Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Разработка GUI приложения Сапер и его решателя на языке Java

Выполнил

студент гр. 3530901/80003\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дзюба Б.А.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глухих М.И.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург   
2019

**Содержание**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc26912944)

[**МЕТОД РЕШЕНИЯ** 4](#_Toc26912945)

[**ЛИСТИНГ ПРОГРАММЫ** 7](#_Toc26912946)

[**JavaSweeper.java** 7](#_Toc26912947)

[**InitialDialog.java** 10](#_Toc26912948)

[**Box.java** 13](#_Toc26912949)

[**Coord.java** 14](#_Toc26912950)

[**Ranges.java** 16](#_Toc26912951)

[**Game.java** 17](#_Toc26912952)

[**Matrix.java** 22](#_Toc26912953)

[**Bomb.java** 23](#_Toc26912967)

[**Flag.java** 25](#_Toc26912968)

[**GameState.java** 27](#_Toc26912969)

[**SolverFrame.java** 28](#_Toc26912969)

[**Solver.java** 31](#_Toc26912969)

[**MoveType.java** 37](#_Toc26912969)

[**MinesGroup.java** 38](#_Toc26912969)

[**ТЕСТЫ** 40](#_Toc26912970)

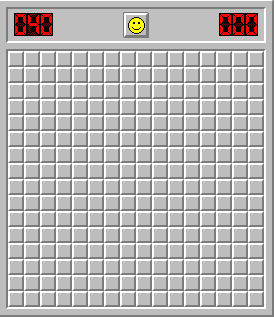
[**СКРИНШОТЫ ПРОГРАММЫ** 44](#_Toc26912971)

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Создать игру Сапер, а также его решатель.

Сапер – компьютерная игра-головоломка. Плоское или объёмное игровое поле разделено на смежные ячейки (квадраты, шестиугольники, кубы и т. п.), некоторые из которых «заминированы»; количество «заминированных» ячеек известно. Целью игры является открытие всех ячеек, не содержащих мины.

Игрок открывает ячейки, стараясь не открыть ячейку с миной. Открыв ячейку с миной, он проигрывает. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано» (в каждом варианте игры соседство определяется по-своему); используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры. «Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки, игрок выигрывает.



GitHub репозиторий проекта: https://github.com/Bogunto14/JavaSweeper

# **МЕТОД РЕШЕНИЯ**

Код разделен на три пакета классов: sweeper, solver.

Класс Main.java – запускает главное окно нашего приложения.

Содержимое пакета sweeper:

* JavaSweeper.java – отвечает за графическое представление основного окна приложения.
* InitialDialog.java – отвечает за графическое представление начального окна с выбором параметров поля.
* Box.java – перечисление для работы с картинками.
* Coord.java – класс для координат клеток.
* Ranges.java – отвечает за хранение размера поля и некоторые операции с ним.
* Game.java – отвечает за саму игру.
* Matrix.java – отвечает за матрицу с бомбами или флагами.
* Bomb.java – отвечает за все операции с бомбами.
* Flag.java – отвечает за все операции с флагами и открытыми клетками.
* GameState.java – отвечает за состояния игры.

Cодержимое пакета solver:

* SolverFrame.java – отвечает за графическое представление окна для запуска решателя.
* Solver.java – алгоритм решателя игры.
* MoveType.java – отвечает за типы ходов.
* MinesGroup.java – отвечает за создание групп мин для алгоритма.

Также имеются изображения, помещённые в пакет img.

Более подробная работа программы:

Пользователь в начальном окне с параметрами поля может задать количество бомб, столбцов и строк, и запустить игру. В окне решателя игры пользователь может включить автоматическое решение сапера (в любой момент может остановить), и, чтобы решатель сделал один ход. Также в окне представлено сообщение о предыдущем ходе: был он логическим, вероятностным или случайным.

Можно играть самому, используя клавиши компьютерной мыши, или же запустить решателя.

Логика решателя имеет два алгоритма, которые используются в зависимости от ситуации на поле. Основной алгоритм позволяет находить все ячейки со 100- и 0-процентной вероятностью нахождения мины.

Основной алгоритм состоит в следующем. Неизвестные ячейки, прилегающие к одной открытой ячейке формируются в группу, в которую записывается также значение ячейки, к которой она прилегает.

