

Содержание

[Введение 5](#_Toc5696)

[1 Постановка задачи 6](#_Toc13190)

[2 Выбор решения 7](#_Toc13343)

[3 Описание программы 8](#_Toc18756)

[3.1 Серверная часть 8](#_Toc9331)

[3.2 Клиентская часть 8](#_Toc25794)

[4 Описание способа организации пользовательского интерфейса 10](#_Toc2359)

[5 Описание результатов работы программы 11](#_Toc6881)

[5.1 Запуск программы. 11](#_Toc6964)

[5.2 Запуск игры. 12](#_Toc8948)

[5.3 Перемещение шашки. 12](#_Toc14202)

[5.4 Конец игры 13](#_Toc25797)

[Заключение 14](#_Toc4537)

[Список используемой литературы 15](#_Toc19474)

[Приложение А. Исходный код 16](#_Toc31324)

[Приложение А.1 – Исходный код Сервера 16](#_Toc13487)

[Приложение А.2 – Исходный код Клиента 26](#_Toc24215)

[Приложение В. UML-диаграммы приложения 55](#_Toc7400)

[Приложение В.1 – UML-диаграмма вариантов использования 55](#_Toc6600)

[Приложение В.2 – UML-диаграмма классов клиента 56](#_Toc30691)

[Приложение В.3 – UML-диаграмма классов сервера 57](#_Toc10958)

[Приложение В.4 – UML-диаграмма деятельности 58](#_Toc2058)

[Приложение В.5 – UML-диаграмма развёртывания 59](#_Toc13647)

[Приложение В.6 – UML-диаграмма последовательности 59](#_Toc10692)

Введение

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. Стоит так же отметить, что приложения написанные на Java не являются платформозависимыми. Платформонезависимость Java опирается на существование виртуальной машины Java (JVM) на каждой поддерживаемой платформе. На разных платформах JVM разные. Но программы на Java могут запуститься на любой из них.

Представленная в данном курсовом проекте программа является простым клиент-серверным приложением, которое позволяет пользователю играть в мультиплеерные шашки.

1. Постановка задачи

Основной задачей данного курсового проекта является закрепление знаний и приобретение практических навыков разработки клиент-серверных приложений на языке Java. Для этого необходимо выполнить программную реализацию модульного приложения «Шашки» на языке Java в среде разработки IntelliJ IDEA.

Программа должна работать так, чтобы два пользователя могли играть в шашки. В проекте предусмотрен графический интерфейс и используются следующие технологии:

1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. Java Stream API

4. Java Multithreading

5. Java Socket и Java Server Socket

1. Выбор решения

Для реализации проекта используются:

* Объектно-ориентированный язык Java;
* Библиотека Swing;
* Библиотека Awt;
* Пакеты, содержащие файлы ресурсов, пользовательский интерфейс;
* Java Stream API;
* IDE IntelliJ IDEA 2022.2.4;

Основной задачей курсового проекта являлась реализация взаимодействия клиента и сервера. Для выполнения данной задачи использовались как UDP так и TCP сокеты.

Взаимодействие между клиентом и сервером должно происходить по принципу “Запрос-Ответ”. При этом для коротких и простых запросов используются более быстрый и простой протокол UDP, а для передаче с сервера больших объёмов информации (например файлов), был использован протокол TCP, способный точно передавать большие объёмы данных.

Интерфейс пользователя реализован с помощью библиотеки Java Swing и содержит в себе стандартный функционал игры «Шашки»:

* Начальное меню;
* Игровые доски для каждого клиента;
* Меню клента;
* Возможность ходить шашками;

1. Описание программы

Программа, состоит из клиентской и серверной части. Сначала запускается сервер, который входит в состояние ожидания запроса клиента.

После чего устройствах запускается 2 клиента, которые по нажатию определённых кнопок, посылают определённый запросы к серверу.

* 1. Серверная часть

Server.java – главный файл серверного процесса, с помощью которого создаётся сервер.

После входа в программу сервер запускается, регистрирует сокеты и встаёт в ожидание запросов от клиентов.

Сервер имеет конструктор создающий нить ServerGame(int PORT),

При получении запроса сервер создаёт нить для его обработки. Метод run(), находящийся в нити, запускает нить ServerGame.

Так же клиентская часть содержит следующие методы:

* public void run() - отправляет сообщения клиентам о начале игры, после чего ожидает сообщения от клиентов, при получении сообщения от клиента отправляет его второму клиенту.
* public void say(String message) - реализует отправку сообщения от сервера клиенту.
  1. Клиентская часть

GuiDemo.java - главный файл проекта, содержащий функции работы программы. Создаёт нить сервера (Server) и нить главного меню (MainThread).

MainThread.java - создаёт и запускает две нити Client, также создаёт фрейм главного меню.

Client .java – главный файл пользовательского процесса, содержащий функции работы программы.

Game.java - файл отвечающий за процесс игры, решает кто, куда и как ходит.

Grid.java - файл отвечающий за создание сетки игрового поля, отрисовки доски.

Так же клиентская часть содержит следующие методы:

* void createCommunicationModuleOn(Socket socket) - выполняет функцию создания коммуникации между сервером и клиентом.
* void makeMove (Move move) –отправляет сообщение серверу о ходе;
* public void run() - получает сообщение с сервера, по содержанию сообщения выполняет определённое действие;
* public void informActionListene(String message) - по полученному сообщения выполняет действие;

.

1. Описание способа организации пользовательского интерфейса

В качестве среды разработки была выбрана программа IntelliJ IDEA. Программа предоставляет все средства, необходимые при разработке графического приложения.

Для реализации пользовательского интерфейса была использована библиотека Swing. Эта библиотека содержит более богатый и удобный набор элементов пользовательского интерфейса, обеспечивает одинаковое восприятие конечными пользователями приложений на разных платформах.

Для вывода меню и досок используется *JFrame*.

Основные кнопки «Начать игру» и «Выход» реализованы с помощью *JButton*.

Для правильного распределения элементов на фрейме используется *JPanel*.

Для дополнительного меню игровых досок используется *JMenuBar* и *JMenuItem*.

Для вывода информации в процессе игры используется *JOptionPane*.

1. Описание результатов работы программы

Среда разработки IntelliJ IDEA предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна клиентской части, алгоритмом программы.

Ниже продемонстрирован результат тестирования функционала программы.

* 1. Запуск программы.

При запуске программы открывается главное меню (рис. 1).

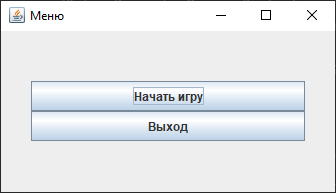


Рисунок – 1 Главное меню приложения

* 1. Запуск игры.

Начать игру можно нажав на кнопку «Начать игру»(рис. 2).

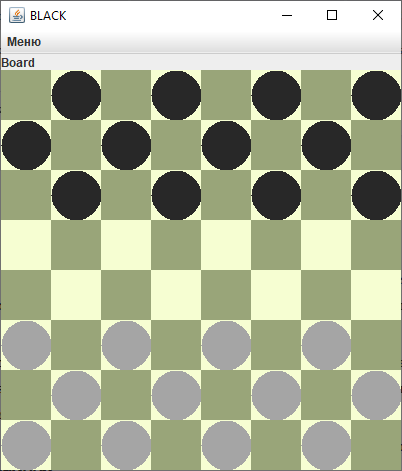
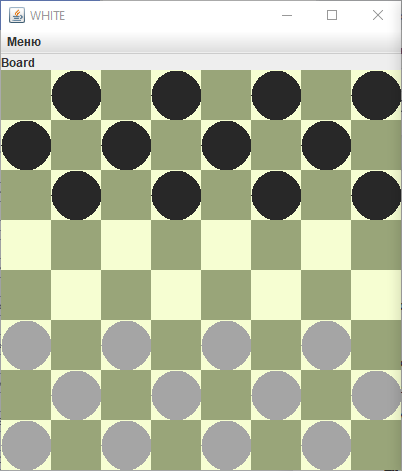


Рисунок – 2 Запуск игры

* 1. Перемещение шашки.

Для перемещения шашки, сначала нужно кликнуть на нужную шашку для её выделения, после чего ходить куда разрешают правила оригинальных шашек. Для выбора другой шашки, нужно произвести повторный клик по на данный момент выбранной шашке, после чего выбрать другую (рис. 3).

