Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

По дисциплине «Программирование на языке Java»

На тему: «Разработка многомодульного приложения на языке Java»

Выполнил:

студент группы 20ВВП2

Духнов О.О.

Приняли:

к.т.н., доцент, Юрова О.В.

к.т.н., доцент, Карамышева Н.С.

Пенза 2023

**Оглавление**

[Введение 5](#_Toc138806030)

[1. Постановка задачи 6](#_Toc138806031)

[2. Выбор решения 7](#_Toc138806032)

[3. Описание программы 8](#_Toc138806033)

[3.1 Серверная часть 8](#_Toc138806034)

[3.2 Клиентская часть 10](#_Toc138806035)

[4. Описание способа организации пользовательского интерфейса 13](#_Toc138806036)

[5. Описание результатов работы программы 14](#_Toc138806037)

[Заключение 23](#_Toc138806038)

[Список используемой литературы 24](#_Toc138806039)

[Приложение А. Исходный код 25](#_Toc138806040)

[Приложение А.1 – Исходный код Сервера 25](#_Toc138806041)

[Приложение А.2 – Исходный код Клиента 43](#_Toc138806042)

[Приложение В. UML-диаграммы приложения 66](#_Toc138806043)

[Приложение В.1 – UML-диаграмма вариантов использования 66](#_Toc138806044)

[Приложение В.2 – UML-диаграмма классов клиента 67](#_Toc138806045)

[Приложение В.3 – UML-диаграмма классов сервера 68](#_Toc138806046)

[Приложение В.4 – UML-диаграмма деятельности 69](#_Toc138806047)

[Приложение В.5 – UML-диаграмма развёртывания 70](#_Toc138806048)

[Приложение В.6 – UML-диаграмма последовательности 71](#_Toc138806049)

# **Введение**

У человека во все времена всегда была подробность в развлечении, общении и обмене какими-либо данными. Постепенно люди придумывали способы, которые могли удовлетворить данные потребности. Сейчас, в двадцать первом веке, благодаря современным технологиям, появилась возможность обмена различными видами информации, такими как видео, фото, музыка, звуки, и так далее по всему миру.

Для реализации приложений, которые бы соответствовали всем вышесказанным требованиям, есть очень популярный язык программирования – Java. Java является мощным и гибким языком программирования, который широко используется для разработки различных приложений, включая веб-приложения, мобильные приложения, игры и многое другое.

В данном курсовом проекте необходимо реализовать карточную игру «Дурак». Согласно Федеральному закону от 29 декабря 2006 года № 244-ФЗ «О государственном регулировании деятельности по организации и проведению азартных игр и о внесении изменений в некоторые законодательные акты Российской Федерации», азартная игра — это основанное на риске соглашение о выигрыше, заключенное двумя или несколькими участниками такого соглашения между собой либо соорганизатором азартной игры. Учитывая изложенное, данная игра не является азартной, так как приложение не требует взносов и выигрыш не несет никакой ценности.

# **1. Постановка задачи**

Разработать систему многомодульных программ, клиент-серверной архитектуры, позволяющую играть в карточную игру «Подкидной Дурак» пользователями в рамках локальной вычислительной сети.

Функции сервера: прием и передача информации от пользователя, сбор статистики по пользователям, работа с колодой, отправка карт пользователю.

Функции клиента: графический интерфейс пользователю, сетевое взаимодействие с сервером.

Операционная система: Windows;

Среда разработки: NetBeans;

Язык программирования Java.

Используемые технологии:

* Java Collections Framework
* Механизм обработки исключительных ситуаций
* Java Stream API
* Java Multithreading
* Сетевое взаимодействие
* Java Swing

# **2. Выбор решения**

Для реализации курсового проекта «Подкидной Дурак» используется язык программирования Java, библиотека Swing, Java коллекции ArrayList и LinkedList, Java Stream Api. В качестве среды разработки выбрана IDE NetBeans 8.2.

Основной задачей курсового проекта является реализация клиент-сервера. Для взаимодействия клиента и сервера были выбраны библиотеки java.net.Socket и java.net.ServerSocket.

На стороне сервера имеется класс, который обрабатывает запросы пользователей по принципу «Запрос-Ответ».

Со стороны клиента исходят запросы на получение данных у сервера или передачу данных серверу. Клиент выбирает нужное действие, а сервер это действие обрабатывает.

Пользовательский интерфейс реализован с помощью Java Swing.

Java Swing – это набор библиотек и классов, предоставляющих разработчикам возможность создания графических интерфейсов пользователя (GUI) на языке Java.

Были использованы следующие компоненты:

* Кнопки (Button), поля ввода (TextField).
* Панели (Panel), которые могут быть использованы для создания композитных компонентов, таких как таблицы и списки.
* Метки (Label) для отображения текста, значков и изображений.
* Диалоги и окна (Dialogs) для отображения сообщений пользователю, запрашивающих информацию или подтверждения.

# **3. Описание программы**

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

После чего на устройствах запускается клиентская часть, которая по нажатию определенных кнопок, посылает определённый запросы к серверу.

## **3.1 Серверная часть**

**GameServer.java** – главный файл серверной части, в котором находится точка входа в программу. Происходит ожидание клиентов для создания с ними связи.

После входа в программу создаётся главный класс GameSession, в котором хранится вся информация об игровой сессии. После этого создаётся ServerSocket, который осуществляет взаимодействие с клиентом.

При получении запроса от клиента, сервер получает индекс действия, которое нужно выполнить, и обрабатывает это действие в классе ServerLogic.

**ServerLogic.java –** класс обработки запросов.

* Selector() – реагирует соответсвующим образом на каждый запрос пользователя. Этот метод принимает от пользователя идентификатор действия, по которому он определяет, что необходимо выполнить, имя пользователя и имя сессии, к которой пользователь подключен.
* SearchSession() – поиск нужной сессии, в которой пользователь находится. Данный метод возвращает индекс сессии.
* SearchPlayer() – метод ищет нужного пользователя по имени сессии и возвращает индекс игрока.
* DefinePlayer() – данный метод определяет, какой пользователь будет отбиваться.
* StrokeTransition() – метод передаёт ход следующему игроку. Возвращает индекс игрока.
* CheckBeat() – проверяет, сколько игроков нажали на кнопку «бито». Возвращает true, если все нажали и false, если не все.

**GameSession.java** – класс, в котором хранится вся информация об одной игровой сессии.

* GetStatusGame() – получает информацию о том, сколько игроков подключилось к сессии.
* MoveDetection() – определяет в первый раз, какой игрок будет ходить по тому, какую игровую «руку» пользователь получил.
* ConnectionToSession() – метод подключения к игровой сессии.

**Deck.java** – класс, в котором хранится информация о колоде.

* Deck() – конструктор создаёт колоду, состоящую из 36 карт.
* AddCard() – добовляет карту в коллекцию колоды.
* SortCard() – метод сортирует всю колоду, каждый раз делает это случайно, чтобы в каждой новой игре было разное расположение карт.
* pollCard() – метод возвращает и удаляет карту из колоды.
* getTrump() – метод определяет, какой «козырь» будет в данной игровой сессии.
* BoostTrump() – метод увеличивает силу козырных карт.

**Card.java** – класс, в котором хранится информация об карте, а именно: сила, путь к иконке, масть и достоинство.

**Player.java ­**– класс, в котором хранится рука игрока в коллекции ArrayList, имя игрока и возможность хода.

**Action.java** – класс, в котором расписаны основные действия, которые нужны для взаимодействия клиента с сервером.

## **3.2 Клиентская часть**

**GameStart.java** – класс, в котором осуществляется создание игровой сессии и её поиск. В нём инициализируются все его элементы Swing.

* WaitingConnection() – метод, который ожидает подключения к игровой сессии. Если все пользователи собрались и есть возможность подключиться, то вызывается метод main класса GamePlay.java

**GamePlay.java** – данный класс содержит основное окно взаимодействия пользователя с игровым интерфейсом. Все элементы интерфейса реализованы с помощью Java Swing.

* PutConnect() – данный метод в отдельном потоке отправляет на сервер информацию о том, какой картой игрок походил.
* FindConnect() – данный метод в отдельном потоке отправляет сообщение серверу с целью получения актуальной информации об игровом столе. Сообщение посылаются каждый пол секунды.
* StartGame() – метод, в котором получаются изначальные данные об игровой сессии.
* InitStartum() – метод начальной инициализации интерфейса.
* StartInitImage() – метод инициализирует деку, игровую «руку» и руку противника.
* repaintGameHand() – метод перерисовки игровой «руки».

**Action.java** – класc, в котором расписаны основные действия, которые нужны для взаимодействия клиента с сервером.

**ServerConnect.java –** класс, в котором происходит сетевое взаимодействие клиента с сервером.

* PutConnectServer() – метод отправляет на сервер карту, которой игрок походил.
* FindConnectServer() – метод получения состояния игрового стола.
* ButConnectServer() – метод отправляет карту на сервер, которой игрок отбился.
* ITakeSessionConnect() – метод отправляет информацию на сервер о том, что игрок берёт карты со стола.
* BeatCardSessionConnection() – метод отправляет информацию о том, что была нажата кнопка «бито».
* UpdateDataSessionConnection() – метод посылает запрос серверу на обновление информации каждого игрока после завершения раунда.

**GameTable.java** – данный класс хранит информацию об игровом столе.

* addCardTable() – метод, который добавляет карту на стол.
* BeatCardTable() – метод, который отбивает карту на столе.

**Player.java** – данный класс содержит информацию об игроке и его соперниках, а именно: имена всех игроков, игровую «руку» главного игрока, определение возможности хода, количество карт в колоде, имя сессии, к которой необходимо подключаться.

* SortPlayers() – сортировка игроков в нужной последовательности для конкретного игрока.
* getIconPath() – распределение иконок между игроками.
* SearchPlayer() – поиск нужного игрока.
* poolCard() – метод получает и удаляет выбранную карту, которую передают по индексу.

**Rules.java** – в данном классе хранятся правила игры «Подкидной Дурак».

* PutCard() – метод, который принимает карту, которую игрок хочет положить на игровой стол. Если данную карту можно добавить, то метод возвращает true, иначе false.
* BeatCard() – данный метод принимает карту, которой игрок хочет отбиться. Если карту можно отбить, то метод возвращает true, иначе false.

# **4. Описание способа организации пользовательского интерфейса**

В качестве среды разработки была выбрана программа NetBeans. 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 вставки изображений и текста был использован jLabel, который позволяет это реализовать.

При взаимодействии с картами был выбран компонент jButton. При нажатии на данный компонент активировалось специальное действие, которое обрабатывало нужное действие.

Для корректного отображения всех элементов был выбран компонент jPanel.

# **5. Описание результатов работы программы**

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

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.

Ниже продемонстрирован результат тестирования функционала программы.

При запуске программа требует ввести имя игрока (рис.1).

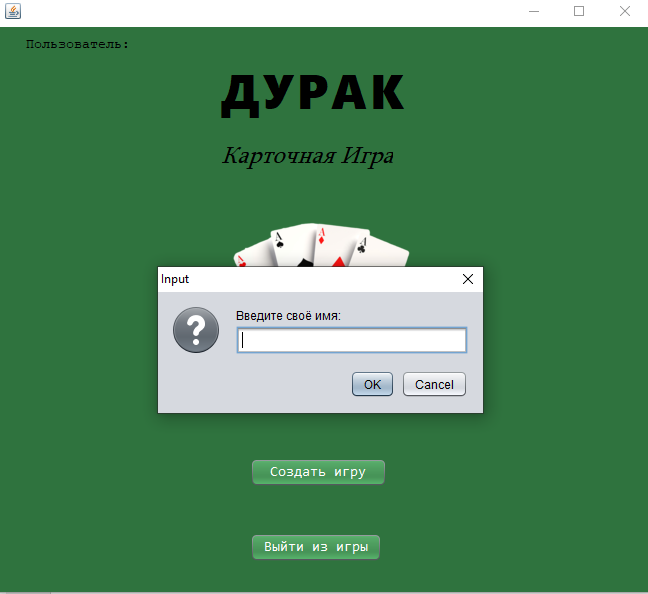


Рисунок 1 - запуск клиента

После того, как пользователь введёт своё имя, его встречает вот такой интерфейс (рис.2).