Для начала нужно преобразовать группы. Для этого:

1. Сравниваем каждую группу с каждой последующей группой.
2. Если группы одинаковые, то вторую удаляем.
3. Если одна группа содержит другую, то вычитаем из большей меньшую.
4. Пересекающиеся группы преобразовываем по следующему принципу:
   1. Создаем новую группу из пересекающихся ячеек.
   2. Рассчитываем количество мин в новой группе, равное количеству мин в группе с большим числом мин минус оставшееся количество ячеек в другой группе после отделения пересекающихся ячеек.
   3. Если рассчитанное количество мин в новой группе не равно количеству мин в группе с меньшим количеством мин, то прекращаем преобразование.
   4. Вычитаем из обоих пересекающихся групп новообразованную группу.
   5. Добавляем новообразованную группу в список групп.
5. Повторяем с п. 1 до тех пор, пока не будет производиться никаких изменений.

В итоге у нас получатся три вида групп:

* количество мин равно нулю
* количество мин равно количеству ячеек в группе
* ненулевое количество мин меньше количества ячеек в группе

Соответственно все ячейки из первой группы можно смело открывать, а из второй группы отмечать. В этом суть основного алгоритма.

Но часто встречаются ситуации, когда нет достоверного решения ситуации. Тогда на помощь приходит следующий алгоритм. Его суть состоит в использовании теории вероятности. Алгоритм работает так: определение вероятности в отдельных ячейках, учитывая влияние нескольких открытых ячеек.

Для вычисления вероятности нахождения мины в ячейке рядом с несколькими открытыми ячейками используем формулу расчета вероятности хотя бы одного события:

Вероятность наступления события А, состоящего в появлении хотя бы одного из событий А1, А2,..., Аn, независимых в совокупности, равна разности между единицей и произведением вероятностей противоположных событий. А=1- (1-A1)\*(1-A2)\*....\*(1-An).

Остается только выбрать одну из ячеек с минимальной вероятностью и сделать ход.

