Discrete Mathematics

CHECKERS

CIS – 143

Semester : 02

Members

* Syed Darain Haider Kazmi
* Mansoob-e-Zahra
* Muhammad Uzyr Warraich

Submitted To

Ms Aneela Zameer

Software Used

VS Code(with GUI java Swing)

**File 1:**

***CheckersGUI:***

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import java.awt.\*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

public class CheckersGUI extends JFrame {

    SoundEffect selectSound;

    SoundEffect placeSound;

    SoundEffect captureSound;

    private JPanel boardPanel;

    private JPanel borderPanel;

    private JPanel topPanel;

    private JPanel bottomPanel;

    private JPanel leftPanel;

    private JPanel rightPanel;

    private JLabel[][] squares;

    private CheckerGame game;

    private int squareSize;

    private int selectedRow = -1;

    private int selectedCol = -1;

    private Color brownColor = new Color(77, 25, 0);

    private Color almondColor = new Color(239, 222, 205);

    private Color selectedColor = new Color(121, 255, 255);

    private int borderThickness = 25;

    private static CheckersGUI instance;

    public static CheckersGUI getInstance() {

        if (instance == null) {

            instance = new CheckersGUI();

        }

        return instance;

    }

    public CheckersGUI() {

        game = new CheckerGame();

        squares = new JLabel[8][8];

        boardPanel = new JPanel(new GridLayout(8, 8));

        borderPanel = new JPanel(new BorderLayout());

        topPanel = new JPanel(new GridLayout(1, 8));

        bottomPanel = new JPanel(new GridLayout(1, 8));

        leftPanel = new JPanel(new GridLayout(8, 1));

        rightPanel = new JPanel(new GridLayout(8, 1));

        borderPanel.setBorder(new EmptyBorder(borderThickness, borderThickness, borderThickness, borderThickness));

        borderPanel.setBackground(brownColor); // Set the background color of the border panel to brown

        squareSize = 75;

        // Load sound effects

         selectSound = new SoundEffect("CheckersGame\\soundeffects\\select.wav"); // Replace with your select sound path

         placeSound = new SoundEffect("CheckersGame\\soundeffects\\placed01.wav"); // Replace with your place sound path

         captureSound = new SoundEffect("CheckersGame\\soundeffects\\capture.wav"); // Replace with your capture sound path

        // Create squares

        for (int i = 0; i < 8; i++) {

            for (int j = 0; j < 8; j++) {

                squares[i][j] = new JLabel();

                squares[i][j].setOpaque(true);

                squares[i][j].setPreferredSize(new Dimension(squareSize, squareSize));

                squares[i][j].setBackground((i % 2 == j % 2) ? almondColor : brownColor);

                squares[i][j].addMouseListener(new SquareClickListener(i, j));

                boardPanel.add(squares[i][j]);

            }

        }

        // Add numbering (1-8) to the side squares

        for (int i = 0; i < 8; i++) {

            JLabel leftLabel = new JLabel(String.valueOf(8 - i), SwingConstants.CENTER);

            leftLabel.setForeground(brownColor);

            leftPanel.add(leftLabel);

            leftPanel.add(new JLabel(" ", SwingConstants.CENTER));

            JLabel rightLabel = new JLabel(String.valueOf(8 - i), SwingConstants.CENTER);

            rightLabel.setForeground(brownColor);

            rightPanel.add(rightLabel);

            rightPanel.add(new JLabel(" ", SwingConstants.CENTER));

        }

        // Add lettering (A-H) to the top and bottom squares

        char[] letters = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};

        for (int j = 0; j < 8; j++) {

            JLabel topLabel = new JLabel(String.valueOf(letters[j]), SwingConstants.CENTER);

            topLabel.setForeground(brownColor);

            topPanel.add(topLabel);

            JLabel bottomLabel = new JLabel(String.valueOf(letters[j]), SwingConstants.CENTER);

            bottomLabel.setForeground(brownColor);

            bottomPanel.add(bottomLabel);

        }

        updateBoard();

        borderPanel.add(boardPanel, BorderLayout.CENTER);

        borderPanel.add(topPanel, BorderLayout.NORTH);

        borderPanel.add(bottomPanel, BorderLayout.SOUTH);

        borderPanel.add(leftPanel, BorderLayout.WEST);

        borderPanel.add(rightPanel, BorderLayout.EAST);

        add(borderPanel, BorderLayout.CENTER);

        setTitle("Checkers By Legends");

        setSize(800, 800);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        setVisible(true);

        setResizable(false);

