

**Содержание**

[**Введение** 4](#_Toc136939478)

[**Постановка задачи** 5](#_Toc136939479)

[**Выбор решения** 6](#_Toc136939480)

[**Описание программы** 7](#_Toc136939481)

[**Описание способа организации пользовательского интерфейса** 8](#_Toc136939482)

[**Описание результатов работы программы** 9](#_Toc136939483)

[**Заключение:** 12](#_Toc136939484)

[**Список литературы** 13](#_Toc136939485)

[**Приложение А** 14](#_Toc136939486)

[**UML – диаграммы** 26](#_Toc136939487)

# **Введение**

Тетрис – игра покорившая детство каждого, кто был рождён не позже 2005. Была разработана советским программистом Алексеем Пажитновым. «Тетрис» представляет собой головоломку, построенную на использовании геометрических фигур «тетрамино» — разновидности полимино, состоящих из четырёх квадратов. Случайные фигурки тетрамино падают сверху в прямоугольный стакан шириной 10 и высотой 20 клеток. В полёте игрок может поворачивать фигурку на 90° и двигать её по горизонтали. Также можно «сбрасывать» фигурку, то есть ускорять её падение, когда уже решено, куда фигурка должна упасть. Фигурка летит до тех пор, пока не наткнётся на другую фигурку либо на дно стакана. Если при этом заполнился горизонтальный ряд из 10 клеток, он пропадает и всё, что выше него, опускается на одну клетку.

В качестве среды разработки была выбрана среда IntellijIDEA, язык программирования – JAVA.

Целью данной курсовой работы является разработка программы на языке JAVA.

# **Постановка задачи**

Разработать игру «Тетрис», на доске размером 10 клеток на 20, 7 видов фигур, счётчик очков, используя среду IntellijIDEA и язык программирования JAVA на ОС WINDOWS. Для создания графического интерфейса приложения используется библиотека javax.swing, а также некоторые другие возможности языка JAVA, такие как:  
1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. Java Stream API

4. Java Multithreading

5. Сетевое взаимодействие.

# **Выбор решения**

Тетрис игра на набирание очков, вся суть – это собирать ряд из кирпичиков. Для этого нам необходимо создать формы для всех кирпичиков, алгоритм для управления кирпичиками, а так же поле игры и подсчёт очков. Подсчёт очков реализуем на сервере, а клиентская часть программы будет полностью обрабатывать всё остальное.

# **Описание программы**

Для реализации этой игры нам понадобится массив фигур, координаты которых необходимо задать вручную. Данная программа реализована на соединении между процессами через протокол UDP. При запуске программы сразу появляется фигура которой можно управлять при помощи клавиш стрелочек, клавиша “ENTER” – пауза, “SPACE” – мгновенное падение фигуры. Пока фигура не упала, следующая появится не может. Как только упавшие фигуру образуют сплошную линию отправляется запрос на сервер с указанием глубины линии, это будет влиять на количество очков возвращаемых сервером. Игра идёт бесконечно пока игрок не проиграет.

Main.java – главный файл в котором находятся все классы игры Tetris.

Tetris\_final – главный класс игры.

Board – класс создающий поле игры.

Shape – отвечает за создание, перемещение и появление фигур.

Tetrominoe - массив в котором хранятся названия фигур

Main – главный класс сервера, создающий его.

Server – класс сервера отвечающий за обработку запроса и отправку ответа обратно клиенту.

Connect - связующий класс необходимый для соединения клиента с сервером

# **Описание способа организации пользовательского интерфейса**

Main.java – главный файл в котором находятся все классы игры Tetris.

Tetris\_final – главный класс игры.

Board – класс создающий поле игры.

Shape – отвечает за создание, перемещение и появление фигур.

Tetrominoe - массив в котором хранятся названия фигур

Main – главный класс сервера, создающий его.

Server – класс сервера отвечающий за обработку запроса и отправку ответа обратно клиенту.

