

**Содержание**

[Введение 3](#_Toc105024078)

[1 Постановка задачи 4](#_Toc105024079)

[2 Выбор решения 5](#_Toc105024080)

[3 Описание программы 6](#_Toc105024081)

[3.1 Серверная часть 6](#_Toc105024082)

[3.2 Клиентская часть 7](#_Toc105024083)

[4 Описание способа организации работы программы 8](#_Toc105024084)

[5 Эксперементальная часть 11](#_Toc105024085)

[Заключение 17](#_Toc105024086)

[Список литературы 18](#_Toc105024087)

[Приложение А «Листинг клиентской части приложения» 19](#_Toc105024088)

[Gomoku.java 19](#_Toc105024089)

[Приложение Б «Листинг серверной части приложения» 26](#_Toc105024090)

[Server.java 26](#_Toc105024091)

[Приложение В «UML диаграммы» 31](#_Toc105024092)

[Диаграмма вариантов использования 31](#_Toc105024093)

[Диаграммы классов 32](#_Toc105024094)

[Диаграмма деятельности 34](#_Toc105024095)

[Диаграмма развёртывания 35](#_Toc105024096)

[Диаграмма последовательности 36](#_Toc105024097)

Введение

В современном мире невозможно представить жизнь человека без взаимодействия с компьютером. Возможность передачи информации на расстоянии интересовала людей всегда. Сегодня люди с легкостью по всему миру могут обмениваться не только сообщениями, но и графикой и звуковой и другими видами информации.

Язык Java является основой практически для всех возможных типов сетевых приложений и всеобщим стандартом для разработки встроенных и мобильных приложений, веб-контента, игр, и корпоративного программного обеспечения. В мире насчитывается более девяти миллионов специалистов, работающих с языком Java, который позволяет эффективно разрабатывать, тестировать, внедрять и использовать программное обеспечение.

Для закрепления полученного материала, была выбрана разработка многомодульного клиент-серверного приложения, а именно развлекательная игра “Пять в ряд”.

# Постановка задачи

Разработать систему многомодульных программ, клиент-серверной архитектуры, позволяющую играть в игру пользователями в рамках локальной сети.

Функции сервера: прием и передача информации от пользователя, сбор статистики о ходе игры.

Функции клиента: взаимодействие с графическим интерфейсом, 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

# Выбор решения

Соединение клиента с сервером реализовано с помощью библиотеки *java.net.Socket*.

Пользовательский интерфейс реализован с помощью Swing (Java)

Сокет — это один конец двустороннего канала связи между двумя программами, работающими в сети. Соединяя вместе два сокета, можно передавать данные между разными процессами (локальными или удаленными). Реализация сокетов обеспечивает инкапсуляцию протоколовсетевого и транспортного уровней. Для сокетов жизнь дескриптора можно разделить на три фазы: открыть (создать) сокет, получить из сокета или отправить сокету и в конце концов закрыть сокет.

Обычно приложение клиент-сервер, использующее сокеты, состоит из двух разных приложений - клиента, инициирующего соединение с целью (сервером), и сервера, ожидающего соединения от клиента.

На стороне клиента, приложение должно знать адрес цели и номер порта. Отправляя запрос на соединение, клиент пытается установить соединение с сервером:

Если события развиваются удачно, при условии, что сервер запущен прежде, чем клиент попытался с ним соединиться, сервер соглашается на соединение. Дав согласие, серверное приложение создает новый сокет для взаимодействия именно с установившим соединение клиентом:

Теперь клиент и сервер могут взаимодействовать между собой, считывая сообщения каждый из своего сокета и, соответственно, записывая сообщения.

# Описание программы

Данная программа состоит из клиентской и серверной части. Первым запускается сервер. На остальных компьютерах локальной сети - клиент. Сервер представлен в виде консоли, клиентская часть - оконное приложение.

Программа осуществляет игровой процессмежду пользователями в локальной сети.

## Серверная часть

**Server.java** – главный файл серверного процесса. Здесь реализуется функция Main. Задается порт и объявляется сокет. Происходит ожидание клиента для создания с ним связи.

## Клиентская часть

**Gomoku.java** – файл, в котором инициализируются элементы Swing, определяются свойства каждого элемента, а также устанавливаются методы, вызывающиеся при взаимодействии с каждым элементом.

Gomoku() – класс, в котором реализуется и создается игровое окно.