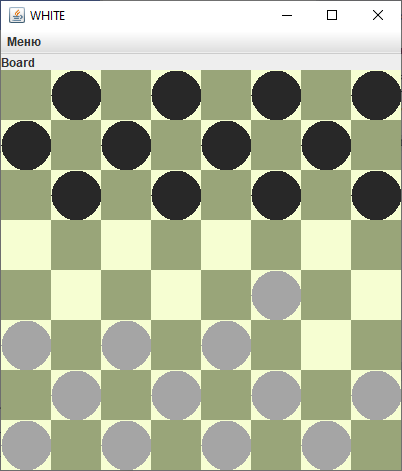
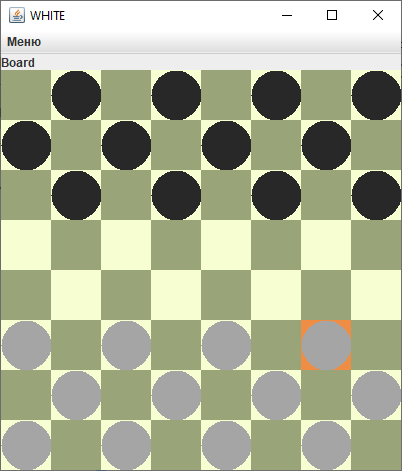


Рисунок – 3 Перемещение шашки

* 1. Конец игры

Игра заканчивается, когда у одного из пользователей не остаётся ходов или заканчиваются все шашки. В конце игры у каждого пользователя появляется окно с результатом игры (рис. 4).

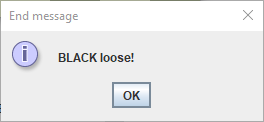
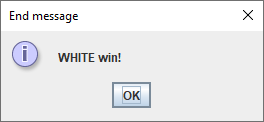
 

Рисунок – 4 Конец игры

Заключение

При выполнении данной курсовой работы были получены навыки разработки программ на языке высокого уровня Java. Были освоены навыки разработки клиент серверных приложений на сокетах и графического интерфейса с помощью библиотеки Swing. Изучены возможности среды разработки IntelliJ IDEA.

В рамках данной работы было разработано клиент серверное приложение «Шашки», обладающий базовым функционалом, необходимым для игры в шашки.

Список используемой литературы

1. Университет Информационных Технологий (ИНТУИТ), 2016.— 285с.
2. Портянкин, Java Swing: Эффектные пользовательские интерфейсы - Издание второе, 2011, 200 с.
3. Package javax.swing // docs.oracle.com : [сайт]. – 2023. – URL: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html (дата обращения: 01.04.2023).
4. Шилдт, Г. Swing. Руководство для начинающих. – М.: Вильямс, 2007. – 704 с.: ил.
5. Берд, Барри Java для чайников / Барри Берд. - М.: Диалектика / Вильямс, 2013. - 521 c.
6. Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубаков – СП: НИУ ИТМО, 2013. – 248 с.

Приложение А. Исходный код

Приложение А.1 – Исходный код Сервера

**Файл Server.java**

package net;  
  
public class Server extends Thread  
{  
  
 CommunicationModule com;  
 ServerGame serverGame;  
  
 public Server(int port)  
 {  
 serverGame = new ServerGame(port);  
 }  
  
 @Override  
 public void run()  
 {  
 serverGame.start();  
 }  
}

**Файл ServerCommunicationModule.java**

package net;  
  
public class ServerCommunicationModule  
{  
 public ServerConnection[] getConnections()  
 {  
 return connections;  
 }  
  
 ServerConnection[] connections;  
  
 public ServerCommunicationModule(ServerConnection[] connections)  
 {  
 this.connections = connections;  
 }  
  
 /\*\*  
 \* Say to everyone.  
 \* @param message  
 \*/  
 public void say(String message)  
 {  
 for(ServerConnection connection : getConnections())  
 {  
 connection.say(message);  
 }  
 }  
  
 /\*\*  
 \* Say to specific communication module.  
 \* @param connection  
 \* @param message  
 \*/  
 public void say(ServerConnection connection, String message)  
 {  
 connection.say(message);  
 }  
  
 public String readLine(ServerConnection connection)  
 {  
 return connection.readLine();  
 }  
}

**Файл ServerConnection.java**

package net;  
  
import java.net.Socket;  
  
public class ServerConnection  
 extends Thread  
{  
 public CommunicationModule getCommunicationModule()  
 {  
 return com;  
 }  
  
 CommunicationModule com;  
  
 public ServerConnection(Socket socket)  
 {  
 com = new CommunicationModule(socket);  
 com.init();  
 }  
  
 public boolean isConnected()  
 {  
 return getCommunicationModule().isConnected();  
 }  
  
 /\*\*  
 \* Facade method.  
 \* @param message  
 \*/  
 public void say(String message)  
 {  
 getCommunicationModule().say(message);  
 }  
  
 public String readLine()  
 {  
 return readLineBlocking();  
 }  
  
 String readLineBlocking()  
 {  
 return getCommunicationModule().readLineBlocking();  
 }  
  
 @Override  
 public void run()  
 {  
  
 }  
}

**Файл ServerGame.java**

package net;  
  
import game.\*;  
import gui.CheckerColor;  
  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.util.Collection;  
import java.util.Scanner;  
  
public class ServerGame extends Thread  
{  
 public Game getGame()  
 {  
 return game;  
 }  
  
 Game game;  
  
 public ServerCommunicationModule getCommunicationModule()  
 {  
 return communicationModule;  
 }  
  
 ServerCommunicationModule communicationModule;  
  
 public ServerConnection[] getConnections()  
 {  
 return connections;  
 }  
  
 ServerConnection[] connections;  
  
 public ServerSocket getServerSocket()  
 {  
 return serverSocket;  
 }  
  
 public void setServerSocket(ServerSocket serverSocket)  
 {  
 this.serverSocket = serverSocket;  
 }  
  
 ServerSocket serverSocket;  
  
 public ServerGame(int port)  
 {  
 try  
 {  
 serverSocket = new ServerSocket(port);  
 }  
 catch (IOException e)  
 {  
 System.err.println("Could not open server socket.");  
 e.printStackTrace(System.err);  
 }  
 }  
  
 @Override  
 public void run()  
 {  
 while (!isConnected())  
 waitForConnections();  
  
 setGame(new Game());  
  
 say("WELCOME");  
 say("PLEASE OCCUPY SEATS");  
  
 getGame().occupySeat(player1());  
 getGame().occupySeat(player2());  
  
 say(player1(), "YOU ARE PLAYING FOR " + getGame().seatName(player1()));  
 say(player2(), "YOU ARE PLAYING FOR " + getGame().seatName(player2()));  
  
 IMessageProcessor processor = new IMessageProcessor()  
 {  
 public Move getLastMove()  
 {  
 return lastMove;  
 }  
  
 public void setLastMove(Move lastMove)  
 {  
 this.lastMove = lastMove;  
 }  
  
 public Move lastMove = null;  
  
 public boolean isCaptureOnLastMove()  
 {  
 return getLastMove().isCapture();  
 }  
  
 @Override  
 public void process(String message)  
 {  
 }  
  
 @Override  
 public void process(String message, ServerConnection player)  
 {  
 if (message == null)  
 throw new IllegalArgumentException("Message cannot be null.");  
  
 if (message.startsWith("MOVE "))  
 {  
 Scanner scanner = new Scanner(message);  
 scanner.next();  
 String from = scanner.next();  
 String to = scanner.next();  
 Move tempMove = new Move(from, to);  
 System.out.println("Server received client move: " + tempMove);  
 Move move = getGame().capture(tempMove);  
 setLastMove(move);  
  
 say(opponentOf(player),"OPPONENT MOVE " + move.toString2());  
  
 getGame().getBoard().go(move);  
 }  
 else  
 {  
 throw new IllegalArgumentException(  
 String.format("Don't know how to process message %s", message));  
 }  
 }  
 };  
  
 ServerConnection winner = null;  
 ServerConnection currentPlayer = player1();  
 boolean hasMoved = false;  
 while (winner == null)  
 {  
 while (!hasMoved || (processor.isCaptureOnLastMove() && hasCaptures(currentPlayer)))  
 {  
 say(currentPlayer, "YOUR TURN");  
  
 processor.process(  
 readLine(currentPlayer), currentPlayer);  
  
 winner = getWinner(currentPlayer);  
 if (winner != null)  
 {  
 break;  
 }  
 hasMoved = true;  
 }  
 currentPlayer = opponentOf(currentPlayer);  
 hasMoved = false;  
  
 }  
  
 if (winner != null)  
 {  
 anounceWinner(winner);  
 }  
 else  
 {  
 }  
 }  
  
 private boolean hasCaptures(ServerConnection currentPlayer)  
 {  
 return getGame().hasCaptures  
 (  
 getGame().seatName(currentPlayer)  
 );  
 }  
  
 private boolean movingAllowed(ServerConnection currentPlayer)  
 {  
 CheckerColor color = getGame().seatName(currentPlayer);  
 return getGame().getAnyAllowedMove(color) != null;  
 }  
  
 public void anounceWinner(ServerConnection player)  
 {  
 say(opponentOf(player), "YOU LOOSE!");  
 say(player, "YOU WIN!");  
 }  
  