Connect - связующий класс необходимый для соединения клиента с сервером

Графический интерфейс был реализован при помощи java swing. Библиотека Graphics нужна для раскраски и обработки фигур и доски.

# **Описание результатов работы программы**

Среда разработки IntellijIDEA предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием методов.

Ниже продемонстрирован результат тестирования программы

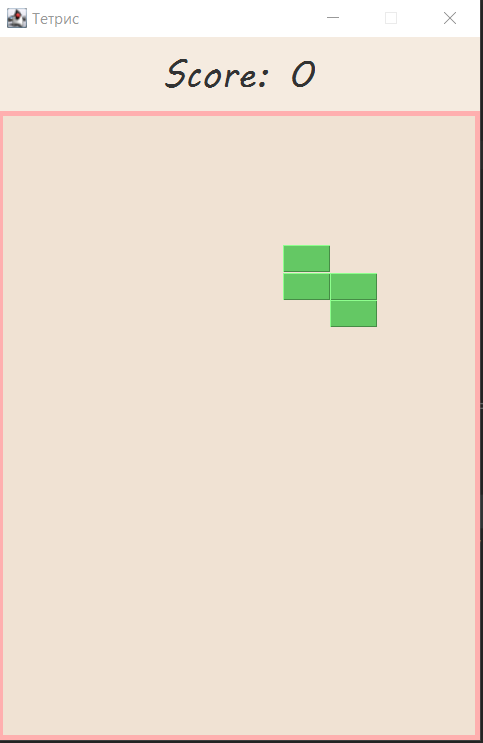


Рисунок 1.Запуск программы

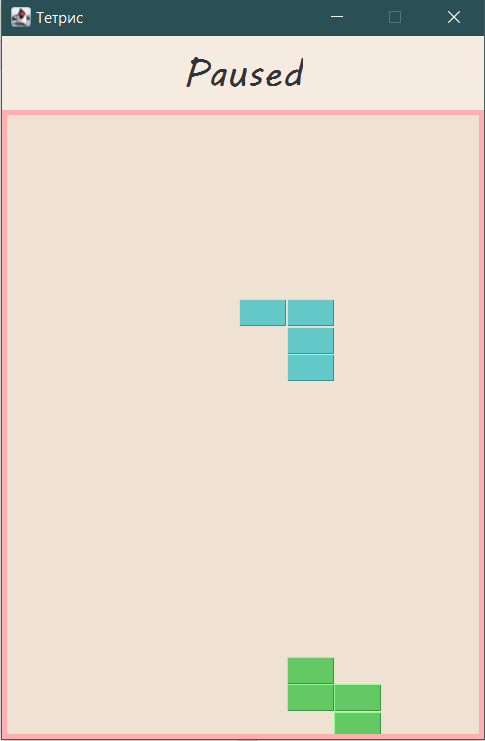


Рисунок 2. Кнопка "Пауза"

