Minesweeper Project Report by “Triple J”

Project status: Off track: A little behind, but catching up.

Project report summary: We have understood the basic idea of the Minesweeper game. We have a working prototype, and we are figuring out how to implement 3 different levels of difficulty.

Project milestones: List and explain more details about what you have done, what you are working on, and future works in chronological order i.e. UML design, class A design, code class A, B, etc.). You can attach date-time (start, finish, etc.) to each milestone. Also, you can also make a note of the completion rate and the remaining % for each milestone.

1. UML Design: Working on adding 3 different levels of difficulty, and a leader board for each level. 50% remaining
2. Method to Flag the mines: 100% remaining
3. User menu: Decide as a team if the user will be able to pause, restart, save, reload each game.

Issues / Problems: “Flagging" the spots with mines like in the real game.

6. Current version of final project

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionListener;

import java.util.\*;

public class Minesweeper {

// The value assigned to cells marked as mines. 10 works

// because no cell will have more than 8 neighboring mines.

private static final int MINE = 10;

// The size in pixels for the frame.

private static final int SIZE = 500;

// The number of mines at generated is the grid size \* this constant

private static final double POPULATION\_CONSTANT = 1.5;

// This fixed amount of memory is to avoid repeatedly declaring

// new arrays every time a cell's neighbors are to be retrieved.

private static Cell[] reusableStorage = new Cell[8];

private int gridSize;

private Cell[][] cells;

private JFrame frame;

private JButton reset;

private JButton giveUp;

private final ActionListener actionListener = actionEvent -> {

Object source = actionEvent.getSource();

if (source == reset) {

createMines();

} else if (source == giveUp) {

revealBoardAndDisplay("You gave up.");

} else {

handleCell((Cell) source);

}

};

private class Cell extends JButton {

private final int row;

private final int col;

private int value;

Cell(final int row, final int col,

final ActionListener actionListener) {

this.row = row;

this.col = col;

addActionListener(actionListener);

setText("");

}

int getValue() {

return value;

}

void setValue(int value) {

this.value = value;

}

boolean isAMine() {

return value == MINE;

}

void reset() {

setValue(0);

setEnabled(true);

setText("");

}

void reveal() {

setEnabled(false);

setText(isAMine() ? "X" : String.valueOf(value));

}

void updateNeighbourCount() {

getNeighbours(reusableStorage);

for (Cell neighbour : reusableStorage) {

if (neighbour == null) {

break;

}

if (neighbour.isAMine()) {

value++;

}

}

}

void getNeighbours(final Cell[] container) {

// Empty all elements first

for (int i = 0; i < reusableStorage.length; i++) {

reusableStorage[i] = null;

}

int index = 0;

for (int rowOffset = -1; rowOffset <= 1; rowOffset++) {

for (int colOffset = -1; colOffset <= 1; colOffset++) {

// Make sure that we don't count ourselves

if (rowOffset == 0 && colOffset == 0) {

continue;

}

int rowValue = row + rowOffset;

int colValue = col + colOffset;

if (rowValue < 0 || rowValue >= gridSize

|| colValue < 0 || colValue >= gridSize) {

continue;

}

container[index++] = cells[rowValue][colValue];

}

}

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass())

return false;

Cell cell = (Cell) obj;

return row == cell.row &&

col == cell.col;

}

@Override

public int hashCode() {

return Objects.hash(row, col);

}

}

private Minesweeper(final int gridSize) {

this.gridSize = gridSize;

cells = new Cell[gridSize][gridSize];

frame = new JFrame("Minesweeper");

frame.setSize(SIZE, SIZE);

frame.setLayout(new BorderLayout());

initializeButtonPanel();

initializeGrid();

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

private void initializeButtonPanel() {

JPanel buttonPanel = new JPanel();

reset = new JButton("Reset");

giveUp = new JButton("Give Up");

reset.addActionListener(actionListener);

giveUp.addActionListener(actionListener);

buttonPanel.add(reset);

buttonPanel.add(giveUp);

frame.add(buttonPanel, BorderLayout.SOUTH);

}

private void initializeGrid() {

Container grid = new Container();

grid.setLayout(new GridLayout(gridSize, gridSize));

for (int row = 0; row < gridSize; row++) {

for (int col = 0; col < gridSize; col++) {

cells[row][col] = new Cell(row, col, actionListener);

grid.add(cells[row][col]);

}

}

createMines();

frame.add(grid, BorderLayout.CENTER);

}

private void resetAllCells() {

for (int row = 0; row < gridSize; row++) {

for (int col = 0; col < gridSize; col++) {

cells[row][col].reset();

}

}

}

private void createMines() {

resetAllCells();

final int mineCount = (int) POPULATION\_CONSTANT \* gridSize;

final Random random = new Random();

// Map all (row, col) pairs to unique integers

Set<Integer> positions = new HashSet<>(gridSize \* gridSize);

for (int row = 0; row < gridSize; row++) {

for (int col = 0; col < gridSize; col++) {

positions.add(row \* gridSize + col);

}

}

// Initialize mines

for (int index = 0; index < mineCount; index++) {

int choice = random.nextInt(positions.size());

int row = choice / gridSize;

int col = choice % gridSize;

cells[row][col].setValue(MINE);

positions.remove(choice);

}

// Initialize neighbor counts

for (int row = 0; row < gridSize; row++) {

for (int col = 0; col < gridSize; col++) {

if (!cells[row][col].isAMine()) {

cells[row][col].updateNeighbourCount();

}

}

}

}

private void handleCell(Cell cell) {

if (cell.isAMine()) {

cell.setForeground(Color.RED);

cell.reveal();

revealBoardAndDisplay("You clicked on a mine!");

return;

}

if (cell.getValue() == 0) {

Set<Cell> positions = new HashSet<>();

positions.add(cell);

cascade(positions);

} else {

cell.reveal();

}

checkForWin();

}

private void revealBoardAndDisplay(String message) {

for (int row = 0; row < gridSize; row++) {

for (int col = 0; col < gridSize; col++) {

if (!cells[row][col].isEnabled()) {

cells[row][col].reveal();

}

}

}

JOptionPane.showMessageDialog(

frame, message, "Game Over",

JOptionPane.ERROR\_MESSAGE

);

createMines();

}

private void cascade(Set<Cell> positionsToClear) {

while (!positionsToClear.isEmpty()) {

// Set does not have a clean way for retrieving

// a single element. This is the best way I could think of.

Cell cell = positionsToClear.iterator().next();

positionsToClear.remove(cell);

cell.reveal();

cell.getNeighbours(reusableStorage);

for (Cell neighbour : reusableStorage) {

if (neighbour == null) {

break;

}

if (neighbour.getValue() == 0

&& neighbour.isEnabled()) {

positionsToClear.add(neighbour);

} else {

neighbour.reveal();

}

}

}

}

private void checkForWin() {

boolean won = true;

outer:

for (Cell[] cellRow : cells) {

for (Cell cell : cellRow) {

if (!cell.isAMine() && cell.isEnabled()) {

won = false;

break outer;

}

}

}

if (won) {

JOptionPane.showMessageDialog(

frame, "You have won!", "Congratulations",

JOptionPane.INFORMATION\_MESSAGE

);

}

}

private static void run(final int gridSize) {

try {

// Totally optional. But this applies the look and

// feel for the current OS to the a application,

// making it look native.

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception ignore) { }

// Launch the program

new Minesweeper(gridSize);

}

public static void main(String[] args) {

final int gridSize = 10;

SwingUtilities.invokeLater(() -> Minesweeper.run(gridSize));

}

}