Joc Bubble Shooter

Ciontu Claudia-Elena
Draghici Andreea-Maria
CR4.S1A

Interactiunea Om-Calculator

Tehnologii utilizate

- **Limbajul de dezvoltare: Java si libraria Swing**
- **Mediul de dezvoltare: IntelliJ IDEA**

Despre joc

- Bubble Shooter este o aplicatie desktop dezvoltata utilizand limbajul de programare Java si mediul de dezvoltare IntelliJ IDEA.
- Pentru functionalitatea interfetei grafice cu utilizatorul s-au reutilizat elemente si obiecte grafice din libraria Java Swing pentru a crea interfata grafica si a reda functionalitate acesteia.

Functionalitatile aplicatiei desktop

Aplicatia dezvoltata are la baza urmatoarele functionalitati:

- Inceputul jocului
- Setarile de configurare a culorilor pentru bile.
- Shooter-ul si incrementarea punctelor obtinute
- Salvarea scorului si al jucatorilor castigatori
- Oprirea jocului

Functionalitatea de incepere a jocului

Pagina principala a aplicatiei este reprezentata in imaginea alaturata:

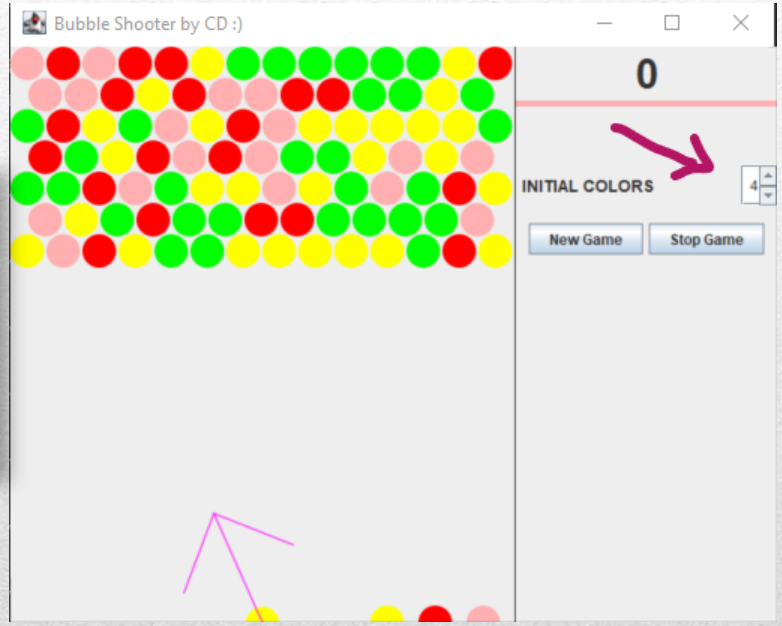
Pentru a incepe un joc nou utilizatorul va apasa pe butonul New Game.


```
JPanel buttonPanel = new JPanel();
newGameButton = new JButton( text: "New Game");
newGameButton.setActionCommand("NEW GAME");
newGameButton.addActionListener(mainFrame);
```

Initializarea butonului de New Game

Functionalitatea de configurare a culorilor pentru bile – Partea 1

Inainte de a incepe un nou joc, utilizatorul poate selecta initial numarul maxim sau minim de culori dorit ca in imaginea alaturata.




```
SpinnerModel colorModell = new SpinnerNumberModel( value: 4,  minimum: 2,  maximum: 8,  stepSize: 1);
colorSpinner = new JSpinner(colorModell);
JLabel colorLabel = new JLabel( text: "INITIAL COLORS");
colorLabel.setFont(new Font(colorLabel.getFont().getName(), Font.CENTER_BASELINE,  size: 14));
colorLabel.setBorder(BorderFactory.createEmptyBorder( top: 0,  left: 5,  bottom: 0,  right: 10));
colorPanel.add(colorLabel, BorderLayout.WEST);
colorPanel.add(colorSpinner, BorderLayout.EAST);
colorPanel.setBorder(BorderFactory.createEmptyBorder( top: 5,  left: 0,  bottom: 10,  right: 0));
```