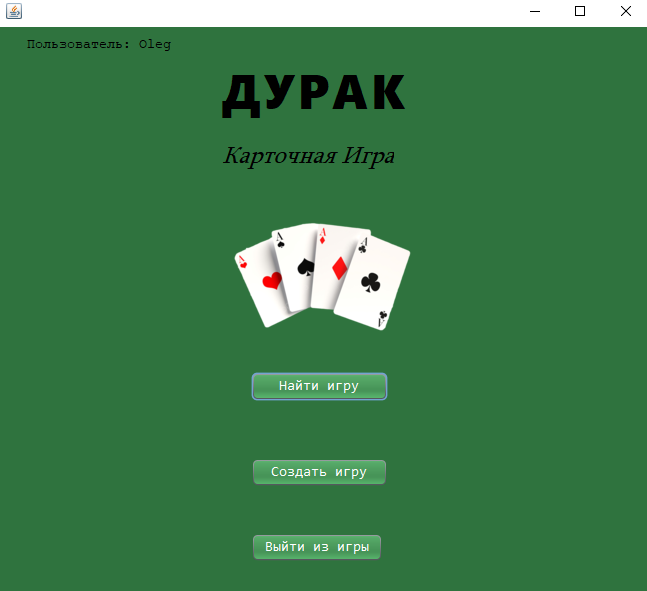


Рисунок 2 – запуск клиента

На выбор у пользователя есть 3 действия:

1. Кнопка «Найти игру». При нажатии на неё программа попросит ввести имя игровой сессии, к которой пользователь желает присоединиться (рис.3).

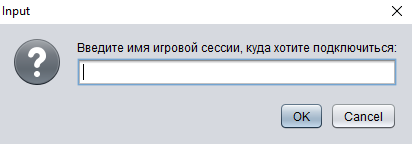


Рисунок 3 - поиск игровой сессии

1. Кнопка «Создать игру». При нажатии на эту кнопку программа запросит у пользователя следующие данные: имя игровой сессии, которую пользователь хочет создать и количество игроков в сессии (рис.4 и рис.5).

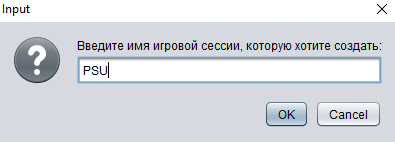


Рисунок 4 - ввод имени игровой сессии

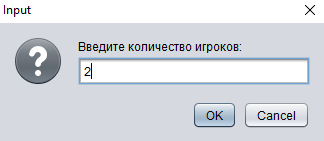


Рисунок 5 - ввод количества игроков

1. Кнопка «Выход из игры». По нажатию на данную кнопку происходит выход из игры.

После того подключения к игровой сессии пользователя встречает интерфейс, который представлен на рисунке 6 и 7, а также очерёдность хода, если игрок ходит первым.

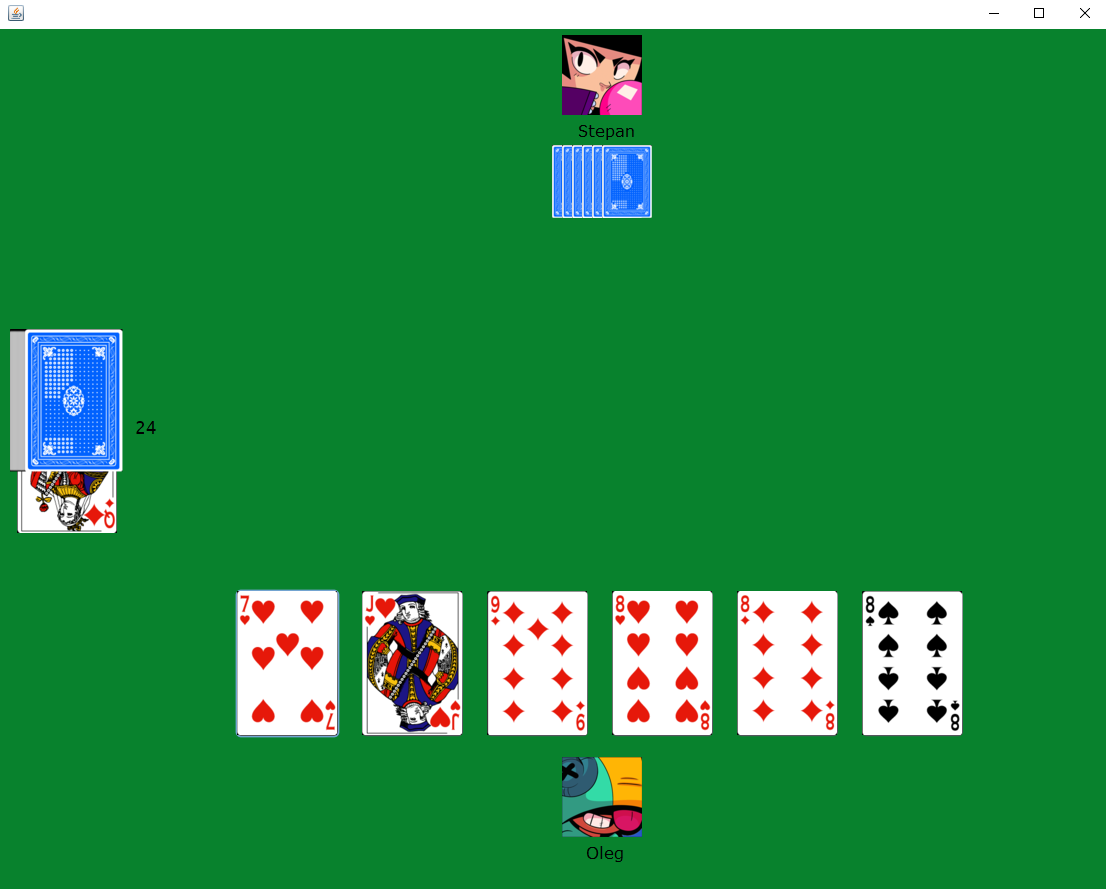


Рисунок 6 - игровое меню

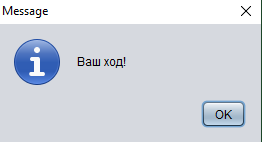


Рисунок 7 - очерёдность хода

У пользователя есть возможность выбирать карту для различных действий.

1. Положить карты на стол (рис.8)

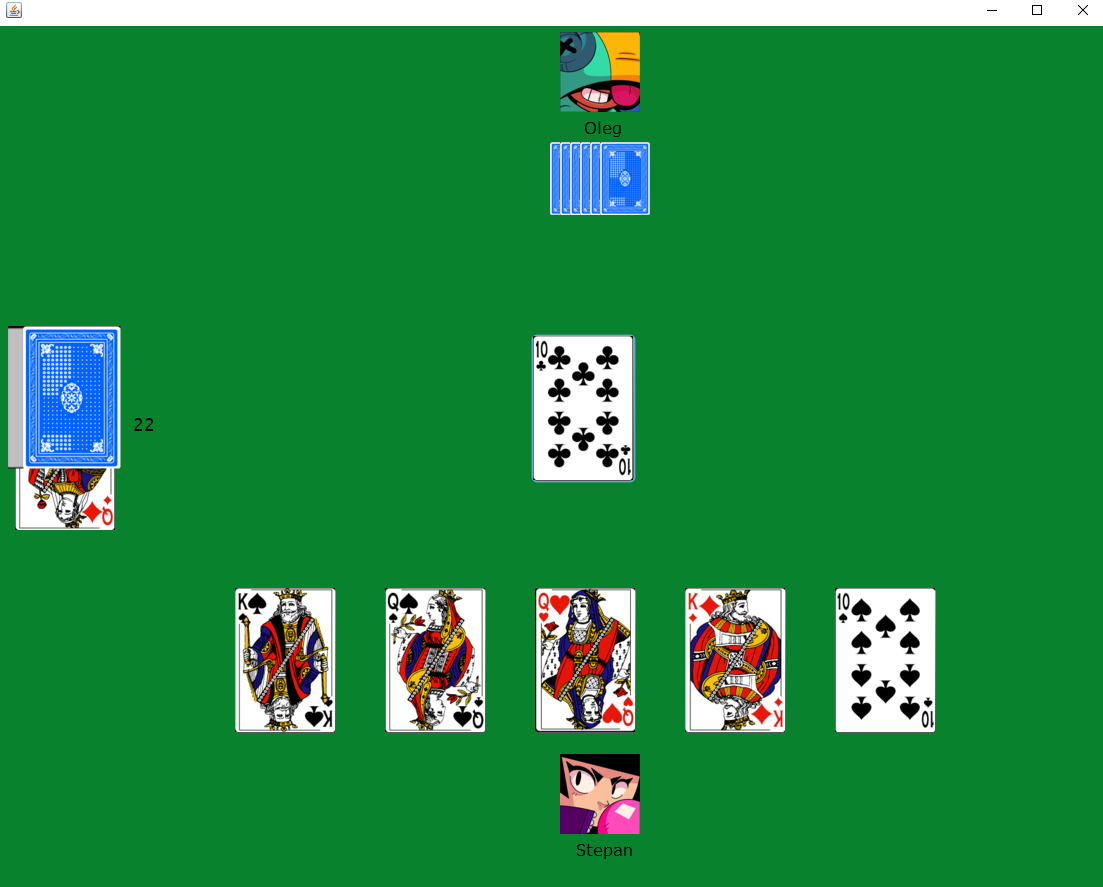


Рисунок 8 - положить карту на стол

1. Отбить карту (рис.9).

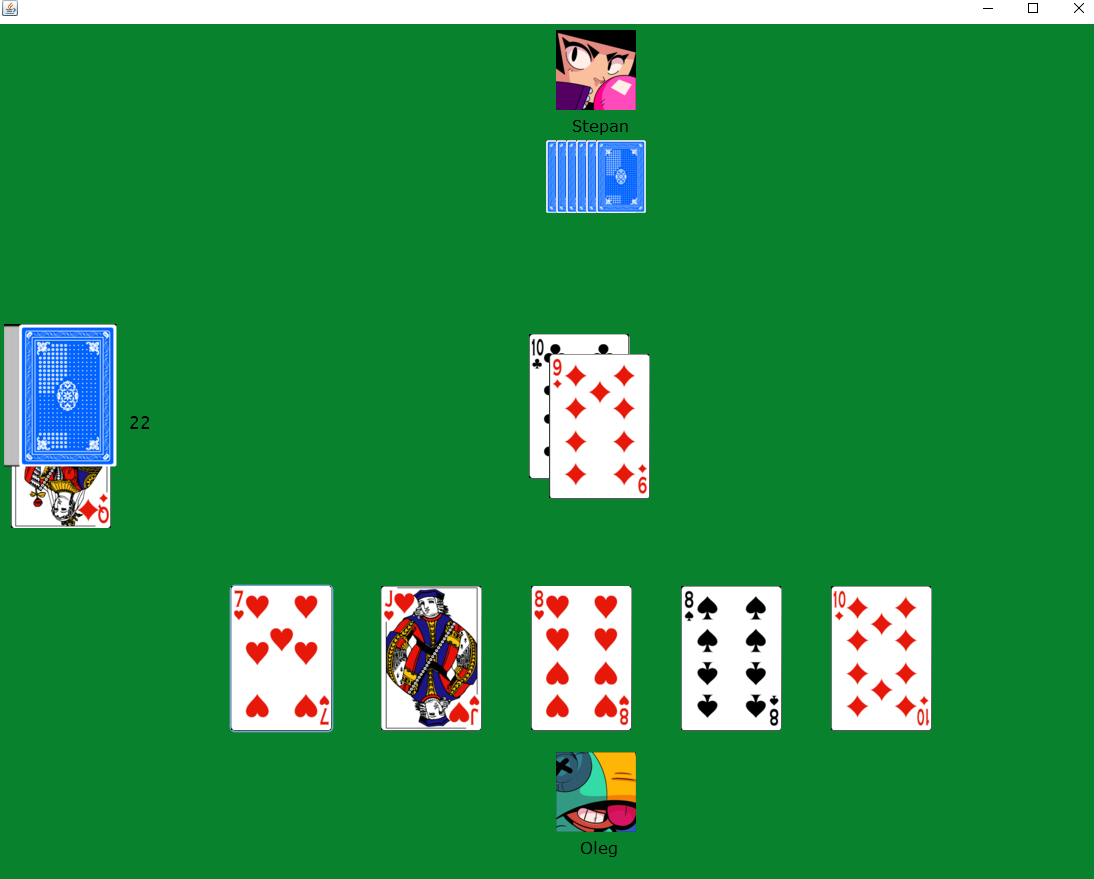


Рисунок 9 - отбивание карты

1. Подкинуть карты (рис. 10).