# **ЛИСТИНГ ПРОГРАММЫ**

## **JavaSweeper.java**

## package sweeper;

## import javax.swing.\*;

## import java.awt.\*;

## import java.awt.event.MouseAdapter;

## import java.awt.event.MouseEvent;

## public class JavaSweeper extends JFrame {

## private Game game;

## private JPanel panel;

## private final int IMAGE\_SIZE = 50;

## private static InitialDialog dialog;

## public static void main(String[] args) {

## new JavaSweeper();

## }

## public JavaSweeper(){

## dialog = new InitialDialog(this);

## setImages();

## restartGame(dialog.getColsN(), dialog.getRowsN(), dialog.getBombsN());

## showRestartDialog("Начальные параметры");

## }

## static void showRestartDialog(String text) {

## dialog.setText(text);

## dialog.setVisible(true);

## }

## void restartGame(int cols, int rows, int bombs) {

## game = new Game(cols, rows, bombs);

## game.start();

## initPanel();

## initFrame();

## }

## private void initPanel() {

## panel = new JPanel() {

## @Override

## public void paintComponent(Graphics g){

## super.paintComponent(g);

## for (Coord coord : Ranges.getAllCoords()) {

## g.drawImage((Image) game.getBox(coord).image,

## coord.x \* IMAGE\_SIZE,

## coord.y \* IMAGE\_SIZE, this);

## }

## }

## };

## panel.addMouseListener(new MouseAdapter() {

## @Override

## public void mousePressed(MouseEvent e) {

## int x = e.getX() / IMAGE\_SIZE;

## int y = e.getY() / IMAGE\_SIZE;

## Coord coord = new Coord(x,y);

## if (!Ranges.inRange(coord)) return;

## if (e.getButton() == MouseEvent.BUTTON1)

## game.pressLeftButton(coord);

## if (e.getButton() == MouseEvent.BUTTON3)

## game.pressRightButton(coord);

## if (e.getButton() == MouseEvent.BUTTON2)

## restartGame(dialog.getColsN(), dialog.getRowsN(), dialog.getBombsN());

## panel.repaint();

## }

## });

## panel.setPreferredSize(new Dimension(

## Ranges.getSize().x \* IMAGE\_SIZE,

## Ranges.getSize().y \* IMAGE\_SIZE));

## add (panel);

## }

## public JPanel getPanel() {

## return panel;

## }

## public void drawClickShadow(Coord coord, Color color) {

## Graphics g = panel.getGraphics();

## g.setColor(new Color(color.getRed(), color.getGreen(), color.getBlue(), 200));

## g.drawRect(coord.x \* IMAGE\_SIZE, coord.y \* IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_SIZE);

## }

## private void initFrame() {

## setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

## setTitle("Сапёр");

## setResizable(false);

## setVisible(true);

## pack();

## setLocationRelativeTo(null);

## setIconImage(getImage("icon"));

## }

## private void setImages(){

## for (Box box : Box.values())

## box.image = getImage(box.name().toLowerCase());

## }

## private Image getImage (String name) {

## String filename = "/img/" + name + ".png";

## ImageIcon icon = new ImageIcon(getClass().getResource(filename));

## return icon.getImage();

## }

## public Game getGame() {

## return game;

## }

## }

## **InitialDialog.java**

## package sweeper;

## import javax.swing.\*;

## import java.awt.\*;

## import java.awt.event.ActionEvent;

## class InitialDialog extends JDialog{

## private final static int BOMBS\_N = 10;

## private final static int COLS\_N = 9;

## private final static int ROWS\_N = 9;

## private JLabel label;

## private JSpinner bombs;

## private JSpinner cols;

## private JSpinner rows;

## private JPanel panel;

## private JavaSweeper parent;

## private Font font;

## InitialDialog(JavaSweeper parent) {

## super(parent, "Новая игра", true);

## this.parent = parent;

## font = new Font(Font.SERIF, Font.PLAIN, 20);

## setFont(font);

## label = new JLabel("", SwingConstants.LEFT);

## label.setHorizontalTextPosition(SwingConstants.LEFT);

## label.setFont(font);

## bombs = new JSpinner(new SpinnerNumberModel(BOMBS\_N, 1, 1000, 1));

## bombs.setFont(font);

## cols = new JSpinner(new SpinnerNumberModel(COLS\_N, 1, 100, 1));

## cols.setFont(font);

## rows = new JSpinner(new SpinnerNumberModel(ROWS\_N, 1, 100, 1));

## rows.setFont(font);

## initPanel();

## getContentPane().add(panel);

## setDefaultCloseOperation(HIDE\_ON\_CLOSE);

## pack();

## setSize(400, 220);

## setLocationRelativeTo(null);

## }

## void setText(String text) {

## label.setText(text);

## }

## private void initPanel() {

## panel = new JPanel();

## panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

## panel.add(label);

## JPanel subPanel = new JPanel(new GridLayout(3, 2));

## JLabel tmp = new JLabel("Количество бомб");

## tmp.setFont(font);

## subPanel.add(tmp);

## subPanel.add(bombs);

## tmp = new JLabel("Количество столбцов");

## tmp.setFont(font);

## subPanel.add(tmp);

## subPanel.add(cols);

## tmp = new JLabel("Количество строк");

## tmp.setFont(font);

## subPanel.add(tmp);

## subPanel.add(rows);

## panel.add(subPanel);

## JButton redeem = new JButton("ОК");

## redeem.setFont(font);

## redeem.addActionListener(this::redeemClicked);

## panel.add(redeem);

## }

## private void redeemClicked(ActionEvent e) {

## int bombsN = getBombsN();

## int colsN = getColsN();

## int rowsN = getRowsN();

## if (bombsN >= colsN \* rowsN) {

## label.setText("Слишком много бомб!");

## return;

## }

## parent.restartGame(getColsN(), getRowsN(), getBombsN());

## setVisible(false);

## label.setText("Начальные параметры");

## }

## int getBombsN() {

## return (int) bombs.getValue();

## }

## int getColsN() {

## return (int) cols.getValue();

## }

## int getRowsN() {

## return (int) rows.getValue();

## }

## }

## **Box.java**

package sweeper;

public enum Box {

ZERO,

NUM1,

NUM2,

NUM3,

NUM4,

NUM5,

NUM6,

NUM7,

NUM8,

BOMB,

OPENED,

CLOSED,

FLAGED,

BOMBED,

NOBOMB;

public Object image;

Box getNextNumberBox() {

return Box.values()[this.ordinal() + 1];

}

int getNumber() {

return this.ordinal();

}

}

## **Coord.java**

package sweeper;

public class Coord {

int x, y;

public Coord (int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

@Override

public int hashCode() {

return ~x ^ y;

}

@Override

public boolean equals(Object o) {

if (o == this)

return true;

if (o instanceof Coord){

Coord to = (Coord)o;

return to.x == x && to.y == y;

}

return super.equals(o);

}

@Override

public String toString() {

return "(" + x + "|" + y + ")";

