import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.canvas.GraphicsContext;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.paint.Color;  
import javafx.scene.input.MouseEvent;  
import javafx.stage.Stage;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Alert.AlertType;  
import javafx.stage.FileChooser;  
import java.io.\*;  
  
public class PositionalGameFrame extends Application {  
  
 private int gridSize = 8; // Default grid size  
 private int cellSize = 50; // Size of each cell  
 private int selectedNodeX = -1;  
 private int selectedNodeY = -1;  
 private boolean isPlayer1Turn = true;  
 private boolean[][] horizontalSticks; // Horizontal sticks between nodes  
 private boolean[][] verticalSticks; // Vertical sticks between nodes  
 private List<String> moves = new ArrayList<>(); // To store game moves  
 private Canvas canvas; // Canvas for drawing the board  
 private TextField gridSizeField; // Text field for entering grid size  
  
 @Override  
 public void start(Stage primaryStage) {  
 BorderPane root = new BorderPane();  
  
 // Configuration panel  
 HBox configPanel = new HBox(10);  
 Label gridSizeLabel = new Label("Grid Size:");  
 gridSizeField = new TextField();  
 gridSizeField.setText(Integer.*toString*(gridSize)); // Set default grid size  
 Button newGameButton = new Button("New Game");  
 newGameButton.setOnAction(e -> createNewGame());  
 configPanel.getChildren().addAll(gridSizeLabel, gridSizeField, newGameButton);  
 root.setTop(configPanel);  
  
 // Create the canvas  
 canvas = new Canvas(gridSize \* cellSize, gridSize \* cellSize);  
 canvas.setOnMousePressed(this::handleMouseClick);  
 root.setCenter(canvas);  
  
 // Control panel  
 HBox controlPanel = new HBox(10);  
 Button saveButton = new Button("Save");  
 saveButton.setOnAction(e -> saveGame());  
 Button loadButton = new Button("Load");  
 loadButton.setOnAction(e -> loadGame());  
 Button exitButton = new Button("Exit");  
 exitButton.setOnAction(e -> primaryStage.close());  
 controlPanel.getChildren().addAll(saveButton, loadButton, exitButton);  
 root.setBottom(controlPanel);  
  
 primaryStage.setScene(new Scene(root));  
 primaryStage.setTitle("Positional Game");  
 primaryStage.show();  
  
 createNewGame();  
 }  
  
 private void createNewGame() {  
 gridSize = Integer.*parseInt*(gridSizeField.getText()); // Get grid size from text field  
 horizontalSticks = new boolean[gridSize][gridSize - 1];  
 verticalSticks = new boolean[gridSize - 1][gridSize];  
 selectedNodeX = -1;  
 selectedNodeY = -1;  
 isPlayer1Turn = true;  
 moves.clear();  
 generateRandomSticks();  
 redraw();  
 }  
  
 private void generateRandomSticks() {  
 Random random = new Random();  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize - 1; j++) {  
 horizontalSticks[i][j] = random.nextBoolean();  
 }  
 }  
 for (int i = 0; i < gridSize - 1; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 verticalSticks[i][j] = random.nextBoolean();  
 }  
 }  
 }  
  
 private void redraw() {  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 gc.clearRect(0, 0, gridSize \* cellSize, gridSize \* cellSize);  
  
 // Draw horizontal sticks  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize - 1; j++) {  
 if (horizontalSticks[i][j]) {  
 gc.strokeLine(j \* cellSize + cellSize, i \* cellSize + cellSize / 2,  
 (j + 1) \* cellSize + cellSize, i \* cellSize + cellSize / 2);  
 }  
 }  
 }  
  
 // Draw vertical sticks  
 for (int i = 0; i < gridSize - 1; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 if (verticalSticks[i][j]) {  
 gc.strokeLine(j \* cellSize + cellSize / 2, i \* cellSize + cellSize,  
 j \* cellSize + cellSize / 2, (i + 1) \* cellSize + cellSize);  
 }  
 }  
 }  
  