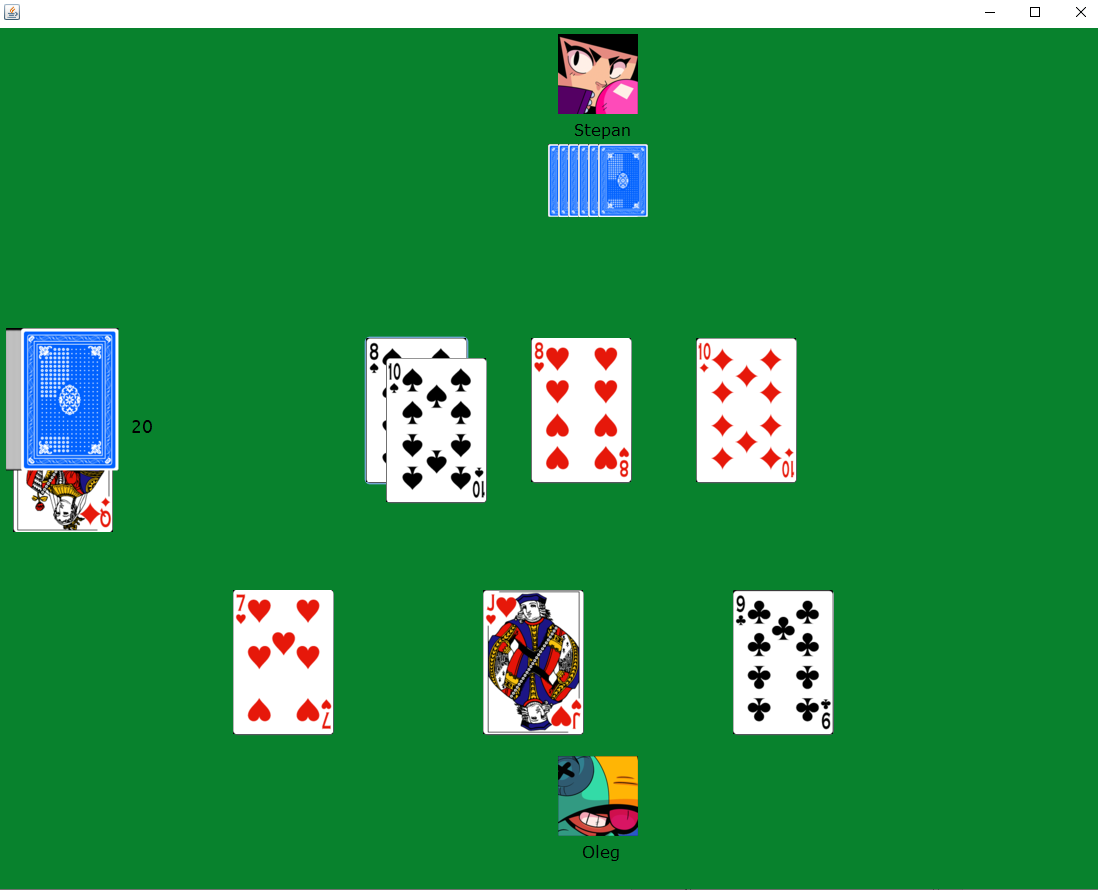


Рисунок 10 - подкидывание карты другому игроку

Также есть у игрока возможность взять карты со стола, для этого нужно нажать кнопку «Беру» (рис.11).

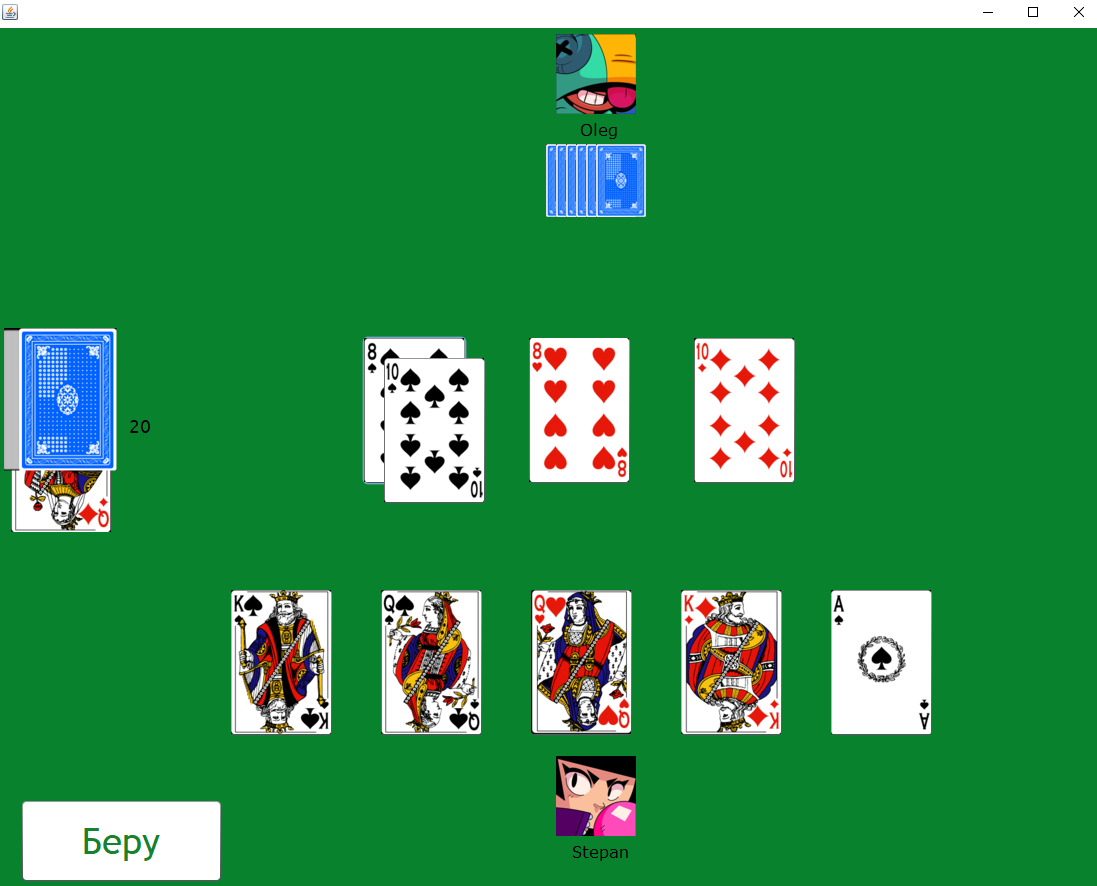


Рисунок 11 - нажатие кнопки "Беру"

Результатом нажатия будет следующее (рис. 12).

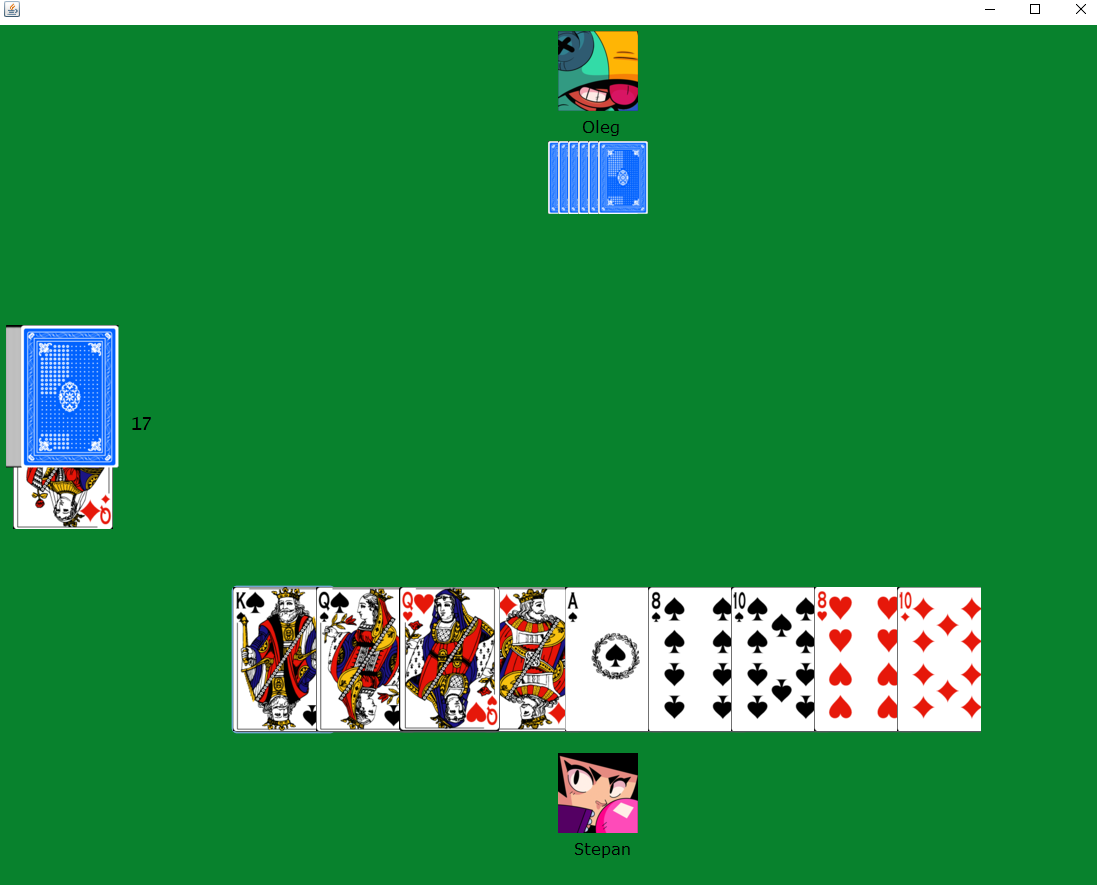


Рисунок 12 - результат взятия

Если игроку нечем подбросить новые карты, то есть возможность использования кнопки «Бито», которая информирует сервер о том, что игрок желает перейти к следующему раунду и ему нечего добавить на стол

(рис.13)