}

}

## **Ranges.java**

## package sweeper;

## import java.util.ArrayList;

## import java.util.Random;

## public class Ranges {

## private static Coord size;

## private static ArrayList<Coord> allCoords;

## private static Random random = new Random();

## 

## public static void setSize(Coord \_size) {

## size = \_size;

## allCoords = new ArrayList<>();

## for (int y = 0 ; y < size.y; y++)

## for (int x = 0; x < size.x; x++)

## allCoords.add(new Coord(x, y));

## }

## public static Coord getSize() {

## return size;

## }

## public static ArrayList<Coord> getAllCoords() {

## return allCoords;

## }

## static boolean inRange(Coord coord){

## return coord.x >= 0 && coord.x < size.x &&

## coord.y >= 0 && coord.y < size.y;

## }

## public static Coord getRandomCoord() {

## return new Coord (random.nextInt(size.x),random.nextInt(size.y));

## }

## public static ArrayList<Coord> getCoordsAround(Coord coord){

## Coord around;

## ArrayList<Coord> list = new ArrayList<>();

## for (int x = coord.x - 1; x <= coord.x + 1; x++)

## for (int y = coord.y - 1; y <= coord.y + 1; y++)

## if (inRange(around = new Coord(x,y)))

## if (!around.equals(coord))

## list.add(around);

## return list;

## }}

## **Game.java**

package sweeper;

public class Game {

private Bomb bomb;

private Flag flag;

private GameState state;

private boolean firstStep;

public Game(Bomb bombs) {

bomb = bombs;

flag = new Flag();

}

public GameState getState() {

return state;

}

Game(int cols, int rows, int bombs) {

Ranges.setSize(new Coord(cols, rows));

bomb = new Bomb(bombs);

flag = new Flag();

}

void start() {

bomb.start();

flag.start();

firstStep = true;

state = GameState.PLAYED;

}

public void testStart() {

flag.start();

state = GameState.PLAYED;

}

Box getBox(Coord coord) {

if (flag.get(coord) == Box.OPENED)

return bomb.get(coord);

else

return flag.get(coord);

}

public Bomb getBomb() {

return bomb;

}

public Flag getFlag() {

return flag;

}

private boolean isFirstStep() {

return firstStep;

}

public void pressLeftButton(Coord coord) {

if (gameOver()) return;

if (isFirstStep()) {

while (bomb.get(coord) == Box.BOMB)

bomb.start();

openBox(coord);

firstStep = false;

} else

openBox(coord);

checkWinner();

}

private void checkWinner() {

if (state == GameState.PLAYED)

if (flag.getCountOfClosedBoxes() == bomb.getTotalBombs())

state = GameState.WINNER;

gameOver();

}

private void openBox (Coord coord) {

if(flag.get(coord) == null) return;

switch (flag.get(coord)) {

case OPENED: setOpenedToClosedBoxesAroundNumber(coord); return;

case FLAGED: return;

case CLOSED:

switch (bomb.get(coord)) {

case ZERO: openBoxesAround(coord); return;

case BOMB: openBombs(coord); return;

default : flag.setOpenedToBox(coord);

}

}

}

private void setOpenedToClosedBoxesAroundNumber(Coord coord) {

if (bomb.get(coord) != Box.BOMB)

if (flag.getCountOfFlagedBoxesAround(coord) == bomb.get(coord).getNumber())

for (Coord around : Ranges.getCoordsAround(coord))

if (flag.get(around) == Box.CLOSED)

openBox(around);

}

private void openBombs(Coord bombed) {

state = GameState.BOMBED;

flag.setBombedToBox(bombed);

for (Coord coord : Ranges.getAllCoords())

if (bomb.get(coord) == Box.BOMB)

flag.setOpenedToClosedBombBox(coord);

else

flag.setNobombToFlagedSafeBox(coord);

}

private void openBoxesAround(Coord coord) {

flag.setOpenedToBox(coord);

for (Coord around : Ranges.getCoordsAround(coord))

if(flag.get(around) != Box.OPENED)

openBox(around);

}

public void pressRightButton(Coord coord) {

if (gameOver()) return;

flag.toggleFlagedToBox(coord);

}

private boolean gameOver() {

switch (state) {

case PLAYED:

return false;

case BOMBED:

JavaSweeper.showRestartDialog("Вы проиграли");

break;

case WINNER:

JavaSweeper.showRestartDialog("Вы выиграли!");

break;

}

return true;

}

}

## **Matrix.java**

## package sweeper;

## public class Matrix {

## private Box[][] matrix;

## public Matrix(Box defaultBox){

## matrix = new Box[Ranges.getSize().x][Ranges.getSize().y];

## for (Coord coord : Ranges.getAllCoords())

## matrix [coord.x] [coord.y] = defaultBox;

## }

## Box get(Coord coord){

## if (Ranges.inRange(coord))

## return matrix[coord.x][coord.y];

## return null;

## }

## public void set(Coord coord, Box box){

## if (Ranges.inRange(coord))

## matrix [coord.x][coord.y] = box;

## }

## }

## **Bomb.java**

package sweeper;

public class Bomb {

private Matrix bombMap;

private int totalBombs;

