Készítsük el a Királynők nevű egyszerű logikai játékot. Ehhez adott egy 8x8-as sakktábla, amelyen a királynőket úgy kell elhelyeznünk balról-jobbra oszloponként,

hogy egyik oszlopba se tegyünk úgy királynőt, hogy a korábbiakat üsse (a királynő függőlegesen, vízszintesen és átlósan is tetszőleges távolságra léphet).

Mielőtt hozzákezd a feladat megoldásának, olvassa végig a feladatlapot, mivel bizonyos funkciók utólagos bevezetése esetlegesen más,

korábbi funkciók újraírását eredményezheti.

Hozza létre kódból a 8x8 darab gombot tartalmazó sakktáblát, és az eseménykezelőt a gombokhoz úgy, hogy mindig csak a sakktábla

legelső üres oszlopába tehessünk királynőt.

Ellenőrízzük, hogy olyan helyre akarja-e tenni a játékos az új királynőt, ahol nem üti egyik korábban elhelyezett sem,

és ne engedjük lerakni a királynőt rossz helyre.

--------------

Ha sikerült mind a 8 királynőt elhelyezni a táblán, akkor gratuláljunk a játékosnak egy dialógus ablakon megjelenített Gratulálok, nyertél!

felirattal, majd indítsuk újra a játékot.

Hozzunk létre egy menüpontot, amellyel újrakezdhetjük a játékot, továbbá egy Visszavonás menüpontot,

amivel egyesével vissza lehet vonni a lépéseket (ezeket érdemes egy vektorban/listben tárolni).

--------------

Jelenítsük meg, hogy adott pályaállás esetén mely mezőkre léphetünk, pl. a gomb háttérszínét színezzük ezeknél át (btn.seBackground(Color.GRAY);).

Hozzunk létre egy menüpontot, amivel megállíthatjuk a játékot. Ekkor semmire sem lehet kattintani (vagy nincs hatása a kattintásnak),

és nem is láthatjuk a pályán lévő feliratokat. A menüpont újbóli használatával a játék mutassa újra a pályaállást és lehessen folytatni a játékot.

Valósítsuk meg a játékot MV architektúrát alkalmazva, azaz egy külön osztályban legyen implementálva a játék táblájának modellje és az egyes műveletei

(lépések, visszavonás, sikerült-e bejárni a táblát).

--------------

Készítsünk egy időzítőt és számlálót, mely alapján tájékozódhat a játékos, hogy mennyi

idő telt el az aktuális játék kezdete óta. Ha lejárt az idő,akkor vesztett a játékos.

package eightqueens;

import eightqueens.model.Engine;

import eightqueens.view.Frame;

public class Launcher {

public static void main(String[] args) {

Engine engine = new Engine();

Frame frame = new Frame(engine);

engine.startNewGame();

frame.showFrame();

}

}

package eightqueens.model;

public class Position {

private int row;

private int column;

public Position(int row, int column) {

this.row = row;

this.column = column;

}

public int getRow() {

return row;

}

public int getColumn() {

return column;

}

/\*

@Override

public boolean equals(Object o) {

if (o instanceof Position) {

Position p = (Position) o;

return row == p.getRow() && column == p.getColumn();

}

return

false;

}

@Override

public int hashCode() {

return row \* 11 + column \* 7;

}\*/

}

package eightqueens.model;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

public class Engine {

private static final int SIZE = 8;

private boolean[][] queens;

private List<Position> queensPositions;

private boolean paused;

public void startNewGame() {

paused = false;

queensPositions = new ArrayList<>();

queens = new boolean[SIZE][SIZE];

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

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

queens[i][j] = false;

}

}

}

public int getSize() {

return SIZE;

}

public void put(int i, int j) {

if (!paused) {

getFirstEmptyColumn().ifPresent(col -> putIfNotInScope(col, i, j));

}

}

private Optional<Integer> getFirstEmptyColumn() {

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

if (columnIsEmpty(j)) {

return Optional.of(j);

}

}

return Optional.empty();

}

private void putIfNotInScope(Integer col, int i, int j) {

if (j == col && isNotInScope(i, j)) {

queens[i][j] = true;

queensPositions.add(new Position(i, j));

}

}

public boolean isNotInScope(int i, int j) {

for (Position p : queensPositions) {

Position p2 = new Position(i, j);

if (isTheSameRow(p, p2) || isTheSameColumn(p, p2) || isTheSameDiagonal(p, p2)) {

return false;

}

}

return true;

}

private boolean isTheSameRow(Position p, Position p2) {

return p.getRow() == p2.getRow();