 /\*\*  
 \* Returns a winner or null if no winner yet.  
 \* @return  
 \*/  
 ServerConnection getWinner(ServerConnection lastMovedPlayer)  
 {  
 CheckerColor color = getGame().seatName(opponentOf(lastMovedPlayer));  
 Collection<Tile> withCheckers = getGame().getTilesWithCheckersOf(color);  
  
 if (withCheckers.isEmpty())  
 {  
 return lastMovedPlayer;  
 }  
  
 return null;  
 }  
  
 /\*\*  
 \* Check if player has won or lost.  
 \* @param player  
 \* @return  
 \*/  
 public boolean winningCondition(ServerConnection player)  
 {  
  
 return false;  
 }  
  
 ServerConnection player(int i)  
 {  
 return getConnections()[i];  
 }  
  
 public ServerConnection player1()  
 {  
 return player(0);  
 }  
  
 public ServerConnection player2()  
 {  
 return opponentOf(player1());  
 }  
  
 public ServerConnection opponentOf(ServerConnection sc)  
 {  
 return getConnections()[0].equals(sc) ? getConnections()[1] : getConnections()[0];  
 }  
  
 private boolean isConnected()  
 {  
 return getConnections() != null  
 && getConnections().length == 2  
 && player1().isConnected()  
 && player2().isConnected();  
 }  
  
 public void waitForConnections()  
 {  
 try  
 {  
 connections = new ServerConnection[]  
 {  
 new ServerConnection(getServerSocket().accept())   
 , new ServerConnection(getServerSocket().accept())   
  
 };  
 setCommunicationModule(new ServerCommunicationModule(connections));  
 }  
 catch (IOException e)  
 {  
 System.err.println("Could not accept connections.");  
 e.printStackTrace(System.err);  
 }  
 }  
 public void say(ServerConnection toPlayer, String aMessage)  
 {  
 getCommunicationModule().say(toPlayer, aMessage);  
 }  
  
 public void say(String message)  
 {  
 getCommunicationModule().say(message);  
 }  
  
 public String readLine(ServerConnection fromPlayer)  
 {  
 return getCommunicationModule().readLine(fromPlayer);  
 }  
  
 public void setCommunicationModule(ServerCommunicationModule communicationModule)  
 {  
 this.communicationModule = communicationModule;  
 }  
  
 public void setGame(Game game)  
 {  
 this.game = game;  
 }  
}

**Файл MainThread.java**

package net;  
  
import static net.GUI.showMenu;  
  
public class MainThread extends Thread  
{  
 int PORT;  
  
 public MainThread(int PORT)  
 {  
 this.PORT = PORT;  
 }  
  
 public void run()  
 {  
 showMenu(PORT);  
 }  
}

**Файл IMessageProcessor.java**

package net;  
  
import game.Move;  
  
public interface IMessageProcessor  
{  
 void process(String message);  
 void process(String message, ServerConnection player);  
 boolean isCaptureOnLastMove();  
 Move getLastMove();  
 void setLastMove(Move move);  
}

**Файл GUI.java**

package net;  
  
import gui.Grid;  
import gui.GridMouseListener;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class GUI  
{  
 public static int PORT;  
  
 public static void showGrid(String NameOfFrame)  
 {  
 JFrame frame = new JFrame(NameOfFrame);  
 JPanel panel = new JPanel();  
 JMenu menu = new JMenu("Меню");  
 JMenuBar menuBar = new JMenuBar();  
 JMenuItem menuItem = new JMenuItem("Выход");  
 Client client = new Client(PORT, NameOfFrame);  
 Grid grid = new Grid(400, 400, 8, 8, client);  
  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 menu.add(menuItem);  
 menuBar.add(menu);  
 frame.setJMenuBar(menuBar);  
 menuItem.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 System.exit(0);  
 }  
 });  
 BoxLayout bl = new BoxLayout(panel, BoxLayout.Y\_AXIS);  
 panel.setLayout(bl);  
  
 panel.add(new JLabel("Board"));  
 grid.addMouseListener(new GridMouseListener());  
 panel.add(grid);  
  
 frame.add(panel);  
  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void showMenu(int newPort)  
 {  
 PORT=newPort;  
  
 JButton StartBtn = new JButton("Начать игру");  
 JButton ExitBtn = new JButton("Выход");  
 JFrame Menu = new JFrame("Меню");  
 JPanel ButtonPanel = new JPanel();  
  
 Menu.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 Menu.setBounds(700,300,350,200);  
  
 ButtonPanel.setLayout(new GridLayout(2,0,0,0));  
  
 StartBtn.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 setVisibleBoards();  
 Menu.setVisible(false);  
 }  
 });  
  
 ExitBtn.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 System.exit(1);  
 }  
 });  
  
 ButtonPanel.add(StartBtn);  
 ButtonPanel.add(ExitBtn);  
 ButtonPanel.setBorder(BorderFactory.createEmptyBorder(50,30,50,30));  
  
 Menu.add(ButtonPanel);  
 Menu.setVisible(true);  
 }  
  
 public static void closeBoards(JFrame frame, Client client)  
 {  
 frame=null;  
 client =null;  
 }  
   
 public static void setVisibleBoards()  
 {  
 javax.swing.SwingUtilities.invokeLater(  
 new Runnable()  
 {  
 @Override  
 public void run()  
 {  
 showGrid("WHITE");  
 }  
 }  
 );  
  
 javax.swing.SwingUtilities.invokeLater(  
 new Runnable()  
 {  
 @Override  
 public void run()  
 {  
 showGrid("BLACK");  
 }  
 }  
 );  
 }  
}

Приложение А.2 – Исходный код Клиента

**Файл Board.java**

package game;  
  
import gui.Checker;  
import gui.CheckerColor;  
  
import java.util.Arrays;  
import java.util.Collection;  
import java.util.LinkedList;  
  
public class Board  
{  
 Tile[] tiles = null;  
 int rows;  
 int cols;  
  
 public int getRows()  
 {  
 return rows;  
 }  
 public int getCols()  
 {  
 return cols;  
 }  
  
 public static CheckerColor[] playableCheckers()  
 {  
 return new CheckerColor[]{CheckerColor.WHITE, CheckerColor.BLACK};  
 }  
  
 public void go(Move move)  
 {  
 go(move.getxFrom(), move.getyFrom(), move.getxTo(), move.getyTo());  
  
 if (move.isCapture())  
 {  
 Tile capture = move.getCapture();  
 capture.setChecker(null);  
 }  
 }  
  
 public void go(int rowFrom, int colFrom, int rowTo, int colTo)  
 {  
 Tile movingTile = getTile(rowFrom, colFrom);  
 Checker checker = movingTile.getChecker();  
  
 Tile movingToTile = getTile(rowTo, colTo);  
  
 movingTile.setChecker(null);  
 movingToTile.setChecker(checker);  
 }  
  
 void initBoard()  
 {  
 for (int i = 0; i < getTiles().length; i++)  
 {  
 getTiles()[i] = Tile.EMPTY();  
 }  
  
 for (int row = 0; row < 3; row++)  
 {  
 for (int col = 0; col < getCols(); col++)  
 {  
 if ( (col + (row % 2)) % 2 != 0)  
 {  
 put(Tile.BLACK(), row, col);  
 }  
 }  
 }  
  
 for (int row = getRows() - 3; row < getRows(); row++)  
 {  
 for (int col = 0; col < getCols(); col++)  
 {  
 if ((col + (row % 2)) % 2 != 0)  
 {  
 put(Tile.WHITE(), row, col);  
 }  
 }  
 }  
 }  
  
 public Tile[] getTiles()  
 {  
 return tiles;  
 }  
  
 public void setTiles(Tile[] tiles)  
 {  
 this.tiles = tiles;  
 }  
  
 public int getWidth()  
 {  
 return getCols();  
 }  
  
 int getHeight()  
 {  
 return getRows();  
 }  
  
 public Board(int rows, int cols)  
 {  
 this.rows = rows;  
 this.cols = cols;  
 this.tiles = new Tile[getRows()\*getCols()];  
 initBoard();  
 }  
 public Board()  
 {  
 this(8,8);  
 }  
 void put(Tile tile, int atRow, int atCol)  
 {  
 tiles[getRows()\*atRow+atCol] = tile;  
 }  
 Tile get(int row, int col)  
 {  
 return getTiles()[getRows()\*row + col];  
 }  
 public Tile tile(int row, int col)  
 {  
 return get(row, col);  
 }  
 Tile getTile(int x,int y)  
 {  
 return tile(x,y);  
 }  
 void setTile(int x, int y, Tile tile)  
 {  
 put(tile,x,y);  
 }  
 boolean isOccupied(int x, int y)  
 {  
 return !isEmpty(x,y);  
 }  
 boolean isEmpty(int x, int y)  
 {  
 return getTile(x,y).isEmpty();  
 }  
 public boolean validTile(int row, int col)  
 {  
 return row >= 0 && row < getWidth() && col >= 0 && col < getHeight();  
 }  
  
 /\*\*  
 \* Returns all left checkers of given color.  
 \* @param color  
 \* @return  
 \*/  
 Collection<Tile> getTilesWithCheckersOf(CheckerColor color)  
 {  
 Collection<Tile> tiles = new LinkedList<>();  
 for (Tile tile : getTiles())  
 {  
 if (tile.isOccupied() && tile.getCheckerColor() == color)  
 {  
 tiles.add(tile);  
 }  
 }  
 return tiles;  
 }  
  
