Міністерство освіти і науки України

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет: Комп’ютерні технології і системи

Кафедра: Комп’ютерні інформаційні технології

Спеціальність: 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедрою "КІТ"

проф. Шинкаренко В.І.

"\_\_\_\_"\_\_\_\_\_\_\_\_2021 р.

З А В Д А Н Н Я

До курсової роботи

студента групи ПЗ 1911(921) Сінькова Георгія

1. Тема проекту: гра “Сапер”.

2. Дата видачі завдання

3. Перелік питань до розробки

1. Опис предметної області та розробка специфікацій (вимоги до програми).

1.1. Постановка задачі.

1.2. Вимоги до програми

2. Розробка об’єктно-орієнтованості моделі.

2.1. Опис відповідальності класів

2.2. Опис відношення між класів.

2.3. Діаграма класів.

2.4. Діаграма послідовності.

3. Розробка та опис інтерфейсної частини класів.

4. Розробка файлів реалізації. Проектування основних алгоритмів.

5. Тестування програми.

6. Відлагодження програми.

7. Приклад роботи програми.

8. Аналіз результатів. Переваги програми за рахунок застосування об’єктно-орієнтованості моделі.

9. Текст програми.

4. Термін виконання курсового проекту 31 травня 2021 року

Керівник курсового проектування: Демидович Інна Миколаївна

Завдання прийняв до виконання: Демидович Інна Миколаївна

**ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

Завданням курсової роботи була розробка гри “Сапер”.

Програма має бути розроблена на мові програмування Java та графічний інтерфейс, за допомогою JFrame у середовищі розробки IntelliJ IDEA.

**Зміст**

Завдання на курсову роботу1

Зміст2

Вступ3

1. Опис предметної області та розробка специфікацій (вимоги до програми)3

1.1. Постановка завдання3

1.2. Вимоги до програми3

1.3. Середовище розробки і мова програмування4

1.4. Методологія програмування4

1.5. Вимоги до вхідних даних4

1.6. Вимоги до вихідних даних4

1.7. Вимоги до функціональності4

2. Розробка об’єктно-орієнтованої моделі4

2.1. Опис розподілу відповідальності між класами та зв’язків між ними4

2.2. Діаграма класів та специфікації класів 5

2.3. Середовище розробки і мова програмування2

3. Розробка та опис інтерфейсної частини класів1

4. Розробка класів. Проектування основних алгоритмів1

5. Тестування роботи програми1

6. Приклад роботи програми1

7. Аналіз результатів та висновок1

8. Текст програми1

**Вступ**

Об'єктно-орієнтоване програмування — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Фундаментальні поняття ООП:

- клас - визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності;

- об'єкт - окремий екземпляр класу;

- метод - можливості об'єкта;

- обмін повідомленнями - передача даних від одного процесу іншому, або надсилання викликів методів;

- успадкування (наслідування) - клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватися цілі дерева успадкування;

- інкапсуляція - приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення;

- абстрагування - спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми;

- поліморфізм - поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення.

**1. Опис предметної області та розробка специфікацій**

**1.1 Постановка завдання**

Задачею курсової роботи є розробка гри “Сапер” в якій потрібно за правилами гри знайти всі бомби.

**1.2 Вимоги до програми**

Програма буде розроблена на мові програмування Java та графічний інтерфейс за допомогою JFrame у середовищі розробки IntelliJ IDEA.

**1.3 Середовище розробки та мова програмування**

Ця курсова робота була написана на мові Java за допомогою середовища Intellij IDEA. Java – строго типізований об'єктно-орієнтована мова програмування загального призначення, розроблений компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Розробка ведеться співтовариством, організованим через Java Community Process; мова і основні реалізують його технології поширюються за ліцензією GPL. Права на торговельну марку належать корпорації Oracle.

Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-якої комп'ютерної архітектурі, для якої існує реалізація віртуальної Java-машини.

**1.4 Методологія програмування**

В роботі була застосована така методології програмування, як об'єктно-орієнтоване програмування.

Об'єктно-орієнтоване програмування (ООП) - методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкування.

