

**МАГИСТЪРСКИ ФАКУЛТЕТ**

**ПРОГРАМА “СОФТУЕРНИ ТЕХНОЛОГИИ В ИНТЕРНЕТ”**

Курсова работа по **INFM119/INFM169**

„ Обектно-ориентирано програмиране в Интернет “

на тема : „ TCP/IP симулатор за играта Цветна конфигурация “

**Разработил: Преподавател:**

Бояна Канатрска доц. Иван Момчев

Ф№ F89795

**СЪДЪРЖАНИЕ**

[1. Задание за разработка 3](#__RefHeading___Toc156378239)

[1.1 Правила на играта “Цветна конфигурация” 3](#__RefHeading___Toc156378240)

[1.2 Примерна игра 4](#__RefHeading___Toc156378241)

[1.3 Комуникационен протокол](#__RefHeading___Toc156378242) 6

[2. Документация за поддръжка](#__RefHeading___Toc156378244) 6

[2.1 Компилация на кода](#__RefHeading___Toc156378245) 6

[3. Ръководство за потребителя](#__RefHeading___Toc156378247) 7

[3.1 Стартиране на сървъра](#__RefHeading___Toc156378248) 7

[3.2 Стартиране на клиента](#__RefHeading___Toc156378249) 7

[3.3 Изход от приложението 8](#__RefHeading___Toc156378250)

[4. Рекламна стратегия](#__RefHeading___Toc156378251) 8

5. [Програмен код](#__RefHeading___Toc156378252) 10

[run.bat 1](#__RefHeading___Toc156378253)0

[Board.java](#__RefHeading___Toc156378254) 11

[GameServer.java 2](#__RefHeading___Toc156378255)2

[Player.java 3](#__RefHeading___Toc156378256)3

[RandomClient.java](#__RefHeading___Toc156378257) 40

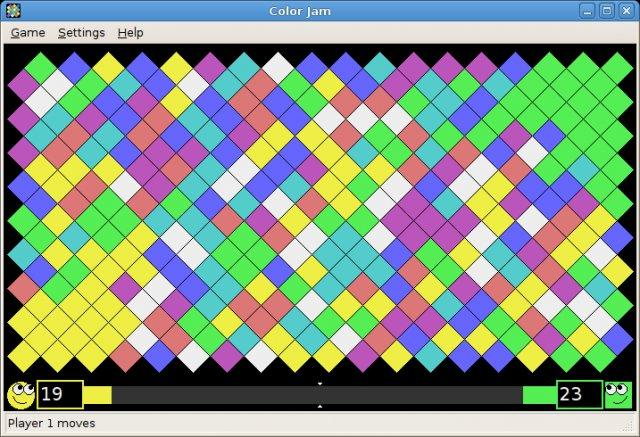
Drawingboard.java........………………………………………………………………………50

## **1. Задание за разработка**

Да се разработи на Java клиент – сървър приложение, базирано на сокети, което да отговаря на следните условия:

* Да се разработи специализиран протокол за сокет комуникация между клиента и сървъра.
* При сървъра да има много нишки, които да се борят за ресурси.
* Да може да бъде управлявана борбата за ресурси чрез промяна на параметри
* При клиента да бъде реализиран графичен интерфейс.

[**1.1 Правила на играта “Цветна конфигурация”**](#__RefHeading___Toc156378240)