 /\*\*  
 \* Gets row of tile.  
 \* @param tile  
 \* @return  
 \*/  
 public int tileRow(Tile tile)  
 {  
 return (Arrays.asList(getTiles()).indexOf(tile) - tileCol(tile)) / getCols();  
 }  
  
 public int tileCol(Tile tile)  
 {  
 return Arrays.asList(getTiles()).indexOf(tile) % getCols();  
 }  
}

**Файл Game.java**

package game;  
  
import gui.Checker;  
import gui.CheckerColor;  
  
import java.util.\*;  
  
public class Game  
{  
 Board board;  
  
 public Board getBoard()  
 {  
 return board;  
 }  
 public Game()  
 {  
 board = new Board();  
 }  
 public Checker winningChecker()  
 {  
 return null;  
 }  
  
 public boolean isValidMove(Move move)  
 {  
 return true;  
 }  
  
 public Object winnerIs()  
 {  
 return winningChecker() == null ? null : occuppantOf(winningChecker().getCheckerColor()) ;  
 }  
  
 public Map<Object, CheckerColor> getSeats()  
 {  
 return seats;  
 }  
  
 Map<Object, CheckerColor> seats = new HashMap<>();  
  
 public Collection<Move> getCapturingMovesFor(CheckerColor color)  
 {  
 return capturingMoves  
 (  
 movesFromTiles  
 (  
 getTilesWithCheckersOf(color)  
 )  
 );  
 }  
  
 public Collection<Move> capturingMoves(Collection<Move> moves)  
 {  
 Collection<Move> capturingMoves = new LinkedList<>(moves);  
 Iterator<Move> iterator = capturingMoves.iterator();  
  
 while (iterator.hasNext())  
 {  
 if (!iterator.next().isCapture())  
 {  
 iterator.remove();  
 }  
 }  
 return capturingMoves;  
 }  
  
 public Collection<Move> allMoves(Tile tile)  
 {  
 return allMoves(getBoard().tileRow(tile), getBoard().tileCol(tile));  
 }  
  
 public boolean hasCaptures(CheckerColor color)  
 {  
 return !getCapturingMovesFor(color).isEmpty();  
 }  
  
 /\*\*  
 \* Returns all moves for the position.  
 \* @param fromRow  
 \* @param fromCol  
 \* @return  
 \*/  
 public Collection<Move> allMoves(int fromRow, int fromCol)  
 {  
 Collection<Move> moves = new LinkedList<>();  
  
 Tile tile = getBoard().get(fromRow, fromCol);  
 Checker checker = tile.getChecker();  
 int direction = checker.isBlack() ? +1 : -1;  
  
 int x = fromRow;  
 int y = fromCol;  
  
 if (checker.isMan())  
 {  
 int coll = fromCol;  
 int colr = fromCol;  
  
 x = x + direction \* 1;   
 int xNext = x + direction \* 1;  
  
 coll = coll - 1;  
 colr = colr + 1;  
  
 int collnext = coll - 1;  
 int colrnext = colr + 1;  
  
 if (getBoard().validTile(x, coll))  
 {  
 Tile tilel = getBoard().get(x, coll);  
 if (tilel.isEmpty())  
 {  
 moves.add(  
 new Move(fromRow, fromCol, x, coll));  
 }  
 else if (getBoard().validTile(xNext, collnext) && getBoard().get(xNext, collnext).isEmpty())  
 {  
 Checker checkerl = tilel.getChecker();  
 if (checkerl.getCheckerColor() != checker.getCheckerColor())   
 {  
 moves.add(  
 new Move(fromRow, fromCol, xNext, collnext, tilel));  
 }  
 }  
 }  
 if (getBoard().validTile(x, colr))  
 {  
 Tile tiler = getBoard().get(x, colr);  
 if (tiler.isEmpty())  
 {  
 moves.add(new Move(fromRow, fromCol, x, colr));  
 }  
 else if (getBoard().validTile(xNext, colrnext) && getBoard().get(xNext,colrnext).isEmpty())  
 {  
 Checker checkerr = tiler.getChecker();  
 if (checkerr.getCheckerColor() != checker.getCheckerColor())   
 {  
 moves.add(  
 new Move(fromRow, fromCol, xNext, colrnext, tiler));  
 }  
 }  
 }  
 }  
 else if (checker.isKing())  
 {  
 x = fromRow - 1;  
 y = fromCol - 1;  
 moves.addAll(  
 collectKingMoves(fromRow, fromCol, -1, -1));  
 moves.addAll(  
 collectKingMoves(fromRow, fromCol, -1, +1));  
 moves.addAll(  
 collectKingMoves(fromRow, fromCol, +1, +1));  
 moves.addAll(  
 collectKingMoves(fromRow, fromCol, +1, -1));  
 }  
  
 return moves;  
 }  
  
 Collection<Move> collectKingMoves(int fromRow, int fromCol, int directionRow, int directionCol)  
 {  
 Checker checker = getBoard().get(fromRow, fromCol).getChecker();  
  
 Collection<Move> moves = new LinkedList<>();  
  
 int x = fromRow + directionRow\*1;  
 int y = fromCol + directionCol\*1;  
 int xNext = x + directionRow\*1;  
 int yNext = y + directionCol\*1;  
 Tile foundOccupied = null;  
  
 while (getBoard().validTile(x,y))  
 {  
 Tile tileC = getBoard().get(x,y);  
 Tile tileNext = getBoard().validTile(xNext, yNext) ? getBoard().get(xNext, yNext) : null;  
  
 if (tileC.isEmpty())  
 {  
 moves.add(new Move(fromRow, fromCol, x, y, foundOccupied));  
 x = x + directionRow\*1;  
 y = y + directionCol\*1;  
 }  
 else if (tileC.isOccupied())  
 {  
 if (foundOccupied != null)   
 {  
 break;  
 }  
 else  
 {  
 foundOccupied = tileC;  
 }  
  
 if (tileNext != null && tileNext.isEmpty()   
 && tileC.getCheckerColor() != checker.getCheckerColor())   
 {  
 x = xNext;  
 y = yNext;  
 }  
 else  
 {  
 break;  
 }  
  
 }  
 xNext = x + directionRow\*1;  
 yNext = y + directionCol\*1;  
 }  
  
 return moves;  
 }  
  
 public Collection<Tile> getTilesWithCheckersOf(CheckerColor color)  
 {  
 return getBoard().getTilesWithCheckersOf(color);  
 }  
  
 /\*\*  
 \* All moves from given tiles.  
 \* @return  
 \*/  
 public Collection<Move> movesFromTiles(Collection<Tile> tiles)  
 {  
 Collection<Move> moves = new LinkedList<>();  
 for(Tile tile : tiles)  
 {  
 moves.addAll(allMoves(tile));  
 }  
  
 return moves;  
 }  
  
 /\*\*  
 \* Find first allowed move  
 \* @return  
 \*/  
 public Move getAnyAllowedMove(CheckerColor color)  
 {  
 Collection<Tile> tiles = getBoard().getTilesWithCheckersOf(color);  
 Iterator iterator = tiles.iterator();  
 while (iterator.hasNext())  
 {  
 Tile tile = (Tile) iterator.next();  
 Collection<Move> allMoves = allMoves(tile);  
 if (!allMoves.isEmpty())  
 {  
 return allMoves.iterator().next();  
 }  
 }  
 return null;  
 }  
  
 public void occupySeat(Object object)  
 {  
 getFreeSeats();  
 seats.put(object, getFreeSeats()[0]);   
 }  
  
 /\*\*  
 \* Tells which seat is occupied by Player  
 \*/  
 public CheckerColor seatName(Object object)  
 {  
 return getSeats().get(object);  
 }  
  
 public Object occuppantOf(CheckerColor color)  
 {  
 Iterator<Map.Entry<Object, CheckerColor>> iterator = getSeats().entrySet().iterator();  
 while (iterator.hasNext())  
 {  
 Map.Entry<Object, CheckerColor> entry = iterator.next();  
 if (entry.getValue() == color)  
 {  
 return entry.getKey();  
 }  
 }  
 return null;  
 }  
  
 List<CheckerColor> occupiedSeats()  
 {  
 return new ArrayList<CheckerColor>(getSeats().values());  
 }  
  
 public CheckerColor[] getFreeSeats()  
 {  
 List<CheckerColor> colors = new ArrayList<>(Arrays.asList(Board.playableCheckers()));  
 colors.removeAll(occupiedSeats());  
 return colors.toArray(new CheckerColor[0]);  
 }  
  
 /\*\*  
 \* For the given move returns the move with capture tile if there is capture. If there is no capture original move is returned.  
 \* @param move  
 \* @return  
 \*/  
 public Move capture(Move move)  
 {  
 Move withCapture = move;  
 Collection<Move> captures = new LinkedList<>  
 (  
 capturesFrom  
 (  
 getBoard().tile  
 (  
 move.getxFrom(), move.getyFrom()  
 )  
 )  
 );  
  