 // Draw stones  
 for (String move : moves) {  
 String[] parts = move.split(",");  
 int x = Integer.*parseInt*(parts[0]);  
 int y = Integer.*parseInt*(parts[1]);  
 boolean isPlayer1 = Boolean.*parseBoolean*(parts[2]);  
 drawStone(x, y, isPlayer1);  
 }  
 }  
  
 private void drawStone(int x, int y, boolean isPlayer1) {  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 gc.setFill(isPlayer1 ? Color.*RED* : Color.*BLUE*);  
 gc.fillOval(x \* cellSize + cellSize / 4, y \* cellSize + cellSize / 4,  
 cellSize / 2, cellSize / 2);  
 }  
  
 private void handleMouseClick(MouseEvent event) {  
 int x = (int) (event.getX() / cellSize);  
 int y = (int) (event.getY() / cellSize);  
  
 if (isValidMove(x, y)) {  
 moves.add(x + "," + y + "," + isPlayer1Turn);  
 drawStone(x, y, isPlayer1Turn);  
 isPlayer1Turn = !isPlayer1Turn;  
 }  
  
 // Check for winner  
 if (!hasValidMoves(isPlayer1Turn)) {  
 announceWinner(!isPlayer1Turn);  
 }  
 }  
  
 private boolean isValidMove(int x, int y) {  
 if (selectedNodeX == -1 && selectedNodeY == -1) {  
 return true; // First move is always valid  
 }  
  
 if (x == selectedNodeX && y == selectedNodeY) {  
 return false; // Can't select the same node twice  
 }  
  
 if (x == selectedNodeX) {  
 // Vertical move  
 int minY = Math.*min*(selectedNodeY, y);  
 int maxY = Math.*max*(selectedNodeY, y);  
 for (int i = minY; i < maxY; i++) {  
 if (!verticalSticks[i][x]) {  
 return false; // Vertical stick is missing  
 }  
 }  
 } else if (y == selectedNodeY) {  
 // Horizontal move  
 int minX = Math.*min*(selectedNodeX, x);  
 int maxX = Math.*max*(selectedNodeX, x);  
 for (int i = minX; i < maxX; i++) {  
 if (!horizontalSticks[y][i]) {  
 return false; // Horizontal stick is missing  
 }  
 }  
 } else {  
 return false; // Diagonal move is not allowed  
 }  
  
 return true;  
 }  
  
 private boolean hasValidMoves(boolean isPlayer1Turn) {  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 if (isValidMove(j, i)) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 private void announceWinner(boolean isPlayer1Winner) {  
 String winner = isPlayer1Winner ? "Player 1" : "Player 2";  
 Alert alert = new Alert(AlertType.*INFORMATION*);  
 alert.setTitle("Game Over");  
 alert.setHeaderText(null);  
 alert.setContentText("Game over! " + winner + " wins!");  
 alert.showAndWait();  
 }  
  
 private void saveGame() {  
 try {  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Save Game");  
 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Positional Game File", "\*.pgf"));  
 File file = fileChooser.showSaveDialog(null);  
 if (file != null) {  
 ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(file));  
 outputStream.writeObject(horizontalSticks);  
 outputStream.writeObject(verticalSticks);  
 outputStream.writeObject(isPlayer1Turn);  
 outputStream.writeObject(moves);  
 outputStream.close();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void loadGame() {  
 try {  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Load Game");  
 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Positional Game File", "\*.pgf"));  
 File file = fileChooser.showOpenDialog(null);  
 if (file != null) {  
 ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(file));  
 horizontalSticks = (boolean[][]) inputStream.readObject();  
 verticalSticks = (boolean[][]) inputStream.readObject();  
 isPlayer1Turn = (boolean) inputStream.readObject();  
 moves = (List<String>) inputStream.readObject();  
 inputStream.close();  
 redraw();  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.canvas.GraphicsContext;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.paint.Color;  
import javafx.scene.input.MouseEvent;  
import javafx.stage.Stage;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Alert.AlertType;  
import javafx.stage.FileChooser;  
import java.io.\*;  
  
public class PositionalGameFrame extends Application {  
  
 private int gridSize = 8; // Default grid size  
 private int cellSize = 50; // Size of each cell  
 private int selectedNodeX = -1;  
 private int selectedNodeY = -1;  
 private boolean isPlayer1Turn = true;  
 private boolean[][] horizontalSticks; // Horizontal sticks between nodes  
 private boolean[][] verticalSticks; // Vertical sticks between nodes  
 private List<String> moves = new ArrayList<>(); // To store game moves  
 private Canvas canvas; // Canvas for drawing the board  
 private TextField gridSizeField; // Text field for entering grid size  
  