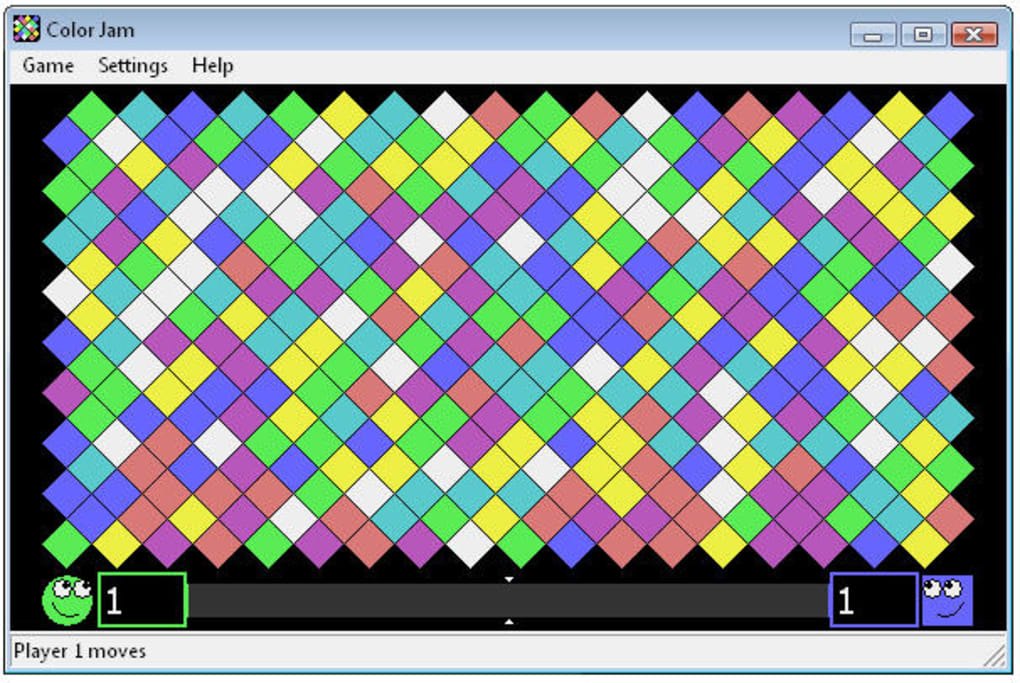


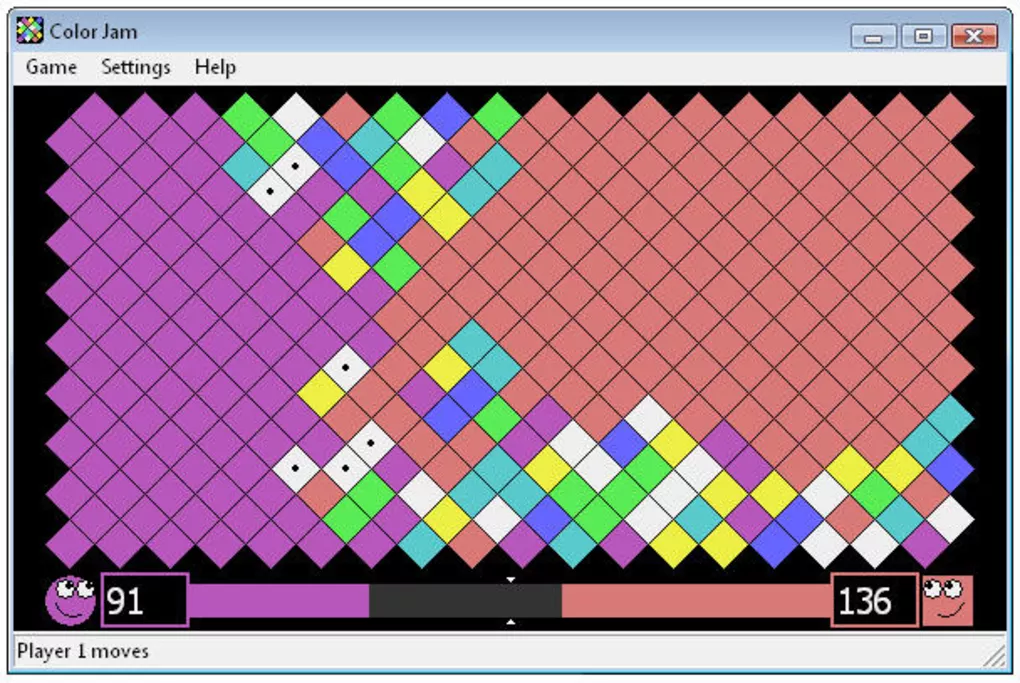
[“Цветна конфигурация”](#__RefHeading___Toc156378240) е проста игра на логически игри, в която промените цвета си, за да спечелите повече територия от опонента си. Това е клон на стара компютърна игра, която излиза в началото на 90-те години.

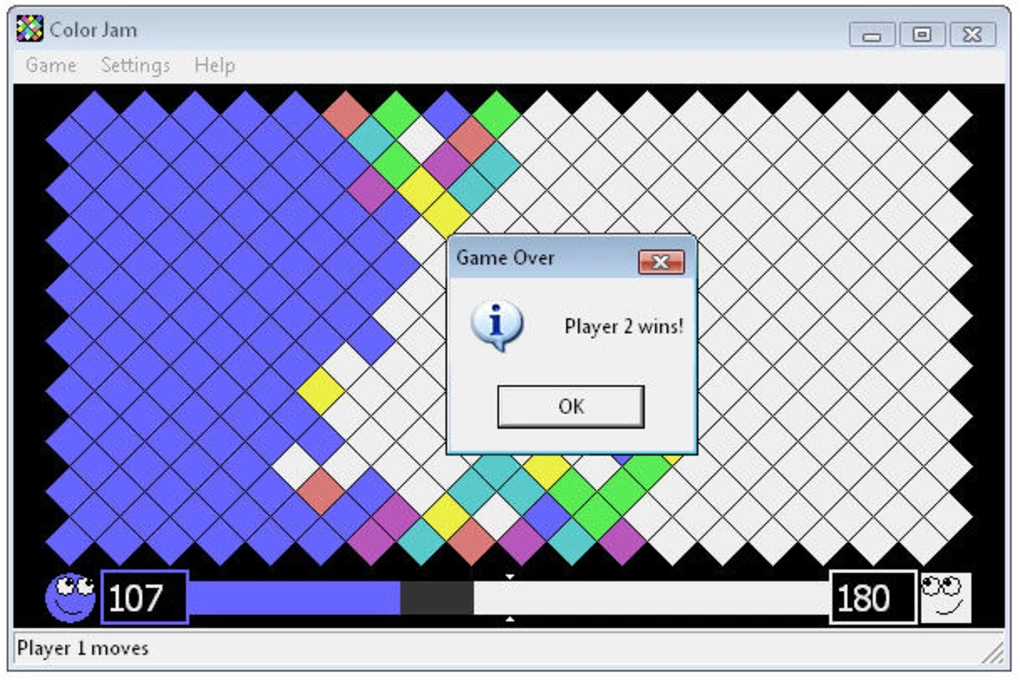
Започвате с един или няколко плочки в долния ляв ъгъл. Компютърът започва горе вдясно. Можете да кликнете върху всеки цвят, за да завладеете съседни плочки от този цвят. Следващия път, когато промените цвета си, те ще се променят с вас. Всеки играч се характеризира с кординатите на една плочка от игралното поле. Тази плочка е негово владение, както и всички околни плочки със същия цвят, които образуват непрекъсваема площадка. На всеки ход играчът има право да смени цвета на плочките, които притежава. Новият цвят може да бъде избран само, ако не бъде ползва от друг играч. Чрез промяна на цвета играчът приобщава нови плочки към своята площадка. Това са плочките, които са в пряко съседство с площадката/дъска. Целта на играта е да се “окупират” колкото се може повече плочки, докато се завземат всички, които не са собственост на някой от играчите. Печели този играч, който превземе най-голям брой плочки. Можете да видите номера на текущите резултати и лентата за напредъка в долната част.