Bomb (int totalBombs){

this.totalBombs = totalBombs;

bombMap = new Matrix(Box.ZERO);

}

public Bomb(int totalBombs, Matrix bombs) {

this.totalBombs = totalBombs;

bombMap = bombs;

}

void start (){

bombMap = new Matrix(Box.ZERO);

for (int j = 0; j < totalBombs; j ++)

placeBomb();

}

Box get (Coord coord){

return bombMap.get(coord);

}

public int bombsN(Coord coord){

return get(coord).getNumber();

}

private void placeBomb(){

while (true) {

Coord coord = Ranges.getRandomCoord();

if (Box.BOMB == bombMap.get(coord))

continue;

placeBombAt(coord);

break;

}

}

private void placeBombAt(Coord coord) {

bombMap.set(coord, Box.BOMB);

incNumbersAroundBomb(coord);

}

private void incNumbersAroundBomb (Coord coord){

for (Coord around : Ranges.getCoordsAround(coord))

if (Box.BOMB != bombMap.get(around))

bombMap.set(around, bombMap.get(around).getNextNumberBox());

}

public int getTotalBombs() {

return totalBombs;

}

}

## **Flag.java**

package sweeper;

public class Flag {

private Matrix flagMap;

private int countOfClosedBoxes;

void start() {

flagMap = new Matrix(Box.CLOSED);

countOfClosedBoxes = Ranges.getSize().x \* Ranges.getSize().y;

}

public Box get(Coord coord) {

return flagMap.get(coord);

}

void setOpenedToBox(Coord coord) {

flagMap.set(coord, Box.OPENED);

countOfClosedBoxes--;

}

void toggleFlagedToBox(Coord coord) {

switch (flagMap.get(coord)){

case FLAGED: setClosedToBox(coord); break;

case CLOSED: setFlagedToBox(coord); break;

}

}

private void setClosedToBox(Coord coord) {

flagMap.set(coord, Box.CLOSED);

}

private void setFlagedToBox(Coord coord) {

flagMap.set(coord, Box.FLAGED);

}

public int getCountOfClosedBoxes() {

return countOfClosedBoxes;

}

void setBombedToBox(Coord coord) {

flagMap.set(coord, Box.BOMBED);

}

void setOpenedToClosedBombBox(Coord coord) {

if (flagMap.get(coord) == Box.CLOSED)

flagMap.set(coord, Box.OPENED);

}

void setNobombToFlagedSafeBox(Coord coord) {

if (flagMap.get(coord) == Box.FLAGED)

flagMap.set(coord, Box.NOBOMB);

}

int getCountOfFlagedBoxesAround(Coord coord) {

int count = 0;

for (Coord around : Ranges.getCoordsAround(coord))

if (flagMap.get(around) == Box.FLAGED)

count++;

return count;

}}

## **GameState.java**

package sweeper;

public enum GameState {

PLAYED,

BOMBED,

WINNER

}

## **SolverFrame.java**

package solver;

import sweeper.Coord;

import sweeper.Game;

import sweeper.GameState;

import sweeper.JavaSweeper;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class SolverFrame extends JFrame implements ActionListener, Solver.CoordAction {

private JavaSweeper gameWindow;

private JButton nextMoveButton;

private JButton solveButton;

private JLabel lastStepLabel;

private Solver solver = new Solver(this);

private boolean solving = false;

private Timer timer = new Timer(100, this);

public static void main(String[] args) {

new SolverFrame();

}

private SolverFrame(){

super("Решатель");

JPanel panel = new JPanel(new GridLayout(3, 1));

nextMoveButton = new JButton("Следующий ход");

nextMoveButton.addActionListener(this);

solveButton = new JButton("");

solveButton.addActionListener(this);

lastStepLabel = new JLabel("Шагов ещё не было");

lastStepLabel.setHorizontalAlignment(SwingConstants.CENTER);

panel.add(nextMoveButton);

panel.add(solveButton);

panel.add(lastStepLabel);

add(panel);

panel.setVisible(true);

setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

setSize(250, 150);

gameWindow = new JavaSweeper();

setLocationRelativeTo(gameWindow);

solved();

timer.start();

setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e){

Game state = gameWindow.getGame();

solver.update(state);

Object source = e.getSource();

if (source == nextMoveButton) {

lastStepLabel.setText(solver.step().getMessage());

