that extends JButton to make up cells with mines and cells without them, flags, and mines. we declare the integer variables row and col and we declare three boolean isRevealed, isMined, isFlagged. The default constructor of the Cell class receive the values of the variables row and col. Set JButton's default display properties.

|  |
| --- |
| public void newGame(boolean isMined, int numMines) {  this.isRevealed = false; // default  this.isFlagged = false; // default  this.isMined = isMined; // given  super.setEnabled(true); // enable button  super.setText(""); // display blank  paint(numMines); //paint() call  } |

NewGame function that initializes variables. Accept numMines as a parameter and pass the factor to the paint() function.

|  |
| --- |
| public void paint(int numMines) {  if(isRevealed) {//isRevealed  super.setBackground(*BG\_REVEALED*);  if(numMines == 1) {  super.setForeground(*FG\_REVEALED\_2)*;  } else if(numMines == 2) {  super.setForeground(*FG\_REVEALED\_2*);  } else if(numMines == 0) {  super.setText("");  } else {  super.setForeground(*FG\_REVEALED\_3*);  }  } else {//not Reveal  super.setBackground(*BG\_NOT\_REVEALED*);  super.setForeground(*FG\_NOT\_REVEALED*);  }  } |

paint function that receives the numMines and specifies the appropriate background color and letter color. If the conditional statement to check if a cell is revealed is true, the background color becomes dark gray. In addition, if there is one mine, the letter color is set to green, if there are two, yellow, there are three, white and if there are zero blank. In another case, if a cell is not revealed, the background color is gray and the letter(flag, mine) color is set to red.

1. **MineMap class**

|  |
| --- |
| public class MineMap {  // package access  int numMines;  boolean[][] isMined = new boolean[GameBoardPanel.*ROWS*][GameBoardPanel.*COLS*];  // default is false  public MineMap() {//constructor  super();  }  public void newMineMap(int numMines) {  this.numMines = numMines;  // Hardcoded for illustration and testing, assume numMines=10  //Mines randomly display  for (int row = 0; row < this.numMines; ++row) {  isMined[(int)(Math.*random*()\*10)][(int)(Math.*random*()\*10)] = true;  }  }  } |

MineMap class that randomly places mines. It declares numMines, which means the total number of mines. newMinewMap() receives numMines as a parameter and repeats the for statement as many times as numMines. Math.random() generates an integer 0~9. This is the size of a two-dimensional grid.

1. **Gameboardpanel class**

|  |
| --- |
| public class GameBoardPanel extends JPanel {  private static final long *serialVersionUID* = 1L; // to prevent serial warning  // Define named constants for the game properties  public static final int *ROWS* = 10;  public static final int *COLS* = 10;  // Define named constants for UI sizes  public static final int *CELL\_SIZE* = 60; // Cell width and height, in pixels  public static final int *CANVAS\_WIDTH* = *CELL\_SIZE* \* *COLS*;  public static final int *CANVAS\_HEIGHT* = *CELL\_SIZE* \* *ROWS*;  // Define properties (package-visible)  /\*\* The game board composes of ROWSxCOLS cells \*/  static Cell *cells*[][] = new Cell[*ROWS*][*COLS*];  /\*\*numMines \*/  static int *numMine*;  public static void setlevel() {  //Difficulty level selection dialog  String[] buttons1 = {"Beginner", "Intermediate", "Advanced"};  int num = JOptionPane.*showOptionDialog*(null, "Select Level(Beginner, Intermediate, Advanced)", "Select Level",  JOptionPane.*YES\_NO\_CANCEL\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*, null, buttons1, "Beginner");  if(num == 0) { //Beginner  *numMine* = 5;  System.*out*.println("Level: Beginner");  }else if(num == 1){//Intermediate  *numMine* = 10;  System.*out*.println("Level: Intermediate");  }else {//Advanced  *numMine* = 15;  System.*out*.println("Level: Advanced");  }  } |