*Важно ограничение:* не можете да изберете същият цвят като противника си!

**1.2 Примерна игра**







**1.3 Комуникационен протокол**

Комуникацията между сървъра и клиентите се осъществява по стандартни TCP сокети. За определен интервал от време сървърът очаква прикачането на клиентите, след което комуникацията протича чрез последователност от съобщения между сървъра и клиентите. На всяка стъпка от комуникацията сървърът изпраща актуалното състояние на играта и изчаква определен интервал време за да може клиента да му върне съобщение за избора на съответния играч.

Съобщението от сървъра към клиента съдържа четири цели положителни числа на първия ред M, N, C и P (1<P<100, P<M<10P, P<N<10P, P<C<5P), разделени с интервал. На втория ред две цели положителни числа, разделени с интервал задават X и Y (1<=X<=M, 1<=Y<=N) координатите на стартовата плочка за играча. Цветът на плочката с координати (X, Y) определя текущият цвят, с който играчът играе. На третият ред са изброени P- цели положителни числа, разделени с интервал, които задават номерата на цветовете блокирани от играчите. Следват N реда от M цели положителни числа, разделени с интервал и задаващи текущото състояние на игралното табло. Всяка плочка е представена от число в интервала от 1 до C, съответстващо на цвета.

Съобщението от клиента към сървъра се състои само от един ред, на който е записано число, показващо към кой от позволените цветове иска да премине играчът.

**2.Документация за поддръжка**

Пълен листинг на програмният текст е представен в 5. Програмен код . Кодът е документиран и организиран в пет класа и три основни: за сървъра, за клиента , диската/панела на играта.

Тъй като продуктът все още е в начален етап на развитие, множество опции са “твърдо” заложени в програмния код. Всяка по-сериозна настройка изисква цялостно изграждане на приложението от програмния код.

**2.1 Компилация на кода**

За да бъде компилирано приложението е нужно да се работи на Java съвместима операционна система. Компилацията се извършва със стандартните развойни средства, предоставени от Sun Microssystems, но също така е възможно използването и на други развойни средства, като JCrator или Eclipse.

Процесът на компилация и стартиране може да се автоматизира, чрез използването на файлове за пакетна обработка (Batch files Windows XP).

Преди компилация кодът трябва да е правилно организиран в йерархия от директории, според изискването за разполагане на пакети в Java.

**3. Ръководство за потребителя**

Играта е реализирана под формата на конзолно приложение, което предоставя възможност за визуализиране на сесията , в която протичат тестовете.

След стартирането на сървъра, в определен времеви интервал, сървърът изчака закачането на клиента. От технологична гледна точка е възможно закачането и на повече клиенти, но по задание сървърът е оптимизиран да поддържа до сто клиента.

/ Тестването протича с четири клиента. /

**3.1 Стартиране на сървъра**

Сървърът се стартира от конзолата на операционната система, а приложението поема грижата да изобрази работния екран.

Windows XP команден ред:

start java.exe -classpath . com.gmail.boianaradkova.Coloring

Преди да бъде изпълнена командата трябва правилно да се зададат пътищата до Java виртуалната машина и тя да е налична, като приложение за съответната компютърна система.

**3.2 Стартиране на клиента**

Клинтът се стартира според инструкциите, които е предоставил създателят му. Тъй като клиентът не е обвързан с конкретен програмен език, стартирането може да се различава в зависимост от използваната платформа за изработката му.

Стартирането на тестовия клиент се извършва в конзолата на операционната система.

* + - * Windows XP команден ред:
      * start java.exe -classpath . com.gmail.boianaradkova.Coloring.RandomStategyClient

Преди да бъде изпълнена командата трябва правилно да се зададат пътищата до Java виртуалната машина и тя да е налична, като приложение за съответната компютърна система.

**3.3 Изход от приложението**

Затварянето на приложенето може да стане, чрез затваряне на конзолния прозорец или затваряне на графично представеният прозорец. Всеки от клиентите може да бъде затворен спрямо инструкциите в приложената към него документация.

**4. Рекламна стратегия**

Тук предложената игра представлява уникален софтуерен продукт, който има за цел да наложи новаторски подход за реализиране на състезания по програмиране. Въпреки безспорните предимства, които проектът прелага, основната добавена стойност ще бъде генерирана от програмите на участниците, взели участие.