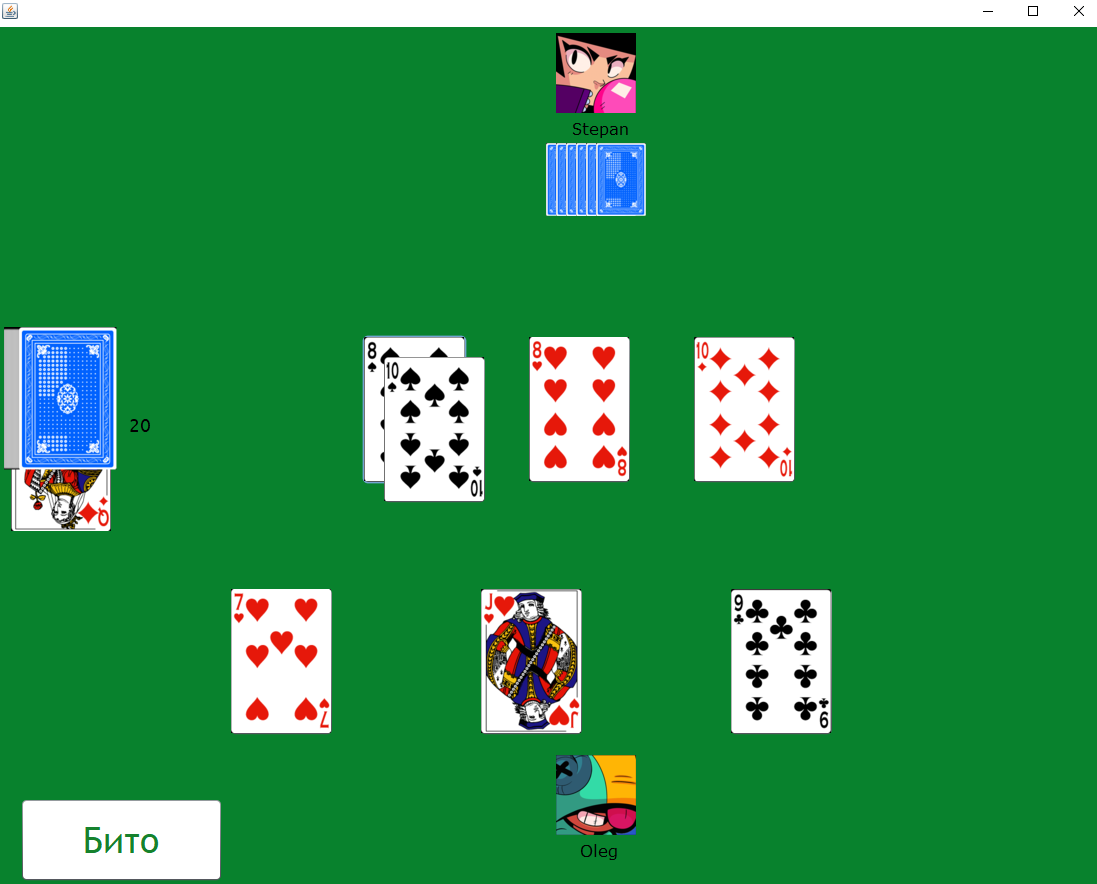


Рисунок 13 - кнопка "Бито"

В конце игры определяется победитель (рис. 14).



Рисунок 14 - определение победителя

**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 NetBeans.

В рамках данной работы было разработано приложение «Подкидной Дупак», обладающий базовым функционалом, необходимым любому подобному приложению в наше время.

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

**GameServer.java:**

package game.server;

import java.io.IOException;

import java.net.Socket;

import java.net.ServerSocket;

import java.util.ArrayList;

public class GameServer

{

public static void main(String[] args) throws IOException

{

ArrayList<GameSession> dataGameSession = new ArrayList();

try (ServerSocket serverSocket = new ServerSocket(8080))

{

System.out.println("Сервер полноценно заработал.");

while(true)

{

System.out.println(serverSocket);

try(Socket socket = serverSocket.accept())

{

System.out.println("Пользователь зашёл.");

dataGameSession = ServerLogic.selector(socket, dataGameSession);

}

System.out.println("Пользователь вышел.");

}

}

catch(IOException exception)

{

throw new RuntimeException(exception);

}

}

}

**GameSession.java:**

package game.server;

import game.player.Card;

import game.player.Deck;

import game.player.GameTable;

import game.player.Player;

import java.util.ArrayList;

public class GameSession

{

int indexTakePlayer = -1; // Индекс игрока, который берёт карты

int indexBeatPlayer = -1; // Индекс игрока, который отбивает карты

int NumberConnections; // Количество подключенных игроков к сессии

int NumberPlayers; // Общее количество игроков, которое должно быть в игре

String portName; // Имя порта

String nameSessions; // Имя сессии

boolean statusSession; // Создана игра или нет

ArrayList<Player> dataPlayers = new ArrayList(); // Данные об игроках

GameTable gameTable = new GameTable(); // Игровой стол

Deck deck = new Deck(); // Игровая колода

GameSession(int NumberPlayers, String nameSessions, String userName)

{

this.NumberConnections = 1;

this.NumberPlayers = NumberPlayers;

this.nameSessions = nameSessions;

Player player = new Player();

player.setUserName(userName);

this.dataPlayers.add(player);

}

// Получение информации о том, сколько игроков подключилось

public boolean GetStatusGame()

{

return NumberPlayers == NumberConnections;

}

// Определение, какой игрок будет ходить

public void MoveDetection()

{

class tempInt

{

int power;

}

ArrayList<tempInt> powerCard = new ArrayList();

for (int i = 0; i < dataPlayers.size(); i++)

{

tempInt temp = new tempInt();

temp.power = 30;

powerCard.add(temp);

for (int j = 0; j < 6; j++)

{

Card card = new Card();

card = dataPlayers.get(i).getHand(j);

if (card.power >= 9 && card.power < powerCard.get(i).power)

{

powerCard.get(i).power = card.power;

}

}

}

int tempPower = 30;

int indexPlayer = 0;

for (int i = 0; i < powerCard.size(); i++)

{

if (powerCard.get(i).power < tempPower)

{

tempPower = powerCard.get(i).power;

indexPlayer = i;

}

}

dataPlayers.get(indexPlayer).turnOrder = 1;

}

// Подключение к игровой сессии

public void ConnectionToSession(String userName)

{

NumberConnections++;

Player player = new Player();

player.setUserName(userName);

this.dataPlayers.add(player);

}

public void InitDeck()

{

deck.SortCard();

deck.getTrump(NumberPlayers);

deck.BoostTrump();

}

public void setHand(int indexPlayer, Card card)

{

dataPlayers.get(indexPlayer).setHand(card);

}

// Установить модификацию возможности хода

public void setTurnOrder(int modifier, int indexPlayer)

{

dataPlayers.get(indexPlayer).turnOrder = modifier;

}

// Узнать модификацию возможности хода

public int getTurnOrder(int indexPlayer)

{

return dataPlayers.get(indexPlayer).turnOrder;

}

}

**ServerLogic.java:**

package game.server;

import game.action.Action;

import game.player.Card;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.Socket;

import java.util.ArrayList;

public class ServerLogic

{

public static ArrayList<GameSession> selector(Socket socket, ArrayList<GameSession> dataGameSession) throws IOException

{

try

(

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Принимает пакеты (читает)

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); // Отправляет пакеты (записывает)

)

{

int numberAction = reader.read();

switch(numberAction)

{

case Action.CREATE\_GAME:

{

String userName = reader.readLine();

String nameSession = reader.readLine();

int numberPlayers = reader.read();

if (SearchSession(dataGameSession, nameSession) == -1)

{

GameSession newGameSession = new GameSession(numberPlayers, nameSession, userName);

dataGameSession.add(newGameSession);

writer.write("Сессия успешно создалась!");

}

else

{

writer.write("Ошибка! Сессия с таким названием уже есть!");

}

break;

}

case Action.CONNECT\_SESSIONS:

{

String userName = reader.readLine();

String nameSession = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession); // Номер сессии в коллекции, к которй нужно подключиться

if (nubmerSession != -1 && dataGameSession.get(nubmerSession).GetStatusGame() == false)

{

dataGameSession.get(nubmerSession).ConnectionToSession(userName);

writer.write("Подключение к сессии...");

if (dataGameSession.get(nubmerSession).NumberPlayers == dataGameSession.get(nubmerSession).NumberConnections)

{

dataGameSession.get(nubmerSession).statusSession = true;

dataGameSession.get(nubmerSession).InitDeck();

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < dataGameSession.get(nubmerSession).NumberPlayers; j++)

{

Card card = new Card();

card = dataGameSession.get(nubmerSession).deck.pollCard();

dataGameSession.get(nubmerSession).setHand(j, card);

}

}

dataGameSession.get(nubmerSession).MoveDetection();

dataGameSession.get(nubmerSession).deck.MoveTrump();

}

}

else

{

writer.write("Ошибка подключения! Сессии с таким названием не существует!");

}

break;

}

case Action.PUT\_CARDS:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int numberPlayers = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

int indexCard = reader.read();

Card card = new Card();

card = dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayers).poolCard(indexCard);

dataGameSession.get(nubmerSession).gameTable.addCardTable(card);

for (int i = 0; i < dataGameSession.get(nubmerSession).dataPlayers.size(); i++)

{

dataGameSession.get(nubmerSession).dataPlayers.get(i).beat = false;

}

int numberPlayerBut = DefinePlayer(dataGameSession, nubmerSession, namePlayer);

if (numberPlayerBut != -1)

{

dataGameSession.get(nubmerSession).gameTable.indexPlayerBut = numberPlayerBut;

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayerBut).turnOrder = 2;

}

break;

}

case Action.FIND\_CARDS:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int numberPlayers = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

// ... !!Добавить отправку количества карт у игроков!!... //

writer.write(dataGameSession.get(nubmerSession).gameTable.round);

writer.flush();

if (dataGameSession.get(nubmerSession).gameTable.getBut == false)

{

writer.write(0);

}

else

{

writer.write(1);

}

writer.flush();

writer.write(dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayers).turnOrder);

writer.flush();

for (int i = 0; i < dataGameSession.get(nubmerSession).gameTable.getSize(); i++)

{

boolean status = dataGameSession.get(nubmerSession).gameTable.getStatus(i);

if (status == true)

{

Card flipCard = new Card();

Card brokenCard = new Card();

writer.write(1);

flipCard = dataGameSession.get(nubmerSession).gameTable.getFlipCard(i);

brokenCard = dataGameSession.get(nubmerSession).gameTable.getBrokenCard(i);

writer.write(flipCard.nominals);

writer.newLine();

writer.write(flipCard.pathInIcon);

writer.newLine();

writer.write(flipCard.suit);

writer.newLine();

writer.write(flipCard.power);

writer.write(brokenCard.nominals);

writer.newLine();

writer.write(brokenCard.pathInIcon);

writer.newLine();

writer.write(brokenCard.suit);

writer.newLine();

writer.write(brokenCard.power);

}

else

{

Card flipCard = new Card();

writer.write(0);

flipCard = dataGameSession.get(nubmerSession).gameTable.getFlipCard(i);

writer.write(flipCard.nominals);

writer.newLine();

writer.write(flipCard.pathInIcon);

writer.newLine();

writer.write(flipCard.suit);

writer.newLine();

writer.write(flipCard.power);

}

writer.flush();

}

break;

}

case Action.WAITING\_CONNECTION:

{

String nameSession = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

if (dataGameSession.get(nubmerSession).GetStatusGame() == true)

{

writer.write("Подключение");

}

else

{

writer.write("Ожидайте остальных игроков!");

}

break;

}

case Action.SESSION\_DATA\_INITAL:

{

String namePlayer = reader.readLine();

String nameSession = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

if (nubmerSession != -1)

{

for (int i = 0; i < dataGameSession.get(nubmerSession).dataPlayers.size(); i++)

{

writer.write(dataGameSession.get(nubmerSession).dataPlayers.get(i).getUserName());

writer.newLine();

}

writer.write("NULL");

writer.newLine();

writer.flush();

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

if (dataGameSession.get(nubmerSession).NumberPlayers != 6)

{

Card card = new Card();

card = dataGameSession.get(nubmerSession).deck.getSuitCard();

writer.write(card.nominals);

writer.newLine();

writer.write(card.pathInIcon);

writer.newLine();

writer.write(card.suit);

writer.newLine();

writer.write(card.power);

}

else

{

writer.write(dataGameSession.get(nubmerSession).deck.getSuitTrump());

}

for (int i = 0; i < 6; i++)

{

Card card = new Card();

card = dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).getHand(i);

writer.write(card.nominals);

writer.newLine();

writer.write(card.pathInIcon);

writer.newLine();

writer.write(card.suit);

writer.newLine();

writer.write(card.power);

}

writer.write("NULL");

writer.newLine();

writer.flush();

int numberCard = dataGameSession.get(nubmerSession).deck.getCardInDeck();

writer.write(numberCard);

writer.write(dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).turnOrder);

writer.flush();

}

else

{

writer.write("Такой сессии не существует!");

writer.newLine();

writer.flush();

}

break;

}

case Action.BUT\_CARDS:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int indexCardHand = reader.read();

int indexCardTable = reader.read();