A setlevel() function that allows you to select the level of difficulty before starting the game. JOptionPane.showOptionDialog() converts Beginner, Intermediate, and Advanced to 0, 1, and 2 integer values, respectively. If number is 0, the easiest difficulty level is selected, so the number of mines becomes 5, and the selected difficulty is printed once. Or if number is 1, the medium difficulty level is selected, so the number of mines becomes 10, and if number is 2, it is the most difficult level, so the number of mines becomes 15, and the selected difficulty level is output once each.

|  |
| --- |
| public static void newGame() {  // Get a new mine map  int numMines = 0;  CellMouseListener.*countFlag* = 3;  *setlevel*();  MineMap mineMap = new MineMap();  mineMap.newMineMap(*numMine*);  // Reset cells, mines, and flags  for (int row = 0; row < *ROWS*; row++) {  for (int col = 0; col < *COLS*; col++) {  // Initialize each cell with/without mine  *cells*[row][col].newGame(mineMap.isMined[row][col], numMines);  }  }  //Timer initialize  View.*main*(null);  View.*count* = 60;  View.*isGameOver* = false;// Announcement GameOver  } |

NewGame() is executed when newGame button is pressed. First, set total number of mines to zero and reset the number of flags to three. Next, run the setlevel() function again to display a window where you can select the difficulty level. To generate mines randomly, we call the newMineMap() function by generating a mineMap class object. Finally, set the timer count for the View class to 60 seconds.

|  |
| --- |
| private int getSurroundingMines(int srcRow, int srcCol) {  int numMines = 0;  for (int row = srcRow - 1; row <= srcRow + 1; row++) {  for (int col = srcCol - 1; col <= srcCol + 1; col++) {  // Need to ensure valid row and column numbers too  if (row >= 0 && row < *ROWS* && col >= 0 && col < *COLS*) {  if (*cells*[row][col].isMined) numMines++;  }  }  }  return numMines;  } |

getSurroundingMines() sequentially searches cells in eight directions around the cell based on the current cell to find mines. Increase numMines one by one each time you find a mine. The range of the for statement is x-1<x<x+1 based on srcRow and srcCol.

|  |
| --- |
| private void revealCell(int srcRow, int srcCol) {  int numMines = getSurroundingMines(srcRow, srcCol);  *cells*[srcRow][srcCol].setText(numMines + "");  *cells*[srcRow][srcCol].isRevealed = true;  *cells*[srcRow][srcCol].paint(numMines); // based on isRevealed  if (numMines == 0) {  // Recursively reveal the 8 neighboring cells  for (int row = srcRow - 1; row <= srcRow + 1; row++) {  for (int col = srcCol - 1; col <= srcCol + 1; col++) {  // Need to ensure valid row and column numbers too  if (row >= 0 && row < *ROWS* && col >= 0 && col < *COLS*) {  if (!*cells*[row][col].isRevealed) revealCell(row, col);  }  }  }  }  } |

revalCell() is a function that causes cells that are consecutively blank to be reevaluated if the numMines of the cell is 0 when the player clicks a cell. If numMines of a cell is not 0, show a number in the cell. First, to determine the number of mines, we call the getSurroundingMines() function to store the return value in numMines, search in 8 directions, and reveal cells that are not mines.

|  |
| --- |
| public boolean checkPlayerWon(){  boolean checkp\_won = true;  for (int row = 0; row < *ROWS*; row++) {  for (int col = 0; col < *COLS*; col++) {  if(!(*cells*[row][col].isMined)) { //if not mines  if(!(*cells*[row][col].isRevealed)) checkp\_won = false;  }  }  }  return checkp\_won;  } |

checkPlayerWon() is checks whether the player has turned all the cells. In other words, it is a function that determines the player won the game. Since it is a function that simply judges, you can use the boolean function to return true or false. Use two for statements to search for cells, Returns false if the cell is not a landmine isRevealed.