Играта е така разработен, че да позволи на участниците максимално да се концентрират върху решаването на поставения проблем, който по същество представлява изграждане на ефективна стратегия за игра, в условията на силна неопредленост. Тъй като симулаторът поставя участниците в условия, в които всеки играе срещу всички останали, залагане на елементи от изкуствения интелект ще направи решенията на участниците уникални по своята реализация.

В съвременните условия на динамирчно развиващи се пазари иновациите са основен двигател на човешкия прогрес, а от там и на усшешния бизнес. Тук предложеният симулатор дава възможност да се провокира креативното мислене в младите хора, чиито идеи в последствие биха намерили приложение при решаването на реални проблеми от ежедневната практика за произвоство на качествен софтуер, позициониран в предния ешалон на научния прогрес.

Състезание по програмиране, базирано на провокативна задача и симулатор с високо качество дава възможностите на млади, способни и творчески ориентирани служителиза по разчупено мислене следствие отразявващо се на бъдещите им предприети решения. Като страничен ефект от такова състезаение може да се добави положителен PR, който носи организирането на съревнования от подобен характер.

Не на последно място, играта позволява да се осъществят множество обстойни тестове и поставяне на предложените решения в различни по сложност ситуации. Визуалното представяне на процеса за тестване предлага изключително интуитивна възможност на проверяващите да наблюдават ходовете предопределящи развитието на играта.

Тук предложената визуална среда може да бъде основа, която да адаптиран за множество логически игри, а на определен етап да се превърне в стандартизирана платформа за провеждане на симулации за игри с логически характер, които противопоставят множество играчи един срещу друг.

**5.Програмен код**

Програмният код е организиран в набор от файлове, както слева:

**run.bat**

cd bin

start java.exe -classpath . com.gmail.boianaradkova.GameServer 3379 5 0

start /MIN java.exe -classpath . com.gmail.boianaradkova.RandomClient 3379 localhost

start /MIN java.exe -classpath . com.gmail.boianaradkova.RandomClient 3379 localhost

start /MIN java.exe -classpath . com.gmail.boianaradkova.RandomClient 3379 localhost

cd ..

**/\*\*Board**

\* Coloring, Version 0.1

\* New Bulgarian University

\*

\* Copyright (c) 2018 Boyana Kantarska

\*/

package com.gmail.boianaradkova;

import java.awt.Color;

import java.awt.Graphics;

import java.util.concurrent.atomic.AtomicBoolean;

/\*\*

\* Board with color tiles.

\*

\* @author Boyana Kantarska

\*/

class Board {

/\*\* Lock for synchronization. \*/

private AtomicBoolean locked = new AtomicBoolean(false);

/\*\* Board width. \*/

private int columns = -1;

/\*\* Board height. \*/

private int rows = -1;

/\*\* Number of colors on the board. \*/

private int numOfColors = -1;

/\*\* Grid of the board. \*/

private int grid[][] = {};

/\*\*

\* Recursive flooding with a new color.

\*

\* @param x Start flooding from coordinate x.

\* @param y Start flooding from coordinate y.

\* @param oldColor Old color which should be replaced.

\* @param newColor New color for replacement.

\*/

private void flood(int x, int y, int oldColor, int newColor) {

if (x < 0) {

return;

}

if (x >= grid.length) {

return;

}

if (y < 0) {

return;

}

if (y >= grid[x].length) {

return;

}

if (grid[x][y] != oldColor) {

return;

}

grid[x][y] = newColor;

flood(x - 1, y, oldColor, newColor);

flood(x, y - 1, oldColor, newColor);

flood(x + 1, y, oldColor, newColor);

flood(x, y + 1, oldColor, newColor);

}

/\*\*

\* Constructor.

\*

\* @param columns Board width.

\* @param rows Board height.

\* @param numOfColors Number of colors on the board.

\*/

public Board(int columns, int rows, int numOfColors) {

locked.set(false);

this.columns = columns;

this.rows = rows;

this.numOfColors = numOfColors;

grid = new int[columns][rows];

/\* Random colors arrangement. \*/

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

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

grid[i][j] = 1 + (int) (Math.random() \* numOfColors);

}

}

}

/\*\*

\* Constructor.

\*

\* @param columns Board width.

\* @param rows Board height.

\* @param numOfColors Number of colors on the board.

\* @param grid Board state.

\*/

public Board(int columns, int rows, int numOfColors, int[][] grid) {

this(columns, rows, numOfColors);

/\* Random colors arrangement. \*/

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

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

this.grid[i][j] = grid[i][j];

}

}

}

/\*\*

\* Board width.

\*

\* @return Width.

\*/

public int getColumns() {

return columns;

}

/\*\*

\* Board height.

\*

\* @return Height.

\*/

public int getRows() {

return rows;

}

/\*\*

\* Number of colors on the board.

\*

\* @return Number of colors.

\*/

public int getNumOfColors() {

return numOfColors;

}

/\*\*

\* Color on a specific position.

\*

\* @param x X coordinate.

\* @param y Y coordinate.

\*

\* @return Color.

\*/

public int getColorIndex(int x, int y) {

return grid[x][y];

}

/\*\*

\* Convert color index from HSV to RGB color value.

\*

\* https://en.wikipedia.org/wiki/HSL\_and\_HSV

\*

\* @param x X coordinate of the cell.

\* @param y Y coordinate of the cell.

\*

\* @return Color object.

