UDC 004.047

CREATING A MINESWEEPER GAME USING THE JAVA PROGRAMMING LANGUAGE

D. MAROZAU, O. GOLUBEVA
Polotsk State University, Belarus

In this article, we will look at the structure of the code on which the game "Minesweeper" is based and show step by step the methods that are used to create games like this. The Java language that is used for building is one of the most common and universal programming languages.

First, let's create a Game Object class (Fig.1), it will represent the cell that will make up our field for the game. In this class, you will need parameters such as:

```
class GameObject {
    int x;
    int y;
    boolean isMine;
    int countMineNeighbors;
    boolean isOpen;
    boolean isFlag;
    String status;
    GameObject (int x, int y, boolean isMine){
        this.x=x;
        this.y=y;
        this.isMine=isMine;
    }
}
```

Figure 1. –Cell coordinates x, y (simple integers);

• cell States: isMine (whether it is a mine or not), isOpen (whether we have already opened the cell or not), isFlag (whether we have set a flag on the cell or not);
• the count of mines adjacent to this cell countMineNeighgors(integer).

We also create a constructor for this class.
The next step is to build the MinesweeperGame class, which will represent the processes that take place inside the game. Here we need parameters (Fig.2) such as:
• the size of the side of the SIDE field (an immutable number);
• two-dimensional array of gameField objects (consisting of objects of the GameObject class);
• displaying the state of cells in the field: MINE, FLAG, CLOSED (immutable rows);
• counters: countFlags (how many flags are available for use), countClosedTiles (number of closed cells);
• the state in which the game is located is Game Stopped (if true, it ends the game and does not allow the player to perform further actions).

```
public class MinesweeperGame {
    private static final int SIDE = 9;
    private GameObject[][] gameField = new GameObject[SIDE][SIDE];
    private int countMinesOnField;
    private static final String MINE = "M";
    private static final String FLAG = "F";
    private static final String CLOSED = "X";
    private int countFlags;
    private boolean isGameStopped;
    private int countClosedTiles = SIDE * SIDE;
```

Figure 2. – Now we can start describing the processes that occur in the game. Let's create an Initialize() method (Fig.3 and 4) that is available for external use.

The method starts the game process and implements a simple interface through the console for the user. The createGame() method enabled by the process initializes objects in the gameField array, randomly creates mines with a given probability, and counts the number of mines in the countMinesOnField field. We also introduced methods to simplify code understanding: Input() and showField(). Which implement the user entering data from the keyboard and displaying the field on the console, respectively.

```
public void initialize() {
    System.out.println("Welcome");
    createGame();
    while (!isGameStopped) {
        showField();
        System.out.print("Open tile [o], set flag [f], exit [x]: ");
        int x, y;
        switch (input()) {
            case "o": {
                System.out.println("Set coordinates");
                System.out.print("x: ");
                x = Integer.parseInt(input());
                System.out.print("y: ");
                y = Integer.parseInt(input());
                openTile(x, y);
                break;
            }
```

Figure 3

```
            case "f": {
                System.out.println("Set coordinates");
                System.out.print("x: ");
                x = Integer.parseInt(input());
                System.out.print("y: ");
                y = Integer.parseInt(input());
                markTile(x, y);
                break;
            }
            case "x": {
                System.out.println("Goodbye");
                isGameStopped = true;

            }
            default:
                System.out.println("Wrong command");
        }
    }
}
```

Figure 4

Now we will create logical sequences that represent the game mechanics. To do this, we need to create several more methods.

First we implement openTile(x,y) (Fig.5), it is intended to open the cell that the user specified. This method checks the cell for a flag and that it is already open. After that, it reduces the number of closed cells of the field by 1 and checks the contents of the object: if the cell is a mine, it starts the process of completing the game. If there is no mine in the cell, it opens it and displays the number of mines in the neighborhood. If there

are no mines in the neighborhood, then the same method is called for all the cells around using recursion. This is followed by a check if the number of mines on the field and closed cells is equal to that the game is won and the win () method is called.

```
private void openTile(int x, int y) {
    if (!gameField[y][x].isOpen && !gameField[y][x].isFlag && !isGameStopped) {
        gameField[y][x].isOpen = true;
        countClosedTiles--;
        gameField[y][x].status = String.valueOf(gameField[y][x].countMineNeighbors);
        if (gameField[y][x].isMine) {
            gameField[y][x].status = MINE;
            gameOver();
        } else {
            gameField[y][x].status = String.valueOf(gameField[y][x].countMineNeighbors);
            if (gameField[y][x].countMineNeighbors == 0) {
                List<GameObject> r = getNeighbors(gameField[y][x]);
                for (GameObject gameObject : r) {
                    if (!gameObject.isOpen) {
                        openTile(gameObject.x, gameObject.y);
                    }
                }
            }
            if (countClosedTiles == countMinesOnField) {
                win();
            }
        }
    }
}
```

Figure 5

The getNeighbors(gameField[y][x]) method is intended to return a list of cells that are neighbors for the specified instance. The gameOver() method included in the method above assigns the value true when called, which is an indicator of the end of the game and other actions are prohibited, which should also be done if you win.

```
private void markTile(int x, int y) {
    if (!gameField[y][x].isOpen && !isGameStopped) {
        if (!gameField[y][x].isFlag) {
            if (countFlags != 0) {
                gameField[y][x].isFlag = true;
                countFlags--;
                gameField[y][x].status = FLAG;
            }
        } else if (gameField[y][x].isFlag) {
            gameField[y][x].isFlag = false;
            countFlags++;
            gameField[y][x].status = CLOSED;
        }
    }
}
```

Figure 6

We will also need the markTile(x,y) method (Fig 6) to implement the ability to put and remove "flags" in the place of the expected mines. To do this, first enter a check for whether the cell is already open and whether

the game is not over. This is followed by a sequence of actions depending on whether the cell has a flag: if it does not exist, we set it and change the cell status in the field; otherwise, we return the cell to its original values.

After implementing all these actions, we can already start the game(Fig 8) by simply creating an object of the MinesweeperGame class and calling its initialize() method (Fig.7).

```
public class Main {
    public static void main(String[] args) {
        MinesweeperGame game=new MinesweeperGame();
        game.initialize();
    }
}
```

Figure 7

```
Welcome
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
#   #   #   #   #   #   #   #   #
Open tile [o], set flag [f], exit [x]:
```

Figure 8

REFERENCES

1. B.Eckel "Java Philosophy"
2. K. Sierra, B.Bates "Head first Java"