Board() – класс, в котором описывается создание кнопок и меток, которые реагируют на нажатие мыши.

actionPerformed() – метод, который реагирует на нажатия кнопок пользователем

doNewGame() – метод, который отвечает за начало новой игры и задаёт стартовые параметры сессии (обнуляет игровое поле, устанавливает ход первого игрока)

doResign() – метод, который определяет «ничью» в игровой сессии и выводит сообщение для пользователя

gameOver() – метод, которые определяет «конец» игры и состояние игрового окна в этот момент

doClickSquare() – метод, который проверяет ячейку на заполненность. В случае её заполненности, выводит пользователю сообщение об ошибке

winner() – класс, в котором определяется «победное» состояние игры

count() – класс, который отвечает за подсчёт ячеек у каждого игрока

paintCompanent() – класс, который отвечает за изображение игрового поля и линии, которой отмечается выигрышная комбинация

drawPiece() – отображение фигуры в ячейке игрового поля

drawWinLine() – отображение выигрышной линии

gameInProgress – отображает состояние игры (запущена ли игра или нет). Используется для «блокировки» кнопок в зависимости от состояния игры

currentPlayer – отображает игрока, которому принадлежит текущий ход

# Описание способа организации работы программы

В качестве среды разработки была выбрана программа ApacheNetBeabs. Программа предоставляет все средства, необходимые при разработке графического приложения. А именно библиотеку JavaSwing.

GUI Builder в среде IDE разрешает основные проблемы, возникающие при создании графического интерфейса Java путем рационализации процесса создания графических интерфейсов, освобождая разработчиков от необходимости изучения особенностей диспетчеров компоновки Swing. Это выполняется путем расширения возможностей конструктора графического интерфейса пользователя IDE NetBeans для поддержки простой парадигмы "Произвольная структура" с простыми правилами компоновки, понятными и простыми в использовании. В процессе проектирования формы GUI Builder предоставляет визуальные средства поддержки, предлагая оптимальное расположение и выравнивание компонентов. GUI Builder способствует переносу пользовательских решений по разработке в функциональный пользовательский интерфейс, реализуемый при помощи диспетчера компоновки GroupLayout и других средств Swing. Благодаря динамической модели размещения компонентов поведение графического интерфейса в GUI Builder во время выполнения соответствует ожидаемому, что позволяет вносить корректировки без изменения установленных взаимосвязей между компонентами. При каждом изменении размеров форм, переключении локалей или применении нового общего стиля графический интерфейс автоматически изменяется в соответствии с новой настройкой вставок и смещений стиля.

В приложении используется окно Jframe и сам игровой процесс. Окна являются основой пользовательского интерфейса любой операционной системы. Они визуально разделяют выполняемые в среде приложения. Окна, используемые в библиотеке Swing, представляют собой окна операционной системы.

Окно JFrame наследует свойства класса JWindow и представляет собой наиболее часто используемое в приложениях окно «общего назначения». Основные отличия окна JFrame от JWindow:

* наличие рамки, которая позволяет изменять размер окна;
* наличие заголовка с названием приложения (заголовок может быть пустым);
* возможность использования системного меню, позволяющее проводить манипуляции с окном и приложением;
* наличие кнопок управления для закрытия и свертывания окна.

Также были использованы кнопки (jButton) новая игра, сдаться и выход.

Библиотека Swing включает абсолютно все придуманные на сегодняшний день элементы управления. К ним относятся кнопки, флажки, переключатели, меню и его элементы, и многое другое. Все эти элементы в библиотеке связаны, поскольку они унаследованы от абстрактного класса AbstractButton, определяющего поведение любого компонента, претендующего на звание элемента управления.Кнопки JButton кроме собственного внешнего вида не включают практически ничего уникального. Поэтому всё, что верно для кнопок, будет верно и для остальных элементов управления. Основное время работы с кнопками связано не столько с их созданием и настройкой, сколько с размещением в контейнере и написанием обработчиков событий. Внешний вид кнопок JButton можно легко изменить, не меняя менеджера внешнего вида и поведения. С интерфейсом кнопок можно делать практически все — сопоставлять каждому действию пользователя своё изображение, убирать рамку, закрашивать в любой цвет, перемещать содержимое по разным углам.

Окно игрового процесса содержит в себе игровое поле, кнопки и строку с состоянием игры. В момент запуска игры на сервер поступает информация об этом и записывает это. Каждый ход игрока отображается на сервере вместе с итоговым результатом игры (победа или ничья). Когда пользователь нажимает кнопку «выход», тем самым закрывая приложение, на сервер поступает информация об этом и он отключает пользователя, а затем и выключается сам.

# Эксперементальная часть

Для начала необходимо запустить сервер.

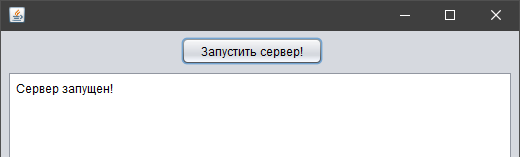


Рисунок 1 – Запуск сервера.