|  |
| --- |
| private class CellMouseListener extends MouseAdapter {  static int *countFlag* = 3;  @Override  public void mouseClicked(MouseEvent e) {  Cell sourceCell = (Cell)e.getSource();    System.*out*.println("You clicked on (" + sourceCell.row + "," + sourceCell.col + ")");  // Left-click to reveal a cell;  if (e.getButton() == MouseEvent.*BUTTON1*) {  if (sourceCell.isMined) { //if hit a mine  System.*out*.println("Game Over");  View. *isGameOver* = true;  sourceCell.setText("\*");  //dialog  String[] buttons2 = {"New Game", "exit"};  int result =JOptionPane.*showOptionDialog*(null, "Game Over!", "Game Result", JOptionPane.*YES\_NO\_CANCEL\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*, null, buttons2, "New Game");  System.*out*.println("Game Result: " + result);  if(result==0){ //if new game  *newGame*();  }else {System.*exit*(0);} ///if exit  } |

Case 1: Left-click\_hit a mine

CellMouseListener class is Gameboardpanel’s Inner class. This class extends MouseAdapter. The countFlag variable representing the number of flags is initialized to 3. The CellMouseListener class receives the MouseEvent object and divides the event into three cases by mouse click.

If the left mouse is hit a mine, it is Game Over. If Game Over, JoptionPane.showOptionDialog is displayed. Choose whether to restart or end the game.

|  |
| --- |
| else { //if not hit a mine  revealCell(sourceCell.row, sourceCell.col);  if(checkPlayerWon()) { //if turned all cell  System.*out*.println("Player Won!!");  //dialog  String[] buttons3 = {"New Game", "exit"};  int result = JOptionPane.*showOptionDialog*(null, "Player Won!!", "Game Result", JOptionPane.*YES\_NO\_CANCEL\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*, null, buttons3, "Player Won!!");  System.*out*.println("Game Result: " + result);  if(result==0){  *newGame*();  }else {System.*exit*(0);}  }  } |

Case 2: Left-click\_not hit mine

If the left mouse is not hit a mine, call revealCell(). Then call checkPlayerWon() to make sure all cells are turned. If you have finished spinning all cells, display player won.

JOptionPane.showOptionDialog() and select whether to restart or end the game.

|  |
| --- |
| else if(e.getButton() == MouseEvent.*BUTTON3*) {  //Right-click to plant/remove the flag.  // If this cell is flagged, remove the flag  // else plant a flag.  while(*countFlag* >= 0) {//if have a flag  if(sourceCell.isFlagged) { //if flag  System.*out*.println("Remove a flag");  System.*out*.println("remain flag: " + *countFlag*);  sourceCell.setText("");  sourceCell.isFlagged = false;  break;  } else if (!(sourceCell.isFlagged)&*countFlag*>0) { //if not flag  System.*out*.println("Plant a flag");  sourceCell.setText(">");  sourceCell.isFlagged = true;  //check isMined  if(*cells*[sourceCell.row][sourceCell.col].isMined) {  revealCell(sourceCell.row, sourceCell.col);  System.*out*.println("You find a flag");  }  *countFlag--*;  System.*out*.println("remain flag: " + *countFlag*);  break;  }  *countFlag--*;  break;  }  if( *countFlag*<0){// if no flag  System.*out*.println("Flag missing!");  JOptionPane.*showMessageDialog*(null, "Flag missing!");  } |

Case 3: Right-click

If the Right mouse is plant or remove Flag. Reduce the count flag each time you plant the flag. When the countFlag is more than 0, if there is a flag in the cell, remove the flag. If there is no flag in the cell, plant the flag. When you plant a flag, call revealCell() if there was a mine in the cell.  
If the flag is used up, float the flag missing dialog.