        // Set the icon image

        ImageIcon icon = new ImageIcon("CheckersGame\\images\\CheckersIcon.jpeg"); // Replace with the path to your icon image

        setIconImage(icon.getImage());

    }

    public CheckersGUI(CheckerGame checkerGame) {

        game = new CheckerGame();

        squares = new JLabel[8][8];

        boardPanel = new JPanel(new GridLayout(8, 8));

        borderPanel = new JPanel(new BorderLayout());

        topPanel = new JPanel(new GridLayout(1, 8));

        bottomPanel = new JPanel(new GridLayout(1, 8));

        leftPanel = new JPanel(new GridLayout(8, 1));

        rightPanel = new JPanel(new GridLayout(8, 1));

        borderPanel.setBorder(new EmptyBorder(borderThickness, borderThickness, borderThickness, borderThickness));

        borderPanel.setBackground(brownColor); // Set the background color of the border panel to brown

        squareSize = 75;

        // Load sound effects

        selectSound = new SoundEffect("CheckersGame\\soundeffects\\select.wav"); // Replace with your select sound path

        placeSound = new SoundEffect("CheckersGame\\soundeffects\\placed01.wav"); // Replace with your place sound path

        captureSound = new SoundEffect("CheckersGame\\soundeffects\\capture.wav"); // Replace with your capture sound path

        // Create squares

        for (int i = 0; i < 8; i++) {

            for (int j = 0; j < 8; j++) {

                squares[i][j] = new JLabel();

                squares[i][j].setOpaque(true);

                squares[i][j].setPreferredSize(new Dimension(squareSize, squareSize));

                squares[i][j].setBackground((i % 2 == j % 2) ? almondColor : brownColor);

                squares[i][j].addMouseListener(new SquareClickListener(i, j));

                boardPanel.add(squares[i][j]);

            }

        }

        // Add numbering (1-8) to the side squares

        for (int i = 0; i < 8; i++) {

            JLabel leftLabel = new JLabel(String.valueOf(8 - i), SwingConstants.CENTER);

            leftLabel.setForeground(brownColor);

            leftPanel.add(leftLabel);

            leftPanel.add(new JLabel(" ", SwingConstants.CENTER));

            JLabel rightLabel = new JLabel(String.valueOf(8 - i), SwingConstants.CENTER);

            rightLabel.setForeground(brownColor);

            rightPanel.add(rightLabel);

            rightPanel.add(new JLabel(" ", SwingConstants.CENTER));

        }

        // Add lettering (A-H) to the top and bottom squares

        char[] letters = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};

        for (int j = 0; j < 8; j++) {

            JLabel topLabel = new JLabel(String.valueOf(letters[j]), SwingConstants.CENTER);

            topLabel.setForeground(brownColor);

            topPanel.add(topLabel);

            JLabel bottomLabel = new JLabel(String.valueOf(letters[j]), SwingConstants.CENTER);

            bottomLabel.setForeground(brownColor);

            bottomPanel.add(bottomLabel);

        }

        updateBoard();

        borderPanel.add(boardPanel, BorderLayout.CENTER);

        borderPanel.add(topPanel, BorderLayout.NORTH);

        borderPanel.add(bottomPanel, BorderLayout.SOUTH);

        borderPanel.add(leftPanel, BorderLayout.WEST);

        borderPanel.add(rightPanel, BorderLayout.EAST);

        add(borderPanel, BorderLayout.CENTER);

        setTitle("Checkers By Legends");

        setSize(800, 800);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        setVisible(true);

        setResizable(false);

        // Set the icon image

        ImageIcon icon = new ImageIcon("CheckersGame\\images\\CheckersIcon.jpeg"); // Replace with the path to your icon image

        setIconImage(icon.getImage());

    }

    void updateBoard() {

        for (int i = 0; i < 8; i++) {

            for (int j = 0; j < 8; j++) {

                CheckerPiece piece = game.getPieceAt(i, j);

                if (piece != null) {

                    ImageIcon icon = new ImageIcon(piece.getImagePath());

                    Image image = icon.getImage();

                    Image scaledImage = image.getScaledInstance(squareSize, squareSize, Image.SCALE\_SMOOTH);