 Iterator<Move> iterator = captures.iterator();  
 while(iterator.hasNext())  
 {  
 Move capture = iterator.next();  
 if (capture.equals(move))  
 {  
 withCapture = capture;  
 break;  
 }  
 }  
  
 return withCapture;  
 }  
  
 public boolean isMoveCapturing(Move move)  
 {  
 Collection<Move> moves = new LinkedList<>();  
 moves.add(move);  
 return !capturingMoves(moves).isEmpty();  
 }  
  
 /\*\*  
 \* True if captures from the given tile.  
 \* @param tile  
 \* @return  
 \*/  
 public Collection<Move> capturesFrom(Tile tile)  
 {  
 Collection<Move> moves = new LinkedList<>(allMoves(tile));  
 Iterator<Move> iterator = moves.iterator();  
 while(iterator.hasNext())  
 {  
 if (!iterator.next().isCapture())  
 {  
 iterator.remove();  
 }  
 }  
 return moves;  
 }  
  
 public boolean hasCaptures(Tile tile)  
 {  
 return !capturesFrom(tile).isEmpty();  
 }  
}

**Файл Move.java**

package game;  
  
public class Move  
{  
 public boolean isCapture()  
 {  
 return getCapture() != null;  
 }  
  
 public Tile getCapture()  
 {  
 return capture;  
 }  
  
 Tile capture = null;  
  
 public void setCapture(Tile tile)  
 {  
 this.capture = tile;  
 }  
  
 public int getxFrom()  
 {  
 return xFrom;  
 }  
  
 public void setxFrom(int xFrom)  
 {  
 this.xFrom = xFrom;  
 }  
  
 public int getyFrom()  
 {  
 return yFrom;  
 }  
  
 public void setyFrom(int yFrom)  
 {  
 this.yFrom = yFrom;  
 }  
  
 public int getxTo()  
 {  
 return xTo;  
 }  
  
 public void setxTo(int xTo)  
 {  
 this.xTo = xTo;  
 }  
  
 public int getyTo()  
 {  
 return yTo;  
 }  
  
 public void setyTo(int yTo)  
 {  
 this.yTo = yTo;  
 }  
  
 int xFrom;  
 int yFrom;  
 int xTo;  
 int yTo;  
  
 @Override  
 /\*\*  
 \* Only coordinates compared.  
 \*/  
 public boolean equals(Object object)  
 {  
 if (object == null)  
 return false;  
  
 Move move = (Move) object;  
  
 return getxFrom() == move.getxFrom()  
 && getyFrom() == move.getyFrom()  
 && getxTo() == move.getxTo()  
 && getyTo() == move.getyTo();  
 }  
  
 public Move(int fromRow, int fromCol, int toRow, int toCol, Tile captureTile)  
 {  
 this(fromRow,fromCol,toRow,toCol);  
 setCapture(captureTile);  
 }  
  
 public Move(int fromRow, int fromCol, int toRow, int toCol)  
 {  
 this(String.format("%d,%d", fromRow, fromCol), String.format("%d,%d", toRow, toCol));  
 }  
  
 /\*\*  
 \* Expecting strings in format "x,y" (no quotes).  
 \*  
 \* @param from  
 \* @param to  
 \*/  
 public Move(String from, String to)  
 {  
 xFrom = Integer.valueOf(from.split(",")[0]);  
 yFrom = Integer.valueOf(from.split(",")[1]);  
 xTo = Integer.valueOf(to.split(",")[0]);  
 yTo = Integer.valueOf(to.split(",")[1]);  
 }  
  
 public String toString2()  
 {  
 return String.format("%d,%d %d,%d", xFrom, yFrom, xTo, yTo);  
 }  
  
 public String toString()  
 {  
 return String.format("A move from [%d,%d] to [%d,%d].", xFrom, yFrom, xTo, yTo);  
 }  
}

**Файл Tile.java**

package game;  
  
import gui.Checker;  
import gui.CheckerColor;  
  
public class Tile  
{  
 public static Tile EMPTY()  
 {  
 return new Tile();  
 }  
  
 public static Tile WHITE()  
 {  
 return new Tile(Checker.Lighter());  
 }  
  
 public static Tile BLACK()  
 {  
 return new Tile(Checker.Darker());  
 }  
  
 public Checker getChecker()  
 {  
 return checker;  
 }  
  
 public void setChecker(Checker checker)  
 {  
 this.checker = checker;  
 }  
  
 Checker checker;  
  
 public Tile()  
 {  
  
 }  
  
 public Tile(Checker checker)  
 {  
 this.checker = checker;  
 }  
  
 public boolean isOccupied()  
 {  
 return !isEmpty();  
 }  
  
 public boolean isEmpty()  
 {  
 return checker==null;  
 }  
  
 public CheckerColor getCheckerColor()  
 {  
 if (isEmpty())  
 {  
 throw new IllegalStateException("Checker not found!");  
 }  
 else  
 {  
 return getChecker().getCheckerColor();  
 }  
 }  
  
 public boolean colorEquals(CheckerColor checkerColor)  
 {  
 return isEmpty() ? false : getChecker().getCheckerColor() == checkerColor;  
 }  
  
 public boolean hasColor(CheckerColor color)  
 {  
 return isEmpty() ? false : getChecker().getCheckerColor() == color;  
 }  
}

**Файл Checker.java**

package gui;  
  
public class Checker  
{  
 public boolean isMan()  
 {  
 return !isKing();  
 }  
 public boolean isKing()  
 {  
 return isKing;  
 }  
  
 public void setKing(boolean king)  
 {  
 isKing = king;  
 }  
  
 boolean isKing;  
 public CheckerColor getCheckerColor()  
 {  
 return checkerColor;  
 }  
  
 public void setCheckerColor(CheckerColor checkerColor)  
 {  
 this.checkerColor = checkerColor;  
 }  
  
 CheckerColor checkerColor;  
  
 public Checker(CheckerColor color)  
 {  
 this.checkerColor = color;  
 }  
  
 public static Checker Darker()  
 {  
 return new Checker(CheckerColor.BLACK);  
 }  
  
 public static Checker Lighter()  
 {  
 return new Checker(CheckerColor.WHITE);  
 }  
  
 public boolean isWhite()  
 {  
 return getCheckerColor() == CheckerColor.WHITE;  
 }  
  
 public boolean isBlack()  
 {  
 return getCheckerColor() == CheckerColor.BLACK;  
 }  
  
 public String toString()  
 {  
 return getCheckerColor().toString();  
 }  
}

**Файл CheckerCircle.java**

package gui;  
  
public class CheckerCircle  
 extends Circle  
{  
 int row;  
 int col;  
  
 public Checker getChecker()  
 {  
 return checker;  
 }  
 Checker checker;  
  
 public CheckerCircle(int row, int col, Rectangle rectangle, Checker checker)  
 {  
 super(rectangle);  
 this.row = row;  
 this.col = col;  
 this.checker = checker;  
 }  
}

**Файл CheckerColor.java**

package gui;  
  
public enum CheckerColor  
{  
 WHITE, BLACK;  
}

package gui;  
  
public class Circle  
 extends java.awt.geom.Ellipse2D.Double  
{  
 Rectangle rectangle;  
 public Circle(Rectangle rectangle)  
 {  
 super(rectangle.getX(), rectangle.getY(), rectangle.getWidth(), rectangle.getHeight());  
 this.rectangle = rectangle;  
 }  
}

**Файл Circle.java**

package gui;  
  
import java.awt.\*;  
  
public class Colors  
{  
 public static final Color RECTANGLE\_COLOR = new Color(153,165,121);  
  
 public static final Color RECTANGLE\_COLOR\_ALTERNATE = new Color(246, 254, 210);  
  
 public static final Color RECTANGLE\_ACTIVE = new Color(238, 141, 67);  
  
 public static final Color CHECKER\_LIGHTER = new Color(165,165,165);  
 public static final Color CHECKER\_DARKER = new Color(40,40,40);  
  
 public static final CheckerColor CHECKER\_COLOR\_LIGHTER = CheckerColor.WHITE;  
 public static final CheckerColor CHECKER\_COLOR\_DARKER = CheckerColor.BLACK;  
  
 public static Color checkerColor(CheckerColor cc)  
 {  
 if (cc == CheckerColor.WHITE)  
 {  
 return CHECKER\_LIGHTER;  
 } else  
 {  
 return CHECKER\_DARKER;  
 }  
 }  
  
 boolean a = false;  
  
 public Color alternateColor(Color color2, Color color1)  
 {  
 Color color = null;  
 if (a)  
 {  
 color = color1;  
 } else  
 {  
 color = color2;  
 }  
 a = !a;  
 return color;  
 }  
}

**Файл Grid.java**

package gui;  
  
import game.Board;  
import game.Game;  
import game.Move;  
import game.Tile;  
import net.Client;  
  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.ListIterator;  
  
public class Grid extends Canvas implements ActionListener  
{  
 Client client;  
  
 int rows;  
 int cols;  
 Dimension dimension;  
 List<Rectangle> rectangles;  
 List<CheckerCircle> circles;  
  
 boolean myTurn;  
  