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

Card butCard = new Card();

butCard.nominals = reader.readLine();

butCard.pathInIcon = reader.readLine();

butCard.suit = reader.readLine();

butCard.power = reader.read();

dataGameSession.get(nubmerSession).gameTable.BeatCardTable(indexCardTable, butCard);

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).poolCard(indexCardHand);

break;

}

case Action.I\_TAKE\_CARD:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

dataGameSession.get(nubmerSession).indexTakePlayer = numberPlayer;

dataGameSession.get(nubmerSession).gameTable.getBut = true;

boolean result = CheckBeat(dataGameSession, nubmerSession);

if (result == true)

{

writer.write(1);

for (int i = 0; i < dataGameSession.get(nubmerSession).gameTable.getSize(); i++)

{

if (dataGameSession.get(nubmerSession).gameTable.getStatus(i) == true)

{

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).setHand(dataGameSession.get(nubmerSession).gameTable.getFlipCard(i));

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).setHand(dataGameSession.get(nubmerSession).gameTable.getBrokenCard(i));

}

else

{

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).setHand(dataGameSession.get(nubmerSession).gameTable.getFlipCard(i));

}

}

dataGameSession.get(nubmerSession).gameTable.ClearTable();

dataGameSession.get(nubmerSession).indexTakePlayer = -1;

dataGameSession.get(nubmerSession).gameTable.getBut = false;

dataGameSession.get(nubmerSession).gameTable.round++;

dataGameSession.get(nubmerSession).gameTable.indexPlayerBut = -1;

}

else

{

writer.write(0);

}

writer.flush();

break;

}

case Action.BEAT\_CARD:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).beat = true;

int transitionPlayer = StrokeTransition(dataGameSession, nubmerSession, dataGameSession.get(nubmerSession).gameTable.indexPlayerBut);

if (transitionPlayer != -1)

{

dataGameSession.get(nubmerSession).dataPlayers.get(transitionPlayer).turnOrder = 1;

}

boolean result = CheckBeat(dataGameSession, nubmerSession);

if (result == true && dataGameSession.get(nubmerSession).indexTakePlayer == -1)

{

dataGameSession.get(nubmerSession).gameTable.ClearTable();

for (int i = 0; i < dataGameSession.get(nubmerSession).NumberPlayers; i++)

{

if (i == dataGameSession.get(nubmerSession).gameTable.indexPlayerBut)

{

dataGameSession.get(nubmerSession).dataPlayers.get(i).turnOrder = 1;

}

else

{

dataGameSession.get(nubmerSession).dataPlayers.get(i).turnOrder = 0;

}

}

dataGameSession.get(nubmerSession).gameTable.round++;

dataGameSession.get(nubmerSession).gameTable.indexPlayerBut = -1;

}

break;

}

case Action.UPDATE\_DATA:

{

String nameSession = reader.readLine();

String namePlayer = reader.readLine();

int nubmerSession = SearchSession(dataGameSession, nameSession);

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, nubmerSession);

boolean check = false;

while (dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).getSizeHand() < 6)

{

if (dataGameSession.get(nubmerSession).deck.getNumberCards() == 0)

{

break;

}

check = true;

dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).setHand(dataGameSession.get(nubmerSession).deck.pollCard());

}

int number = 0;

for (int i = 0; i < dataGameSession.get(nubmerSession).NumberPlayers; i++)

{

if (dataGameSession.get(nubmerSession).dataPlayers.get(i).getSizeHand() < 6)

{

number += 6 - dataGameSession.get(nubmerSession).dataPlayers.get(i).getSizeHand();

}

}

number = dataGameSession.get(nubmerSession).deck.getNumberCards() - number;

if (number <= 0)

{

number = 0;

}

writer.write(number);

writer.flush();

if (check == true)

{

writer.write(1);

}

else

{

writer.write(0);

}

writer.flush();

if (check == true)

{

for (int i = 0; i < dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).getSizeHand(); i++)

{

Card card = new Card();

card = dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).getHand(i);

writer.write(card.nominals);

writer.newLine();

writer.write(card.pathInIcon);

writer.newLine();

writer.write(card.suit);

writer.newLine();

writer.write(card.power);

}

writer.write("NULL");

writer.newLine();

writer.flush();

}

writer.write(dataGameSession.get(nubmerSession).dataPlayers.get(numberPlayer).turnOrder);

writer.flush();

}

}

return dataGameSession;

}

}

// Поиск существующей сессии

// Возвращает номер сессии, если такая сессия есть. Если нет, то возвращает -1

private static int SearchSession(ArrayList<GameSession> dataGameSession, String nameSession)

{

for (int i = 0; i < dataGameSession.size(); i++)

{

if (dataGameSession.get(i).nameSessions == null ? nameSession == null : dataGameSession.get(i).nameSessions.equals(nameSession))

{

return i;

}

}

return -1;

}

// Поиск нужного игрока

private static int SearchPlayer(ArrayList<GameSession> dataGameSession, String namePlayer, int numberSession)

{

for (int i = 0; i < dataGameSession.get(numberSession).dataPlayers.size(); i++)

{

if (dataGameSession.get(numberSession).dataPlayers.get(i).getUserName().equals(namePlayer))

return i;

}

return -1;

}

// Определить игрока, который будет отбиваться

private static int DefinePlayer(ArrayList<GameSession> dataGameSession, int numberSession, String namePlayer)

{

if (dataGameSession.get(numberSession).gameTable.indexPlayerBut == -1)

{

int numberPlayer = SearchPlayer(dataGameSession, namePlayer, numberSession);

if (dataGameSession.get(numberSession).dataPlayers.size() == numberPlayer + 1)

{

return 0;

}

else

{

return numberPlayer + 1;

}

}

return -1;

}

// Определить игрока, кому переходит ход

private static int StrokeTransition(ArrayList<GameSession> dataGameSession, int numberSession, int beatPlayer)

{

if (dataGameSession.get(numberSession).NumberPlayers == 2)

{

return -1;

}

else

{

if (dataGameSession.get(numberSession).dataPlayers.size() == beatPlayer + 1)

{

return 0;

}

else

{

return beatPlayer + 1;

}

}

}

// Проверка на биту

private static boolean CheckBeat(ArrayList<GameSession> dataGameSession, int numberSession)

{

for (int i = 0; i < dataGameSession.get(numberSession).dataPlayers.size(); i++)

{

if (dataGameSession.get(numberSession).dataPlayers.get(i).turnOrder == 1 && dataGameSession.get(numberSession).dataPlayers.get(i).beat != true)

{

return false;

}

}

return true;

}

// Определить игрока, кому переходит ход

private static int StrokeTransitionNext(ArrayList<GameSession> dataGameSession, int numberSession, int beatPlayer)

{

if (dataGameSession.get(numberSession).NumberPlayers == 2)

{

if (beatPlayer == 0)

{

return 1;

}

return 0;

}

else

{

if (dataGameSession.get(numberSession).dataPlayers.size() == beatPlayer + 1)

{

return 0;

}

return beatPlayer + 1;

}

}

}

**Deck.java**

package game.player;

import java.util.LinkedList;

import java.util.Random;

public class Deck

{

private Card suitTrump = new Card(); // Козырная масть

private LinkedList<Card> deck = new LinkedList(); // Карты в деке

// Конструктор инициализации карт

public Deck()

{

//

// Инициализация карт масти "крести"

//

deck.add(AddCard(0, "6","../Game Durak/src/image/6\_clubs.png", "clubs"));

deck.add(AddCard(1, "7","../Game Durak/src/image/7\_clubs.png", "clubs"));

deck.add(AddCard(2, "8","../Game Durak/src/image/8\_clubs.png", "clubs"));

deck.add(AddCard(3, "9","../Game Durak/src/image/9\_clubs.png", "clubs"));

deck.add(AddCard(4, "10","../Game Durak/src/image/10\_clubs.png", "clubs"));

deck.add(AddCard(5, "J","../Game Durak/src/image/j\_clubs.png", "clubs"));

deck.add(AddCard(6, "Q","../Game Durak/src/image/q\_clubs.png", "clubs"));

deck.add(AddCard(7, "K","../Game Durak/src/image/k\_clubs.png", "clubs"));

deck.add(AddCard(8, "A","../Game Durak/src/image/ace\_clubs.png", "clubs"));

//

// Инициализация карт масти "буби"

//

deck.add(AddCard(0, "6","../Game Durak/src/image/6\_diams.png", "diams"));

deck.add(AddCard(1, "7","../Game Durak/src/image/7\_diams.png", "diams"));

deck.add(AddCard(2, "8","../Game Durak/src/image/8\_diams.png", "diams"));

deck.add(AddCard(3, "9","../Game Durak/src/image/9\_diams.png", "diams"));

deck.add(AddCard(4, "10","../Game Durak/src/image/10\_diams.png", "diams"));

deck.add(AddCard(5, "J","../Game Durak/src/image/j\_diams.png", "diams"));

deck.add(AddCard(6, "Q","../Game Durak/src/image/q\_diams.png", "diams"));

deck.add(AddCard(7, "K","../Game Durak/src/image/k\_diams.png", "diams"));

deck.add(AddCard(8, "A","../Game Durak/src/image/ace\_diams.png", "diams"));

//

// Инициализация карт масти "черви"

//

deck.add(AddCard(0, "6","../Game Durak/src/image/6\_hearts.png", "hearts"));

deck.add(AddCard(1, "7","../Game Durak/src/image/7\_hearts.png", "hearts"));

deck.add(AddCard(2, "8","../Game Durak/src/image/8\_hearts.png", "hearts"));

deck.add(AddCard(3, "9","../Game Durak/src/image/9\_hearts.png", "hearts"));

deck.add(AddCard(4, "10","../Game Durak/src/image/10\_hearts.png", "hearts"));

deck.add(AddCard(5, "J","../Game Durak/src/image/j\_hearts.png", "hearts"));

deck.add(AddCard(6, "Q","../Game Durak/src/image/q\_hearts.png", "hearts"));

deck.add(AddCard(7, "K","../Game Durak/src/image/k\_hearts.png", "hearts"));

deck.add(AddCard(8, "A","../Game Durak/src/image/ace\_hearts.png", "hearts"));

//

// Инициализация карт масти "вини"

//

deck.add(AddCard(0, "6","../Game Durak/src/image/6\_spades.png", "spades"));

deck.add(AddCard(1, "7","../Game Durak/src/image/7\_spades.png", "spades"));

deck.add(AddCard(2, "8", "../Game Durak/src/image/8\_spades.png", "spades"));

deck.add(AddCard(3, "9", "../Game Durak/src/image/9\_spades.png", "spades"));

deck.add(AddCard(4, "10", "../Game Durak/src/image/10\_spades.png", "spades"));

deck.add(AddCard(5, "J", "../Game Durak/src/image/j\_spades.png", "spades"));

deck.add(AddCard(6, "Q", "../Game Durak/src/image/q\_spades.png", "spades"));

deck.add(AddCard(7, "K", "../Game Durak/src/image/k\_spades.png", "spades"));

deck.add(AddCard(8, "A", "../Game Durak/src/image/ace\_spades.png", "spades"));

}

// Добавление карт в коллекцию

private Card AddCard(int power, String nominals, String pathInIcon, String suit)

{

Card filling = new Card();

filling.power = power;

filling.suit = suit;

filling.nominals = nominals;

filling.pathInIcon = pathInIcon;

return filling;

}

// Метод сортировки карт в колоде

public void SortCard()

{

LinkedList<Card> newDeck = new LinkedList();

int[] arr = new int[36];

Random random = new Random();

for (int i = 0; i < 36; i++)

arr[i] = I;

for(int i = 0; i < arr.length; i++)

{

int index = i + random.nextInt(arr.length - i);

int temp = arr[i];

arr[i] = arr[index];

arr[index] = temp;

}

for (int i = 0; i < arr.length; i++)

{

Card card = new Card();

card = deck.get(arr[i]);

newDeck.add(card);

}

deck = newDeck;

}

}

Приложение А.2 – Исходный код Клиента

**GameStart.java:**

public class GameStart extends javax.swing.JFrame

{

private static String userName;

private static String sessionToConnect;

private static boolean visible = true;

public GameStart()

{

initComponents();