} else if (source == solveButton) {

if (solving) {

solved();

} else {

nextMoveButton.setEnabled(false);

solveButton.setText("Остановить решателя");

solving = true;

}

} else if (source == timer) {

if (solving) {

lastStepLabel.setText(solver.step().getMessage());

if (state.getState() != GameState.PLAYED) {

solved();

}

}

}

}

private void solved() {

nextMoveButton.setEnabled(true);

solving = false;

solveButton.setText("Начать решение");

}

@Override

public void performCoordAction(Coord coord) {

gameWindow.drawClickShadow(coord, Color.RED);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

gameWindow.getPanel().repaint();

}

}

## **Solver.java**

package solver;

import sweeper.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import java.util.Set;

public class Solver {

private Game state;

private Bomb bombs;

private Flag flags;

private CoordAction action;

public Solver(CoordAction action){

this.action = action;

}

public interface CoordAction {

void performCoordAction(Coord coord);

}

public void update(Game newstate){

assert newstate != null;

state = newstate;

bombs = newstate.getBomb();

flags = newstate.getFlag();

}

private List<MinesGroup> genGroups() {

List<MinesGroup> groups = new ArrayList<>();

for (Coord coord : Ranges.getAllCoords()) {

if (flags.get(coord) == Box.OPENED) {

int number = bombs.bombsN(coord) - countCellsAround(coord);

if (number == 0) continue;

MinesGroup group = new MinesGroup(number);

Set<Coord> coords = group.group;

for (Coord around : Ranges.getCoordsAround(coord)) {

if (flags.get(around) == Box.CLOSED) {

coords.add(around);

}

}

groups.add(group);

}

}

boolean changes;

do {

changes = false;

for (int i = 0; i < groups.size(); i++) { // Проходим по списку групп

MinesGroup groupI = groups.get(i);

for (int j = i + 1; j < groups.size(); j++) { // Сравниваем ее с остальными группами

MinesGroup groupJ = groups.get(j);

if (groupI.equals(groupJ)) { // Если группы одинаковые, то вторую удаляем

groups.remove(j--);

continue;

}

MinesGroup parent; // Большая группа

MinesGroup child; // Меньшая группа

if (groupI.group.size() > groupJ.group.size()) {

parent = groupI;

child = groupJ;

} else {

parent = groupJ;

child = groupI;

}

if (parent.contains(child)) { // Если большая группа содержит меньшую, то вычитаем из большей меньшую

parent.minusAssign(child);

changes = true; // Фиксируем факт изменения групп

}

// Пересекающиеся группы будут устранены автоматически

}

}

} while (changes); // Повторяем до тех пор, пока не будет производиться никаких изменений

return groups;

}

private boolean logicalMove(List<MinesGroup> groups) {

for (MinesGroup group : groups) {

if (group.minesAround == 0) { // Количество мин равно нулю

openBox(group.group.iterator().next());

return true;

} else if (group.minesAround == group.group.size()) { // Количество мин равно количеству ячеек в группе

flagBox(group.group.iterator().next());

return true;

}

}

return false;

}

private boolean probabilityMove(List<MinesGroup> groups) {

Coord size = Ranges.getSize();

double[][] grid = new double[size.getX()][size.getY()];

int countBombs = 0;

for (MinesGroup group : groups) { // Нахождение вероятностей для каждой клетки с возможной миной

for (Coord coord : group) {

double prob = ((double) group.minesAround) / group.group.size();

double old = grid[coord.getX()][coord.getY()];

grid[coord.getX()][coord.getY()] = 1.0 - (1.0 - old) \* (1.0 - prob);

}

countBombs += group.minesAround;

}

// Все остальные клетки также имеют вероятность содержания бомбы равную остаточному количеству бомб

// поделённому на количество закрытых ячеек, не принадлежащих ни одной группе.

int countClosedReminder = 0;

for (Coord coord : Ranges.getAllCoords()) {

// сначала сосчитаем такие ячейки

if (flags.get(coord) == Box.CLOSED && grid[coord.getX()][coord.getY()] == 0.0)

countClosedReminder++;

}

final int countBombsReminder = bombs.getTotalBombs() - countBombs;

for (Coord coord : Ranges.getAllCoords()) {

// теперь определим их вероятнности

if (flags.get(coord) == Box.CLOSED && grid[coord.getX()][coord.getY()] == 0.0)

grid[coord.getX()][coord.getY()] = ((double) countBombsReminder) / countClosedReminder;

}

// Выбор лучшего варианта

double bestProb = 2.0;