**1.5 Вимоги до вхідних даних**

Вхідні дані:

* натискання правої кнопки мишки
* натискання лівої кнопки мишки
* натискання коліщатка мишки

**1.6 Вимоги до вихідних даних**

Вихідні дані:

* ігрове поле
* результат натискання на клітинку(вивід бомби номера и т.д.)
* вивід повідомлення про кінець гри або перемогу у ній

**1.7 Вимоги до функціональності**

Функціональні характеристики програми:

1. Генерація у випадкових місцях бомб
2. Відкриття закритих клітинок
3. Встановлення прапору
4. Встановлення номерів навколо бомб
5. Закінчення гри при відкриті бомби

**2. Розробка об’єктно-орієнтованої моделі**

**2.1 Опис розподілу відповідальності між класами та зв’язків між ними**

|  |  |  |
| --- | --- | --- |
| Клас | Відповідальність | Зв’язки |
| **JavaSweeper** | Ініціалізація форми  Ініціалізація панелі  Загрузка картинок |  |
| **Game** | Натискання клавіш мишки  Відкриття кліток на полі | Агрегація:  **Bomb** – створення, розміщення та робота с бомбами  **Flag** – створення, розміщення та робота с прапорами, відкритими та закритими клітинками  **GameState** – отримання стану |
| **Bomb** | Установка бомб и номерів | Агрегація:  **Matrix** – для створення масиву |
| **Flag** | Установка прапора, відкритих та закритих клітинок | Агрегація:  **Matrix** – для створення масиву |
| **Matrix** | Створення матриці | Агрегація:  **Box** – отримання номерів перечислення |
| **Ranges** | Робота с координатами | Реалізація:  **Game**  **Bomb**  **Matrix**  Агрегація:  **Coord** – створення координат |
| **Coord** | Зберігання координат |  |
| **Box**(enum) | Перечислення картинок |  |
| **GameState**(enum) | Перечислення стану гри |  |

**2.2 Діаграма класів та специфікації класів**

На малюнку 1 зображена специфікація класів ігри ‹‹Сапер ››

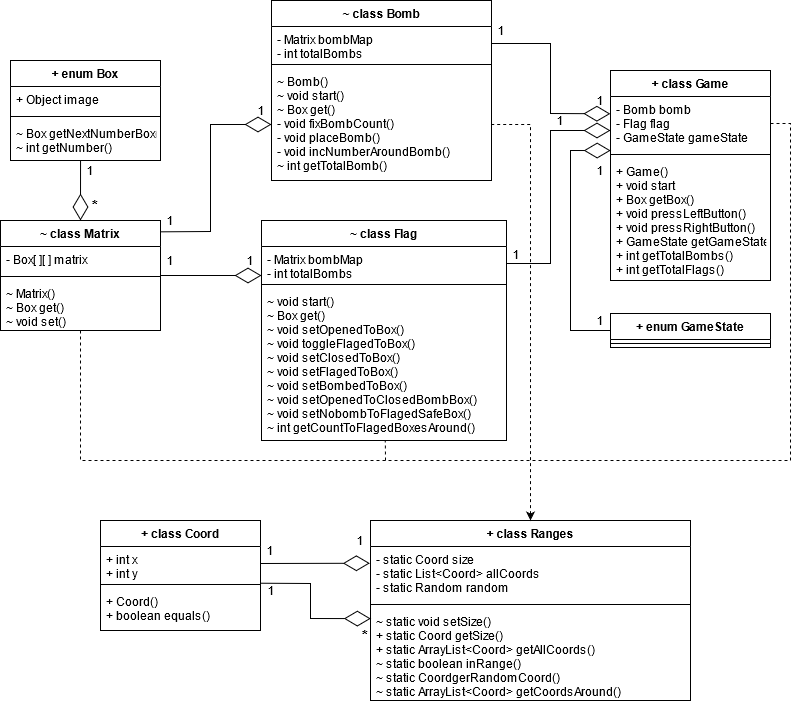


Рис. 1 – специфікації класів ігри ‹‹Сапер››

**2.3 Діаграма послідовності**

На малюнку 2-3 зображено діаграми послідовностей для натискання на праву та ліву кнопку миші