 @Override  
 public void start(Stage primaryStage) {  
 BorderPane root = new BorderPane();  
  
 // Configuration panel  
 HBox configPanel = new HBox(10);  
 Label gridSizeLabel = new Label("Grid Size:");  
 gridSizeField = new TextField();  
 gridSizeField.setText(Integer.*toString*(gridSize)); // Set default grid size  
 Button newGameButton = new Button("New Game");  
 newGameButton.setOnAction(e -> createNewGame());  
 configPanel.getChildren().addAll(gridSizeLabel, gridSizeField, newGameButton);  
 root.setTop(configPanel);  
  
 // Create the canvas  
 canvas = new Canvas(gridSize \* cellSize, gridSize \* cellSize);  
 canvas.setOnMousePressed(this::handleMouseClick);  
 root.setCenter(canvas);  
  
 // Control panel  
 HBox controlPanel = new HBox(10);  
 Button saveButton = new Button("Save");  
 saveButton.setOnAction(e -> saveGame());  
 Button loadButton = new Button("Load");  
 loadButton.setOnAction(e -> loadGame());  
 Button exitButton = new Button("Exit");  
 exitButton.setOnAction(e -> primaryStage.close());  
 controlPanel.getChildren().addAll(saveButton, loadButton, exitButton);  
 root.setBottom(controlPanel);  
  
 primaryStage.setScene(new Scene(root));  
 primaryStage.setTitle("Positional Game");  
 primaryStage.show();  
  
 createNewGame();  
 }  
  
 private void createNewGame() {  
 gridSize = Integer.*parseInt*(gridSizeField.getText()); // Get grid size from text field  
 horizontalSticks = new boolean[gridSize][gridSize - 1];  
 verticalSticks = new boolean[gridSize - 1][gridSize];  
 selectedNodeX = -1;  
 selectedNodeY = -1;  
 isPlayer1Turn = true;  
 moves.clear();  
 generateRandomSticks();  
 redraw();  
 }  
  
 private void generateRandomSticks() {  
 Random random = new Random();  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize - 1; j++) {  
 horizontalSticks[i][j] = random.nextBoolean();  
 }  
 }  
 for (int i = 0; i < gridSize - 1; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 verticalSticks[i][j] = random.nextBoolean();  
 }  
 }  
 }  
  
 private void redraw() {  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 gc.clearRect(0, 0, gridSize \* cellSize, gridSize \* cellSize);  
  
 gc.setLineWidth(3.0); // Set line width for sticks  
  
 // Draw horizontal sticks  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize - 1; j++) {  
 if (horizontalSticks[i][j]) {  
 gc.strokeLine(j \* cellSize, (i + 1) \* cellSize,  
 (j + 1) \* cellSize, (i + 1) \* cellSize);  
 }  
 }  
 }  
  
 // Draw vertical sticks  
 for (int i = 0; i < gridSize - 1; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 if (verticalSticks[i][j]) {  
 gc.strokeLine((j + 1) \* cellSize, i \* cellSize,  
 (j + 1) \* cellSize, (i + 1) \* cellSize);  
 }  
 }  
 }  
  
 gc.setLineWidth(1.0); // Reset line width for cell borders  
  
 // Draw cell borders  
 for (int i = 0; i <= gridSize; i++) {  
 gc.strokeLine(0, i \* cellSize, gridSize \* cellSize, i \* cellSize); // Horizontal lines  
 gc.strokeLine(i \* cellSize, 0, i \* cellSize, gridSize \* cellSize); // Vertical lines  
 }  
  
 // Draw stones  
 for (String move : moves) {  
 String[] parts = move.split(",");  
 int x = Integer.*parseInt*(parts[0]);  
 int y = Integer.*parseInt*(parts[1]);  
 boolean isPlayer1 = Boolean.*parseBoolean*(parts[2]);  
 drawStone(x, y, isPlayer1);  
 }  
 }  
  
  
  
 private void drawStone(int x, int y, boolean isPlayer1) {  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 gc.setFill(isPlayer1 ? Color.*RED* : Color.*BLUE*);  
 gc.fillOval(x \* cellSize + cellSize / 4, y \* cellSize + cellSize / 4,  
 cellSize / 2, cellSize / 2);  
 }  
  
 private void handleMouseClick(MouseEvent event) {  
 double mouseX = event.getX();  
 double mouseY = event.getY();  
  