                    ImageIcon scaledIcon = new ImageIcon(scaledImage);

                    squares[i][j].setIcon(scaledIcon);

                    squares[i][j].setHorizontalAlignment(JLabel.CENTER);

                    squares[i][j].setVerticalAlignment(JLabel.CENTER);

                } else {

                    squares[i][j].setIcon(null);

                }

            }

        }

    }

    private class SquareClickListener extends MouseAdapter {

        private int row;

        private int col;

        public SquareClickListener(int row, int col) {

            this.row = row;

            this.col = col;

        }

        @Override

        public void mouseClicked(MouseEvent e) {

            if (selectedRow == -1 && selectedCol == -1) {

                CheckerPiece piece = game.getPieceAt(row, col);

                if (piece != null && game.isRedTurn() == (piece.getColor() == Color.RED)) {

                    selectedRow = row;

                    selectedCol = col;

                    squares[row][col].setBackground(selectedColor);

                    selectSound.play(); // Play select sound

                }

            } else {

                if (game.movePiece(selectedRow, selectedCol, row, col)) {

                    if (Math.abs(selectedRow - row) == 2) {

                        captureSound.play(); // Play capture sound if a piece was captured

                    } else {

                        placeSound.play(); // Play place sound

                    }

                    selectedRow = -1;

                    selectedCol = -1;

                    updateBoard();

                    for (int i = 0; i < 8; i++) {

                        for (int j = 0; j < 8; j++) {

                            squares[i][j].setBackground((i % 2 == j % 2) ? almondColor : brownColor);

                        }

                    }

                } else {

                    selectedRow = -1;

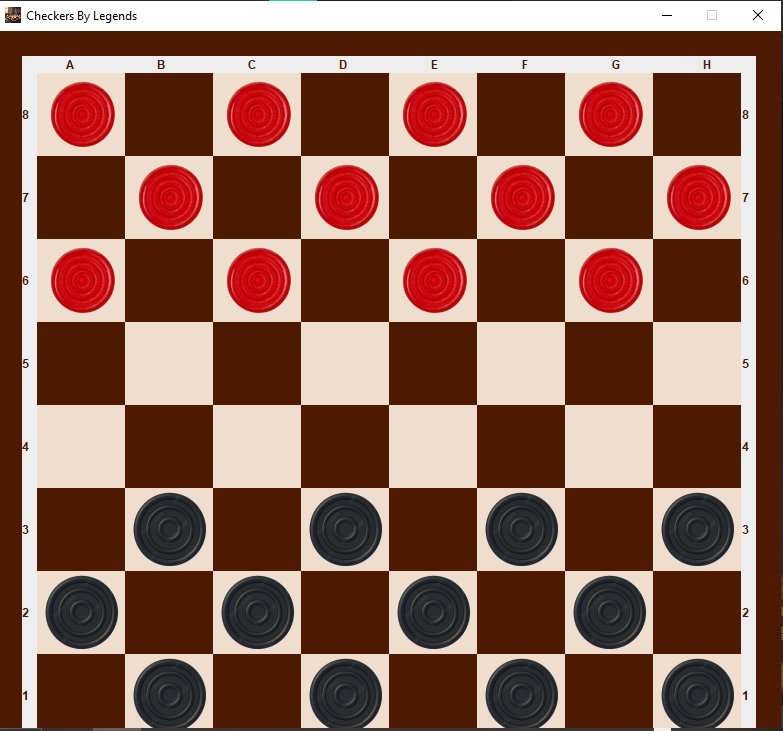
                    selectedCol = -1;

                    squares[row][col].setBackground((row % 2 == col % 2) ? almondColor : brownColor);

                }

            }

}

 *Result*

**File 2:**

***CheckersPieces:***

import java.awt.Color;

public class CheckerPiece {

    private Color color;

    private boolean isKing;

    private String imagePath;

    public CheckerPiece(Color color, String imagePath) {

        this.color = color;

        this.isKing = false;

        this.imagePath = imagePath;

    }

    public Color getColor() {

        return color;

    }

    public boolean isKing() {

        return isKing;

    }

  public void makeKing() {

        isKing = true;

    }

    public String getImagePath() {

        return imagePath;

    }

    public void setImagePath(String imagePath) {

        this.imagePath = imagePath;