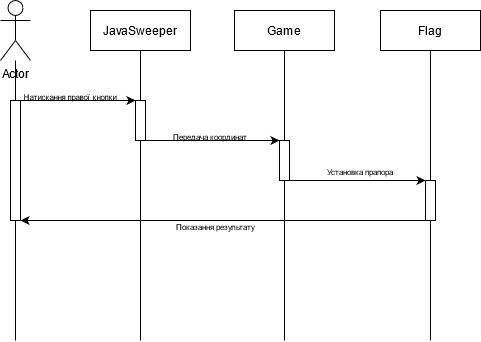


Рис. 2 – діаграма послідовності натискання на праву кнопку миші

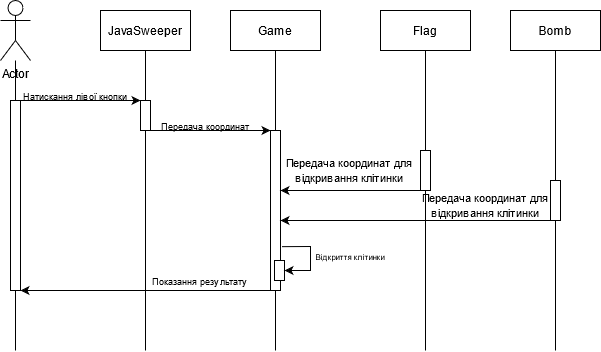


Рис. 3 – діаграма послідовності натискання на ліву кнопку миші

**3. Розборка та опис інтерфейсної частини класів**

**3.1 JavaSweppew**

**Поля:**

**Game game** – посилання на об’єкт класу Game

**JPanel panel** – посилання на об’єкт класу JPanel

**JLabel label** – посилання на об’єкт класу JLabel

**final int COLS = 9** – зберігає кількість колонок

**final int ROWS = 9** – зберігає кількість рядків

**final int BOMBS = 10** – зберігає кількість бомб

**final int IMAGE\_SIZE = 50** – зберігає розмір картинки

**Методи:**

**static void main (String [ ] args)**

**JavaSweeper ()**

**void initLabel ()** – ініціалізація напису

**void initPanel ()** – ініціалізація панелі

**String getMessage ()** – повертає напис-повідомлення про стан гри

**void initFrame ()** – ініціалізація форми

**void setImages ()** – загрузка всіх картинок

**Image getImage (String name)** – отримання всіх картинок

**3.2 Game**

**Поля:**

**Bomb bomb** – посилання на об’єкт класу Bomb

**Flag flag** – посилання на об’єкт класу Flag

**GameState state** – посилання на об’єкт класу GameState

**Методи:**

**GameState getState ()** – повернення стану гри

**Game (int cols, int rows, int bombs)** – ініціалізація об’єкта в залежності від кількості стовбців, рядків та бомб

**void start ()** – головна функція, яка запускає гру

**Box getBox (Coord coord)** – повернення ігрової клітинки

**void pressLeftButton (Coord coord)** – натискання лівої клавіші мишки

**void checkWinner ()** – перевірка на виграш

**void openBox (Coord coord)** – відкриття клітини

**void setOpenedToClosedBoxesAroundNumber (Coord coord)** – відкриття закритих клітин навколо

**void openBombs (Coord coord)** – відкриття бомби

**void openBoxesAround (Coord coord)** – відкриття клітин навколо

**void pressRightButton (Coord coord)** – натискання правої клавіші мишки

**Boolean gameOver ()** – кінець гри

**3.3 Bomb**

**Поля:**

**Matrix bombMap** – створення масиву бомб

**int totalBombs** – зберігає кількість бомб

**Методи:**

**Bomb (int totalBombs)** – ініціалізація об’єкта в залежності від кількості бомб

**void start ()** – встановлення бомб на полі гри

**Box get (Coord coord)** – отримання бомби по заданої координаті

**void fixBombsCount ()** – встановлює максимально фіксовану кількість бомб

**void placeBomb ()** – установка бомб

**void incNumberAroundBomb (Coord coord)** – установка номерів навколо бомб

**int getTotalBombs ()** – отримати всі бомбі

**3.4 Flag**

**Поля:**

**Matrix flagMap** – створення масиву прапорів

**int countOfClosedBoxes** – кількість закритих кліток

**Методи:**

**void start ()**

**Box get (Coord coord)**

**void setOpenedToBox (Coord coord)** – встановлення відкритої клітинки

**void toggleFlagedToBox (Coord coord)** –

**void setClosedToBox (Coord coord)** – встановлення закритої клітинки

**void setFlagedToBox (Coord coord)** – встановлення прапора

**int getCountOfClosedBoxes ()** –

**void setBombedToBox (Coord coord)** – встановлення

**void setOpenedToClosedBombBox (Coord coord)** – встановлення відкритої клітинки замість закритої

**void setNobombToFlagedSafeBox (Coord coord)** – встановлення

**int getCountToFlagedBoxesAround (Coord coord)** – отримання

**3.5 Matrix**

**Поля:**

**Box [ ][ ] matrix** – двовимірний масив перечислення “Box”

**Методи:**

**Matrix (Box defaultBox)** – створення масиву клітинок

**Box get (Coord coord)** – повернення клітинки по заданих координатах

**void set (Coord coord, Box box)** – присвоєння по заданих координатах потрібної клітинки

**3.6 Ranges**

**Поля:**

**Coord size** – зберігає розмір

**ArrayList<Coord> allCoords** – список всіх координат

**Random random – new Random ()** – генерує випадкову координату для бомби

**Методи:**

**static void setSize (Coord \_size)** – встановлення розміру поля

**static CoordgetSize ()** – повернення розміру

**static ArrayList<Coord> getAllCoords ()** – повернення всіх координат

**static Boolean inRange (Coord coord)** – перевірка не переповнення

**static Coord getRandomCoord ()** – повернення випадкової координати бомби

**static ArrayList<Coord> getCoordsAround (Coord coord)** – повернення координат навколо заданої координати

**3.7 Coord**

**Поля:**

**int x** – зберігає x

**int y** – зберігає y

**Методи:**

**Coord (int x, int y)** – ініціалізація координат класу “Coord”

**boolean equals (Object o)** – перевірка поточної координати та наступної

**3.8 Box**

**Поля:**

**ZERO**

**NUM1**

**NUM2**

**NUM3**

**NUM4**

**NUM5**

**NUM6**

**NUM7**

**NUM8**

**BOMB**

**OPENED**

**CLOSED**

**FLAGED**

**BOMBED**

**NOBOMB**

**Object image** – зберігання картинки

**Методи:**

**Box getNextNumberBox ()** – отримання наступного номера перечислення

**int getNumber ()** – отримання поточного номера перечислення

**3.9 GameState**

**Поля:**

**PLAYED**

**BOMBED**

**WINNER**

**4. Проектування основних алгоритмів**

У програмі є два основні алгоритми які змушують її працювати. Перший - це натискання на праву клавішу мишки. При натисканні ми встановлюємо чи прибираємо прапор на клітинці.

Нижче наведений алгоритм натискання правої кнопки мишки у вигляді блок-схеми:

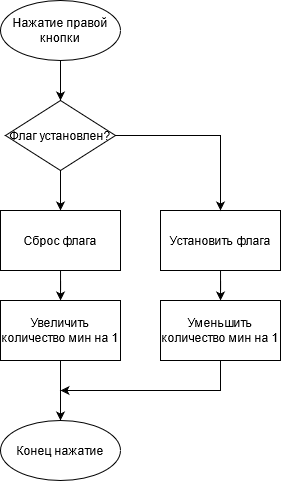


Рис. 3 – Блок-схема алгоритму натискання правої кнопки мишки

Другий важливий алгоритм - це натискання на ліву клавішу мишки. При натисканні ми відкриваємо закриту клітинку і після цього дивимось, є там бомба чи ні. Та в залежності від результату ми продовжуємо гру чи буде кінець гри.