Initalizarea culorilor pentru bile

Functionalitatea de configurare a culorilor pentru bile – Partea 2

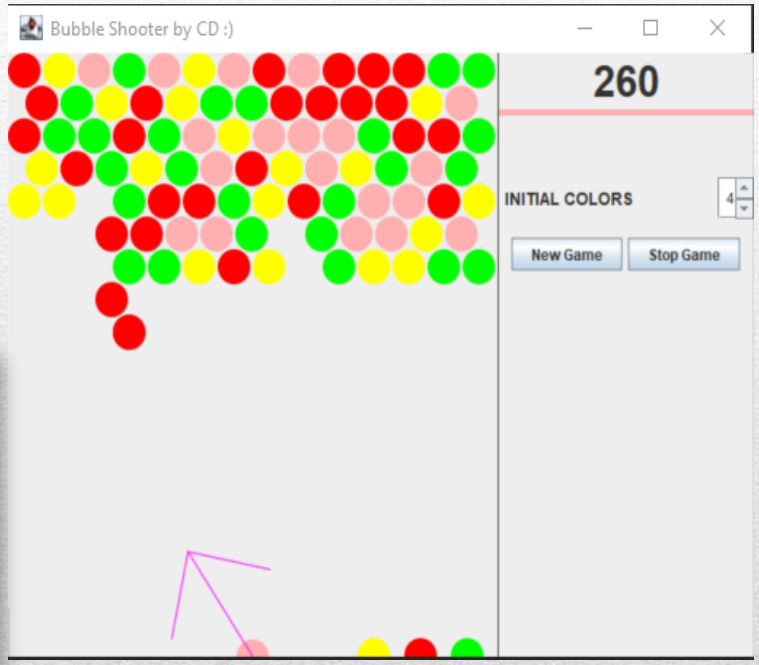
Culorile pentru bile sunt generate aleatoru de algoritmul din imaginea atasata.

Numarul maxim de posibilitati pentru culori este 8, iar numarul minim este 2.

Astfel in functie de dorinta utilizatorului, se vor genera culorile in functie de cate isi doreste acesta in intervalul (2,8).

```java
/**
 * static method for getting a random color that can be set
 * as the color of a bubble
 *
 * @param bound the number of possibilities when choosing randomly, maximal 8
 * @return the generated color
 */
public static Color getRandomColor(int bound) {
    int rnd = (int) (bound <= 8 ? Math.random() * bound : Math.random() * 8);
    switch (rnd) {
        case 0:
            return Color.green;
        case 1:
            return Color.pink;
        case 2:
            return Color.yellow;
        case 3:
            return Color.red;
        case 4:
            return Color.cyan;
        case 5:
            return Color.magenta;
        case 6:
            return Color.orange;
        case 7:
            return Color.black;
        default:
            break;
    }
    return null;
}
```

Functionalitatea pentru shooter si incrementarea punctelor – Partea 1



Shooter-ul este redat de catre sageata din imaginea alaturata, utilizatorul poate trage catre o grupare de doua sau mai multe bile.

O data tintita grupa de bile de aceeasi culoare cu bila, aceasta dispare si punctele se incrementeaza si afisate in sectiunea de sus din stanga a imaginii alaturate.

Functionalitatea pentru shooter si incrementarea punctelor – Partea 2

```java
/**
 * marks the bubbles that have the same color as the one given with
 * the coordinates (these ones must be somehow connected to the
 * given one with a series of bubbles of the same color)
 *
 * @param row row-index of the bubble
 * @param col column-index of the bubble
 */
private void markColor(int row, int col) {

    try {
        bubbles.get(row).get(col).mark();
        for (Bubble b : getNeighbours(row, col)) {
            if (b.isVisible() && !b.isMarked()) {
                if (b.getColor().equals(bubbles.get(row).get(col).getColor())) {
                    markColor(b.getRow(), b.getCol());
                }
            }
        }
    } catch (Exception e) {
        throw new RuntimeException("Failed to mark color due to: " + e.getMessage());
    }

}
```