Coord bestCoord = null;

for (Coord coord : Ranges.getAllCoords()) {

double prob = grid[coord.getX()][coord.getY()];

if (flags.get(coord) == Box.CLOSED && prob < bestProb) {

bestProb = prob;

bestCoord = coord;

}

}

if (bestCoord == null)

return false;

openBox(bestCoord);

return true;

}

public MoveType step() {

if (state.getState() != GameState.PLAYED)

return MoveType.NoStep;

Coord size = Ranges.getSize();

if (flags.getCountOfClosedBoxes() == size.getX() \* size.getY()){

openBox(Ranges.getRandomCoord());

return MoveType.First; // Первый ход

}

List<MinesGroup> groups = genGroups();

if (logicalMove(groups)) {

return MoveType.Logical; // Логический ход

} else if (probabilityMove(groups))

return MoveType.Probability; // Вероятностный ход

randomMove();

return MoveType.Random; // Случайный ход

}

private void openBox(Coord coord){

action.performCoordAction(coord);

state.pressLeftButton(coord);

}

private void flagBox(Coord coord){

action.performCoordAction(coord);

state.pressRightButton(coord);

for (Coord place : Ranges.getCoordsAround(coord)){

if (flags.get(place) != Box.CLOSED && state.getState() == GameState.PLAYED)

state.pressLeftButton(place);

}

}

private int countCellsAround(Coord coord){

int n = 0;

for (Coord place : Ranges.getCoordsAround(coord)){

if (flags.get(place) == Box.FLAGED)

n++;

}

return n;

}

private void randomMove(){

Optional<Coord> move = Ranges.getAllCoords()

.stream()

.filter(coord -> flags.get(coord) == Box.CLOSED)

.findFirst();

if (move.isPresent())

openBox(move.get());

else

throw new IllegalStateException("There must be at least 1 closed cell");

}

}

## **MoveType.java**

package solver;

public enum MoveType {

First("Первый ход"),

Logical("Логический ход"),

Probability("Вероятностный ход"),

Random("Случайный ход"),

NoStep("Игра закончена");

MoveType(String \_message){

message = \_message;

}

private String message;

public String getMessage(){

return message;

}

}

## **MinesGroup.java**

package solver;

import sweeper.Coord;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

import java.util.function.Consumer;

public class MinesGroup implements Iterable<Coord> {

int minesAround;

Set<Coord> group;

MinesGroup(int minesAround) {

this.minesAround = minesAround;

group = new HashSet<>();

}

void minusAssign(MinesGroup other) {

minesAround -= other.minesAround;

group.removeAll(other.group);

}

boolean contains(MinesGroup other) {

return group.containsAll(other.group);

}

@Override

public String toString() {

return "{" + minesAround + ":" + group.toString() + "}";

}

@Override

public boolean equals(Object other){

if (other instanceof MinesGroup) {

final MinesGroup otherGroup = (MinesGroup) other;

boolean eg = group.equals(otherGroup.group);

if (eg && minesAround != otherGroup.minesAround) {

System.out.println("!!!!");

minesAround = minesAround > otherGroup.minesAround ? minesAround : otherGroup.minesAround;

}

return minesAround == otherGroup.minesAround && eg;

}

return false;

}

@Override

public int hashCode(){

return ~minesAround ^ group.hashCode();

}

@Override

public Iterator<Coord> iterator() {

return group.iterator();

}

@Override

public void forEach(Consumer<? super Coord> action) {

group.forEach(action);

}

}

# **ТЕСТЫ**

Были созданы тесты в классе TestSolver.java в пакете testSolver, проверяющие работу программы.

package testSolver;

import org.junit.Test;

import solver.MoveType;

import solver.Solver;

import sweeper.\*;

import static org.junit.Assert.assertEquals;

public class TestSolver {

@Test

public void logicalMove() {

Ranges.setSize(new Coord(5, 5));

/\*

00000

01110

01\*10

01110

00000

\*/

Matrix bombMap = new Matrix(Box.ZERO);

for (int x = 1; x < 4; x++) {

for (int y = 1; y < 4; y++) {

bombMap.set(new Coord(x, y), Box.NUM1);

}

}

bombMap.set(new Coord(2, 2), Box.BOMB);

Bomb bombs = new Bomb(2, bombMap);

Game game = new Game(bombs);

game.testStart();

game.pressLeftButton(new Coord(0,0));

Solver solver = new Solver(coord -> assertEquals(new Coord(2, 2), coord));

solver.update(game);

assertEquals(MoveType.Logical, solver.step());