Алгоритм натискання лівої кнопки мишки у вигляді блок-схеми:

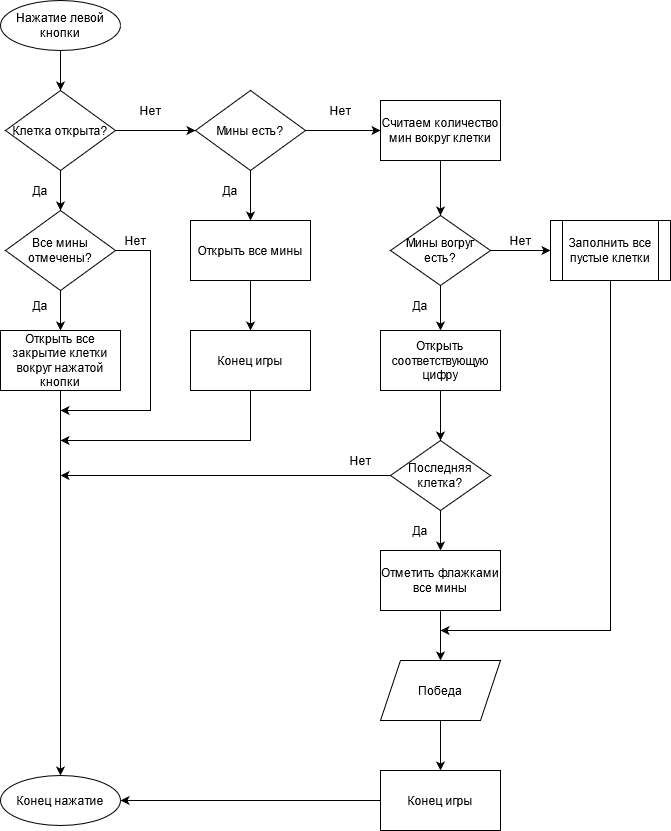


Рис. 4 – Блок-схема алгоритму натискання на ліву кнопку мишки

**5. Тестування програми**

Тестувати даний кінцевий проект майже не можливо, тому що ніяких вхідних і вихідних даних не маю як таких. Все що можливо протестувати, це як реагує програма на натискання на ігрове поле. Саме тестування на справність програми проводилось паралельно розробці програми. При створенні кожної функції та додаванні у виконувану функцію програма постійно тестувалась.

**6. Приклад роботи програми**

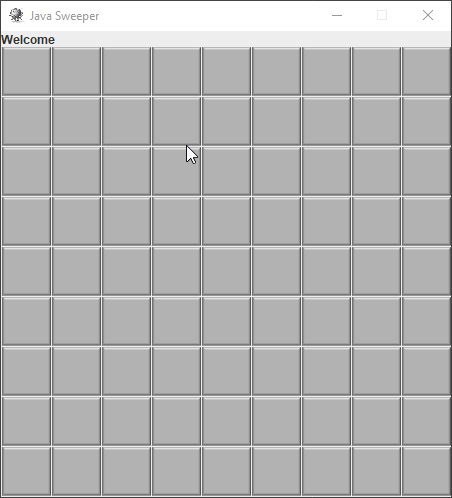


Рис. 5 – При заході у гру є надпис “Welcome”

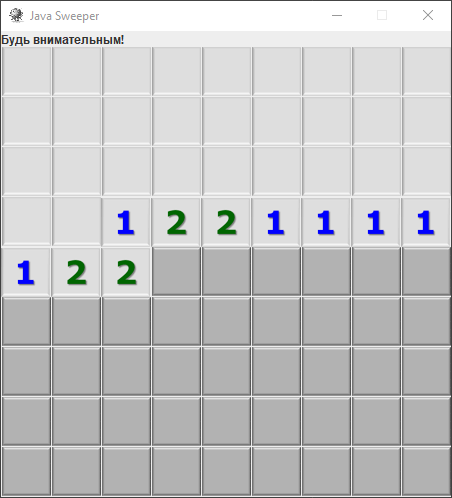


Рис. 6 – Після першого ходу з'являється надпис “Будь внимательным!”

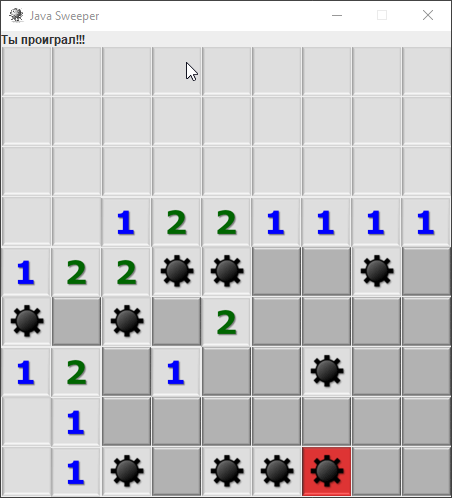


Рис. 7 – Якщо потрапляєш на бомбу з'являється надпис “Ты проиграл!!!”