Далее запускаем клиент.

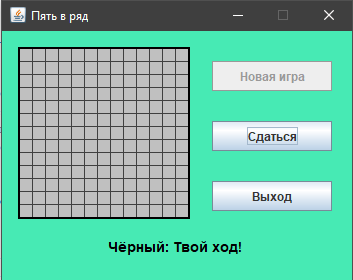


Рисунок 2 – GUI игрового окна.

Сервер получил информацию о подключении пользователя и начале игрового сеанса.

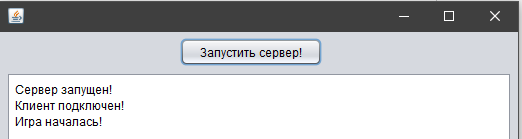


Рисунок 3 – Информация на сервере

В ходе игры каждый игрок выбирает клетку, в которую он хочет поместить свою фигуру. Надпись под игровым полем сообщает, какой из игроков должен совершать ход.

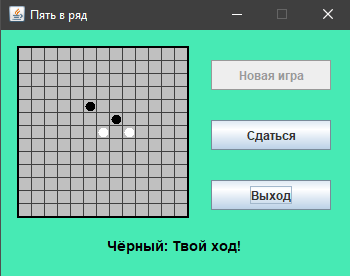


Рисунок 4 – Игровой процесс

Если один из игроков нажимает на ячейку, которая уже занята фигурой, приложение сообщает, что нужно выбрать свободную ячейку.

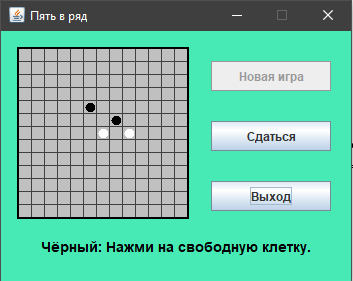


Рисунок 5 – Сообщение о неверном ходе

После того, как одному из игроков удаётся выставить 5 или более своих фигур в ряд (во всех направлениях), выигранная комбинация помечается красной линеей и появляется информация о победе.



Рисунок 6 – Победа одного из игроков

В ходе игры на сервере отображается информация о ходе каждого игрока и информация о победе.

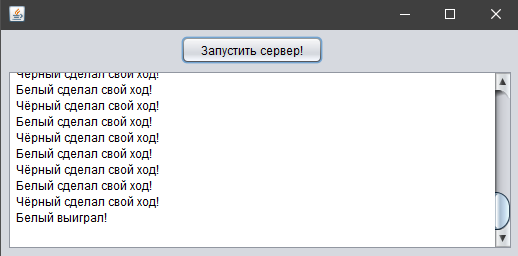


Рисунок 7 – Информация на сервере.

По окончании игры пользователь может начать новую игру или выйти из приложения. В процессе игры один из игроков может сдаться, нажав на соответствующую кнопку. В этом случае победа засчитается противнику, и эта информация отобразится на сервере.



Рисунок 8 – Один из игроков сдаётся.

В случае, если все ячейки игрового поля заполнены и победитель не был определен, игра заканчивается «ничьей». Эта информация также отображается на сервере.

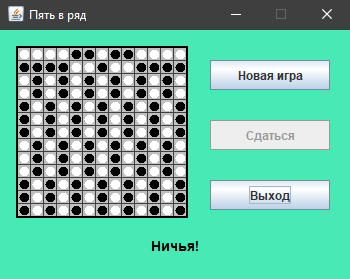


Рисунок 9 – Результат игры «Ничья»

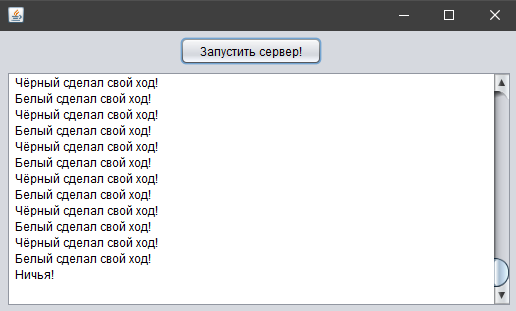


Рисунок 10 – Результат игры «Ничья» на сервере

В случае выхода клиента из приложения сервер закрывает соединение.