}

private void CreateGameActionPerformed(java.awt.event.ActionEvent evt) {

sessionToConnect = JOptionPane.showInputDialog("Введите имя игровой сессии, которую хотите создать:");

String numberPlayers = JOptionPane.showInputDialog("Введите количество игроков:");

try (Socket socket = new Socket("127.0.0.1", 8080);)

{

try

(

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Принимает пакеты (читает)

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); // Отправляет пакеты (записывает)

)

{

//

// Отправка на сервер запроса на создание игровой сессии

// Отправляем имя игрока, имя сессии и кол-ва игроков на сессии

//

writer.write(Action.CREATE\_GAME);

writer.write(userName);

writer.newLine();

writer.write(sessionToConnect);

writer.newLine();

writer.write(Integer.valueOf(numberPlayers));

writer.newLine();

writer.flush();

String info = null;

while (info == null)

{

info = reader.readLine();

try

{

Thread.sleep(200);

}

catch (InterruptedException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

System.out.println(info);

WaitingConnectionThread waitingConnection = new WaitingConnectionThread();

waitingConnection.start();

JOptionPane.showMessageDialog(MainPanel, "Пожалуйста, дождитесь подключения остальных игроков!");

while(visible == true)

{

try

{

Thread.sleep(1000);

}

catch (InterruptedException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

setVisible(false);

}

}

catch (IOException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void SearchGameActionPerformed(java.awt.event.ActionEvent evt) {

sessionToConnect = JOptionPane.showInputDialog("Введите имя игровой сессии, куда хотите подключиться:");

try (Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Принимает пакеты (читает)

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); // Отправляет пакеты (записывает)

)

{

writer.write(Action.CONNECT\_SESSIONS);

writer.write(userName);

writer.newLine();

writer.write(sessionToConnect);

writer.newLine();

writer.flush();

String info = null;

while (info == null)

{

info = reader.readLine();

try

{

Thread.sleep(200);

}

catch (InterruptedException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

System.out.println(info);

WaitingConnectionThread waitingConnection = new WaitingConnectionThread();

waitingConnection.start();

JOptionPane.showMessageDialog(MainPanel, "Пожалуйста, дождитесь подключения остальных игроков!");

while(visible == true)

{

try

{

Thread.sleep(1000);

}

catch (InterruptedException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

setVisible(false);

}

}

catch (IOException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void ExitGameActionPerformed(java.awt.event.ActionEvent evt) {

this.dispose();

}

// Ожидание подключения к игровой сессии

private void WaitingConnection() throws IOException

{

String connectionStatus = null;

while(!"Подключение".equals(connectionStatus))

{

try(Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Принимает пакеты (читает)

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); // Отправляет пакеты (записывает)

)

{

writer.write(Action.WAITING\_CONNECTION);

writer.write(sessionToConnect);

writer.newLine();

writer.flush();

try

{

Thread.sleep(1000);

}

catch (InterruptedException ex)

{

Logger.getLogger(GameStart.class.getName()).log(Level.SEVERE, null, ex);

}

connectionStatus = reader.readLine();

System.out.println(connectionStatus);

}

}

}

String[] args = null;

GamePlay.StartGame(userName, sessionToConnect);

visible = false;

//setVisible(false);

}

**GamePlay.java:**

public class GamePlay extends javax.swing.JFrame

{

private static int round; // Раунд игры

private static Player player = new Player();

private static GameTable gameTable = new GameTable();

public GamePlay()

{

initComponents();

}

private void jgameTableMouseClicked(java.awt.event.MouseEvent evt) {

Rules rules = new Rules();

switch(player.getTurnOrder())

{

case 0:

{

showMessageDialog(jMainPanel, "Дождитесь своего ход!");

break;

}

case 1:

{

int indexCard = -1;

for (int i = 0; i < player.getNumberCard(); i++)

{

if (player.getPressButton(i) == true)

{

indexCard = i;

}

}

if (indexCard == -1)

{

showMessageDialog(jMainPanel, "Выберите карту!");

}

else

{

boolean result = rules.PutCard(player.getCard(indexCard), gameTable);

if (result == true)

{

gameTable.addCardTable(player.getCard(indexCard));

addCardTable(player.getCard(indexCard), gameTable.getSize() - 1);

try

{

PutConnect(indexCard);

}

catch (IOException ex)

{

Logger.getLogger(GamePlay.class.getName()).log(Level.SEVERE, null, ex);

}

player.poolCard(indexCard);

repaintGameHand(player.getNumberCard());

if (player.getNumberCard() == 0 && player.getCardInDeck() == 0)

{

showMessageDialog(jMainPanel, "Вы выиграли!");

this.dispose();

}

}

else

{

showMessageDialog(jMainPanel, "Эту карту нельзя положить!");

}

}

}

}

}

private void jReadyActionPerformed(java.awt.event.ActionEvent evt) {

ServerConnect serverConnect = new ServerConnect();

if (jReady.getText().equals("Бито"))

{

try

{

serverConnect.BeatCardSessionConnection(player.sessionToConnect, player.getUserName());

}

catch (IOException ex)

{

Logger.getLogger(GamePlay.class.getName()).log(Level.SEVERE, null, ex);

}

//player.setTurnOrder(0);

}

else if (jReady.getText().equals("Беру"))

{

player.setTurnOrder(0);

ITakeConnectionThread myThread = new ITakeConnectionThread();

myThread.start();

}

}

// Отправляет сообщение на сервер, какой картой походил игрок

private void PutConnect(int indexCard) throws IOException

{

ServerConnect serverConnect = new ServerConnect();

serverConnect.PutConnectServer(player.getSessionToConnect(), player.getUserName(), indexCard);

}

// Старт игры. Здесь получаются изначальные данные о игровой сессии

public static void StartGame(String m\_userName, String m\_sessionToConnection)

{

player.setUserName(m\_userName);

player.setSessionToConnect(m\_sessionToConnection);

try(Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

)

{

writer.write(Action.SESSION\_DATA\_INITAL);

writer.write(player.getUserName());

writer.newLine();

writer.write(player.getSessionToConnect());

writer.newLine();

writer.flush();

String playerName = reader.readLine();

while (!playerName.equals("NULL"))

{

player.addPlayerNames(playerName);

playerName = reader.readLine();

}

if (player.getSizePlayerNames() != 6)

{

Card card = new Card();

card.nominals = reader.readLine();

card.pathInIcon = reader.readLine();

card.suit = reader.readLine();

card.power = reader.read();

player.setSuitTrump(card);

}

else

{

Card card = new Card();

card.suit = reader.readLine();

player.setSuitTrump(card);

}

String nominals;

while (true)

{

Card card = new Card();

nominals = reader.readLine();

if (nominals.equals("NULL"))

break;

card.nominals = nominals;

card.pathInIcon = reader.readLine();

card.suit = reader.readLine();

card.power = reader.read();

player.addCard(card);

}

player.setCardInDeck(reader.read());

player.setTurnOrder(reader.read());

}

}

catch(IOException ex)

{

System.out.println("Подключение...");

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

StartGame(m\_userName, m\_sessionToConnection);

}

player.setNumberPlays(player.getSizePlayerNames());

String args[] = null;

main(args);

}

// Начальная инициализация интерфейса

private static void InitStartum()

{

player.getIconPath();

player.SortPlayers();

switch(player.getNumberPlays())

{

case 2:

{

setPlayerIcon(player.getPlayerPathIcon(0), player.getPlayerNames(0), jPanelPlayers.getWidth() / 2);

setRivalIcon(player.getPlayerPathIcon(1), player.getPlayerNames(1), jPanelRival.getWidth() / 2, 6);

break;

}

case 3: case 4: case 5: case 6:

{

setPlayerIcon(player.getPlayerPathIcon(0), player.getPlayerNames(0), jPanelPlayers.getWidth() / 2);

int positionIcon = jPanelRival.getWidth() / player.getNumberPlays();

for (int i = 1, j = 0; i < player.getNumberPlays(); i++, j++)

{

setRivalIcon(player.getPlayerPathIcon(i), player.getPlayerNames(i), positionIcon + positionIcon \* j, 6);

}

break;

}

}

}

static class ButtonGameTable implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

String action = e.getActionCommand();

int indexCard = parseInt(action);

int indexCardBeat = -1;

for (int i = 0; i < player.getNumberCard(); i++)

{

if (player.getPressButton(i) == true)

{

indexCardBeat = i;

}

}

if (indexCardBeat == -1)

{

showMessageDialog(jMainPanel, "Выберите карту!");

}

switch(player.getTurnOrder())

{

case 0:

{

showMessageDialog(jMainPanel, "Дождитесь своего хода!");

break;

}

case 1:

{

showMessageDialog(jMainPanel, "Это сделать невозможно!");

break;

}

case 2:

{

if (gameTable.getStatus(indexCard) == false)

{

Rules rules = new Rules();

ServerConnect serverConnect = new ServerConnect();

boolean result = rules.BeatCard(player.getCard(indexCardBeat), gameTable.getFlipCard(indexCard));

if (result == true)

{

gameTable.setStatus(indexCard, result);

gameTable.setBrokenCard(player.getCard(indexCardBeat), indexCard);

while (CheckStatusThread != false);

try

{

synchronized(findConnect)

{

findConnect.interrupt();

}

serverConnect.ButConnectServer(player.getSessionToConnect(), player.getUserName(), indexCard, player.getCard(indexCardBeat), indexCardBeat);

player.poolCard(indexCardBeat);

repaintGameHand(player.getNumberCard());

UpdateCardTable();

synchronized(findConnect)

{

findConnect.notify();

}

}

catch (IOException ex)

{

Logger.getLogger(GamePlay.class.getName()).log(Level.SEVERE, null, ex);

}

}

else

{

showMessageDialog(jMainPanel, "Этой картой нельзя отбить!");

}

}

else

{

showMessageDialog(jMainPanel, "Эта карта уже отбита!");

}

break;

}

}

System.out.println("Нажатие кнопки! От - " + e.getActionCommand() + "\n");

}

}

static class FindConnectionThread extends Thread

{

public void run()

{

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

ServerConnect serverConnect = new ServerConnect();

while (true)

{

CheckStatusThread = true;

GameTable newGameTable = new GameTable();

newGameTable = serverConnect.FindConnectServer(player.getSessionToConnect(), player.getUserName());

if (CheckGameTable(newGameTable) == true)

{

player.setTurnOrder(serverConnect.turnOrder);

if (player.getTurnOrder() == 1)

{

jReady.setText("Бито");

jReady.setVisible(true);

}

else if (player.getTurnOrder() == 2)

{

jReady.setText("Беру");

jReady.setVisible(true);

}

gameTable = newGameTable;

if (gameTable.getSize() != 0)

{

tempGameTable = gameTable;

}

UpdateCardTable();

}

//if (newGameTable.checkTakePlayer != 0)

//{

player.setTurnOrder(serverConnect.turnOrder);

if (player.getTurnOrder() == 1)

{

jReady.setText("Бито");

jReady.setVisible(true);

}

//}

if ((gameTable.getSize() == 0 || CheckForRepulsed() == false) && newGameTable.checkTakePlayer == 0)

{

if (player.getTurnOrder() == 1)

{

jReady.setVisible(false);

}

}

if (gameTable.getSize() == 0 || CheckForRepulsed() == true)

{

if (player.getTurnOrder() == 2)

{

jReady.setVisible(false);

}

}

round = serverConnect.round;

if (prevRound != round)

{

Player prevPlayer = new Player();

//prevPlayer = player;

prevPlayer.setPlayerCard(serverConnect.UpdateDataSessionConnection(player.sessionToConnect, player.getUserName()));

if (prevPlayer.getNumberCard() != 0)

{

player.setPlayerCard(prevPlayer.getPlayerCard());

}

player.setTurnOrder(serverConnect.turnOrder);

player.setCardInDeck(serverConnect.numberCards);

Integer numberCardInDeck = serverConnect.numberCards;

//int check = serverConnect.check;

jNumberCard.setText(numberCardInDeck.toString());

repaintGameHand(player.getNumberCard());

jReady.setVisible(false);

serverConnect.check = 0;

if (player.getTurnOrder() == 1)