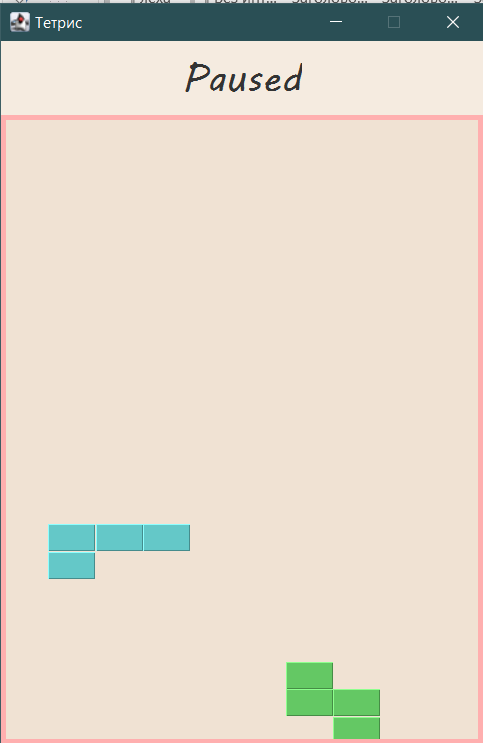


Рисунок 3.Перемещение и поворот фигуры

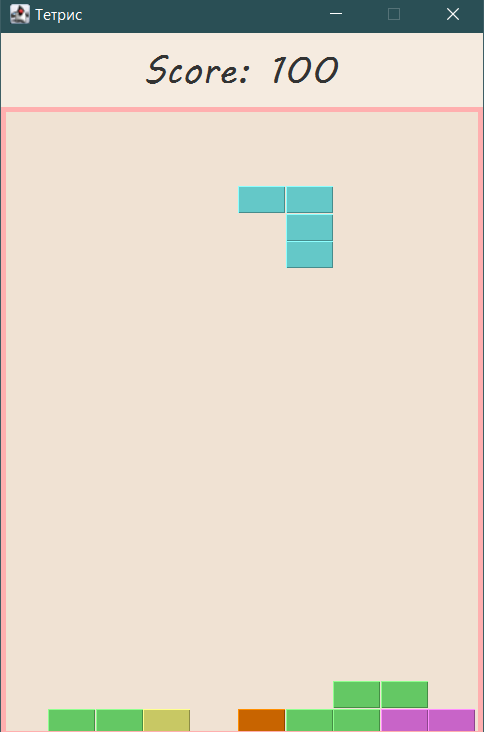


Рисунок 4. Считывание строки и увеличение результата

Таблица 1. Описание поведения программы при тестировании.

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Открытие формы и появление первой фигуры. | Верно |
| Кнопка «Пауза» | Нажатие клавиши «Enter», а затем застывание экрана | Верно |
| Нажатие клавиш перемещения и поворота фигур | Управление фигуркой в поле формы. | Верно |
| Возможность перемещения фигур у грани формы | Невозможность перевернуть фигуру у стен формы | Верно |
| Считывание последней строки и увеличение результата | Успешное выполнение, вывод количества полученных очков | Верно |

# **Заключение:**

Таким образом, в процессе создания данного проекта разработана программа, реализующая игру «Тетрис», в IntellijIDEA.

При выполнении данной курсовой работы были улучшены навыки разработки программ и углублены знания языка программирования JAVA.

# **Список литературы**

1. "Java. Руководство для начинающих", Герберт Шилдт
2. Интернет – ресурс javarush.com
3. Интернет – ресурс Docs.oracle.com
4. Интернет – ресурс Java – online.ru

# **Приложение А**

**Листинг программы**

**Main.java(Client)**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.IOException;  
import java.util.\*;  
import java.util.Timer;  
  
// это класс для всех форм и их поворотов  
class Shape {  
 Tetrominoe pieceShape; // форма фигуры  
 int coords[][]; // координаты фигуры  
 int[][][] coordsTable; // координаты фигуры во всех ее поворотах  
 public Shape()  
 {  
 initShape(); // инициализировать форму  
 }  
 void initShape()  
 {  
 coords = new int[4][2]; // инициализировать размер куска  
 setShape(Tetrominoe.NoShape); // установите форму на NoShape  
 }  
  
 // установить форму фигуры в заданную форму и установить координаты фигуры в координаты заданной формы  
 protected void setShape(Tetrominoe shape) {  
  
 coordsTable = new int[][][]{  
 { { 0, 0 }, { 0, 0 }, { 0, 0 }, { 0, 0 } },  
 { { 0, -1 }, { 0, 0 }, { -1, 0 }, { -1, 1 } },  
 { { 0, -1 }, { 0, 0 }, { 1, 0 }, { 1, 1 } },  
 { { 0, -1 }, { 0, 0 }, { 0, 1 }, { 0, 2 } },  
 { { -1, 0 }, { 0, 0 }, { 1, 0 }, { 0, 1 } },  
 { { 0, 0 }, { 1, 0 }, { 0, 1 }, { 1, 1 } },  
 { { -1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } },  
 { { 1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } }  
 };  
  