1. **View class**

|  |
| --- |
| public class View extends JPanel{  private static final long *serialVersionUID* = 1L;  // count variable  static int *count* = 60;  int width = 75, height = 75;  int x = 260, y = 15;  private static JLabel *lbl*;  static boolean *isGameOver* = false;  public View(boolean *isGameOver*) { //create View label  View.*isGameOver= isGameOver;*  *lbl* = new JLabel();  *lbl*.setOpaque(true);  *lbl*.setBounds(x, y, width, height);  *lbl*.setForeground(Color.*BLUE*);  *lbl*.setText(*count* + "");  *lbl*.setFont(new Font("맑은고딕", Font.*PLAIN*, 50));  *lbl*.setHorizontalAlignment(JLabel.*CENTER*);    super.setLayout(null);  super.add(*lbl*);  super.setPreferredSize(new Dimension(100, 100));  }  public static void main(String[] args){  Timer timer=new Timer();  TimerTask task=new TimerTask(){  @Override  public void run() { //Auto-generated method stub  if(*count* >= 0){  System.*out*.println("[카운트다운 : "+*count*+"]");  l*bl*.setText(*count* + "");  *count*--;  }else{ //time over  timer.cancel();  System.*out*.println("[카운트다운 : 종료]");  //dialog  String[] buttons4 = {"New Game", "exit"};  int result = JOptionPane.*showOptionDialog*(null, "Time Over!", "Game Result", JOptionPane.*YES\_NO\_CANCEL\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*, null, buttons4, "Game Over!");  System.*out*.println("Game Result: " + result);  if(result==0){  GameBoardPanel.*newGame*();  }else {System.*exit*(0);}  }  if(*isGameOver*) {//if GameOver  timer.cancel(); //then, Stop timer  }  }  }  } |

View class is a function that operates the timer. There is a static count variable in the class. In the View constructor, you create a View label. Create Timer and TimerTask objects in the main() function to override run(). Decrease the count variable by 1 every second, and terminate the timer if the count variable becomes less than zero. Create a dialog to decide whether to start the game anew or end the game.

If the game ends before the timer ends (Game Over / Player Won), the timer ends with if(isGameOver).

1. **MineSweeperMain class**

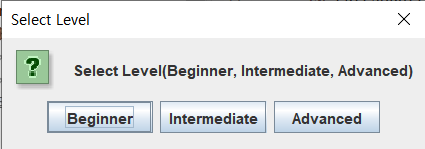
|  |
| --- |
| public class MineSweeperMain extends JFrame{  private static final long *serialVersionUID* = 1L; // to prevent serial warning  View timer = new View(); //create View Object  //create GameBoardPanel Object  GameBoardPanel board = new GameBoardPanel();  JButton btnNewGame = new JButton("New Game");//create JButton    // Constructor to set up all the UI and game components  public MineSweeperMain() {  Container cp = this.getContentPane(); // JFrame's content-pane  cp.setLayout(new BorderLayout()); // in 10x10 GridLayout    cp.add(timer, BorderLayout.*NORTH*); // Timer dispose  cp.add(board, BorderLayout.*CENTER*); // Grid dispose  cp.add(btnNewGame, BorderLayout.*SOUTH*); // newGame Button dispose  btnNewGame.addActionListener(new ActionListener() { // newGame Button action  @Override  public void actionPerformed(ActionEvent e) {  GameBoardPanel.*newGame*();  }  });  GameBoardPanel.*newGame*();  pack(); // Pack the UI components, instead of setSize()  setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // handle window-close button  setTitle("Mineswepper");  setVisible(true); // show it  }  public static void main(String[] args) {  new MineSweeperMain();  }  } |

The MineSweeper Main class places the generated Timer label and 10x10 Grid in the JFrame. It also creates a JButton and places a new Game Button. When positioning, place each with BorderLayout.NORTH/CENTER/SOUTH using the BorderLayout attribute.