 CheckerCircle activeCircle;  
 CheckerColor myColor;  
  
 public void setMyColor(CheckerColor color)  
 {  
 this.myColor = color;  
 }  
 public CheckerColor getMyColor()  
 {  
 return myColor;  
 }  
  
 public List<CheckerCircle> getCircles()  
 {  
 return circles;  
 }  
  
 public int getRectangleRow(Rectangle rectangle)  
 {  
 return (getRectangles().indexOf(rectangle) - getRectangleCol(rectangle)) / getRows();  
 }  
  
 public int getRectangleCol(Rectangle rectangle)  
 {  
 return getRectangles().indexOf(rectangle) % getCols();  
 }  
  
 public Rectangle getRectangleXY(int x, int y)  
 {  
 Rectangle rectangle = null;  
  
 for (Rectangle r : getRectangles())  
 {  
 if (r.contains(x,y))  
 {  
 rectangle = r;  
 break;  
 }  
 }  
 return rectangle;  
 }  
  
 Rectangle getActiveRectangle()  
 {  
 Rectangle active = null;  
 for (Rectangle r : getRectangles())  
 {  
 if (r.isActive())  
 {  
 active = r;  
 break;  
 }  
 }  
 return active;  
 }  
  
 List<Rectangle> getActiveRectangles()  
 {  
 List<Rectangle> temporary = new LinkedList<>(getRectangles());  
 ListIterator<Rectangle> iterator = temporary.listIterator();  
 while (iterator.hasNext())  
 {  
 if (!iterator.next().isActive())  
 iterator.remove();  
 }  
 return temporary;  
 }  
  
 void unsetActiveRectangles()  
 {  
 for (Rectangle rectangle : getRectangles())  
 {  
 if (rectangle.isActive())  
 {  
 rectangle.setActive(false);  
 }  
 }  
 }  
  
 public void setActiveRectangle(Rectangle r)  
 {  
 int row = getRectangleRow(r);  
 int col = getRectangleCol(r);  
 Tile tile = getBoard().tile(row,col);  
  
 if(getActiveRectangle() == null)  
 {  
 if (tile.isEmpty())  
 {  
 return;  
 }  
  
 if (!tile.colorEquals(getMyColor()))  
 {  
 return;  
 }  
  
 if (!getGame().allMoves(tile).isEmpty())  
 {  
 if (getGame().hasCaptures(getMyColor()))  
 {  
 if (getGame().hasCaptures(tile))  
 {  
 r.setActive(true);  
 }  
 }  
 else  
 {  
 r.setActive(true);  
 }  
 }  
 else  
 {  
 return;  
 }  
 }  
 else   
 {  
 if (tile.isEmpty())  
 {  
 Rectangle r1 = getActiveRectangle();  
  
 int toRow = getRectangleRow(r);  
 int toCol = getRectangleCol(r);  
 int fromRow = getRectangleRow(r1);  
 int fromCol = getRectangleCol(r1);  
  
 List<Move> allMoves = new LinkedList<>(getGame().allMoves(fromRow, fromCol));  
 Move tempMove = new Move(fromRow, fromCol, toRow, toCol);  
  
 if (!allMoves.contains(tempMove))  
 {  
 return;  
 }  
  
 Move move = allMoves.get(allMoves.indexOf(tempMove));  
  
 r.setActive(true);  
  
 getBoard().go(move);  
 getClient().makeMove(move);  
  
 setMyTurn(false);  
  
 unsetActiveRectangles();  
 }  
 }  
 }  
  
 public void resetBoard(Client client)  
 {  
 this.client=client;  
 }  
  
 public Grid(int width, int height, int rows, int cols, Client client)  
 {  
 this.dimension = new Dimension(width, height);  
 this.rows = rows;  
 this.cols = cols;  
  
 this.client = client;  
 getClient().start();  
  
 client.setActionListener(this);  
  
 rectangles = new LinkedList<>();  
 for (int row = 0; row < getRows(); row++)  
 {  
 for (int col = 0; col < getCols(); col++)  
 {  
 Rectangle newRectangle = makeRectangle(row, col);  
 getRectangles().add(newRectangle);  
 }  
 }  
 }  
  
 List<Rectangle> getRectangles()  
 {  
 return rectangles;  
 }  
  
 Dimension getDimension()  
 {  
 return dimension;  
 }  
  
 public int getWidth()  
 {  
 return (int)getDimension().getWidth();  
 }  
  
 public int getHeight()  
 {  
 return (int)getDimension().getHeight();  
 }  
  
 int getCols()  
 {  
 return cols;  
 }  
  
 int getRows()  
 {  
 return rows;  
 }  
  
 CheckerCircle makeCheckerCircle(Tile tile, int row, int col)  
 {  
 Checker checker = tile.getChecker();  
 return new CheckerCircle(row, col, getRectangle(row, col), checker);  
 }  
  
 Rectangle makeRectangle(int row, int col)  
 {  
 Rectangle r = new Rectangle(  
 (int) getWidth() / getCols() ,  
 (int) getHeight() / getRows());  
 r.setLocation((int)(col\*r.getWidth()), (int)(row\*r.getHeight()));  
 return r;  
 }  
  
 public Rectangle getRectangle(int row, int col)  
 {  
 return getRectangles().get(getRows()\*row + col);  
 }  
  
 public Game getGame()  
 {  
 return getClient().getGame();  
 }  
 public Board getBoard()  
 {  
 return getClient().getBoard();  
 }  
  
 @Override  
 public void paint(Graphics g)  
 {  
 Graphics2D g2 = (Graphics2D) g;  
 Colors colors = new Colors();  
 ListIterator iterator = getRectangles().listIterator();  
 for (int row = 0; row < getRows(); row++)  
 {  
 for (int col = 0; col < getCols(); col++)  
 {  
 Rectangle rectangle = (Rectangle)iterator.next();  
 g2.setPaint(colors.alternateColor(Colors.RECTANGLE\_COLOR, Colors.RECTANGLE\_COLOR\_ALTERNATE));  
 if (rectangle.isActive())  
 {  
 g2.setPaint(Colors.RECTANGLE\_ACTIVE);  
 }  
 g2.fill(rectangle);  
 }  
 g2.setPaint(colors.alternateColor(Colors.RECTANGLE\_COLOR, Colors.RECTANGLE\_COLOR\_ALTERNATE));  
 }  
  
 for (int row = 0; row < getRows(); row++)  
 {  
 for (int col = 0; col < getCols(); col++)  
 {  
 Tile tile = getBoard().tile(row, col);  
 if (tile.isOccupied())  
 {  
 CheckerCircle checkerCircle = makeCheckerCircle(tile, row, col);  
 g2.setPaint(Colors.checkerColor(checkerCircle.getChecker().getCheckerColor()));  
 g2.fill(checkerCircle);  
 }  
 }  
 g2.setPaint(colors.alternateColor(Colors.RECTANGLE\_COLOR, Colors.RECTANGLE\_COLOR\_ALTERNATE));  
 }  
 }  
  
 @Override  
 public java.awt.Dimension getPreferredSize()  
 {  
 return new java.awt.Dimension(getWidth(), getHeight());  
 }  
  
 /\*\*  
 \* This is where messages are received. Parse them from here.  
 \* @param e  
 \*/  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 String message = e.getActionCommand();  
  
 if (message.equals("YOU ARE PLAYING FOR BLACK"))  
 {  
 setMyColor(CheckerColor.BLACK);  
 }  
 else if (message.equals("YOU ARE PLAYING FOR WHITE"))  
 {  
 setMyColor(CheckerColor.WHITE);  
 }  
 else if (message.equals("YOUR TURN"))  
 {  
 setMyTurn(true);  
 }  
 else if (message.startsWith("OPPONENT MOVE "))  
 {  
 }  
 repaint();  
 }  
  
 public void rectangleClicked(Rectangle r)  
 {  
 if (!isMyTurn())  
 {  
 return;  
 }  
  
 if (getActiveRectangle() != null)  
 {  
 if (getActiveRectangle().equals(r))  
 {  
 r.setActive(false);  
 }  
 else  
 {  
 if (getActiveRectangles().size() == 2)  
 {  
 unsetActiveRectangles();  
 }  
  
 setActiveRectangle(r);  
 }  
 } else   
 {  
 setActiveRectangle(r);  
 }  
 }  
  
 public Client getClient()  
 {  
 return client;  
 }  
  
 public boolean isMyTurn()  
 {  
 return myTurn;  
 }  
  
 public void setMyTurn(boolean myTurn)  
 {  
 this.myTurn = myTurn;  
 }  
}

**Файл GridMouseListener.java**

package gui;  
  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
  
public class GridMouseListener implements MouseListener  
{  
 @Override  
 public void mouseClicked(MouseEvent e)  
 {  
 Grid grid = (Grid) e.getSource();  
  