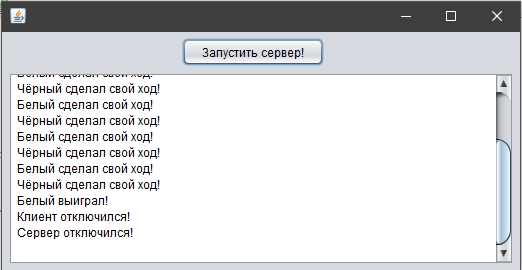


Рисунок 11 – Закрытие соединения.

# Заключение

В ходе выполнения данной курсовой работы были получены навыки разработки программ на языке высокого уровня. Были освоены приемы создания графического интерфейса. Усвоены механизмы реализации меню. Были закреплены знания и приобретены практические навыки разработки многомодульных приложений на языке Java. Изучены основные возможности среды программирования NetBeans.

# Список литературы

1) Герберт Шилдт. «Java. Руководство для начинающих. Современные методы создания, компиляции и выполнения программ на Java»

2) Джошуа Блох. «Java. Эффективное программирование»

3) Б. Эванс, Д. Флэнаган. «Java. Справочник разработчика»  
4) Джитер Кевин Уэйн, Седжвик Роберт. «Алгоритмы на Java»

5) Роберт Сэдгевик, Кевин Уэйн. «Алгоритмы на Java 4 — е издание»

6) [Орфали Роберт](https://www.labirint.ru/authors/104748/), [Харки Дэн](https://www.labirint.ru/authors/104749/) «JAVA и CORBA. В приложениях клиент-сервер»

7) [Мухамедзянов Р. Р.](https://biblioclub.ru/index.php?page=author_red&id=77483) «JAVA. Серверные приложения: практическое пособие»

Приложение А «Листинг клиентской части приложения»

Gomoku.java

package com.mycompany.kursa;

import java.awt.\*;

import java.awt.event.\*;

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.logging.Level;

import java.util.logging.Logger;

import javax.swing.\*;

/\*\*

\* Эта панель позволяет двум пользователям играть в Го Моку против друг

\* друга. Черные всегда начинают игру. Когда у игрока получается пять фигур подряд,

\* этот игрок выигрывает. Игра заканчивается ничьей, если доска заполнена

\* до победы любого из игроков.

\*/

public class Gomoku extends JPanel {

private static Socket clientSocket; //сокет для общения

private static BufferedReader in; // поток чтения из сокета

private static BufferedWriter out; // поток записи в сокет

/\*\*

\* Основная процедура позволяет запускать GoMoku как

\* отдельное приложение. Открывается окно с игрой; программа

\* заканчивается, когда пользователь закрывает окно.

\*/

public static void main(String[] args) {

try {

clientSocket = new Socket("localhost", 4004); // этой строкой мы запрашиваем

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

JFrame window = new JFrame("Пять в ряд");

Gomoku content = new Gomoku();

window.setContentPane(content);

window.pack();

Dimension screensize = Toolkit.getDefaultToolkit().getScreenSize();

window.setLocation( (screensize.width - window.getWidth())/2,

(screensize.height - window.getHeight())/2 );

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

window.setResizable(false);

window.setVisible(true);

} catch (IOException ex) {

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

}

}

private JButton newGameButton; // Кнопка для новой игры.

private JButton resignButton; // Кнопка, которая позволяет игроку сдаться.

private JButton endGameButton;

private JLabel message; // Отображение сообщения пользователю.

public Gomoku() {

setLayout(null); // Создание игрового поля

setPreferredSize( new Dimension(350,250) );

setBackground(new Color(71,234,180));

/\* Создание компонентов и отображение их в панели \*/

Board board = new Board();

add(board);

add(newGameButton);

add(resignButton);

add(endGameButton);

add(message);

board.setBounds(16,16,172,172);

newGameButton.setBounds(210, 30, 120, 30);

resignButton.setBounds(210, 90, 120, 30);

endGameButton.setBounds(210, 150, 120, 30);

message.setBounds(0, 200, 350, 30);

}

// ----------------------- Вложенный класс -----------------------------------

class Board extends JPanel implements ActionListener, MouseListener {

int[][] board;

static final int EMPTY = 0, // Пустая клетка.

WHITE = 1, // Белая фигура.

BLACK = 2; // Чёрная фигура.

boolean gameInProgress; // Запущена ли игра

int currentPlayer; // Чей ход. Возможные значения черный или белый

// Работает только если игра запущена

int win\_r1, win\_c1, win\_r2, win\_c2; // Когда игрок получает 5 или более

// фигур подряд, ячейки определяются

// (win\_r1,win\_c1) и (win\_r2,win\_c2)

// Красная линия рисуется между этими

// ячейками. Если 5 фигур подряд нет

// значение win\_r1 устанавливается -1.

//

/\*\*

\* Конструктор. Создает кнопки и метки. Реагирует на

\* нажатие мыши. Создаёт игровое поле и начинает игру.