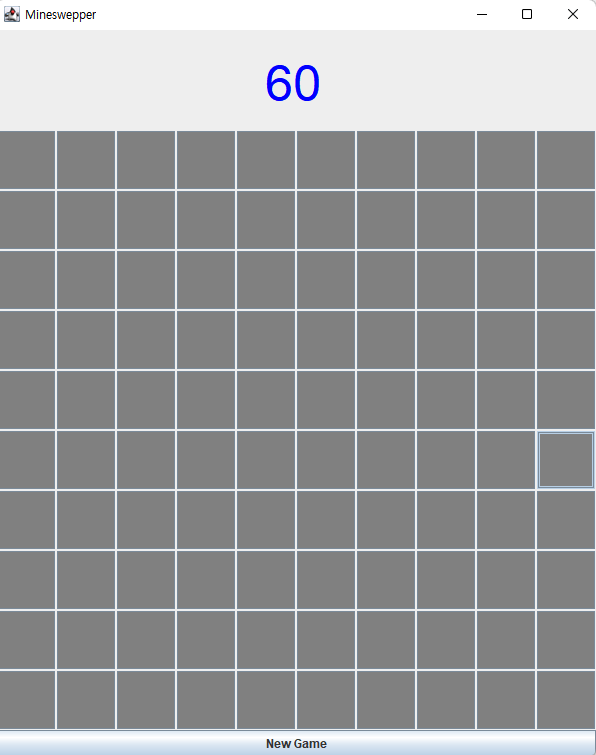
The newGameJButton uses the addActionListener() function to press Button to activate the newGame() on the GameBoard Panel.