{

showMessageDialog(jMainPanel, "Ваш ход!");

}

}

prevRound = round;

CheckStatusThread = false;

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

}

}

}

static class ITakeConnectionThread extends Thread

{

public void run()

{

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

ServerConnect serverConnect = new ServerConnect();

boolean result = false;

while (result != true)

{

while (CheckStatusThread == true);

try

{

synchronized(findConnect)

{

findConnect.interrupt();

}

result = serverConnect.ITakeSessionConnect(player.sessionToConnect, player.getUserName());

}

catch (IOException ex)

{

Logger.getLogger(GamePlay.class.getName()).log(Level.SEVERE, null, ex);

}

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

}

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

{

if (tempGameTable.getStatus(i) == true)

{

player.addCard(tempGameTable.getFlipCard(i));

player.addCard(tempGameTable.getBrokenCard(i));

}

else

{

player.addCard(tempGameTable.getFlipCard(i));

}

}

// tempGameTable.ClearTable();

repaintGameHand(player.getSizeHand());

jgameTable.removeAll();

jgameTable.repaint();

synchronized(findConnect)

{

findConnect.notify();

}

}

}

**ServerConnect.java:**

package game.action;

import game.durak.Card;

import game.durak.GameTable;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.Socket;

import java.util.LinkedList;

public class ServerConnect

{

public int turnOrder;

public int round;

public int numberCards;

public int check;

// Отправить на сервер карту, которой игрок походил

public boolean PutConnectServer(String sessionToConnect, String namePlayer, int indexCard) throws IOException

{

try(Socket socket = new Socket("127.0.0.1", 8080))

{

try(BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())))

{

writer.write(Action.PUT\_CARDS);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.write(indexCard);

writer.flush();

}

}

catch(IOException ex)

{

System.out.println("Идёт отправка карты...");

try

{

Thread.sleep(500);

}

catch(InterruptedException ie) {}

PutConnectServer(sessionToConnect, namePlayer, indexCard);

}

return true;

}

// Получение состояния игрового стола

public GameTable FindConnectServer(String sessionToConnect, String namePlayer)

{

GameTable newGameTable = new GameTable();

try(Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()))

)

{

writer.write(Action.FIND\_CARDS);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.flush();

int status = 0;

int index = 0;

round = reader.read();

newGameTable.checkTakePlayer = reader.read();

turnOrder = reader.read();

while (status != -1)

{

status = reader.read();

if (status == 1)

{

Card flipCard = new Card();

Card brokenCard = new Card();

flipCard.nominals = reader.readLine();

flipCard.pathInIcon = reader.readLine();

flipCard.suit = reader.readLine();

flipCard.power = reader.read();

brokenCard.nominals = reader.readLine();

brokenCard.pathInIcon = reader.readLine();

brokenCard.suit = reader.readLine();

brokenCard.power = reader.read();

newGameTable.addCardTable(flipCard);

newGameTable.setBrokenCard(brokenCard, index);

newGameTable.setStatus(index, true);

}

else if (status == 0)

{

Card flipCard = new Card();

flipCard.nominals = reader.readLine();

flipCard.pathInIcon = reader.readLine();

flipCard.suit = reader.readLine();

flipCard.power = reader.read();

newGameTable.addCardTable(flipCard);

newGameTable.setStatus(index, false);

}

index++;

}

}

}

catch(IOException ex)

{

System.out.println("Идёт получение карты...");

try

{

Thread.sleep(2500);

}

catch(InterruptedException ie) {}

FindConnectServer(sessionToConnect, namePlayer);

}

return newGameTable;

}

// Отбить карту противника

public void ButConnectServer(String sessionToConnect, String namePlayer, int indexCardTable, Card butCard, int indexCardHand) throws IOException

{

try (Socket socket = new Socket("127.0.0.1", 8080))

{

try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())))

{

writer.write(Action.BUT\_CARDS);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.write(indexCardHand);

writer.flush();

writer.write(indexCardTable);

writer.write(butCard.nominals);

writer.newLine();

writer.write(butCard.pathInIcon);

writer.newLine();

writer.write(butCard.suit);

writer.newLine();

writer.write(butCard.power);

writer.flush();

}

}

catch(IOException ex)

{

System.out.println("Идёт отбитие карты...");

try

{

Thread.sleep(2500);

}

catch(InterruptedException ie) {}

ButConnectServer(sessionToConnect, namePlayer, indexCardTable, butCard, indexCardHand);

}

}

// Послать сообщение на сервер о том, что взял карты со стола

public boolean ITakeSessionConnect(String sessionToConnect, String namePlayer) throws IOException

{

try (Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()))

)

{

writer.write(Action.I\_TAKE\_CARD);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.flush();

int result = reader.read();

if (result == 1)

{

return true;

}

else

{

return false;

}

}

}

}

public void BeatCardSessionConnection(String sessionToConnect, String namePlayer) throws IOException

{

try (Socket socket = new Socket("127.0.0.1", 8080))

{

try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())))

{

writer.write(Action.BEAT\_CARD);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.flush();

}

}

catch(IOException ex)

{

System.out.println("Сейчас отобьём...");

try

{

Thread.sleep(2500);

}

catch(InterruptedException ie) {}

BeatCardSessionConnection(sessionToConnect, namePlayer);

}

}

public LinkedList<Card> UpdateDataSessionConnection(String sessionToConnect, String namePlayer)

{

LinkedList<Card> newHand = new LinkedList();

try (Socket socket = new Socket("127.0.0.1", 8080))

{

try

(

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()))

)

{

writer.write(Action.UPDATE\_DATA);

writer.write(sessionToConnect);

writer.newLine();

writer.write(namePlayer);

writer.newLine();

writer.flush();

numberCards = reader.read();

check = reader.read();

if (check == 1)

{

String nominals;

while (true)

{

Card card = new Card();

nominals = reader.readLine();

if (nominals.equals("NULL"))

break;

card.nominals = nominals;

card.pathInIcon = reader.readLine();

card.suit = reader.readLine();

card.power = reader.read();

newHand.add(card);

}

}

turnOrder = reader.read();

}

}

catch(IOException ex)

{

System.out.println("Сейчас отобьём...");

try

{

Thread.sleep(2500);

}

catch(InterruptedException ie) {}

UpdateDataSessionConnection(sessionToConnect, namePlayer);

}

return newHand;

}

}

**GameTable.java**

public class GameTable

{

private class CardFlip

{

public Card flipCard; // Карта, которую нужно отбить

public Card brokenCard; // Карта, которой отбили

public boolean status; // Состояние карты. false - ещё не отбита, true - отбита

}

public ArrayList<CardFlip> cardFlip = new ArrayList();

public int checkTakePlayer = -1;

public void addCardTable(Card flipCard)

{

CardFlip cardFlipTemp = new CardFlip();

cardFlipTemp.flipCard = flipCard;

cardFlip.add(cardFlipTemp);

}

public boolean BeatCardTable(int indexCard, Card brokenCard)

{

if (cardFlip.get(indexCard).status == false)

{

cardFlip.get(indexCard).brokenCard = brokenCard;

cardFlip.get(indexCard).status = true;

return true;

}

return false;

}

}

**Player.java**

public class Player

{

private ArrayList<String> playerNames = new ArrayList(); // Имена всех игроков

private LinkedList<Card> playerCard = new LinkedList(); // Рука игрока

private ArrayList<InfoPlayer> infoPlayer = new ArrayList(); // Информация об игроках

private Card suitTrump = new Card(); // Козырь

private int turnOrder; // Определение возможности хода. 0 - ходить нельзя, 1 - можно ходить или подкидывать, 2 можно отбиваться

private int numberCardInDeck; // Количество карт в колоде

private int numberPlays; // Количество игроков в сессии

private String userName; // Имя главного игрока

public String sessionToConnect; // Имя сессии, к которой нужно подключаться

// Сортировка имён игроков в нужной последовательности

public void SortPlayers()

{

ArrayList<String> sortPlayerNames = new ArrayList();

int nameMain = 0;

while(!playerNames.get(nameMain).equals(userName))

{

nameMain++;

}

for (int i = nameMain; i < playerNames.size(); i++)

{

sortPlayerNames.add(playerNames.get(i));

}

for (int i = 0; i < nameMain; i++)

{

sortPlayerNames.add(playerNames.get(i));

}

playerNames = sortPlayerNames;

}

public void setPlayerCard(LinkedList<Card> newPlayerCard)

{

playerCard.clear();

playerCard = newPlayerCard;

}

}

**Rules.java:**

package game.durak;

public class Rules

{

// Добавление карты на стол. True - можно добавить, false - нельзя

public boolean PutCard(Card card, GameTable gameTable)

{

if (gameTable.getSize() == 0)

{

return true;

}

else

{

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

{

if (gameTable.getStatus(i) == true)

{

Card flipCard = new Card();

Card brokenCard = new Card();

flipCard = gameTable.getFlipCard(i);

brokenCard = gameTable.getBrokenCard(i);

if (flipCard.nominals.equals(card.nominals) || brokenCard.nominals.equals(card.nominals))

{

return true;

}

}

else

{

Card flipCard = new Card();

flipCard = gameTable.getFlipCard(i);

if (flipCard.nominals.equals(card.nominals))

{

return true;

}

}

}

}

return false;

}

// Отбивание карты

public boolean BeatCard(Card beatCard, Card card)

{

if (beatCard.power > 8 && beatCard.power > card.power)

{

return true;