\*/

public Board() {

setBackground(Color.LIGHT\_GRAY);

addMouseListener(this);

resignButton = new JButton("Сдаться");

resignButton.addActionListener(this);

newGameButton = new JButton("Новая игра");

newGameButton.addActionListener(this);

endGameButton = new JButton("Выход");

endGameButton.addActionListener(this);

message = new JLabel("",JLabel.CENTER);

message.setFont(new Font("Bahnschrift SemiBold", Font.BOLD, 14));

message.setForeground(Color.BLACK);

board = new int[13][13];

doNewGame();

}

/\*\*

\* Реагирует на нажатие одной из кнопок пользователем.

\*/

public void actionPerformed(ActionEvent evt) {

Object src = evt.getSource();

if (src == newGameButton)

doNewGame();

else if (src == resignButton)

doResign();

else if (src == endGameButton){

try {

send("exit");

clientSocket.close();

in.close();

out.close();

System.exit(0);

} catch (IOException ex) {

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

}

}

}

/\*\*

\* Начало новой игры;

\* Когда пользователь нажимает "Новая игра".

\*/

void doNewGame() {

send("StartGame");

for (int row = 0; row < 13; row++) // Заполняет игровое поле пустыми ячейками

for (int col = 0; col < 13; col++)

board[row][col] = EMPTY;

currentPlayer = BLACK; // Чёрный ходит первым.

message.setText("Чёрный: Твой ход! ");

gameInProgress = true;

newGameButton.setEnabled(false);

resignButton.setEnabled(true);

win\_r1 = -1;

repaint();

}

// Текущий игрок сдаётся;

void doResign() {

if (currentPlayer == WHITE){

message.setText("Белый сдаётся. Чёрный победил!");

send("BlackWin");

}

else{

message.setText("Чёрный сдаётся. Белый победил!");

send("WhiteWin");

}

newGameButton.setEnabled(true);

resignButton.setEnabled(false);

gameInProgress = false;

}

/\*\*

\* Когда игра заканчивается. Параметр str

\* отображается как сообщение, и кнопки неактивны.

\* Игра не запущена.

\*/

void gameOver(String str) {

message.setText(str);

newGameButton.setEnabled(true);

resignButton.setEnabled(false);

gameInProgress = false;

}

void doClickSquare(int row, int col) {

/\* Проверка клетки, если пользователь нажал на заполненую

клетку, выдает сообщение об ошибке

\*/

if ( board[row][col] != EMPTY ) {

if (currentPlayer == BLACK)

message.setText("Чёрный: Нажми на свободную клетку.");

else

message.setText("Белый: Нажми на свободную клетку.");

return;

}

/\* Проверка заполнена ли всё игровое поле или сдедующий ход

выигрышный. Если так, то игра заканчивается. Если нет,

Ход переходит к другому игроку \*/

board[row][col] = currentPlayer; // Сделать ход

repaint();

if (winner(row,col)) { // Проверка на победителя

if (currentPlayer == WHITE){

gameOver("Белый победил!");

send("WhiteWin");

}

else{

gameOver("Чёрный победил!");

send("BlackWin");

}

return;

}

boolean emptySpace = false; // Проверка на заполненность игрового поля.

for (int i = 0; i < 13; i++)

for (int j = 0; j < 13; j++)

if (board[i][j] == EMPTY)

emptySpace = true;

if (emptySpace == false) {

gameOver("Ничья!");

send("draw");

return;

}

/\* Продолжение игры. Ход следующего игрока \*/

if (currentPlayer == BLACK) {

currentPlayer = WHITE;

send("BlackStep");

message.setText("Белый: Твой ход! ");

}

else {

currentPlayer = BLACK;

send("WhiteStep");

message.setText("Чёрный: Твой ход! ");

}

}

/\*\*

\* Вызывается после каждого хода игрока

\* Определяет, выигрышный ли ход подсчитывая количество

\* заполненных ячеек в линии во всех направлениях

\* (row,col). Если заполнено 5 или более

\* ячеек подряд, то игра заканчивается

\*/

private boolean winner(int row, int col) {

if (count( board[row][col], row, col, 1, 0 ) >= 5)

return true;

if (count( board[row][col], row, col, 0, 1 ) >= 5)

return true;

if (count( board[row][col], row, col, 1, -1 ) >= 5)

return true;

if (count( board[row][col], row, col, 1, 1 ) >= 5)

return true;

/\*Игра не выиграна\*/

win\_r1 = -1;

return false;

}

private int count(int player, int row, int col, int dirX, int dirY) {

int ct = 1; //Количество ячеек подряд у игрока.