    }

}

A screenshot of a game

Description automatically generated *Result*

**File 3:**

***CheckersGame:***

import java.awt.Color;

import javax.swing.SwingUtilities;

public class CheckerGame {

    private CheckerPiece[][] board;

    private boolean redTurn;

        // Load the winning sound effect

    // private SoundEffect winSound;

    public CheckerGame() {

        board = new CheckerPiece[8][8];

        redTurn = true;

        initializeBoard();

        //winSound = new SoundEffect("soundeffects\\clapping and cheering.wav"); // Replace with your win sound path

    }

    private void initializeBoard() {

        // Place red pieces in starting positions (rows 0-2)

        for (int row = 0; row < 3; row++) {

            for (int col = row % 2; col < 8; col += 2) {

                board[row][col] = new CheckerPiece(Color.RED, "CheckersGame\\images\\Red\_piece.png");

            }

        }

        // Place black pieces in starting positions (rows 5-7)

        for (int row = 5; row < 8; row++) {

            for (int col = row % 2; col < 8; col += 2) {

                board[row][col] = new CheckerPiece(Color.BLACK, "CheckersGame\\images\\Black\_piece.png");

            }

        }

    }

    public void resetGame() {

        redTurn = true;

        initializeBoard();

    }

    public CheckerPiece getPieceAt(int row, int col) {

        return board[row][col];

    }

    public boolean movePiece(int fromRow, int fromCol, int toRow, int toCol) {

        CheckerPiece piece = getPieceAt(fromRow, fromCol);

        if (piece != null && isValidMove(fromRow, fromCol, toRow, toCol)) {

            board[toRow][toCol] = piece;

            board[fromRow][fromCol] = null;

            // Handle pawn promotion

            if (!piece.isKing() && (toRow == 0 || toRow == 7)) {

                piece.makeKing();

                piece.setImagePath(piece.getColor() == Color.RED ? "CheckersGame\\images\\Red\_king\_piece.png" : "CheckersGame\\images\\Black\_king\_piece.png");

            }

            // Handle capturing pieces

            if (Math.abs(fromRow - toRow) == 2) {

                int capturedRow = (fromRow + toRow) / 2;

                int capturedCol = (fromCol + toCol) / 2;

                board[capturedRow][capturedCol] = null;

            }

              // Check if the game is won

              String winner = isGameOver();

              if (winner != null) {

                  // Play the sound in a new thread

                  new Thread(() -> {

                      SoundEffect winSound = new SoundEffect("CheckersGame\\soundeffects\\splash 03.wav");

                      winSound.play();

                  }).start();

                  // Display the winning frame

                  SwingUtilities.invokeLater(() -> new WinningFrame(winner).setVisible(true));

                }

            redTurn = !redTurn;

            return true;

        }

        return false;

    }

    private boolean isValidMove(int startX, int startY, int endX, int endY) {

        // Check if the end position is within the bounds of the board

        if (endX < 0 || endX >= 8 || endY < 0 || endY >= 8 || board[endX][endY] != null) {

            return false; // Destination is outside the board or already occupied

        }

        CheckerPiece piece = board[startX][startY];

        int direction = (piece.getColor() == Color.RED) ? 1 : -1; // Forward direction for regular pieces

        // Check if the move is a regular move (one step diagonally)

        if (Math.abs(endX - startX) == 1 && Math.abs(endY - startY) == 1) {

            if (piece.isKing() || (endX - startX == direction)) {

                return true; // King pieces can move in all directions; regular pieces can only move forward

            }

        }

        // Check if the move is a capturing move (jump over opponent's piece)

        if (Math.abs(endX - startX) == 2 && Math.abs(endY - startY) == 2) {

            int capturedX = (startX + endX) / 2;

            int capturedY = (startY + endY) / 2;

            // Check if there is an opponent's piece to capture

            CheckerPiece capturedPiece = board[capturedX][capturedY];

            if (capturedPiece != null && capturedPiece.getColor() != piece.getColor()) {

                if (piece.isKing() || (endX - startX == 2 \* direction)) {

                    return true; // Allow capturing moves for kings in all directions, and regular pieces only forward

                }

            }

        }

        return false; // Invalid move

    }

    private boolean isKingRow(int row, Color color) {

        return (color == Color.RED && row == 7) || (color == Color.BLACK && row == 0);

    }

    public boolean isRedTurn() {

        return redTurn;