}

private boolean isTheSameColumn(Position p, Position p2) {

return p.getColumn() == p2.getColumn();

}

private boolean isTheSameDiagonal(Position p, Position p2) {

return Math.abs(p.getRow() - p2.getRow()) == Math.abs(p.getColumn() - p2.getColumn());

}

private boolean columnIsEmpty(int j) {

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

if (queens[i][j]) {

return false;

}

}

return true;

}

public boolean isQueen(int i, int j) {

return queens[i][j];

}

/\*public boolean won() {

if (SIZE == queensPositions.size()) {

startNewGame();

return true;

}

return false;

}

public void undo() {

if (!paused && !queensPositions.isEmpty()) {

Position lastQueen = getLastQueenPosition();

removeQueen(lastQueen);

queensPositions.remove(lastQueen);

}

}\*/

private Position getLastQueenPosition() {

return queensPositions.get(queensPositions.size() - 1);

}

private void removeQueen(Position pos) {

queens[pos.getRow()][pos.getColumn()] = false;

}

public boolean canPutHere(int i, int j) {

return isNotInScope(i, j) && j == queensPositions.size();

}

public void togglePause() {

paused = !paused;

}

public boolean isPaused() {

return paused;

}

}

package eightqueens.view;

//import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

//import javax.swing.JOptionPane;

import eightqueens.model.Engine;

import javax.swing.UIManager;

public class Frame extends JFrame {

//private static final long serialVersionUID = 8316572961171616624L;

private Engine engine;

private JMenuItem pauseMenuItem;

public Frame(Engine engine) {

super("8 Queens");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.engine = engine;

applyNimbusLookAndFeelTheme();

}

public void showFrame() {

createFields();

setMenu();

pack();

setVisible(true);

}

private void applyNimbusLookAndFeelTheme(){

try {

for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (Exception ignored) {

}

}

private void createFields() {

getContentPane().setLayout(new GridLayout(engine.getSize(), engine.getSize()));

for (int i = 0; i < engine.getSize(); ++i) {

for (int j = 0; j < engine.getSize(); ++j) {

JButton field = new JButton();

//field.setBackground(getFieldBackground(i, j));

field.setPreferredSize(new Dimension(80, 80));

field.setFont(field.getFont().deriveFont(30.0f));

addFieldActionListener(i, j, field);

getContentPane().add(field);

}

}

}

private void addFieldActionListener(int i, int j, JButton field) {

field.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

engine.put(i, j);

updateFields();

//checkVictoryCondition();

}

});

}

/\*private void checkVictoryCondition() {

if (engine.won()) {

JOptionPane.showMessageDialog(this, "Oh nice.");

updateFields();

}

}\*/

private void setMenu() {

JMenuBar menuBar = new JMenuBar();

JMenu menu = new JMenu("Game");

menuBar.add(menu);

//addRestartMenuItem(menu);

//addUndoMenuItem(menu);

//addPauseMenuItem(menu);

setJMenuBar(menuBar);

}

/\*private void addRestartMenuItem(JMenu menu) {

JMenuItem menuItem = new JMenuItem("Restart");

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

engine.startNewGame();

updateFields();

updatePauseMenuItem();

}

});

menu.add(menuItem);

}

private void addUndoMenuItem(JMenu menu) {

JMenuItem menuItem = new JMenuItem("Undo");

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

//engine.undo();

updateFields();

}

});

menu.add(menuItem);

}

private void addPauseMenuItem(JMenu menu) {

pauseMenuItem = new JMenuItem("Pause");

pauseMenuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

engine.togglePause();

updatePauseMenuItem();

updateFields();

}

});

menu.add(pauseMenuItem);

}

private void updatePauseMenuItem() {

pauseMenuItem.setText(engine.isPaused() ? "Resume" : "Pause");

}\*/

private void updateFields() {

for (int i = 0; i < engine.getSize(); ++i) {

for (int j = 0; j < engine.getSize(); ++j) {

Component c = getContentPane().getComponent(i \* engine.getSize() + j);

JButton field = (JButton) c;

field.setText(getFieldText(i, j));

//field.setBackground(getFieldBackground(i, j));

}

}

}

private String getFieldText(int i, int j) {

return !engine.isPaused() && engine.isQueen(i, j) ? "♕" : "";

}

/\*private Color getFieldBackground(int i, int j) {

if (!engine.isPaused() && engine.canPutHere(i, j)) {

return Color.BLUE;

}

return (i + j) % 2 == 0 ? Color.GRAY : Color.WHITE; }\*/}