}

}

}

Приложение В. UML-диаграммы приложения

Приложение В.1 – UML-диаграмма вариантов использования

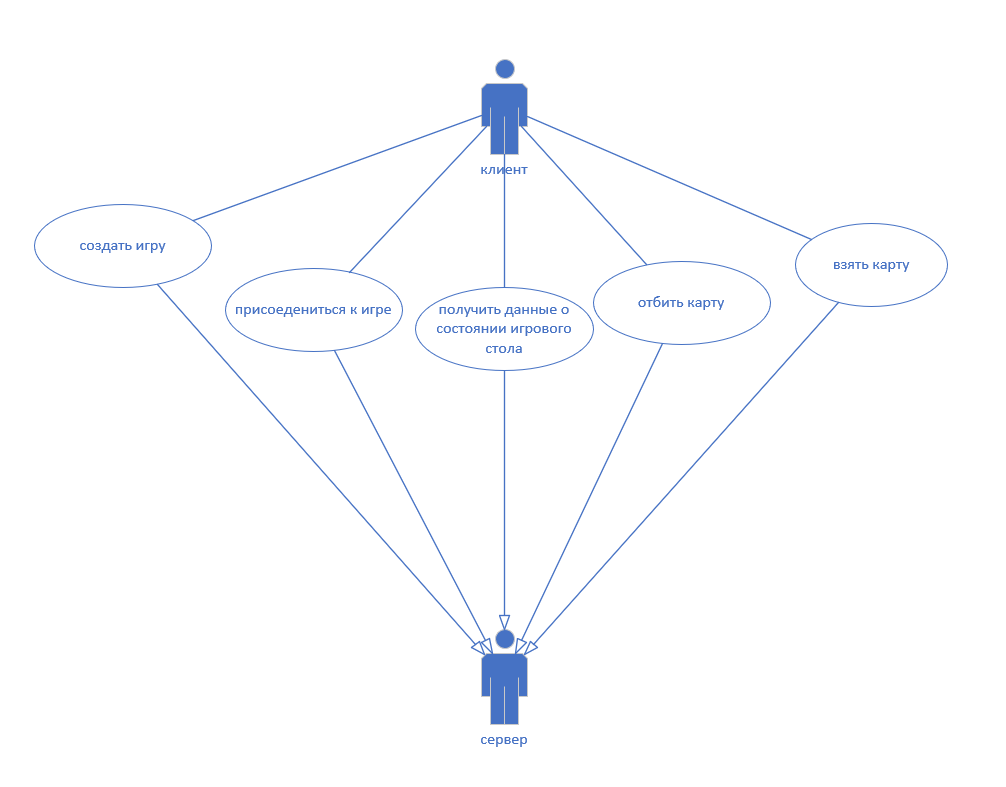


Рисунок B1 – UML диаграмма вариантов использования

Приложение В.2 – UML-диаграмма классов клиента

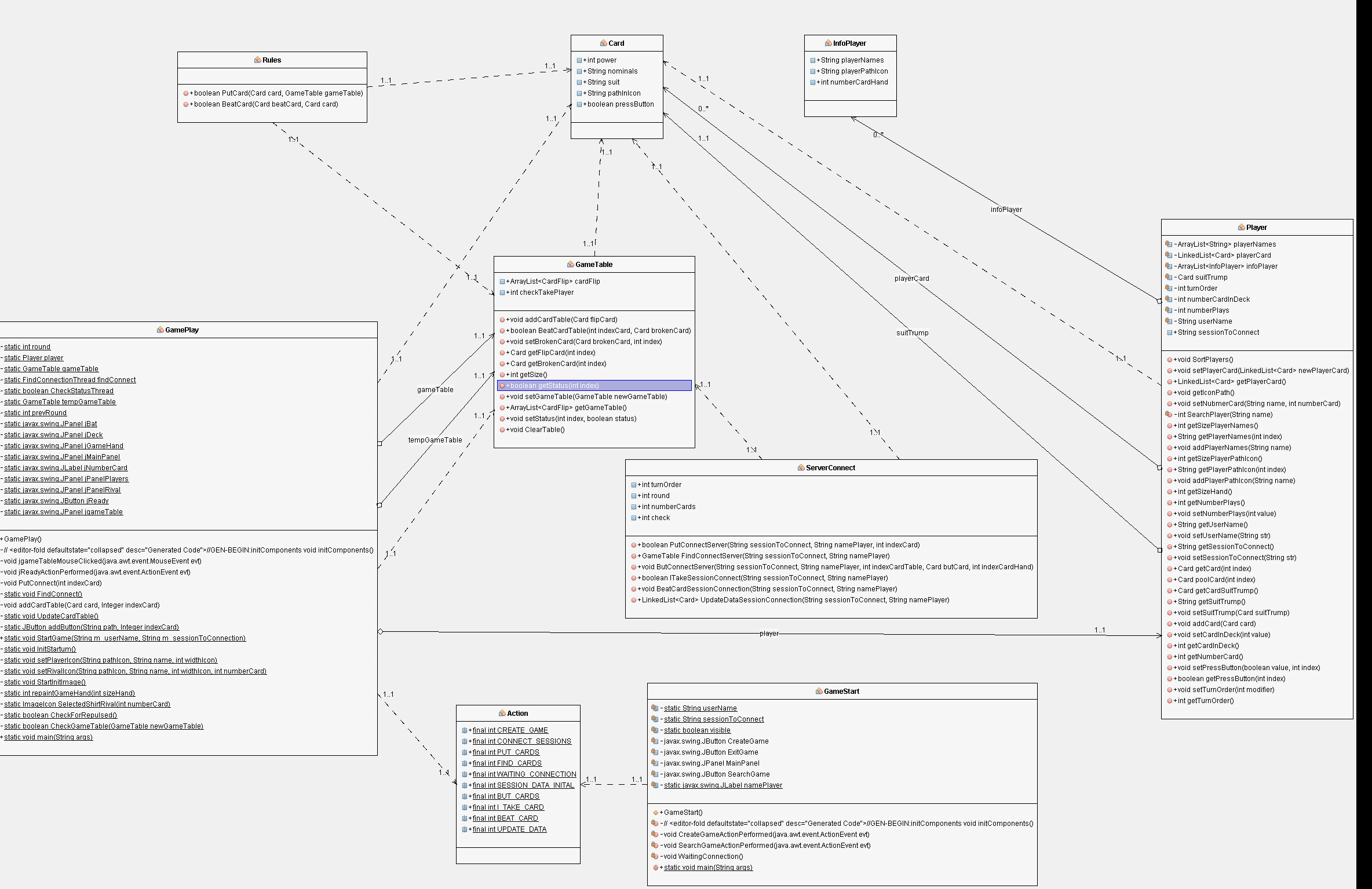


Рисунок B2 – UML диаграмма классов клиента

Приложение В.3 – UML-диаграмма классов сервера

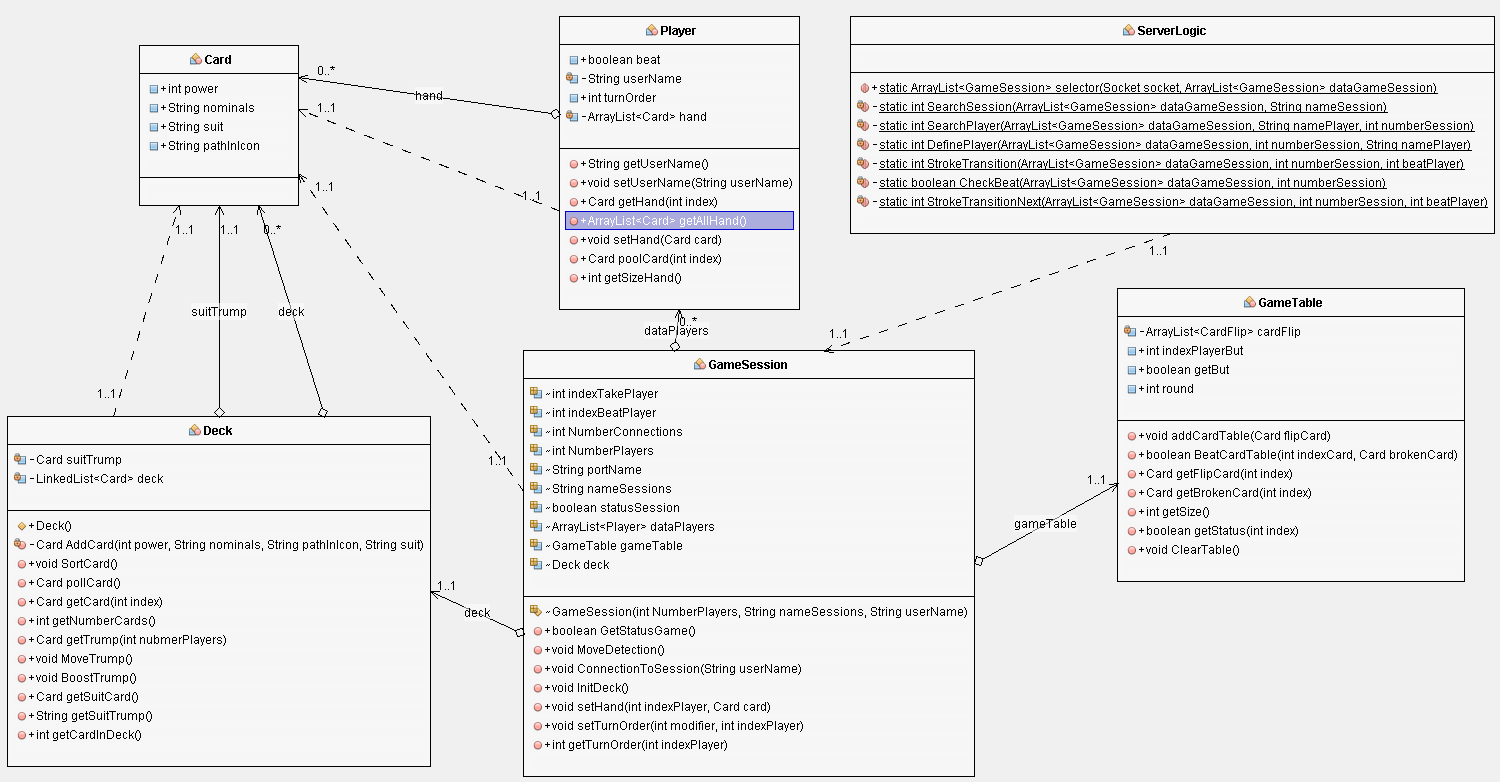


Рисунок B3 – UML диаграмма классов сервера

Приложение В.4 – UML-диаграмма деятельности

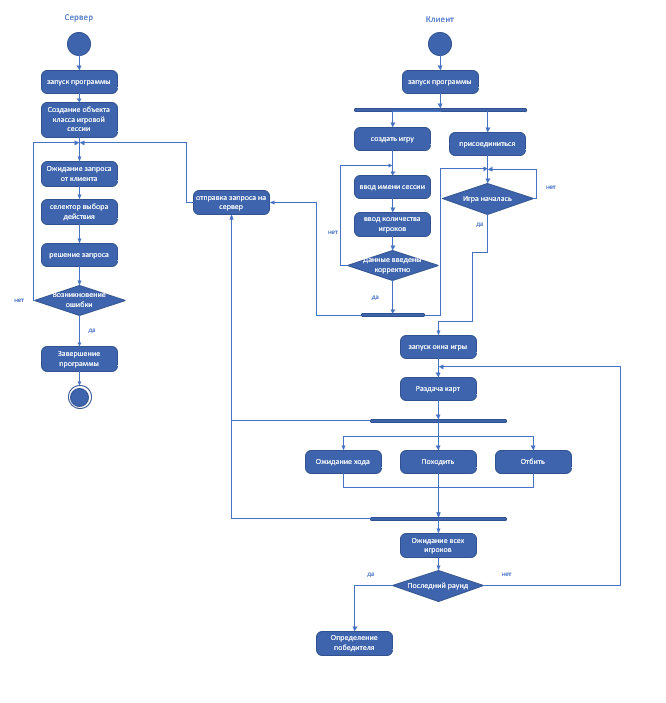


Рисунок B4 – UML диаграмма деятельности

Приложение В.5 – UML-диаграмма развёртывания

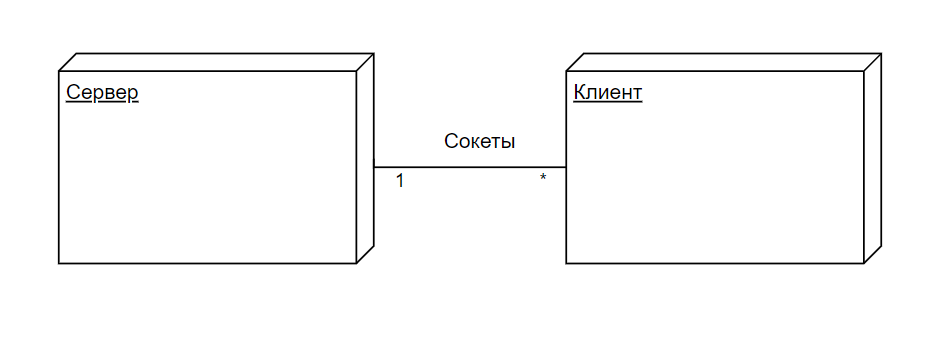


Рисунок B5 – UML диаграмма развёртывания

Приложение В.6 – UML-диаграмма последовательности

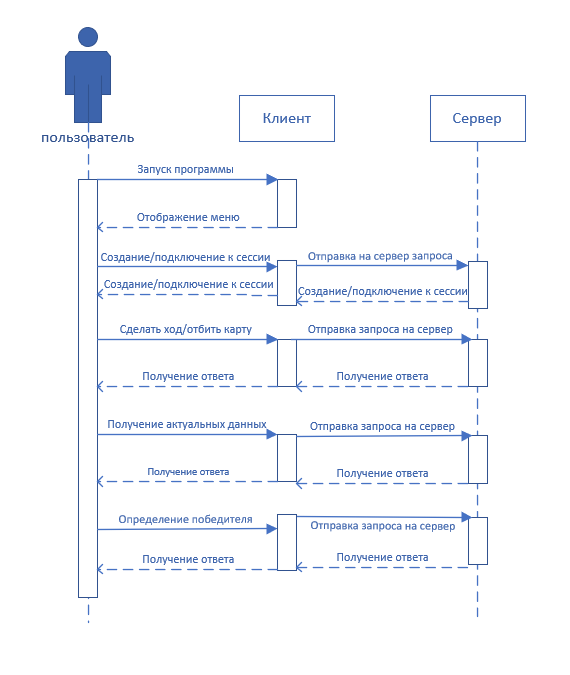


Рисунок B6 – UML диаграмма последовательности