 // Check if the click is within the bounds of the sticks  
 int x = (int) (mouseX / cellSize);  
 int y = (int) (mouseY / cellSize);  
 double offsetX = mouseX % cellSize;  
 double offsetY = mouseY % cellSize;  
 boolean isHorizontal = Math.*abs*(offsetY - cellSize / 2) < Math.*abs*(offsetX - cellSize / 2);  
  
 if (isHorizontal && offsetY >= cellSize / 4 && offsetY <= cellSize \* 3 / 4) {  
 // Horizontal stick clicked  
 if (isValidMove(x, y) && isValidMove(x, y + 1)) {  
 moves.add(x + "," + y + "," + isPlayer1Turn);  
 drawStone(x, y, isPlayer1Turn);  
 isPlayer1Turn = !isPlayer1Turn;  
 }  
 } else if (!isHorizontal && offsetX >= cellSize / 4 && offsetX <= cellSize \* 3 / 4) {  
 // Vertical stick clicked  
 if (isValidMove(x, y) && isValidMove(x + 1, y)) {  
 moves.add(x + "," + y + "," + isPlayer1Turn);  
 drawStone(x, y, isPlayer1Turn);  
 isPlayer1Turn = !isPlayer1Turn;  
 }  
 }  
  
 // Check for winner  
 if (!hasValidMoves(isPlayer1Turn)) {  
 announceWinner(!isPlayer1Turn);  
 }  
 }  
  
  
 private boolean isValidMove(int x, int y) {  
 if (selectedNodeX == -1 && selectedNodeY == -1) {  
 return true; // First move is always valid  
 }  
  
 if (x == selectedNodeX && y == selectedNodeY) {  
 return false; // Can't select the same node twice  
 }  
  
 if (x == selectedNodeX) {  
 // Vertical move  
 int minY = Math.*min*(selectedNodeY, y);  
 int maxY = Math.*max*(selectedNodeY, y);  
 for (int i = minY; i < maxY; i++) {  
 if (!verticalSticks[i][x]) {  
 return false; // Vertical stick is missing  
 }  
 }  
 } else if (y == selectedNodeY) {  
 // Horizontal move  
 int minX = Math.*min*(selectedNodeX, x);  
 int maxX = Math.*max*(selectedNodeX, x);  
 for (int i = minX; i < maxX; i++) {  
 if (!horizontalSticks[y][i]) {  
 return false; // Horizontal stick is missing  
 }  
 }  
 } else {  
 return false; // Diagonal move is not allowed  
 }  
  
 return true;  
 }  
  
 private boolean hasValidMoves(boolean isPlayer1Turn) {  
 for (int i = 0; i < gridSize; i++) {  
 for (int j = 0; j < gridSize; j++) {  
 if (isValidMove(j, i)) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 private void announceWinner(boolean isPlayer1Winner) {  
 String winner = isPlayer1Winner ? "Player 1" : "Player 2";  
 Alert alert = new Alert(AlertType.*INFORMATION*);  
 alert.setTitle("Game Over");  
 alert.setHeaderText(null);  
 alert.setContentText("Game over! " + winner + " wins!");  
 alert.showAndWait();  
 }  
  
 private void saveGame() {  
 try {  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Save Game");  
 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Positional Game File", "\*.pgf"));  
 File file = fileChooser.showSaveDialog(null);  
 if (file != null) {  
 ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(file));  
 outputStream.writeObject(horizontalSticks);  
 outputStream.writeObject(verticalSticks);  
 outputStream.writeObject(isPlayer1Turn);  
 outputStream.writeObject(moves);  
 outputStream.close();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void loadGame() {  
 try {  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Load Game");  
 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Positional Game File", "\*.pgf"));  
 File file = fileChooser.showOpenDialog(null);  
 if (file != null) {  
 ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(file));  
 horizontalSticks = (boolean[][]) inputStream.readObject();  
 verticalSticks = (boolean[][]) inputStream.readObject();  
 isPlayer1Turn = (boolean) inputStream.readObject();  
 moves = (List<String>) inputStream.readObject();  
 inputStream.close();  
 redraw();  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}