}

@Test

public void probabilityMove() {

Ranges.setSize(new Coord(6, 7));

Matrix bombMap = new Matrix(Box.ZERO);

bombMap.set(new Coord(0, 0), Box.NUM2);

bombMap.set(new Coord(1, 0), Box.BOMB);

bombMap.set(new Coord(2, 0), Box.NUM3);

bombMap.set(new Coord(3, 0), Box.BOMB);

bombMap.set(new Coord(4, 0), Box.NUM1);

bombMap.set(new Coord(5, 0), Box.ZERO);

bombMap.set(new Coord(0, 1), Box.NUM2);

bombMap.set(new Coord(1, 1), Box.BOMB);

bombMap.set(new Coord(2, 1), Box.NUM3);

bombMap.set(new Coord(3, 1), Box.NUM1);

bombMap.set(new Coord(4, 1), Box.NUM1);

bombMap.set(new Coord(5, 1), Box.ZERO);

bombMap.set(new Coord(0, 2), Box.NUM1);

bombMap.set(new Coord(1, 2), Box.NUM1);

bombMap.set(new Coord(2, 2), Box.NUM2);

bombMap.set(new Coord(3, 2), Box.NUM1);

bombMap.set(new Coord(4, 2), Box.NUM1);

bombMap.set(new Coord(5, 2), Box.ZERO);

bombMap.set(new Coord(0, 3), Box.ZERO);

bombMap.set(new Coord(1, 3), Box.ZERO);

bombMap.set(new Coord(2, 3), Box.NUM1);

bombMap.set(new Coord(3, 3), Box.BOMB);

bombMap.set(new Coord(4, 3), Box.NUM1);

bombMap.set(new Coord(5, 3), Box.ZERO);

bombMap.set(new Coord(0, 4), Box.NUM1);

bombMap.set(new Coord(1, 4), Box.NUM1);

bombMap.set(new Coord(2, 4), Box.NUM2);

bombMap.set(new Coord(3, 4), Box.NUM1);

bombMap.set(new Coord(4, 4), Box.NUM1);

bombMap.set(new Coord(5, 4), Box.ZERO);

bombMap.set(new Coord(0, 5), Box.NUM1);

bombMap.set(new Coord(1, 5), Box.BOMB);

bombMap.set(new Coord(2, 5), Box.NUM1);

bombMap.set(new Coord(3, 5), Box.ZERO);

bombMap.set(new Coord(4, 5), Box.ZERO);

bombMap.set(new Coord(5, 5), Box.ZERO);

bombMap.set(new Coord(0, 6), Box.NUM1);

bombMap.set(new Coord(1, 6), Box.NUM1);

bombMap.set(new Coord(2, 6), Box.NUM1);

bombMap.set(new Coord(3, 6), Box.ZERO);

bombMap.set(new Coord(4, 6), Box.ZERO);

bombMap.set(new Coord(5, 6), Box.ZERO);

/\* 0123456

-------

0 2\*3\*10

1 2\*3110

2 112110

3 001\*10

4 112110

5 1\*1000

6 111000

\*/

Bomb bombs = new Bomb(2, bombMap);

Game game = new Game(bombs);

game.testStart();

game.pressLeftButton(new Coord(0,1));

game.pressLeftButton(new Coord(2,1));

Solver solver = new Solver(coord -> assertEquals(new Coord(4, 0), coord));

solver.update(game);

assertEquals(MoveType.Probability, solver.step());

}

}

# **СКРИНШОТЫ ПРОГРАММЫ**

# 

Рис.1 Внешний вид начального окна

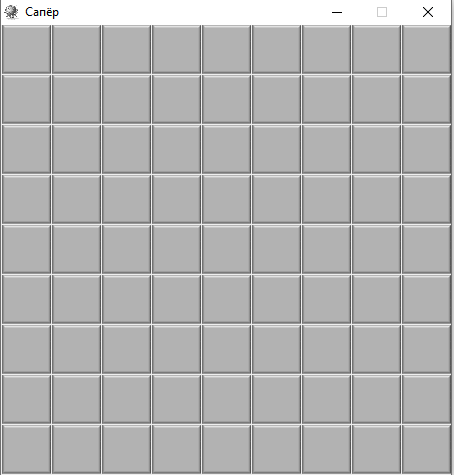


Рис. 2 Внешний вид игрового поля



Рис. 3 Игра

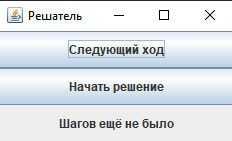


Рис. 4 Внешний вид окна решателя