```java
// MemoryGame.java
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.swing.Timer;

public class MemoryGame extends JFrame {

    private CartaMemory[][] carte;
    private int dimensioneGriglia;
    private CartaMemory primaCarta, secondaCarta;
    private boolean bloccaGriglia;
    private JLabel lblTentativi, lblPunteggio, lblTempo;
    private JButton btnNuovaPartita;
    private int tentativi;
    private int punteggio;
    private int coppieRimanenti;
    private Timer timer;
    private int secondiTrascorsi;

    public MemoryGame(int dimensione) {
        super("Memory Game");
        this.dimensioneGriglia = dimensione;
        this.carte = new CartaMemory[dimensione][dimensione];
        this.bloccaGriglia = false;
        this.tentativi = 0;
        this.punteggio = 1000;
        this.coppieRimanenti = (dimensione * dimensione) / 2;
        this.secondiTrascorsi = 0;

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        initComponenti();
        initPannelli();
        initAscoltatori();
        initTimer();

        distribuisciCarte();
```

```
        pack();
        setVisible(true);
    }

    private void initComponenti() {
        // Creazione griglia di carte
        for (int i = 0; i < dimensioneGriglia; i++) {
            for (int j = 0; j < dimensioneGriglia; j++) {
                carte[i][j] = new CartaMemory();
                carte[i][j].setCoordinate(i, j);
            }
        }

        // Etichette informative
        lblTentativi = new JLabel("Tentativi: 0");
        lblPunteggio = new JLabel("Punteggio: 1000");
        lblTempo = new JLabel("Tempo: 0s");
        btnNuovaPartita = new JButton("Nuova Partita");
    }

    private void initPannelli() {
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new BorderLayout(10, 10));

        // Pannello griglia
        JPanel pnlGriglia = new JPanel(new GridLayout(dimensioneGriglia,
dimensioneGriglia, 5, 5));
        for (int i = 0; i < dimensioneGriglia; i++) {
            for (int j = 0; j < dimensioneGriglia; j++) {
                pnlGriglia.add(carte[i][j]);
            }
        }

        // Pannello informazioni
        JPanel pnlInfo = new JPanel(new FlowLayout(FlowLayout.CENTER, 15,
5));
        pnlInfo.add(lblTentativi);
        pnlInfo.add(lblPunteggio);
        pnlInfo.add(lblTempo);
        pnlInfo.add(btnNuovaPartita);

        // Aggiunta pannelli al contentPane
        contentPane.add(pnlGriglia, BorderLayout.CENTER);
        contentPane.add(pnlInfo, BorderLayout.SOUTH);

        // Setting bordi
        pnlGriglia.setBorder(BorderFactory.createEmptyBorder(10, 10, 5,
10));
        pnlInfo.setBorder(BorderFactory.createEmptyBorder(5, 10, 10, 10));
```

```
    }

    private void initAscoltatori() {
        // Aggiungi listener a tutte le carte
        AscoltaCarta ascoltaCarta = new AscoltaCarta(this);
        for (int i = 0; i < dimensioneGriglia; i++) {
            for (int j = 0; j < dimensioneGriglia; j++) {
                carte[i][j].addMouseListener(ascoltaCarta);
            }
        }

        // Listener per il bottone nuova partita
        btnNuovaPartita.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                nuovaPartita();
            }
        });
    }

    private void initTimer() {
        TimerListener timerListener = new TimerListener(this);
        timer = new Timer(1000, timerListener);
    }

    private void distribuisciCarte() {
        // Genera coppie di valori
        List<Integer> valori = new ArrayList<>();
        for (int i = 0; i < (dimensioneGriglia * dimensioneGriglia) / 2;
i++) {
            valori.add(i);
            valori.add(i);
        }

        // Mescola i valori
        Collections.shuffle(valori);

        // Assegna i valori alle carte
        int indice = 0;
        for (int i = 0; i < dimensioneGriglia; i++) {
            for (int j = 0; j < dimensioneGriglia; j++) {
                carte[i][j].setValore(valori.get(indice++));
                carte[i][j].copri();
            }
        }
    }

    public void gestisciCartaClic(CartaMemory carta) {
        if (bloccaGriglia || carta.isScoperta()) {
            return;
```

```
        }

        // Avvia il timer al primo click
        if (tentativi == 0 && primaCarta == null) {
            timer.start();
        }

        // Scopri la carta
        carta.scopri();

        // Prima carta cliccata
        if (primaCarta == null) {
            primaCarta = carta;
            return;
        }

        // Seconda carta cliccata
        secondaCarta = carta;
        tentativi++;
        aggiornaContatore();

        // Controlla se le carte sono uguali
        if (primaCarta.getValore() == secondaCarta.getValore()) {
            // Coppia trovata
            primaCarta = null;
            secondaCarta = null;
            coppieRimanenti--;

            // Controlla se il gioco è finito
            if (coppieRimanenti == 0) {
                timer.stop();
                JOptionPane.showMessageDialog(this,
                        "Complimenti! Hai completato il memory in " +
tentativi +
                        " tentativi.\nPunteggio finale: " + punteggio,