    }

    public String isGameOver() {

        int redPieces = 0;

        int blackPieces = 0;

        for (int row = 0; row < 8; row++) {

            for (int col = 0; col < 8; col++) {

                CheckerPiece piece = board[row][col];

                if (piece != null) {

                    if (piece.getColor() == Color.RED) {

                        redPieces++;

                    } else if (piece.getColor() == Color.BLACK) {

                        blackPieces++;

                    }

                }

            }

        }

        if (redPieces == 0) {

            return "black"; // Black wins

        } else if (blackPieces == 0) {

            return "red"; // Red wins

        } else {

            // Check for king capture (no valid moves for a player)

            if (!hasValidMoves(redTurn ? Color.RED : Color.BLACK)) {

                return redTurn ? "black" : "red"; // Opponent wins if current player has no valid moves

            }

        }

        return null; // No winner yet

    }

    private boolean hasValidMoves(Color color) {

        // Iterate through all remaining pieces of the player with the specified color

        for (int row = 0; row < 8; row++) {

            for (int col = 0; col < 8; col++) {

                CheckerPiece piece = board[row][col];

                if (piece != null && piece.getColor() == color) {

                    // Check for any valid capture or advance move for the piece

                    if (isValidMove(row, col, row + (color == Color.RED ? 1 : -1), col + 1) || // Diagonal right (capture or advance)

                        isValidMove(row, col, row + (color == Color.RED ? 1 : -1), col - 1) || // Diagonal left (capture or advance)

                        (piece.isKing() && (isValidMove(row, col, row - (color == Color.RED ? 1 : -1), col + 1) || // King: Diagonal right (capture or advance)

                                            isValidMove(row, col, row - (color == Color.RED ? 1 : -1), col - 1)))) { // King: Diagonal left (capture or advance)

                        return true; // Valid move found, player has options

                    }

                }

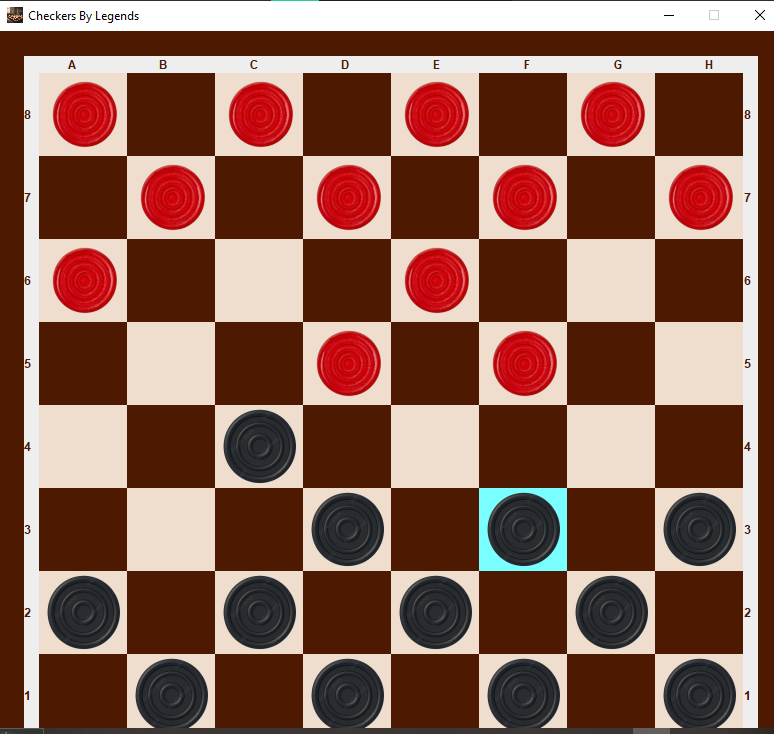
            }

        }

        return false; // No valid moves found for all pieces of the specified color

      }

}

*Result* 

**File 4:**

***Sound Effect:***

import javax.sound.sampled.\*;

import java.io.File;

import java.io.IOException;

public class SoundEffect {

    private Clip clip;

    public SoundEffect(String soundFileName) {

        try {

            File soundFile = new File(soundFileName);

            AudioInputStream audioIn = AudioSystem.getAudioInputStream(soundFile);

            clip = AudioSystem.getClip();

            clip.open(audioIn);

        } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {

            e.printStackTrace();

        }

    }

    public void play() {

        if (clip != null) {

            clip.setFramePosition(0); // Rewind to the beginning

            clip.start();