\*/

private Color getColor(int x, int y) {

int Hi;

int R = 0, G = 0, B = 0;

double H, S, V;

double f, p, q, t;

int val = grid[x][y];

int min\_val = 1;

int max\_val = numOfColors;

val -= min\_val;

max\_val -= min\_val;

H = 360 \* val / max\_val;

S = 0.8;

V = 0.8;

Hi = (int) (H / 60) % 6;

f = H / 60 - Hi;

p = V \* (1 - S);

q = V \* (1 - f \* S);

t = V \* (1 - (1 - f) \* S);

switch (Hi) {

case 0:

R = (int) (V \* 255);

G = (int) (t \* 255);

B = (int) (p \* 255);

break;

case 1:

R = (int) (q \* 255);

G = (int) (V \* 255);

B = (int) (p \* 255);

break;

case 2:

R = (int) (p \* 255);

G = (int) (V \* 255);

B = (int) (t \* 255);

break;

case 3:

R = (int) (p \* 255);

G = (int) (q \* 255);

B = (int) (V \* 255);

break;

case 4:

R = (int) (t \* 255);

G = (int) (p \* 255);

B = (int) (V \* 255);

break;

case 5:

R = (int) (V \* 255);

G = (int) (p \* 255);

B = (int) (q \* 255);

break;

}

while (R > 0xFF) {

R >>= 1;

}

while (G > 0xFF) {

G >>= 1;

}

while (B > 0xFF) {

B >>= 1;

}

return new Color(R, G, B);

}

/\*\*

\* Check for the synchronization lock.

\*

\* @return True if the board is locked and false if the board is not locked.

\*/

public boolean isLocked() {

return locked.get();

}

/\*\*

\* Lock and unlock the board.

\*

\* @param locked State.

\*/

public void setLocked(boolean locked) {

this.locked.set(locked);

}

/\*\*

\* Change the color of region starting on specific coordinate.

\*

\* @param x X coordinate.

\* @param y Y coordinate.

\* @param color New color.

\*/

public void change(int x, int y, int color) {

flood(x, y, grid[x][y], color);

}

/\*\*

\* Draws the board on a specific graphic context.

\*

\* @param g Graphic context.

\* @param width Width of the drawing area.

\* @param height Height of the drawing area.

\*/

public void draw(Graphics g, int width, int height) {

if (getColumns() == 0) {

return;

}

if (getRows() == 0) {

return;

}

int cellSize = 0;

int a = (width - 1) / (getColumns() + 1);

int b = (height - 1) / (getRows() + 1);

if (a < b) {

cellSize = a;

} else {

cellSize = b;

}

cellSize--;

int xOffset = width / 2 - (getColumns() \* (cellSize + 1)) / 2;

int yOffset = height / 2 - (getRows() \* (cellSize + 1)) / 2;

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

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

g.setColor(getColor(i, j));

g.fillRect(xOffset + i \* (cellSize + 1), yOffset + j \* (cellSize + 1), cellSize, cellSize);

}

}

}

/\*\* Representing the board as a string. \*/

public String toString() {

String text = "";

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

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

text += grid[i][j];

if (i < columns - 1) {

text += " ";

}

}

if (j < rows - 1) {

text += "\n";

}

}

return text;

}

}

**\*\* GameServer**

\* Coloring, Version 0.1

\* New Bulgarian University

\*

\* Copyright (c) 2018 Boyana Kantarska

\*/

package com.gmail.boianaradkova;

import java.awt.Graphics;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Collections;

import java.util.Vector;

import javax.swing.JFrame;

/\*\*

\* The game engine.

\*

\* @author Boyana Kantarska

\*/

@SuppressWarnings("serial")

public class GameServer extends JFrame {

/\*\* Port number for the game server. \*/

static private int port = -1;

/\*\* Clients have given seconds to connect. \*/

static private int connectionTimeout = -1;

/\*\* Each client should response in a given seconds. \*/

static private int responseTimeout = -1;

/\*\* Index of the player on turn. \*/

private int playingIndex = -1;

/\*\* Game board with lock. \*/

private Board board = null;

/\*\* Holding handle to each player as thread. \*/

private Vector<Player> players = new Vector<>();

/\*\* Drawing area for the board. \*/

private DrawingPanel canvas;

/\*\* Order of the players should be random. \*/

private void shufflePlayers() {

Collections.shuffle(players);

}

/\*\* Matching the playing player index with the handle. \*/

private Player playingNow() {

return players.elementAt(playingIndex);

}

/\*\*

\* Check for blocked colors.

\*

\* @param playerIndex Player on turn.

\* @param color New selected color.

\*

\* @return True if the color is not in use and false if the color is in use.

\*/

private boolean isColorUsed(int playerIndex, int color) {

boolean isUsed = false;

for (int p = 0; p < players.size(); p++) {

/\* If the player is the same do nothing. \*/

if (p == playerIndex) {

continue;

}

/\* The color should correspond. \*/

if (color != ((Player) players.elementAt(p)).getColor()) {

continue;

}

isUsed = true;

break;

}

return isUsed;

}

/\*\*

\* Check the position in first positioning of the player.

\*

\* @param playerIndex Index of the player to be positioned.

\* @param x X coordinate on the board.

\* @param y Y coordinate on the board.

\* @param color Color which should be used.

\*

\* @return True if the position is available and false if the position is not

\* available.