int r, c;

r = row + dirX;

c = col + dirY;

while ( r >= 0 && r < 13 && c >= 0 && c < 13 && board[r][c] == player ) {

// Ячейка игрового поля заполнена одним из игроков

ct++;

r += dirX; //Переместиться к следующей ячейке

c += dirY;

}

win\_r1 = r - dirX;

win\_c1 = c - dirY; // Последняя просмотренная ячейка вне игрового поля

// или не содержит фигур ни одного из игроков.

r = row - dirX;

c = col - dirY;

while ( r >= 0 && r < 13 && c >= 0 && c < 13 && board[r][c] == player ) {

// Ячейка в игровом поле и содержит фигуру одного из игроков.

ct++;

r -= dirX; // Переместиться к следующей ячейке

c -= dirY;

}

win\_r2 = r + dirX;

win\_c2 = c + dirY;

// В этом случае, (win\_r1,win\_c1) и (win\_r2,win\_c2) отмечают конечные точки

// линии с фигурой одного из игроков.

return ct;

}

/\*\*

\* Изображение игрового поля и фигур. Если игра выиграна,

\* рисует красную линию по фигурам

\*/

public void paintComponent(Graphics g) {

super.paintComponent(g);

g.setColor(Color.DARK\_GRAY);

for (int i = 1; i < 13; i++) {

g.drawLine(1 + 13\*i, 0, 1 + 13\*i, getSize().height);

g.drawLine(0, 1 + 13\*i, getSize().width, 1 + 13\*i);

}

g.setColor(Color.BLACK);

g.drawRect(0,0,getSize().width-1,getSize().height-1);

g.drawRect(1,1,getSize().width-3,getSize().height-3);

/\* Фигура в игровом поле \*/

for (int row = 0; row < 13; row++)

for (int col = 0; col < 13; col++)

if (board[row][col] != EMPTY)

drawPiece(g, board[row][col], row, col);

/\* Если игры была выиграна, win\_r1 >= 0. Рисует красную линию

по выигрышным фигурами \*/

if (win\_r1 >= 0)

drawWinLine(g);

}

/\*\*

\* Отображение фигуры в ячейке (row,col). Цвет выбирается по

\* параметру, который установлен как BLACK или WHITE.

\*/

private void drawPiece(Graphics g, int piece, int row, int col) {

if (piece == WHITE)

g.setColor(Color.WHITE);

else

g.setColor(Color.BLACK);

g.fillOval(3 + 13\*col, 3 + 13\*row, 10, 10);

}

// Рисует красную линию из (win\_r1,win\_c1) в (win\_r2,win\_c2)

private void drawWinLine(Graphics g) {

g.setColor(Color.RED);

g.drawLine( 8 + 13\*win\_c1, 8 + 13\*win\_r1, 8 + 13\*win\_c2, 8 + 13\*win\_r2 );

if (win\_r1 == win\_r2)

g.drawLine( 8 + 13\*win\_c1, 7 + 13\*win\_r1, 8 + 13\*win\_c2, 7 + 13\*win\_r2 );

else

g.drawLine( 7 + 13\*win\_c1, 8 + 13\*win\_r1, 7 + 13\*win\_c2, 8 + 13\*win\_r2 );

}

/\*\*

\* Реагирует на нажатие игрока по игровому полю.

\* Если игра не запущена, выдаёт ошибку. Иначе ищет строку и столбец

\* куда нажал игрок и вызывает doClickSquare()

\*/

public void mousePressed(MouseEvent evt) {

if (gameInProgress == false)

message.setText("Нажми \"Новая игра\" чтобы начать игру.");

else {

int col = (evt.getX() - 2) / 13;

int row = (evt.getY() - 2) / 13;

if (col >= 0 && col < 13 && row >= 0 && row < 13)

doClickSquare(row,col);

}

}

private void send(String msg) {

try {

out.write(msg + "\n");

out.flush();

} catch (IOException ignored) {}

}

public void mouseReleased(MouseEvent evt) { }

public void mouseClicked(MouseEvent evt) { }

public void mouseEntered(MouseEvent evt) { }

public void mouseExited(MouseEvent evt) { }

}

}

Приложение Б «Листинг серверной части приложения»

Server.java

package com.mycompany.servergomoku;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author verba

\*/

public class Server extends javax.swing.JFrame {

/\*\*

\* Creates new form Server

\*/

public Server() {

initComponents();

}

private static Socket clientSocket; //сокет для общения

private static ServerSocket server; // серверсокет

private static BufferedReader in; // поток чтения из сокета

private static BufferedWriter out; // поток записи в сокет

String command;