Рис. 8 – Якщо на всіх бомбах буде прапор з'являється надпис “Ты выиграл!!!”

**7. Аналіз результатів та висновок**

В результаті виконання курсової роботи була розроблена і реалізована гра «Сапер». Програмний продукт створений в ході роботи виконує всі заплановані функціональні вимоги.

В ході виконання даної роботи були отримані дуже корисні навички розробки програм методом ООП. Також було отримано додатковий досвід в програмуванні на мові Java, створення графічної форми JFrame і все що із нею пов’язано. Слід додати, що були також отримані навички в проектуванні програмного продукту, проведенні тестування та складання документації.

**8. Текст програми**

JavaSweeper.java

import javax.swing.\*;  
import java.awt.\*;  
import sweeper.Box;  
import sweeper.Coord;  
import sweeper.Game;  
import sweeper.Ranges;  
  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
  
public class JavaSweeper extends JFrame {  
  
 private Game game;  
  
 private JPanel panel;  
 private JLabel label;  
 private final int COLS = 9;  
 private final int ROWS = 9;  
 private final int BOMBS = 10;  
 private final int IMAGE\_SIZE = 50;  
  
  
 public static void main(String[] args) {  
 new JavaSweeper();  
 }  
  
 private JavaSweeper() {  
 game = new Game(COLS, ROWS, BOMBS);  
 game.start();  
 setImages();  
 initLabel();  
 initPanel();  
 initFrame();  
 }  
  
 private void initLabel() { // инициализация надписи  
 label = new JLabel("Welcome");  
 add(label, BorderLayout.*NORTH*);  
 }  
   
 private void initPanel() { // инициализация панели  
 panel = new JPanel() {  
 @Override  
 protected 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() { // добавление мышечного адаптера  
 @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 (e.getButton() == MouseEvent.*BUTTON1*)  
 game.pressLeftButton(coord);  
 if (e.getButton() == MouseEvent.*BUTTON2*)  
 game.start();  
 if (e.getButton() == MouseEvent.*BUTTON3*)  
 game.pressRightButton(coord);  
 label.setText(getMessage());  
 panel.repaint();  
 }  
 });  
 panel.setPreferredSize(new Dimension(  
 Ranges.*getSize*().x \* IMAGE\_SIZE,  
 Ranges.*getSize*().y \* IMAGE\_SIZE)); // устанавливаем розмер панели  
 add(panel);  
 }  
  
 private String getMessage() {  
 return switch (game.getState()) {  
 case *PLAYED* -> "Будь внимательным!";  
 case *BOMBED* -> "Ты проиграл!!!";  
 case *WINNER* -> "Ты выйграл!!!";  
 };  
 }  
   
 private void initFrame() { // инициализация формы  
 setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*); // полное закрите программы  
 setTitle("Java Sweeper");  
 setResizable(false); // запрет на изменение розмера окна  
 setVisible(true); // видно форму  
 setIconImage(getImage("icon")); // устанавливаем иконку  
 pack(); // устанавливаем минимальный розмер контейнера  
 setLocationRelativeTo(null); // устанавливаем окно по центру  
 }  
  
 private void setImages() { // загрузка всех .png  
 for (Box box : Box.*values*())  
 box.image = getImage(box.name().toLowerCase());  
 }  
  
 private Image getImage(String name) { // получение всех .png из папки rec  
 String filename = "img/" + name.toLowerCase() + ".png";  
 ImageIcon icon = new ImageIcon(getClass().getResource(filename));  
 return icon.getImage();  
 }  
}