\*/

private boolean isGoodPosition(int playerIndex, int x, int y) {

boolean isGood = true;

int color = board.getColorIndex(x, y);

for (int p = 0; p < playerIndex; p++) {

if (((Player) players.elementAt(p)).getColor() != color) {

continue;

}

if (((Player) players.elementAt(p)).getX() != x) {

continue;

}

if (((Player) players.elementAt(p)).getY() != y) {

continue;

}

isGood = false;

break;

}

return isGood;

}

/\*\*

\* Constructor with parameters.

\*

\* @param title Game server title.

\*/

public GameServer(String title) {

super(title);

canvas = new DrawingPanel(board);

this.getContentPane().add(canvas);

}

/\*\*

\* Draw visual component.

\*

\* @param g Graphic context.

\*/

@Override

public void paint(Graphics g) {

if (board == null) {

return;

}

// board.draw(g, this.getWidth(), this.getHeight());

}

/\*\*

\* Initialize board and players. Should be executed after the players are

\* connected.

\*/

public void init() {

Thread.currentThread().setPriority(Thread.MIN\_PRIORITY);

int x, y;

/\* Choosing rаndom number of colors. \*/

int numOfColors = (players.size() + 1) + (int) (Math.random() \* 4 \* players.size() - 1);

/\* Choosing random dimensions of the board. \*/

int width = (players.size() + 1) + (int) (Math.random() \* 9 \* players.size() - 1);

int heigth = (players.size() + 1) + (int) (Math.random() \* 9 \* players.size() - 1);

/\* Board creation. \*/

board = new Board(width, heigth, numOfColors);

/\* Initializing players and positioning on the board. \*/

for (int p = 0; p < players.size(); p++) {

do {

x = (int) (Math.random() \* board.getColumns());

y = (int) (Math.random() \* board.getRows());

} while (isGoodPosition(p, x, y) == false);

((Player) players.elementAt(p)).init(x, y, board.getColorIndex(x, y));

}

/\* Players should play turns in random order. \*/

shufflePlayers();

/\* Players' threads should be started. \*/

for (int p = 0; p < players.size(); p++) {

((Player) players.elementAt(p)).setPriority(Thread.MIN\_PRIORITY);

((Player) players.elementAt(p)).start();

}

System.out.println("Game server initialization ...");

canvas.setBoard(board);

}

/\*\*

\* Turn of a player.

\*

\* @param player Player who is on turn.

\*/

synchronized public void doTurn(Player player) {

/\* Synchronization is needed because the board is only one. \*/

while (board.isLocked() == true) {

try {

wait();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

/\* Only one player on time should play. \*/

board.setLocked(true);

if (player.isActive() == true) {

for (int p = 0; p < players.size(); p++) {

if ((players.elementAt(p)) == player) {

playingIndex = p;

}

}

/\* Game state is send via TCP socket. \*/

player.write(this.toString());

/\* Player's move is received via TCP socket. \*/

int color = player.read(responseTimeout);

/\* Player answer should be valid. \*/

if (color < 1 || color > board.getNumOfColors() || isColorUsed(playingIndex, color) == true)

player.setNotActive();

else {

board.change(player.getX(), player.getY(), color);

player.setColor(color);

}

}

canvas.repaint();

System.out.println( this.toString() );

/\* Unlock and notify the others. \*/

board.setLocked(false);

notifyAll();

}

/\*\*

\* Add player to the list of the players.

\*/

public void addPlayer(Player player) {

players.add(player);

}

/\*\*

\* Presenting game state as string.

\*/

public String toString() {

/\* Text of the message sent by the server. \*/

String text = "";

text += board.getColumns() + " " + board.getRows() + " " + board.getNumOfColors() + " " + players.size() + "\n";

text += (1 + playingNow().getX()) + " " + (1 + playingNow().getY()) + "\n";

for (int p = 0; p < players.size(); p++) {

text += ((Player) players.elementAt(p)).getColor();

if (p != players.size() - 1) {

text += " ";

}

}

text += "\n";

text += board + "\n";

return text;

}

/\*\* Wait players to connect. \*/

public void host() {

long start = System.currentTimeMillis();

try {

ServerSocket server = new ServerSocket(port);

server.setSoTimeout(connectionTimeout \* 1000);

System.out.println("Server started on port " + port + " ...");

System.out.println("Server will wait for clients to connect " + connectionTimeout + " seconds ...");

do {

Socket client = null;

try {

client = server.accept();

} catch (Exception ex) {

}

if (client != null) {

addPlayer(new Player("" + System.currentTimeMillis(), client, this));

}

} while ((System.currentTimeMillis() - start) < connectionTimeout \* 1000);

} catch (Exception ex) {

ex.printStackTrace();

}

System.out.println("Connection timeout, " + players.size() + " clients connected ...");

}

/\*\*

\* Main method.

\*

\* @param args Command line parameters.

\*/

public static void main(String args[]) {

port = Integer.valueOf(args[0]);

connectionTimeout = Integer.valueOf(args[1]);

responseTimeout = Integer.valueOf(args[2]);

GameServer game = new GameServer("Game Server ...");

game.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

game.setSize(800, 600);

// game.setVisible(true);

game.host();

game.init();

}

}

**/\*\* Player**

\* Coloring, Version 0.1

\* New Bulgarian University

\*

\* Copyright (c) 2018 Boyana Kantarska

\*/

package com.gmail.boianaradkova;

import java.io.IOException;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

/\*\*

\* Presentation of each player.