        }

    }

}

**File 5:**

***SplashFrame:***

import javax.swing.\*;

import java.awt.\*;

public class SplashFrame extends JFrame {

    public SplashFrame() {

        SoundEffect splashSound = new SoundEffect("CheckersGame\\soundeffects\\splash.wav"); // Add your splash screen sound file here

        splashSound.play();

        // Set up the frame

        setTitle("Splash Frame");

        setSize(800, 800);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE); // Dispose this frame only

        setLocationRelativeTo(null); // Center the frame

        setResizable(false);

        setUndecorated(true);

        // Set the icon image

        ImageIcon icon = new ImageIcon("CheckersGame\\images\\CheckersIcon.jpeg"); // Replace with the path to your icon image

        setIconImage(icon.getImage());

        // Create a panel with the background image

        JPanel backgroundPanel = new JPanel() {

            @Override

            protected void paintComponent(Graphics g) {

                super.paintComponent(g);

                ImageIcon imageIcon = new ImageIcon("CheckersGame\\images\\Starting . . ..png"); // Replace "splash\_background.jpg" with the path to your image

                Image image = imageIcon.getImage();

                g.drawImage(image, 0, 0, getWidth(), getHeight(), this);

            }

        };

        backgroundPanel.setLayout(new BorderLayout());

        // Add the panel to the frame

        add(backgroundPanel);

        // Display this frame for 5 seconds

       Timer timer = new Timer(5000, e -> {

        dispose(); // Close this frame

        SwingUtilities.invokeLater(() -> new CheckersGUI().setVisible(true)); // Open the main frame

         });

        timer.setRepeats(false); // Only fire once

        timer.start();

    }

}

*Result*  A screenshot of a game

Description automatically generated

**File 6:**

***Winning Frame:***

import javax.swing.\*;

import java.awt.\*;

public class WinningFrame extends JFrame {

    public WinningFrame(String color) {

        setTitle("\" Congratulations! \"");

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setSize(800, 800);

        setLocationRelativeTo(null);

        setResizable(false);

        setUndecorated(true);

        // Set the icon image

        ImageIcon icon = new ImageIcon("CheckersGame\\images\\CheckersIcon.jpeg"); // Replace with the path to your icon

                                                                                   // image

        setIconImage(icon.getImage());

        JPanel contentPanel = new JPanel();

        contentPanel.setBackground(Color.WHITE);

        // Load the image based on the color

        ImageIcon image;

        if (color.equals("red")) {

            image = new ImageIcon("CheckersGame\\images\\RedWin.jpeg");

        } else if (color.equals("black")) {

            image = new ImageIcon("CheckersGame/images/BlackWin.jpeg");

        } else {

            image = new ImageIcon("CheckersGame/images/Tie.jpeg");

        }

        // Scale the image to fit within 800x800

        double aspectRatio = image.getIconWidth() / (double) image.getIconHeight();

        int maxWidth = Math.min(image.getIconWidth(), 800);

        int maxHeight = (int) (maxWidth / aspectRatio);

        Image scaledImage = image.getImage().getScaledInstance(maxWidth, maxHeight, Image.SCALE\_SMOOTH);

        image = new ImageIcon(scaledImage);

        JLabel imageLabel = new JLabel(image);

        imageLabel.setHorizontalAlignment(JLabel.CENTER);

        contentPanel.add(imageLabel);

        // Display the winning message in metallic golden color

        JLabel messageLabel = new JLabel();

        String message = (color.equals("red") ? "\"Red Wins!\""

                : (color.equals("black") ? "\"Black Wins!\"" : "It's a Tie!"));

        messageLabel.setText(message);

        messageLabel.setFont(new Font("Times New Romanl", Font.BOLD, 40));

        messageLabel.setForeground(new Color(211, 175, 55)); // Metallic golden color

        messageLabel.setHorizontalAlignment(JLabel.CENTER);

        messageLabel.setBounds(0, 70, 800, 40); // Adjust vertical position

        imageLabel.add(messageLabel); // Add message label to image label

        JButton closeButton = new JButton("Quit Game");

        closeButton.setFont(new Font("Times New Roman", Font.BOLD, 14));

        closeButton.setHorizontalAlignment(JButton.CENTER);

        closeButton.setBounds(250, 125, 125, 25); // Adjust button position

        closeButton.setForeground(Color.WHITE);

