Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Программирование для мобильных платформ**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Нагалевский

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.А. Приходько

**Тема работы:** Разработка графического приложения игры типа “Balls”.

**Постановка задачи:**

Разноцветные шарики падают на квадратное поле в клеточку, игрок имеет возможность их поменять местами при наличии рядом свободного поля. При выстраивании в ряд 5 шариков одного цвета они исчезают.

**Описание решения:**

1. Инициализация игрового поля: Создаем игровое поле, представленное двумерным массивом grid, где каждая ячейка содержит цвет шарика и заполняем поле случайными цветами из массива BALL\_COLORS.
2. Управление игрой: Реализуем обработчик события mouseClicked, который будет реагировать на клики игрока. При клике на два соседних шарика, которые можно поменять местами, поменяйте их позиции в массиве grid.
3. Анимация и обновление состояния:

* Создаем таймер, который будет выполняться с определенной частотой (например, каждую секунду). Этот таймер будет отвечать за движение шариков вниз и проверку на совпадения.
* В методе moveBallsDown, сдвигаем шарики вниз, заполняя пустые ячейки.
* В методе checkMatches, проверяем наличие горизонтальных и вертикальных рядов из пяти или более одинаковых цветов шариков. Если такие ряды найдены, удаляйте их, устанавливая соответствующие ячейки в null.

1. Моргание выбранного шара: Реализуем таймер, который будет периодически мигать выбранным шаром, чтобы игрок мог легко узнать, какой шар выбран для перемещения.
2. Отрисовка: В методе paint, отрисовывываем игровое поле и шары на нем. Моргание выбранного шара реализуем изменяя его цвет на белый каждую секунду.

На рисунке 1 представлена UML-диаграмма классов.

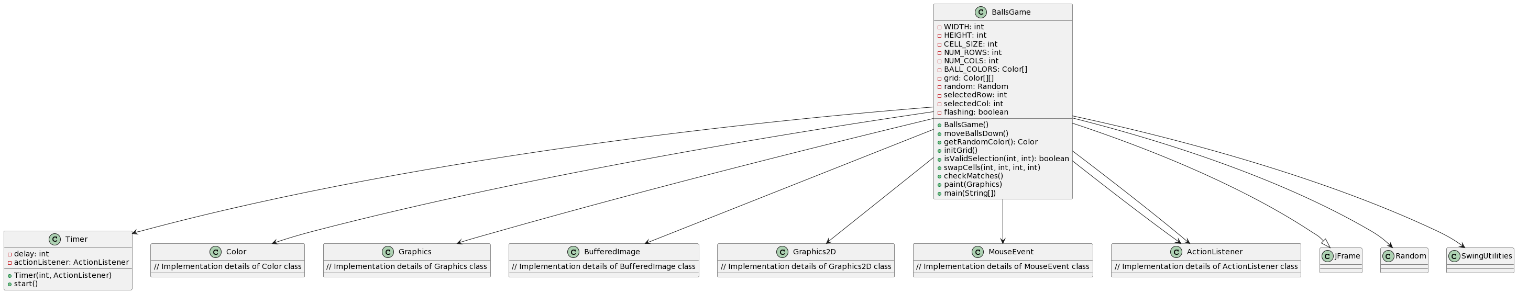


Рисунок 1 – UML-диаграмма

Также на рисунке 2 представлены диаграмма последовательности обработки события mouseCliked.

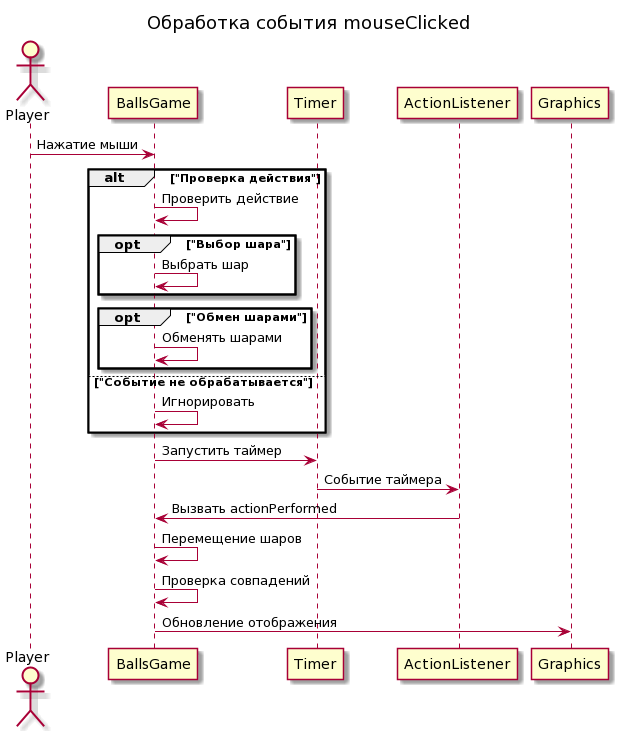


Рисунок 2 – диаграмма последовательности.