Marcarea bilelor care au aceeasi culoare cu cea data de shooter

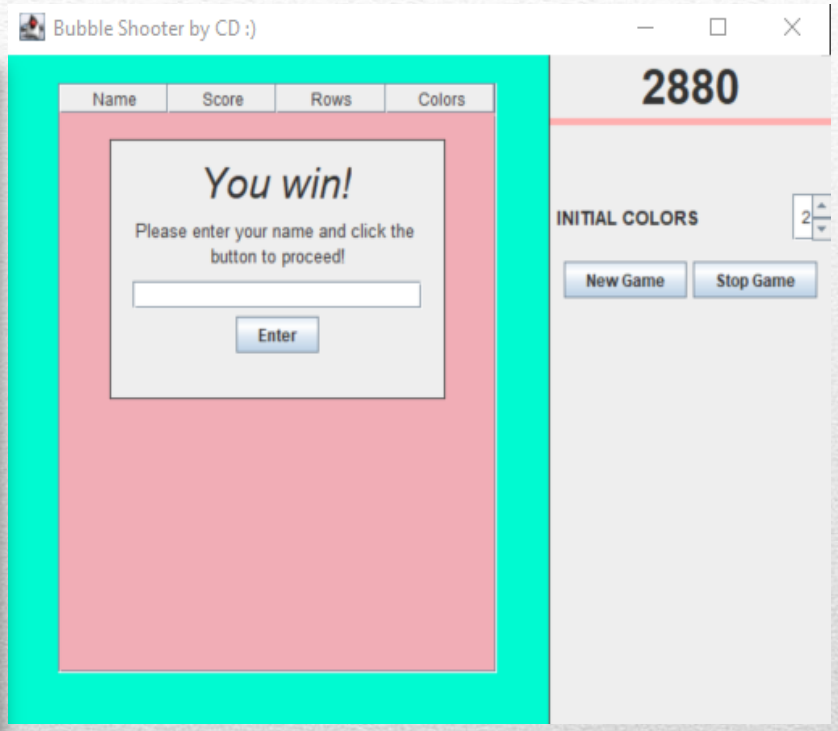
Contorizarea bilelor marcate de aceeasi culoare

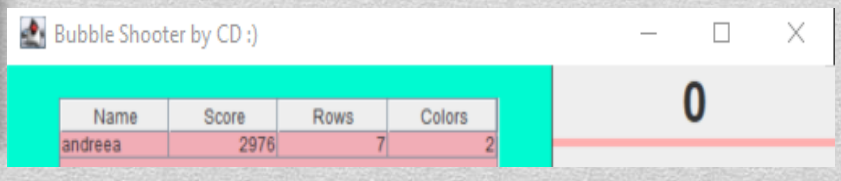
```java
/**
 * counts the marked bubbles in the bubble-matrix
 *
 * @return number of the marked bubbles
 */
private int countMarked() {
    try {
        int ret = 0;
        for (RowList r : bubbles) {
            for (Bubble b : r) {
                if (b.isMarked() && b.isVisible()) {
                    ret++;
                }
            }
        }
        return ret;
    } catch (Exception e) {
        throw new RuntimeException("Failed to count marked bubble due to: " + e.getMessage());
    }
}
```

Functionalitatea de salvare a scorului si al jucatorilor castigatori –Partea 1

La finalizarea jocului, va aparea un modal cu mesajul You win!, unde utilizatorul va trebui sa isi introduca numele pentru a fi inregistrat in clasament.

De altfel, aceste informatii vor fi salvate si intr-un fisier extern .text, astfel incat pe viitor la o noua sesiune de joc noii utilizatori sa poata vedea direct in interfata scorurile salvate, cat si numele jucatorilor castigatori, ca in a doua imagine atasata de jos.