Thread thread = new Thread() {

@Override

public void run() {

try {

server = new ServerSocket(4004); // серверсокет прослушивает порт 4004

try {

sleep(1);

} catch (InterruptedException ex) {

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

}

jTextArea1.append("Сервер запущен!\n");

clientSocket = server.accept();

jTextArea1.append("Клиент подключен!\n");

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

while (true){

get();

if("exit".equals(command)){

jTextArea1.append("Клиент отключился!\n");

clientSocket.close();

in.close();

out.close();

break;

}

if("StartGame".equals(command)){jTextArea1.append("Игра началась!\n");}

if("BlackWin".equals(command)){jTextArea1.append("Чёрный выиграл!\n");}

if("WhiteWin".equals(command)){jTextArea1.append("Белый выиграл!\n");}

if("BlackStep".equals(command)){jTextArea1.append("Чёрный сделал свой ход!\n");}

if("WhiteStep".equals(command)){jTextArea1.append("Белый сделал свой ход!\n");}

if("draw".equals(command)){jTextArea1.append("Ничья!\n");}

}

jTextArea1.append("Сервер отключился!\n");

server.close();

}

catch (IOException e)

{

System.err.println(e);

}

}

};

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jButton1 = new javax.swing.JButton();

jScrollPane1 = new javax.swing.JScrollPane();

jTextArea1 = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jButton1.setText("Запустить сервер!");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jTextArea1.setColumns(20);

jTextArea1.setRows(5);

jScrollPane1.setViewportView(jTextArea1);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1)

.addContainerGap())

.addGroup(layout.createSequentialGroup()

.addGap(180, 180, 180)

.addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 142, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap(196, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jButton1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 374, Short.MAX\_VALUE)

.addContainerGap())

);

pack();

}// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

thread.start();

}

private void send(String msg) {

try {

out.write(msg + "\n");

out.flush();

} catch (IOException ignored) {}

}

private void get() {

try {

command = in.readLine();

} catch (IOException ignored) {}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

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

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

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new Server().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTextArea jTextArea1;

// End of variables declaration

}

Приложение В «UML диаграммы»

Диаграмма вариантов использования

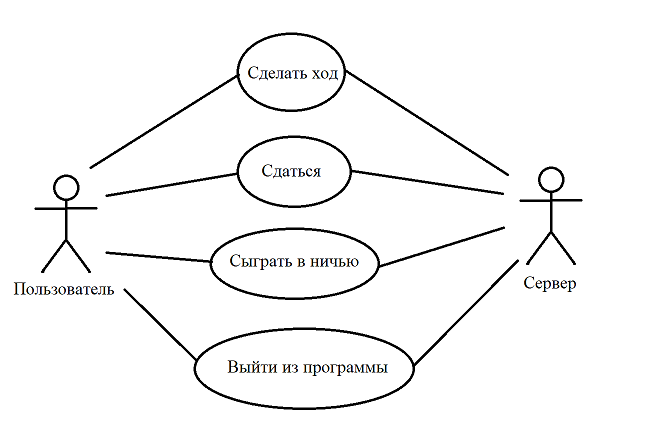


Рисунок В1 – UML диаграмма вариантов использования.

Диаграммы классов

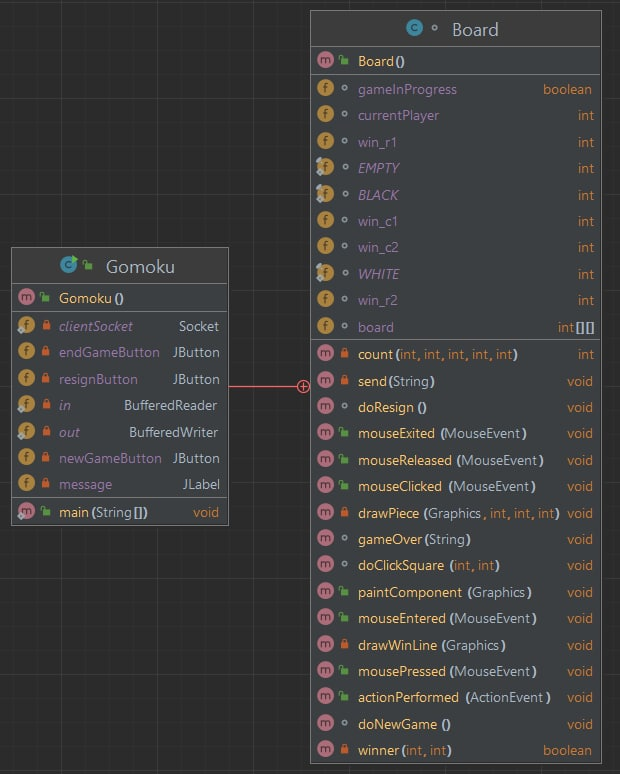


Рисунок В2 – UML диаграмма классов Клиента*.*



Рисунок В3 – UML диаграмма классов Сервера.