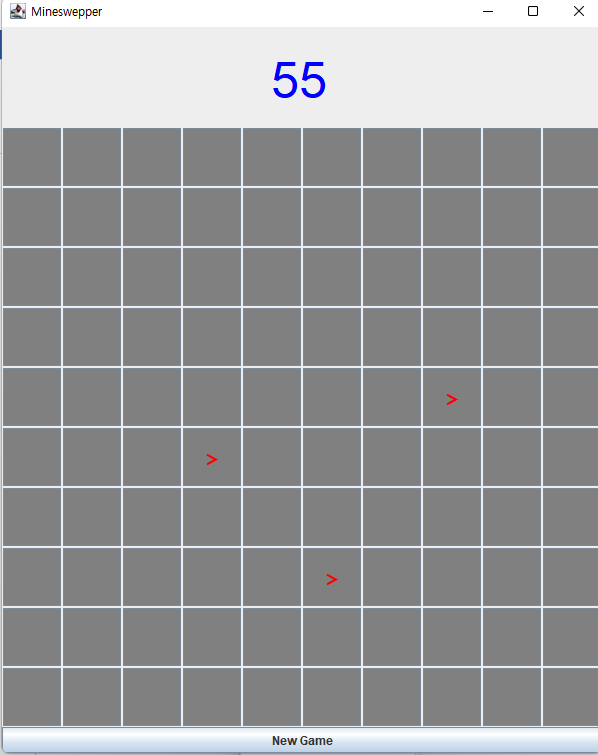
* Results/Output

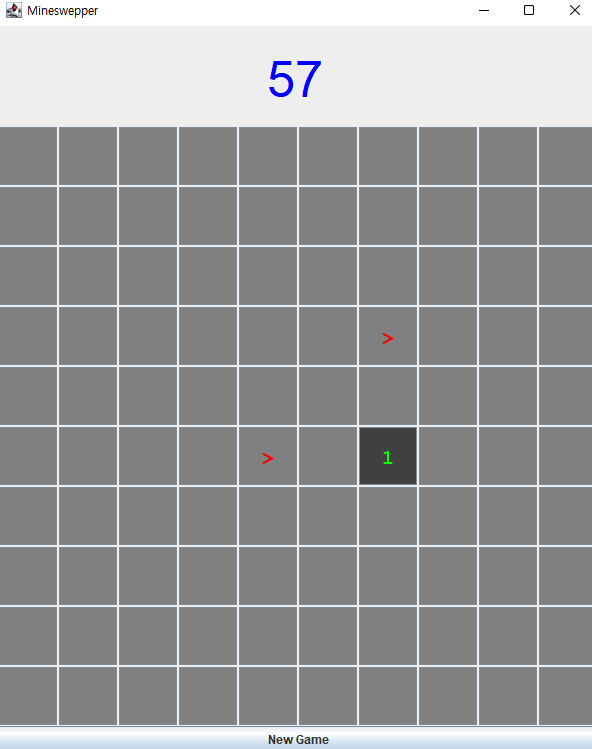
**Select Level dialog**



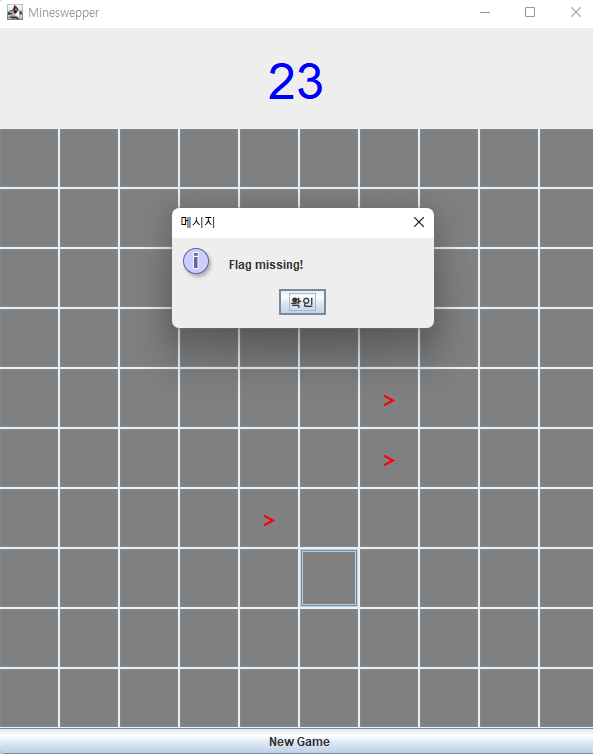
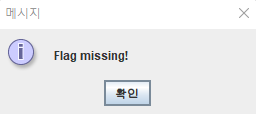
**Minesweeper Game Screen**