Game.java

package sweeper;  
  
public class Game {  
  
 private Bomb bomb;  
 private Flag flag;  
 private GameState state;  
  
 public GameState getState() {  
 return state;  
 }  
  
 public Game(int cols, int rows, int bombs) {  
 Ranges.*setSize*(new Coord(cols, rows));  
 bomb = new Bomb(bombs);  
 flag = new Flag();  
 }  
  
 public void start() {  
 bomb.start();  
 flag.start();  
 state = GameState.*PLAYED*;  
 }  
  
 public Box getBox(Coord coord) {  
 if (flag.get(coord) == Box.*OPENED*)  
 return bomb.get(coord);  
 else  
 return flag.get(coord);  
 }  
  
 public void pressLeftButton(Coord coord) { // нажатие левой клавиши мышки  
 if (gameOver())  
 return;  
 openBox(coord);  
 checkWinner();  
 }  
  
 private void checkWinner() { // проверка на выигрыш  
 if (state == GameState.*PLAYED*)  
 if (flag.getCountOfClosedBoxes() == bomb.getTotalBombs())  
 state = GameState.*WINNER*;  
 }  
  
 private void openBox(Coord coord) { // открытие клетки  
 switch (flag.get(coord)) {  
 case *OPENED*: setOpenedToClosedBoxesAroundNumber(coord); return;  
 case *FLAGED*: return;  
 case *CLOSED*:  
 switch (bomb.get(coord)) {  
 case *ZERO* -> openBoxesAround(coord);  
 case *BOMB* -> openBombs(coord);  
 default -> flag.setOpenedToBox(coord);  
 }  
 }  
 }  
  
 void setOpenedToClosedBoxesAroundNumber(Coord coord) { // открытие закрытых клеток вокруг  
 if (bomb.get(coord) != Box.*BOMB*)  
 if (flag.getCountToFlagedBoxesAround(coord) == bomb.get(coord).getNumber())  
 for (Coord around : Ranges.*getCoordsAround*(coord))  
 if (flag.get(around) == Box.*CLOSED*)  
 openBox(around);  
 }  
  
 private void openBombs(Coord bombed) { // открытие бомбы  
 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) { // открытие клеток вокруг  
 flag.setOpenedToBox(coord);  
 for (Coord around : Ranges.*getCoordsAround*(coord))  
 openBox(around);  
 }  
  
 public void pressRightButton(Coord coord) { // нажатие правой клавиши мышки  
 if (gameOver())  
 return;  
 flag.toggleFlagedToBox(coord);  
 }  
  
 private boolean gameOver() { // конец игры  
 if (state == GameState.*PLAYED*)  
 return false;  
 start();  
 return true;  
 }  
}

Bomb.java

package sweeper;  
  
class Bomb {  
 private Matrix bombMap;  
 private int totalBombs;  
  
 Bomb(int totalBombs) {  
 this.totalBombs = totalBombs;  
 fixBombsCount();  
 }  
  
 void start() {  
 bombMap = new Matrix(Box.*ZERO*);  
 for (int j = 0; j < totalBombs; j++)  
 placeBomb(); // устанавливаем бомбы  
 }  
  
 Box get(Coord coord) {  
 return bombMap.get(coord);  
 }  
  
 private void fixBombsCount() {  
 int maxBombs = Ranges.*getSize*().x \* Ranges.*getSize*().y / 2; // бомб не больше половины клеток  
 if (totalBombs > maxBombs)  
 totalBombs = maxBombs;  
 }  
  
 private void placeBomb() { // установка бомбы  
 while (true) { // проверка бомбы на бомбу  
 Coord coord = Ranges.*getRandomCoord*();  
 if (Box.*BOMB* == bombMap.get(coord))  
 continue;  
 bombMap.set(coord, Box.*BOMB*);  
 incNumberAroundBomb(coord);  
 break;  
 }  
 }  
  
 private void incNumberAroundBomb(Coord coord) { // устанавливаем номера вокруг бомбы  
 for (Coord around : Ranges.*getCoordsAround*(coord))  
 if (Box.*BOMB* != bombMap.get(around))  
 bombMap.set(around, bombMap.get(around).getNextNumberBox());  
 }  
  
 int getTotalBombs() { // получаем все бомбы  
 return totalBombs;  
 }  
}

Flag.java

package sweeper;  
  
class Flag {  
 private Matrix flagMap;  
 private int countOfClosedBoxes;  
  
 void start() {  
 flagMap = new Matrix(Box.*CLOSED*);  
 countOfClosedBoxes = Ranges.*getSize*().x \* Ranges.*getSize*().y;  
 }  
  