Functionalitatea de salvare a scorului si al jucatorilor castigatori –Partea 2

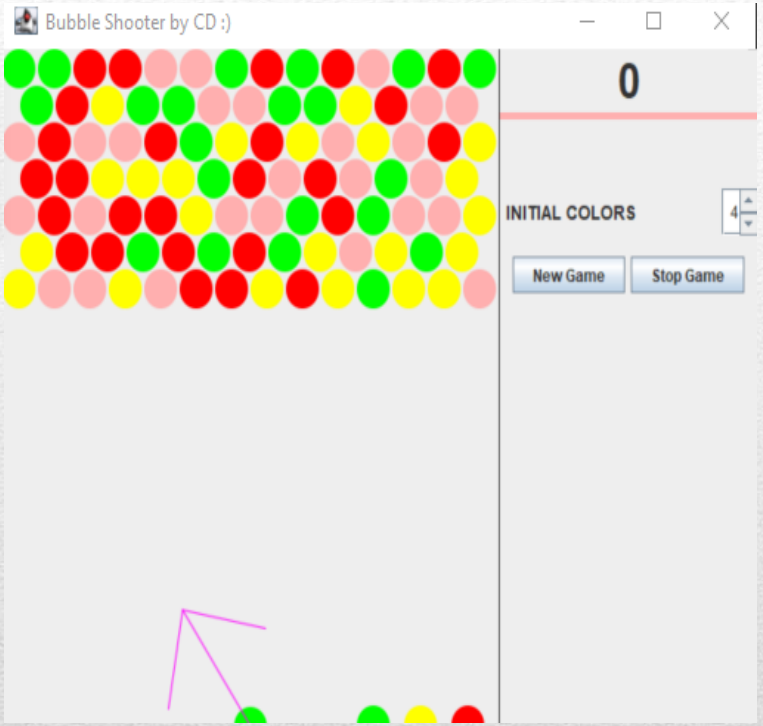
Metodele pentru salvarea scorului intr-un fisier text si de incarcare din fisierul text apoi propagat in interfata grafica.

```java
/**
 * writes the highscores to a file named "bubble_shooter_score.text"
 */
private void saveHighscores() {
    try {
        ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream(fileName));
        os.writeObject(highscores);
        os.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * reads the highscores from the fie named "bubble_shooter_score.text"
 * if it exists
 */
private void loadHighscores() {
    try {
        File f = new File( pathname: "bubble_shooter_score.text");
        if (f.exists()) {
            ObjectInputStream os = new ObjectInputStream(new FileInputStream(fileName));
            highscores = (Highscores) os.readObject();
            os.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Functionalitatea de oprire a jocului

Pentru a opri fortat un joc utilizatorul va apasa pe butonul Stop Game din imaginea alaturata.




```
stopGameButton = new JButton( text: "Stop Game");
stopGameButton.setActionCommand("STOP GAME");
stopGameButton.addActionListener(mainFrame);
buttonPanel.add(newGameButton);
buttonPanel.add(stopGameButton);
```

Initializarea butonului de Stop Game

Legatura dintre PC cu un controller extern

Legatura interfetei grafice cu controller-ul extern se face via Universal Serial Bus 3.0, acesta arata ca in figura alaturata.

```
1 usage   claudia ciontu +1
public void paintComponent(Graphics2D graphics, Point coordinate) {

    graphics.setColor(Color.MAGENTA);
    Point mouseLocation = MouseInfo.getPointerInfo().getLocation();

    int x = mouseLocation.x - coordinate.x;
    int y = mouseLocation.y - coordinate.y;

    if ((0 <= x) && (x < Constants.FIELD_SIZE_X) && (0 <= y) && (y < Constants.FIELD_SIZE_Y)) {
        point = mouseLocation;
    }

    x = point.x - coordinate.x;
    y = point.y - coordinate.y;

    double angle = Math.atan((double) (x - Constants.FIELD_SIZE_X / 2) / (Constants.FIELD_SIZE_Y - y));

    drawArrowLines(graphics, angle);
}
```

Cod pentru reactiile de baza