****

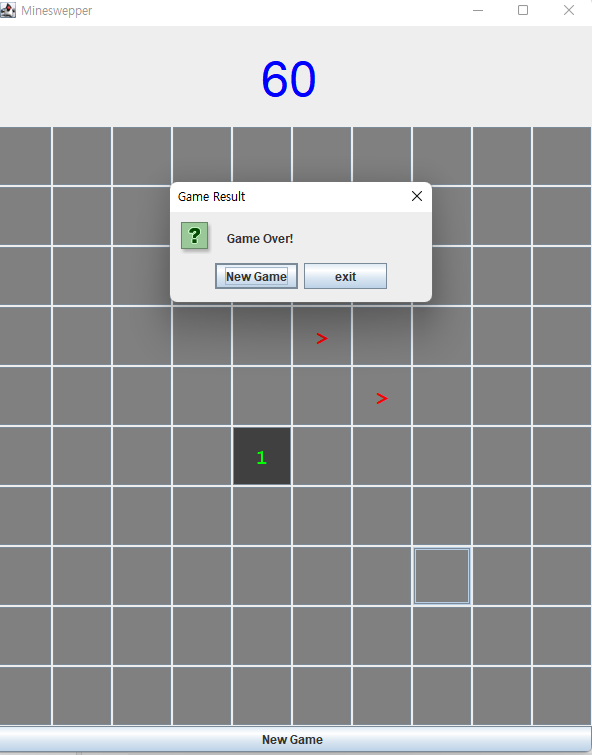
**Hit a mine(Reveal Flag) Plant Flag**

****

**Flag missing**

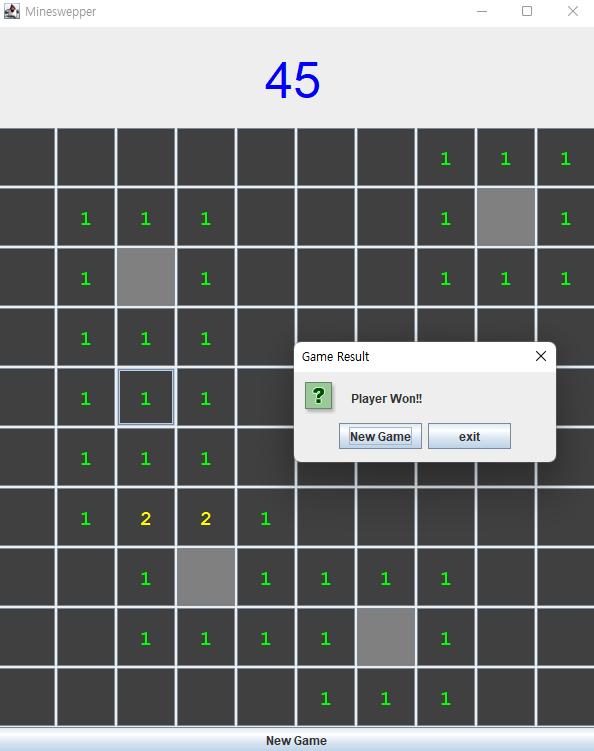
****

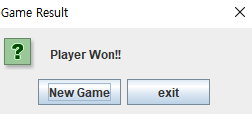
**Game Over**

****

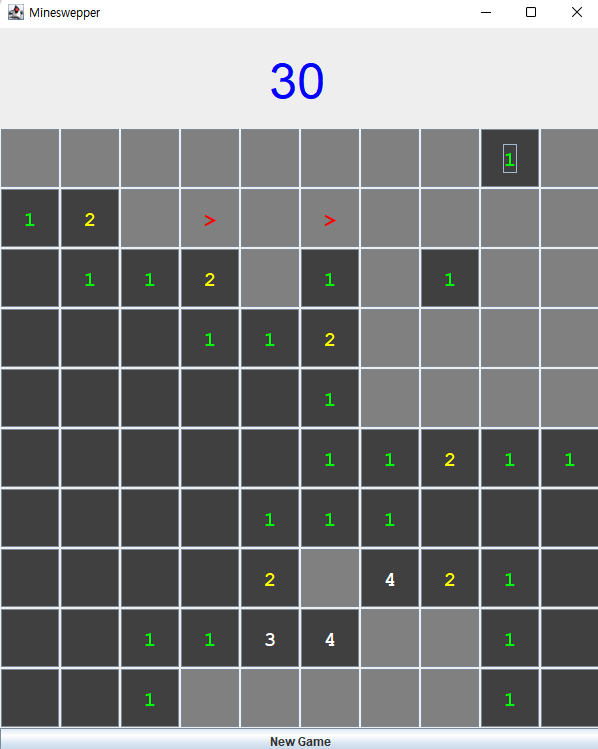
****

**player won**

****

****

**Total**

****

**Conclusion**

Through the Mine Sweeper Project, We can create a simple GUI program by collaborating with team members using Java. In this Project, All types of Java's inheritance concept and Java grammar learned so far are utilized and understand the grammar and library functions of Java FX. We were able to create a complete game by adding our own functions and GUI from the existing MineSweeper code.

First of all, we tried to understand the existing MineSweeper code. We focused on the relationship between various classes and each other. In addition, many factors had to be considered, such as overridden functions, access to parent classes in child classes, and Inner classes. The biggest challenge for us was the GUI. It was hard to understand GUI elements and attributes that everyone saw for the first time. Working together to implement the code and sharing each other's knowledge.

Through this project, we learned a lot of Java grammar and properties of Java FX that went into simple MineSweeper code implementation. We have also grown further through constant knowledge sharing with our team members. It was a really meaningful Team Project.