 Rectangle r = grid.getRectangleXY(e.getX(), e.getY());  
 grid.rectangleClicked(r);  
  
 grid.repaint();  
 }  
  
 @Override  
 public void mousePressed(MouseEvent e)  
 {  
  
 }  
  
 @Override  
 public void mouseReleased(MouseEvent e)  
 {  
  
 }  
  
 @Override  
 public void mouseEntered(MouseEvent e)  
 {  
  
 }  
  
 @Override  
 public void mouseExited(MouseEvent e)  
 {  
  
 }  
}

**Файл Rectangle.java**

package gui;  
  
public class Rectangle  
 extends java.awt.Rectangle  
{  
 public boolean isActive()  
 {  
 return isActive;  
 }  
  
 public void setActive(boolean active)  
 {  
 isActive = active;  
 }  
  
 boolean isActive;  
  
 public Rectangle(int x, int y)  
 {  
 super(x, y);  
 }  
}

**Файл Client.java**

package net;  
  
import game.Board;  
import game.Game;  
import game.Move;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.IOException;  
import java.net.Socket;  
import java.util.Scanner;  
  
public class Client extends Thread  
{  
 ActionListener actionListener;  
 String colorOfCircle;  
 public ActionListener getActionListener()  
 {  
 return actionListener;  
 }  
 public void setActionListener(ActionListener actionListener)  
 {  
 this.actionListener = actionListener;  
 }  
  
 Game game;  
 public void setGame(Game game)  
 {  
 this.game = game;  
 }  
 public Game getGame()  
 {  
 return game;  
 }  
 public Board getBoard ()  
 {  
 return getGame().getBoard();  
 }  
  
 /\*\*  
 \* With communication module the client speaks to the socket.  
 \*/  
 CommunicationModule communicationModule;  
 public void setCommunicationModule(CommunicationModule communicationModule)  
 {  
 this.communicationModule = communicationModule;  
 }  
  
 public CommunicationModule getCommunicationModule()  
 {  
 return communicationModule;  
 }  
  
 public void resetGame()  
 {  
 setGame(new Game());  
 }  
  
 public Client(int port, String color)  
 {  
 this.colorOfCircle = color;  
 try  
 {  
 createCommunicationModuleOn(new Socket("localhost", port));  
 communicationModule.init();  
  
 setGame(new Game());  
 }  
 catch (IOException e)  
 {  
 System.err.println("Can not create communication module on port " + port);  
 e.printStackTrace();  
 }  
 }  
  
 public void createCommunicationModuleOn(Socket socket)  
 {  
 setCommunicationModule(new CommunicationModule(socket));  
 }  
  
 public void makeMove(Move move)  
 {  
 getCommunicationModule().say("MOVE " + move.toString2());  
 }  
  
 public void capture(int row, int col)  
 {  
 getCommunicationModule().say(  
 String.format("CAPTURE %d%,%d", row, col ));  
 }  
  
 public void informActionListener(String message)  
 {  
 getActionListener().actionPerformed(new ActionEvent(this, 1, message));  
 }  
  
 @Override  
 public void run()  
 {  
 String message;  
 do  
 {  
 message = getCommunicationModule().getLineBlocking();  
 IMessageProcessor messageProcessor = new IMessageProcessor()  
 {  
 public Move getLastMove()  
 {  
 return lastMove;  
 }  
  
 public void setLastMove(Move lastMove)  
 {  
 this.lastMove = lastMove;  
 }  
  
 public Move lastMove = null;  
  
 public boolean isCaptureOnLastMove()  
 {  
 return getLastMove().isCapture();  
 }  
  
 public void process(String message)  
 {  
 if (null == message)  
 throw new IllegalArgumentException("Message cannot be null");  
  
 if (message.equals("YOUR TURN"))  
 {  
 informActionListener("YOUR TURN");  
 }  
 else if (message.startsWith("OPPONENT MOVE "))  
 {  
 informActionListener(message);  
  
 Scanner scanner = new Scanner(message);  
 scanner.next();  
 scanner.next();  
 String from = scanner.next();  
 String to = scanner.next();  
 Move tempMove = new Move(from, to);  
 Move move = getGame().capture(tempMove);  
 setLastMove(move);  
 echo("Client received opponent move: " + move);  
  
 getBoard().go(move);  
 }  
 else if (message.equals("YOU ARE PLAYING FOR BLACK"))  
 {  
 informActionListener("YOU ARE PLAYING FOR BLACK");  
 }  
 else if (message.equals("YOU ARE PLAYING FOR WHITE"))  
 {  
 informActionListener("YOU ARE PLAYING FOR WHITE");  
 }  
 else if (message.startsWith("PLEASE OCCUPY SEATS"))  
 {  
 }  
 else if (message.startsWith("YOU WIN!"))  
 {  
 JOptionPane.showMessageDialog(null,  
 colorOfCircle + " win!",  
 "End message",  
 JOptionPane.INFORMATION\_MESSAGE);  
 }  
 else if (message.startsWith("YOU LOOSE!"))  
 {  
 JOptionPane.showMessageDialog(null,  
 colorOfCircle + " loose!",  
 "End message",  
 JOptionPane.INFORMATION\_MESSAGE);  
 }  
 else if (message.equals("WELCOME"))  
 {

}  
 else  
 {  
 throw new IllegalArgumentException(  
 String.format("Don't know how to process message \"%s\"", message));  
 }  
 }  
  
 @Override  
 public void process(String message, ServerConnection player) { }  
  
 public void echo(String message)  
 {  
 System.out.println(message);  
 }  
 };  
 messageProcessor.process(message);  
 }  
 while (!"END".equals(message));  
 }  
}

**Файл CommunicationModule.java**

package net;  
  
import java.io.PrintWriter;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.IOException;  
import java.net.Socket;  
  
public class CommunicationModule  
{  
 private Socket socket;  
 private PrintWriter out ;  
 private BufferedReader in ;  
  
 public CommunicationModule(Socket socket)  
 {  
 this.socket = socket;  
 }  
  
 public void setSocket(Socket socket)  
 {  
 this.socket = socket;  
 }  
  
 public Socket getSocket()  
 {  
 return socket;  
 }  
  
 public void setOut(PrintWriter out)  
 {  
 this.out = out;  
 }  
  
 public PrintWriter getOut()  
 {  
 return out;  
 }  
  
 public void setIn(BufferedReader in)  
 {  
 this.in = in;  
 }  
  
 public BufferedReader getIn()  
 {  
 return in;  
 }  
  
 public void init()  
 {  
 try  
 {  
 setOut(new PrintWriter(  
 getSocket().getOutputStream(), true));  
 setIn(new BufferedReader(  
 new InputStreamReader(  
 getSocket().getInputStream())));  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 public String readLine()  
 {  
 String ret = null;  
 try  
 {  
 ret = in.readLine();  
 }  
 catch (IOException e)  
 {  
 System.err.println("Could not read from socket."+e);  
 }  
 return ret;  
 }  
  
 public String readLineBlocking()  
 {  
 return getLineBlocking();  
 }  
  
 public void say(String msg)  
 {  
 while (null == getOut())  
 {  
 System.err.println("Out not initialized yet!");  
 }  
 getOut().println(msg);  
 }  
  
 public boolean isConnected()  
 {  
 return getSocket().isConnected();  
 }  
  
 String getLineBlocking()  
 {  
 String str = null;  
 while (null == getIn())  
 {  
 System.out.println("In not initialized yet!");  
 }  
 while (null == (str = readLine()))  
 {  
 }  
 return str;  
 }  
}