\*

\* @author Boyana Kantarska

\*/

class Player extends Thread {

/\*\* Connection socket for each player. \*/

private Socket socket;

/\*\* Input stream. \*/

private BufferedReader in;

/\*\* Output stream. \*/

private PrintWriter out;

/\*\* Handle to the game object. \*/

private GameServer game;

/\*\* Player's name. \*/

private String name;

/\*\* X initial coordinate on the board. \*/

private int x;

/\*\* Y initial coordinate on the board. \*/

private int y;

/\*\* Current color used. \*/

private int color;

/\*\* Total score of the player. \*/

private int score;

/\*\* Player is active until the response is correct. \*/

private boolean active;

/\*\*

\* Constructor.

\*

\* @param name Name of the player.

\* @param socket Socket handle.

\* @param game Game handle.

\*

\* @throws IOException If the socket is broken.

\*/

public Player(String name, Socket socket, GameServer game) throws IOException {

this.game = game;

this.name = name;

this.score = 0;

this.socket = socket;

this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

this.out = new PrintWriter(socket.getOutputStream());

}

/\*\*

\* Initialize player in the beginning of the game.

\*

\* @param color Color of the player.

\* @param x X coordinate.

\* @param y Y Coordinate.

\*/

public void init(int x, int y, int color) {

this.color = color;

this.x = x;

this.y = y;

active = true;

System.out.println("Player " + name + " initialized ...");

}

/\*\* Sets the player not to be active. \*/

public void setNotActive() {

active = false;

}

/\*\*

\* Check is the player active.

\*

\* @return True if the player is active and false if the player is not active.

\*/

public boolean isActive() {

return active;

}

/\*\*

\* Player's x initial coordinate.

\*

\* @return X coordinate.

\*/

public int getX() {

return x;

}

/\*\*

\* Player's y initial coordinate.

\*

\* @return Y coordinate.

\*/

public int getY() {

return y;

}

/\*\*

\* Set new player's color.

\*

\* @param color New color.

\*/

public void setColor(int color) {

this.color = color;

}

/\*\*

\* Get current player's color.

\*

\* @return Color.

\*/

public int getColor() {

return color;

}

/\*\*

\* Read data from the socket in given timeout.

\*

\* @param timeout Number or seconds to wait for response.

\*

\* @return Chosen color.

\*/

public int read(int timeout) {

int color = 0;

String str = "";

try {

Thread.currentThread().sleep(1000 \* timeout);

} catch (InterruptedException ex) {

System.err.println("Thread can not be interrupted: " + ex);

}

try {

str = in.readLine();

} catch (IOException ex) {

str = "";

System.err.println("Receive socket message failed: " + ex);

}

try {

color = (new Integer(str)).intValue();

} catch (Exception ex) {

color = 0;

System.err.println("Incorrect data receieved: " + ex.toString());

}

return color;

}

/\*\*

\* Write data into socket.

\*

\* @param str Data which should be written.

\*/

public void write(String str) {

out.print(str);

out.flush();

}

/\*\* Try to make move on each thread loop.

\* The thread is started fulfilling the "true " condition the game is working.

\* The thread falls asleep by returning a number between 0 and 10 during this time.

\* Exception Capture Condition, which transmit directly about the error.

\*

\*/

public void run() {

while (true) {

game.doTurn(this);

try {

sleep((long) (Math.random() \* 100));

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

}

**/\*\* RandomClient**

\* Coloring, Version 0.1

\* New Bulgarian University

\*

\* Copyright (c) 2018 Boyana Kantarska

\*/

package com.gmail.boianaradkova;

import java.awt.Graphics;

import java.io.BufferedWriter;

import java.io.DataInputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.net.Socket;

import javax.swing.JFrame;

/\*\*

\* Simple client with random playing strategy.

\*

\* @author Boyana Kantarska

\*/

@SuppressWarnings("serial")

public class RandomClient extends JFrame {

/\*\* Socket for client-server communication. \*/

private Socket socket = null;

/\*\* Input stream with information from the server. \*/

private DataInputStream in = null;

/\*\* Output stream with information for the server. \*/

private PrintWriter out = null;

/\*\* Server port. \*/

private int port = -1;

/\*\* Server URL address. \*/

private String address = "";

/\*\* Game board with lock. \*/

private Board board = null;

/\*\*

\* Constructor.

\*

\* @param port TCP/IP communication port.

\* @param address Server URL address.

\*/

public RandomClient(int port, String address) {

super();

this.port = port;

this.address = address;

/\*\*

\* This constructo call and create whit "super"

\* port and address to the Server.

\*

\*/

do {

try {

socket = new Socket(address, port);

} catch (Exception ex) {

socket = null;

}

} while (socket == null);

System.out.println("Client connected ...");

try {

in = new DataInputStream(socket.getInputStream());

out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

} catch (IOException ex) {

System.err.println("Input/Output streams are not available form the socket: " + ex.toString());

}

/\*\*

\* Loop "do" run the Constructor whit "Socket port and network address".

\* By checking for exceptions and errors in the code, returns the socket blank.

\* If the reason for exception isn't dropped, code execution continues with

\* an "while" loop until connection is established.