 Box get(Coord coord) {  
 return flagMap.get(coord);  
 }  
  
 void setOpenedToBox(Coord coord) { // установка открытой клетки  
 flagMap.set(coord, Box.*OPENED*);  
 countOfClosedBoxes--;  
 }  
  
 void toggleFlagedToBox(Coord coord) {  
 switch (flagMap.get(coord)){  
 case *FLAGED* -> setClosedToBox(coord);  
 case *CLOSED* -> setFlagedToBox(coord);  
 }  
 }  
  
 private void setClosedToBox(Coord coord) { // установка закрытой клетки  
 flagMap.set(coord, Box.*CLOSED*);  
 }  
  
 private void setFlagedToBox(Coord coord) { // установка флага  
 flagMap.set(coord, Box.*FLAGED*);  
 }  
  
  
 int getCountOfClosedBoxes() { // получаем количества закрытых клеток  
 return countOfClosedBoxes;  
 }  
  
 public void setBombedToBox(Coord coord) {  
 flagMap.set(coord, Box.*BOMBED*);  
 }  
  
 public void setOpenedToClosedBombBox(Coord coord) { // установка открытой клетки заместь закрытой  
 if (flagMap.get(coord) == Box.*CLOSED*)  
 flagMap.set(coord, Box.*OPENED*);  
 }  
  
 public void setNobombToFlagedSafeBox(Coord coord) { // установка NOBOMB клетки заместь флага  
 if (flagMap.get(coord) == Box.*FLAGED*)  
 flagMap.set(coord, Box.*NOBOMB*);  
 }  
  
 int getCountToFlagedBoxesAround(Coord coord) {  
 int count = 0;  
 for (Coord around : Ranges.*getCoordsAround*(coord))  
 if (flagMap.get(around) == Box.*FLAGED*)  
 count++;  
 return count;  
 }  
}

Matrix.java

package sweeper;  
  
class Matrix {  
 private Box[][] matrix;  
  
 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;  
 }  
  
 void set(Coord coord, Box box) {  
 if(Ranges.*inRange*(coord))  
 matrix[coord.x][coord.y] = box;  
 }  
}

Ranges.java

package sweeper;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
public class Ranges { // хранение поля  
 private static Coord *size*;  
 private static ArrayList<Coord> *allCoords*;  
 private static Random *random* = new Random();  
  
 public static void setSize(Coord \_size) { // устанавливаем розмер поля  
 *size* = \_size;  
 *allCoords* = new ArrayList<>();  
 for (int y = 0; y < *size*.y; y++) // перебор всех координат  
 for (int x = 0; x < *size*.x; x++)  
 *allCoords*.add(new Coord(x, y));  
 }  
  
 public static Coord getSize() { // возращаем розмер  
 return *size*;  
 }  
  
 public static ArrayList<Coord> getAllCoords() { // возращаем все координаты  
 return *allCoords*;  
 }  
  
 static boolean inRange(Coord coord) { // проверка на переполнение масива  
 return coord.x >= 0 && coord.x < *size*.x &&  
 coord.y >=0 && coord.y < *size*.y;  
 }  
   
 static Coord getRandomCoord() { // возращаем рандомние координаты бомбы  
 return new Coord(*random*.nextInt(*size*.x),  
 *random*.nextInt(*size*.y));  
 }  
   
 static ArrayList<Coord> getCoordsAround(Coord coord) { // возращаем координаты вокруг заданой координаты  
 Coord around;  
 ArrayList<Coord> list = new ArrayList<Coord>();  
  
 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;  
 }  
}

Coord.java

package sweeper;  
  
  
public class Coord { // координаты  
 public int x;  
 public int y;  
  
 public Coord(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (o instanceof Coord) {  
 Coord to = (Coord)o;  
 return to.x == x && to.y == y;  
 }  
 return super.equals(o);  
 }  
}

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();  
 }  
}

GameState.java

package sweeper;  
  
public enum GameState {  
 *PLAYED*,  
 *BOMBED*,  
 *WINNER*}