Приложение В. UML-диаграммы приложения

Приложение В.1 – UML-диаграмма вариантов использования



Рисунок 5 – UML-Диаграмма вариантов использования

Приложение В.2 – UML-диаграмма классов клиента

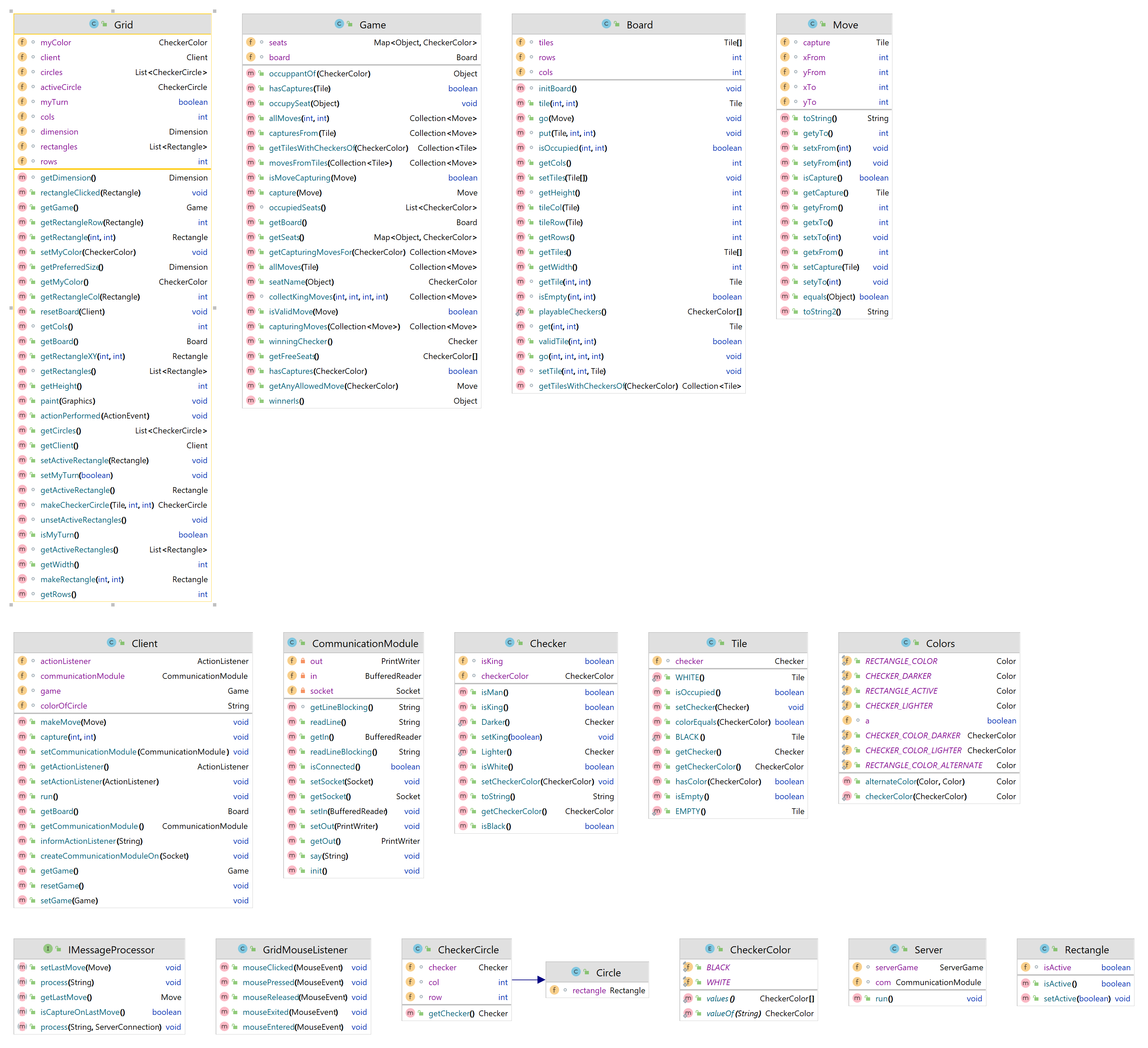


Рисунок 6 – UML-диаграмма классов клиента

Приложение В.3 – UML-диаграмма классов сервера

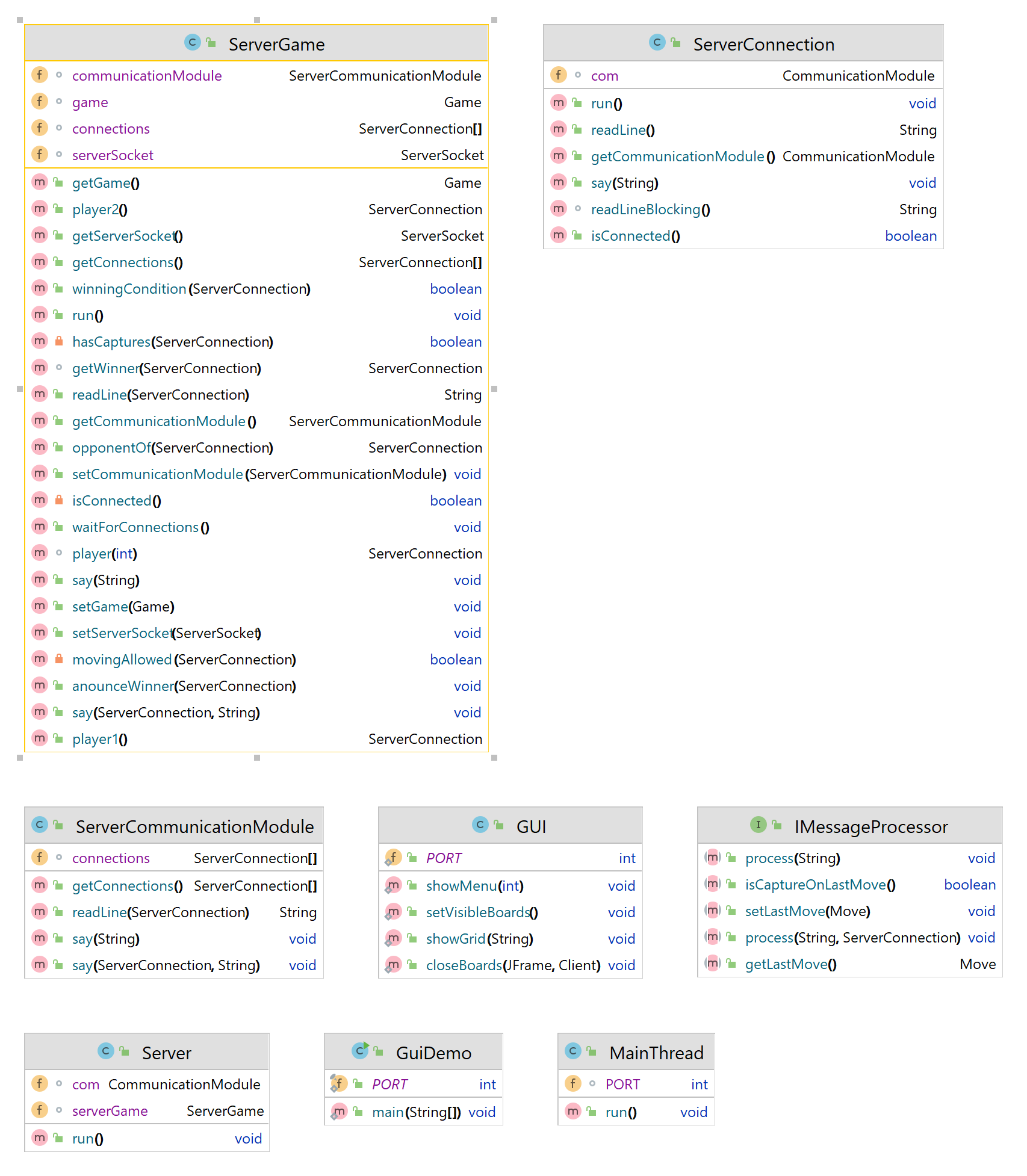


Рисунок 7 – UML-диаграмма классов сервера

Приложение В.4 – UML-диаграмма деятельности

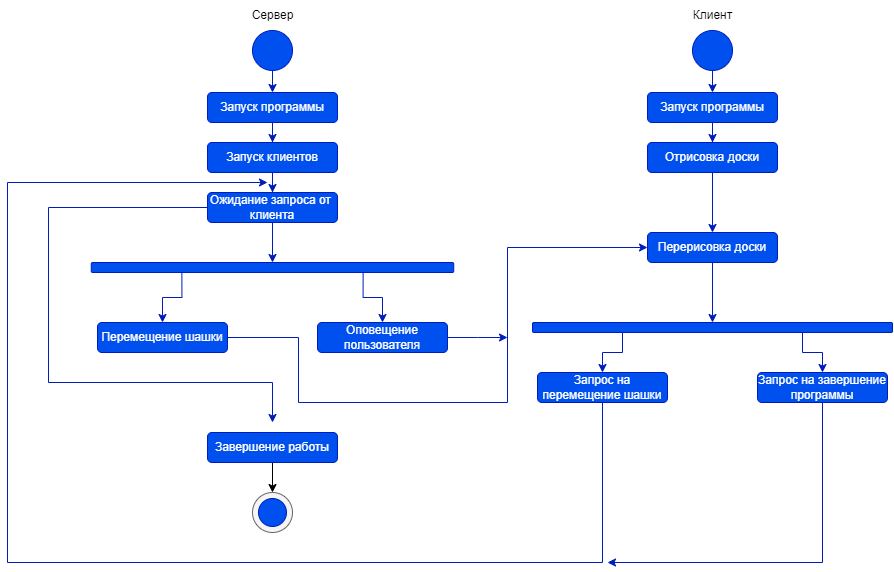


Рисунок 8 – UML-диаграмма деятельности

Приложение В.5 – UML-диаграмма развёртывания



Рисунок 9 – UML-диаграмма развёртывания

Приложение В.6 – UML-диаграмма последовательности

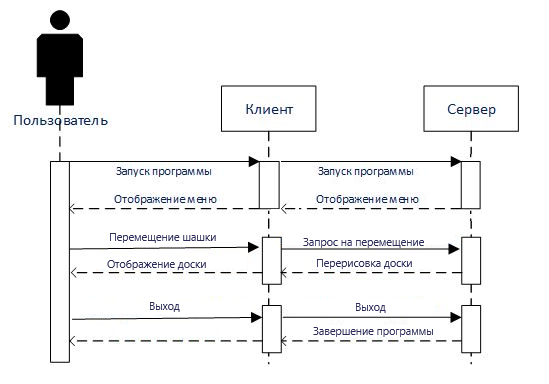


Рисунок 10 – UML-диаграмма последовательности