На рисунке 3 – 4 представлено выполненииe программы.

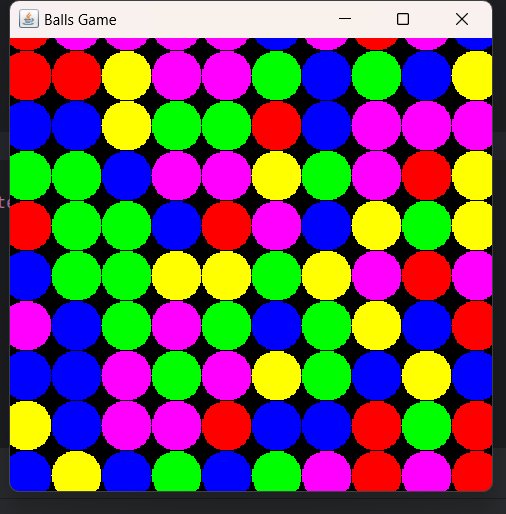


Рисунок 3 – Пример 1.

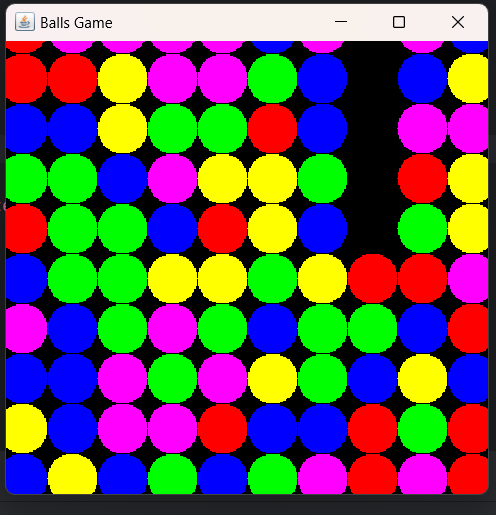


Рисунок 4 – Пример 2.

**Листинг программ**

Файл BallsGame.java

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.awt.image.BufferedImage;  
import java.util.Random;  
public class BallsGame extends JFrame {  
 private static final int *WIDTH* = 400;  
 private static final int *HEIGHT* = 400;  
 private static final int *CELL\_SIZE* = 40;  
 private static final int *NUM\_ROWS* = *HEIGHT* / *CELL\_SIZE*;  
 private static final int *NUM\_COLS* = *WIDTH* / *CELL\_SIZE*;  
 private static final Color[] *BALL\_COLORS* = {Color.*RED*, Color.*BLUE*, Color.*GREEN*, Color.*MAGENTA*, Color.*YELLOW*};  
  
 private final Color[][] grid;  
 private final Random random = new Random();  
 private int selectedRow = -1;  
 private int selectedCol = -1;  
 private boolean flashing = false;  
  
 public BallsGame() {  
 // определеим игровое поле  
 grid = new Color[*NUM\_ROWS*][*NUM\_COLS*];  
 initGrid();  
 addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 int clickX = e.getX() / *CELL\_SIZE*;  
 int clickY = e.getY() / *CELL\_SIZE*;  
 if (isValidSelection(clickX, clickY)) {  
 if (selectedRow == -1) {  
 selectedRow = clickY;  
 selectedCol = clickX;  
 } else {  
 swapCells(selectedRow, selectedCol, clickY, clickX);  
 selectedRow = -1;  
 selectedCol = -1;  
 }  
 repaint();  
 }  
 }  
 });  
  
 Timer timer = new Timer(1000, new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 moveBallsDown();  
 checkMatches();  
 repaint();  
 }  
 });  
 timer.start();  
  
 setTitle("Balls Game");  
 setSize(*WIDTH*, *HEIGHT*);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
  