        closeButton.setBackground(new Color(211, 175, 55));

        closeButton.addActionListener(e -> System.exit(0));

        imageLabel.add(closeButton); // Add button to image label

        JButton restartButton = new JButton("Restart Game");

        restartButton.setFont(new Font("Times New Roman", Font.BOLD, 13));

        restartButton.setHorizontalAlignment(JButton.CENTER);

        restartButton.setBounds(430, 125, 125, 25); // Adjust button position

        restartButton.setForeground(Color.WHITE);

        restartButton.setBackground(new Color(211, 175, 55));

        restartButton.addActionListener(e -> {

            dispose(); // Close the winning frame

            new CheckersGUI(); // Restart the game

        });

        imageLabel.add(restartButton); // Add button to image label

        add(contentPanel);

        setVisible(true);

    }

}



*Result(for red)*

*Result(for black)*



**File 7:**

***Game Runner***

import javax.swing.\*;

public class GameRunner {

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            SplashFrame splashFrame = new SplashFrame();

            splashFrame.setVisible(true);

            Timer timer = new Timer(5000, e -> {

                splashFrame.dispose();

            });

            timer.setRepeats(false);

            timer.start();

        });

    }

}

**Attached Files**

***Splashframe:***



***Pieces pics:***







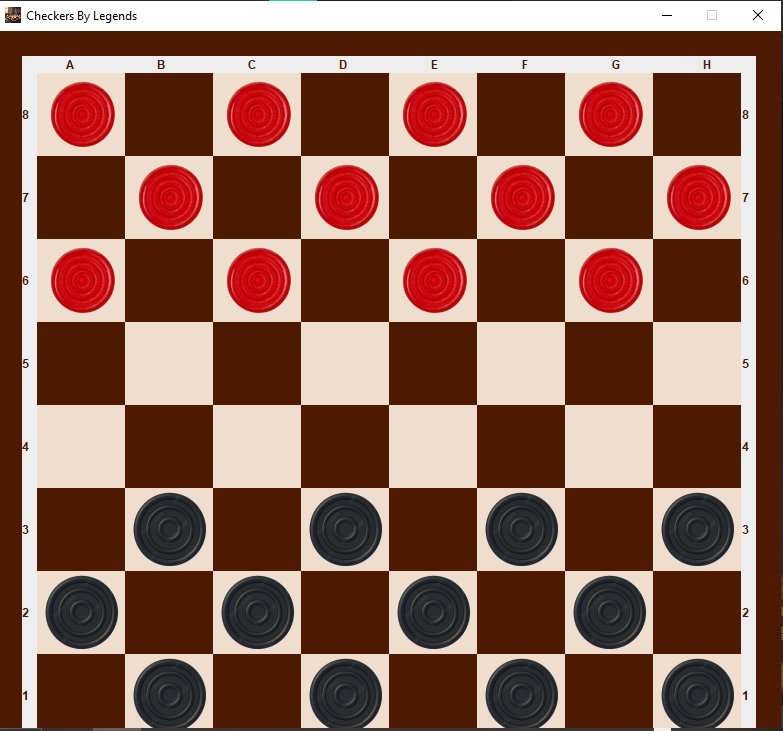


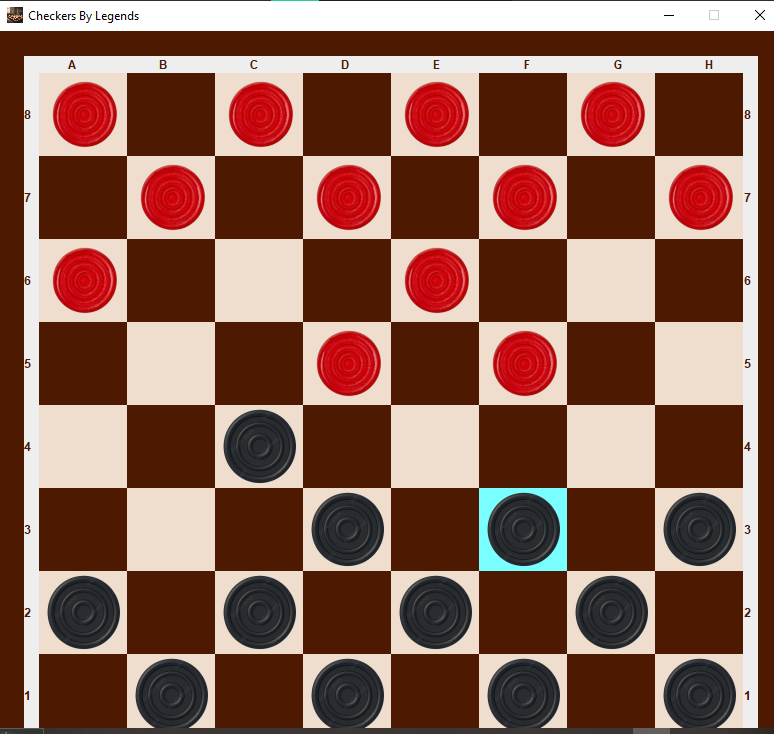
***Winning Frames:***

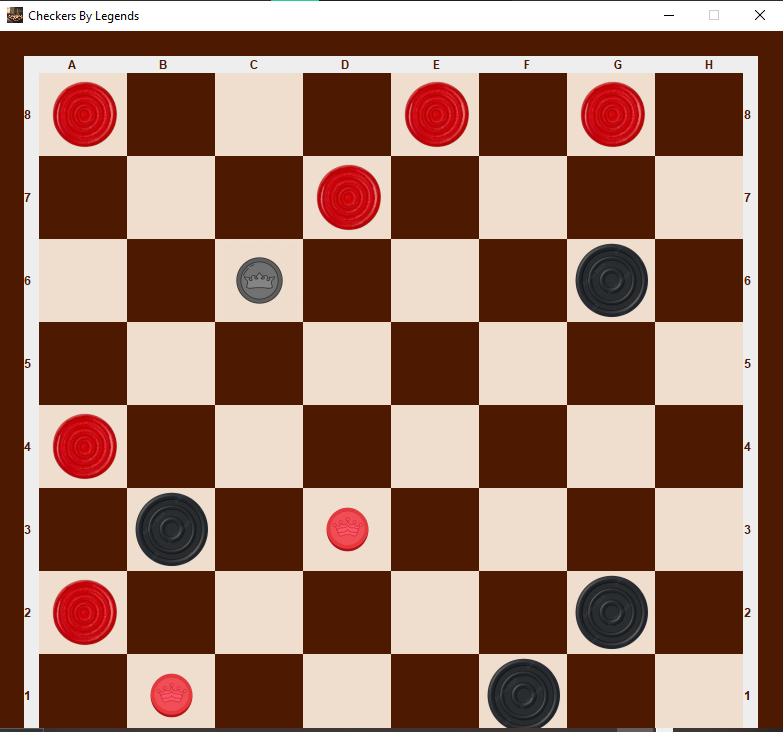




***Game board:***







***NOTE:***

Sound is also available in our Checkers game but it cannot be shown in word file but we can give you its demo.

**Discrete Mathematics Concepts**

Graph Theory:

Board Representation: The checkers board can be represented as a graph where each square is a node, and each possible legal move (from one square to another) is an edge. This helps in visualizing and analyzing possible moves and sequences of moves.

Connectivity and Pathfinding: Graph theory helps in finding paths and connections between nodes. This is crucial for understanding moves, sequences of captures, and potential winning strategies.

Combinatorics:

Counting Positions: Combinatorics is used to count the number of possible board positions, which is essential for estimating the complexity of the game.

Move Combinations: This concept helps in determining the number of possible moves at any given point, and the different ways pieces can be arranged on the board.

Game Theory:

Optimal Strategies: Game theory helps in finding optimal strategies for players. It involves analyzing payoffs, making decisions based on possible moves of the opponent, and minimizing losses while maximizing gains.

Minimax Algorithm: This is a common algorithm used in AI for checkers, where the program simulates all possible moves to minimize the possible loss for a worst-case scenario.

Logic and Boolean Algebra:

Rule Enforcement: Logic is used to enforce the rules of the game, ensuring that moves are legal and that sequences follow the game's constraints.

Decision Making: Logical operators and conditions are used to make decisions about the best moves, check for end-game conditions, and determine winning or losing states.

Set Theory:

Piece Management: Pieces on the board can be considered as elements of a set, where set operations help manage the pieces, such as adding or removing pieces after moves.

Board States: Different board states can be viewed as sets, and transitions between states can be analyzed using set operations.