\*

\*/

/\* Communication done in a separate thread. \*/

Thread thread = new Thread() {

@Override

/\*\*

\*The function allows to start the new thread with

\*next run() method.

\*

\*/

public void run() {

String line = "";

while (line != null) {

int M = 0, N = 0, C = 0, P = 0;

int X = 0, Y = 0;

int usedColors[] = null;

int board[][] = null;

/\*\*

\* Enter an empty string that enters in "while" loop.

\* Declares the coordinates: M , N , C , P , X , Y and

\* arrays : usedColors[] (sets the use of colors) and

\* board[][](two-dimensional array, sets the coordinates

\* of the individual customer's board.

\*

\*/

try {

/\* Parsing of the messages in the communication protocol. \*/

line = in.readLine() + " ";

M = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = line.substring(line.indexOf(' ') + 1);

N = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = line.substring(line.indexOf(' ') + 1);

C = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = line.substring(line.indexOf(' ') + 1);

P = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = in.readLine() + " ";

X = (new Integer(line.substring(0, line.indexOf(' ')))).intValue() - 1;

line = line.substring(line.indexOf(' ') + 1);

Y = (new Integer(line.substring(0, line.indexOf(' ')))).intValue() - 1;

/\*\*

\* "try" block sets the beginning of the graphic drawing through the defined

\* coordinates and methods:

\* readLine- draws according to its set values and transfers to the next line.

\* subString- starts at its first index and ends at its predetermined.

\* indexOf-returns the index from the first string to the last symbol by

\* setting whit intValues.

\*

\*/

line = in.readLine() + " ";

usedColors = new int[P];

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

usedColors[i] = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = line.substring(line.indexOf(' ') + 1);

}

/\*\*

\* Color and graphic drawing are set.

\*/

board = new int[M][N];

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

line = in.readLine() + " ";

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

board[i][j] = (new Integer(line.substring(0, line.indexOf(' ')))).intValue();

line = line.substring(line.indexOf(' ') + 1);

}

}

/\*\*

\* In "for" loop the colors and frame for the player are predetermined.

\*

\*/

/\* Board creation. \*/

RandomClient.this.board = new Board(M, N, C, board);

} catch (IOException ex) {

System.err.println("Incorrect imput data: " + ex.toString());

}

/\*\*

\* The exception checks for failures of failed or interrupted operations.

\* When a problem occurs, a message is printed and returns a string of

\* text information with a correct prefix to the problem.

\*

\*/

int color = 0;

boolean done;

do {

done = true;

color = 1 + (int) (Math.random() \* C);

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

if (color == usedColors[i]) {

done = false;

}

}

} while (done == false);

/\*\*

\*In addition to the exception check, a variable is entered- "color"

\*of an integer type of data that serves as a comparison by the primitive

\*type "boolean". Its function is to return the argument to the object.

\*

\*/

/\* Response from the client to the server. \*/

out.println(color); /\* Closing the flow for color printing. \*/

out.flush(); /\* Close the buffer entry. \*/

/\* Redraw GUI. \*/

repaint();

/\*\*

\* A method that controls the update through a cycle retrieves

\* the repainting components.

\*/

try {

Thread.sleep(100);

} catch (InterruptedException e) {

}

}

}

};

thread.start(); /\* Thread Execution Method. \*/

}

/\*\*

\* Draw visual component.

\*

\* @param g Graphic context.

\*/

@Override /\* Function allowing the drawing method to be executed. \*/

public void paint(Graphics g) {

if (board == null) {

return;

}

board.draw(g, this.getWidth(), this.getHeight());

}

/\*\*

\* The condition "if" verifies whether the board is initialized.

\* If not, it takes the current objects to determine the size and length and height.

\*

\*/

/\*\* Finalize internal state of the objects. \*/

@Override

public void finalize() {

try {

in.close();

} catch (IOException ex) {

}

out.close();

try {

socket.close();

} catch (IOException ex) {

}

}

/\*\*

\* The method collects excess garbage, which is out of control of the developer.

\* Closes the stream by checking again for exceptions and errors.

\*/

/\*\*

\* Main method.

\*

\* @param args Command line parameters.

\*/

public static void main(String[] args) {

int port = Integer.valueOf(args[0]);

String address = "" + args[1];

RandomClient client = new RandomClient(port, address);

client.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

client.setSize(800, 600);

client.setVisible(true);

}

}

**/\*\*Drawing board.**

\* Coloring, Version 0.1

\* New Bulgarian University

\*

\* Copyright (c) 2018 Boyana Kantarska

\*/

package com.gmail.boianaradkova;

import java.awt.Graphics;

import javax.swing.JPanel;

/\*\*

\* Drawing board.

\*

\* @author Boyana Kantarska

\*/

@SuppressWarnings("serial")

class DrawingPanel extends JPanel {

/\*\* Handle to the board which should be drawn. \*/

private Board board;

/\*\*

\* Constructor.

\*

\* @param board Handle to the board.

\*/

DrawingPanel(Board board) {

this.board = board;

}

/\*\*

\* Control paint method.

\*

\* @param g Graphic context.

\*/

public void paintComponent(Graphics g) {

super.paintComponent(g);

board.draw(g, getWidth(), getHeight());

}

/\*\*

\* Change the board handle.

\*

\* @param board Board handle.

\*/

public void setBoard(Board board) {

this.board = board;

}

}