 Timer flashTimer = new Timer(500, new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 flashing = !flashing;  
 repaint();  
 }  
 });  
 flashTimer.start();  
 }  
  
 private void moveBallsDown() {  
 for (int col = 0; col < *NUM\_COLS*; col++) {  
 for (int row = *NUM\_ROWS* - 1; row >= 0; row--) {  
 if (grid[row][col] == null) {  
 for (int aboveRow = row - 1; aboveRow >= 0; aboveRow--) {  
 if (grid[aboveRow][col] != null) {  
 grid[row][col] = grid[aboveRow][col];  
 grid[aboveRow][col] = null;  
 break;  
 }  
 }  
 }  
 }  
 }  
 }  
  
 private Color getRandomColor() {  
 return *BALL\_COLORS*[random.nextInt(*BALL\_COLORS*.length)];  
 }  
  
 private void initGrid() {  
 for (int row = 0; row < *NUM\_ROWS*; row++) {  
 for (int col = 0; col < *NUM\_COLS*; col++) {  
 grid[row][col] = getRandomColor();  
 }  
 }  
 }  
  
 private boolean isValidSelection(int row, int col) {  
 return row >= 0 && row < *NUM\_ROWS* && col >= 0 && col < *NUM\_COLS*;  
 }  
  
 private void swapCells(int row1, int col1, int row2, int col2) {  
 Color temp = grid[row1][col1];  
 grid[row1][col1] = grid[row2][col2];  
 grid[row2][col2] = temp;  
 }  
  
 private void checkMatches() {  
 // горизонтальные совпадений  
 for (int row = 0; row < *NUM\_ROWS*; row++) {  
 for (int col = 0; col < *NUM\_COLS* - 4; col++) {  
 Color currentColor = grid[row][col];  
 if (currentColor != null) {  
 boolean isMatch = true;  
 for (int i = 1; i < 5; i++) {  
 if (grid[row][col + i] == null || !grid[row][col + i].equals(currentColor)) {  
 isMatch = false;  
 break;  
 }  
 }  
 if (isMatch) {  
 for (int i = 0; i < 5; i++) {  
 grid[row][col + i] = null;  
 }  
 }  
 }  
 }  
 }  
  
 // вертикальные совпадений  
 for (int row = 0; row < *NUM\_ROWS* - 4; row++) {  
 for (int col = 0; col < *NUM\_COLS*; col++) {  
 Color currentColor = grid[row][col];  
 if (currentColor != null) {  
 boolean isMatch = true;  
 for (int i = 1; i < 5; i++) {  
 if (grid[row + i][col] == null || !grid[row + i][col].equals(currentColor)) {  
 isMatch = false;  
 break;  
 }  
 }  
 if (isMatch) {  
 for (int i = 0; i < 5; i++) {  
 grid[row + i][col] = null;  
 }  
 }  
 }  
 }  
 }  
 }  
  
 @Override  
 public void paint(Graphics g) {  
 BufferedImage buffer = new BufferedImage(*WIDTH*, *HEIGHT*, BufferedImage.*TYPE\_INT\_RGB*);  
 Graphics2D g2d = buffer.createGraphics();  
  
 for (int row = 0; row < *NUM\_ROWS*; row++) {  
 for (int col = 0; col < *NUM\_COLS*; col++) {  
 Color ballColor = grid[row][col];  
 if (ballColor != null) {  
 g2d.setColor(ballColor);  
 g2d.fillOval(col \* *CELL\_SIZE*, row \* *CELL\_SIZE*, *CELL\_SIZE*, *CELL\_SIZE*);  
 }  
 }  
 }  
  
 // Моргание выбранного шара  
 if (selectedRow != -1 && selectedCol != -1) {  
 Color selectedColor = grid[selectedRow][selectedCol];  
 if (selectedColor != null) {  
 if (flashing) {  
 g2d.setColor(selectedColor);  
 } else {  
 g2d.setColor(Color.*WHITE*);  
 }  
 g2d.fillOval(selectedCol \* *CELL\_SIZE*, selectedRow \* *CELL\_SIZE*, *CELL\_SIZE*, *CELL\_SIZE*);  
 }  
 }  
  
 g.drawImage(buffer, 0, 0, this);  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 BallsGame game = new BallsGame();  
 game.setVisible(true);  
 }  
 });  
 }  
}