Диаграмма деятельности

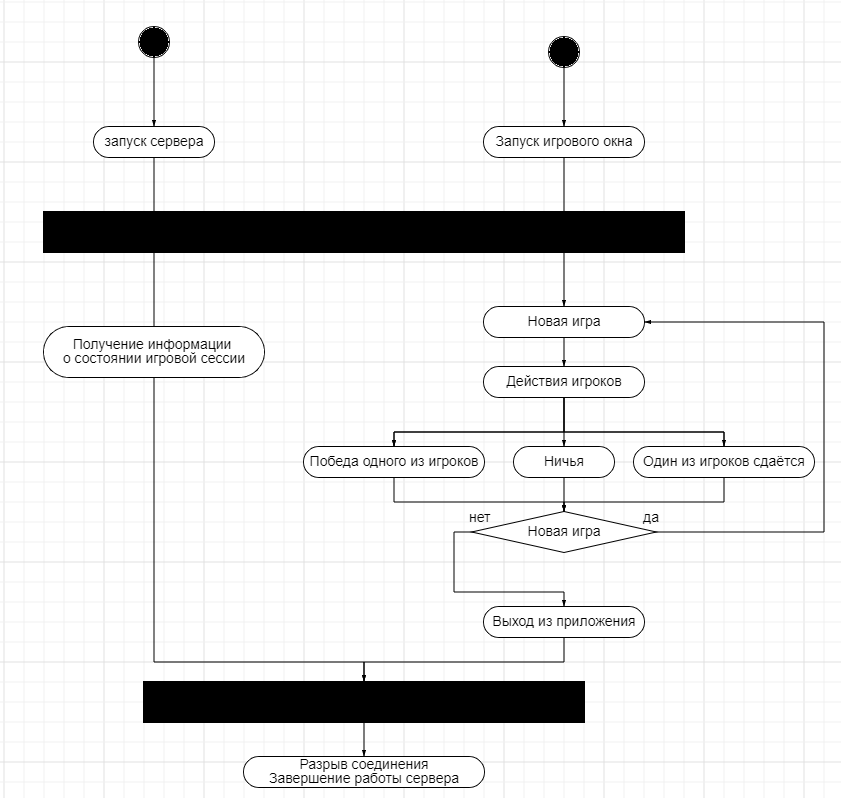


Рисунок В4 – UML диаграмма деятельности.

Диаграмма развёртывания

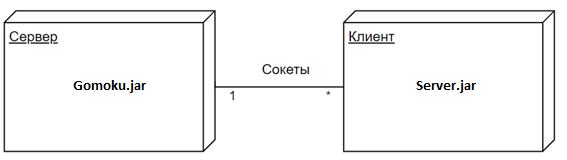


Рисунок В5 – UML диаграмма развертывания.

Диаграмма последовательности

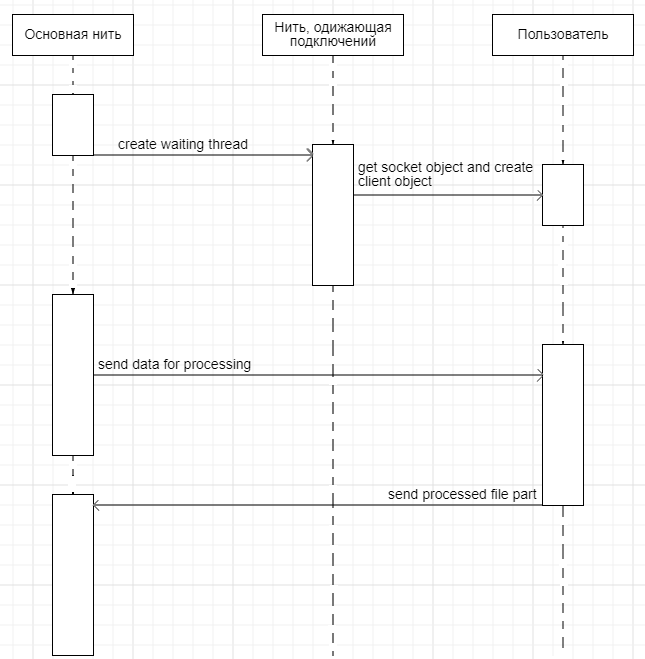


Рисунок В6 – UML диаграмма последовательности.