                        "Vittoria!", JOptionPane.INFORMATION_MESSAGE);
            }
        } else {
            // Coppia non trovata, attendi e copri le carte
            bloccaGriglia = true;
            Timer timerCopri = new Timer(1000, new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    primaCarta.copri();
                    secondaCarta.copri();
                    primaCarta = null;
                    secondaCarta = null;
                    bloccaGriglia = false;
                }
            });
```

```
            timerCopri.setRepeats(false);
            timerCopri.start();

            // Riduci il punteggio
            punteggio = Math.max(0, punteggio - 50);
            lblPunteggio.setText("Punteggio: " + punteggio);
        }
    }

    private void aggiornaContatore() {
        lblTentativi.setText("Tentativi: " + tentativi);
    }

    public void aggiornaTempo() {
        secondiTrascorsi++;
        lblTempo.setText("Tempo: " + secondiTrascorsi + "s");
    }

    private void nuovaPartita() {
        timer.stop();
        tentativi = 0;
        punteggio = 1000;
        coppieRimanenti = (dimensioneGriglia * dimensioneGriglia) / 2;
        secondiTrascorsi = 0;
        primaCarta = null;
        secondaCarta = null;
        bloccaGriglia = false;

        distribuisciCarte();

        lblTentativi.setText("Tentativi: 0");
        lblPunteggio.setText("Punteggio: 1000");
        lblTempo.setText("Tempo: 0s");
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                int dimensione = 4; // Dimensione di default
                try {
                    String input = JOptionPane.showInputDialog(null,
                            "Inserisci la dimensione della griglia (4, 6,
8):",
                            "Memory Game", JOptionPane.QUESTION_MESSAGE);

                    if (input != null) {
                        dimensione = Integer.parseInt(input);
                        if (dimensione % 2 != 0 || dimensione < 4 ||
dimensione > 8) {
```

```
                            dimensione = 4;
                        }
                    }
                } catch (NumberFormatException e) {
                    // Usa il valore di default
                }

                new MemoryGame(dimensione);
            }
        });
    }
}

// CartaMemory.java
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.BorderFactory;
import javax.swing.JButton;

public class CartaMemory extends JButton {

    private int valore;
    private boolean scoperta;
    private int riga, colonna;

    private static final Color COLORE_COPERTA = new Color(70, 130, 180);
    private static final Color[] COLORI_VALORI = {
        Color.RED, Color.BLUE, Color.GREEN, Color.YELLOW,
        Color.ORANGE, Color.MAGENTA, Color.CYAN, Color.PINK,
        new Color(128, 0, 0), new Color(0, 128, 0), new Color(0, 0, 128),
        new Color(128, 128, 0), new Color(128, 0, 128), new Color(0, 128,
128),
        new Color(70, 130, 180), new Color(210, 105, 30)
    };

    public CartaMemory() {
        super();
        this.valore = -1;
        this.scoperta = false;

        setPreferredSize(new Dimension(80, 80));
        setFont(new Font("Arial", Font.BOLD, 24));
        setBorder(BorderFactory.createRaisedBevelBorder());

        copri();
    }

    public void setValore(int valore) {
        this.valore = valore;
```

```
    }

    public int getValore() {
        return valore;
    }

    public void setCoordinate(int riga, int colonna) {
        this.riga = riga;
        this.colonna = colonna;
    }

    public void scopri() {
        scoperta = true;

        // Usa l'indice del valore per determinare il colore, con gestione
overflow
        Color colore = COLORI_VALORI[valore % COLORI_VALORI.length];

        setBackground(colore);
        setText(String.valueOf(valore));
        setForeground(getContrastColor(colore));
    }

    public void copri() {
        scoperta = false;
        setBackground(COLORE_COPERTA);
        setText("?");
        setForeground(Color.WHITE);
    }

    public boolean isScoperta() {
        return scoperta;
    }

    // Calcola un colore contrastante per il testo
    private Color getContrastColor(Color colore) {
        double luminanza = (0.299 * colore.getRed() + 0.587 *
colore.getGreen() + 0.114 * colore.getBlue()) / 255;
        return luminanza > 0.5 ? Color.BLACK : Color.WHITE;
    }
}

// AscoltaCarta.java
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class AscoltaCarta extends MouseAdapter {

    private MemoryGame gioco;
```

```
    public AscoltaCarta(MemoryGame gioco) {
        this.gioco = gioco;
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getButton() == MouseEvent.BUTTON1) {
            CartaMemory carta = (CartaMemory) e.getSource();
            gioco.gestisciCartaClic(carta);
        }
    }
}

// TimerListener.java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TimerListener implements ActionListener {

    private MemoryGame gioco;

    public TimerListener(MemoryGame gioco) {
        this.gioco = gioco;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        gioco.aggiornaTempo();
    }
}
```