 for (int i = 0; i < 4 ; i++) { // для количества строк в форме  
  
 for (int j = 0; j < 2; ++j) { // для количества столбцов в форме  
  
 coords[i][j] = coordsTable[shape.ordinal()][i][j]; // установить координаты фигуры в координаты заданной формы  
 }  
 }  
  
 pieceShape = shape;  
 }  
  
 void setX(int index, int x) { coords[index][0] = x; } // установить координату x куска  
 void setY(int index, int y) { coords[index][1] = y; } // установить координату y куска  
 public int x(int index) { return coords[index][0]; } // получить координату x куска  
 public int y(int index) { return coords[index][1]; } // получить координату y куска  
 public Tetrominoe getShape() { return pieceShape; } // получить форму куска  
  
 // установить форму куска в случайную форму каждый раз, когда создается новый кусок  
 public void setRandomShape() {  
 Random r = new Random(); // создать случайный объект  
 int x = Math.abs(r.nextInt()) % 7 + 1; // получить случайное число от 1 до 7  
 Tetrominoe[] values = Tetrominoe.values(); // получить все формы  
 setShape(values[x]); // установить форму куска в случайную форму  
 }  
  
 // получить минимальную координату x куска  
 public int minX() {  
  
 int m = coords[0][0];  
  
 for (int i=0; i < 4; i++) { // для количества строк в форме  
  
 m = Math.min(m, coords[i][0]); // получить минимальную координату x куска  
 }  
  
 return m;  
 }  
  
  
 // получить минимальную координату у куска  
 public int minY() {  
  
 int m = coords[0][1];  
  
 for (int i=0; i < 4; i++) {  
  
 m = Math.min(m, coords[i][1]);  
 }  
 return m; //  
 }  
 // повернуть фигуру влево  
 public Shape rotateLeft() {  
 if (pieceShape == Tetrominoe.SquareShape) // если фигура квадратная  
 return this; // вернуть фигуру без поворота  
 Shape result = new Shape(); // создать новую фигуру  
 result.pieceShape = pieceShape; // установите форму новой фигуры в форму старой  
 for (int i = 0; i < 4; ++i) { // для количества строк в форме  
  
 result.setX(i, y(i)); // устанавливает координату x новой формы на координату y части  
 result.setY(i, -x(i)); //устанавливает координату y новой формы на отрицательную координату x части  
 }  
  
 return result;  
 }  
 // повернуть фигуру вправо  
 public Shape rotateRight() {  
  
 if (pieceShape == Tetrominoe.SquareShape)  
 return this;  
  
 Shape result = new Shape(); // создаем новую форму  
 result.pieceShape = pieceShape; // устанавливаем форму новой формы в соответствии с формой фигуры  
  
 for (int i = 0; i < 4; ++i) {  
  
 result.setX(i, -y(i)); // устанавоивает координату x новой формы на отрицательную координату y части  
 result.setY(i, x(i)); //устанавливает координату y новой формы на координату x части  
 }  
  
 return result;  
 }  
}  
  
enum Tetrominoe { NoShape, ZShape, SShape, LineShape,  
 TShape, SquareShape, LShape, MirroredLShape };  
  
// это класс доски  
class Board extends JPanel {  
  
 static final long serialVersionUID = 1L;  
 final int BOARD\_WIDTH = 10; // ширина  
 final int BOARD\_HEIGHT = 22; // высота  
 final int INITIAL\_DELAY = 100; // начальная задержка таймера  
 final int PERIOD\_INTERVAL = 300; // интервал таймера  
  
 Timer timer;  
 boolean isFallingFinished = false; // проверить, закончил ли падение кусок  
 boolean isStarted = false;  
 boolean isPaused = false;  
 int numLinesRemoved = 0; // количество удаленных строк  
 int curX = 0; // текущая координата x  
 int curY = 0; // текущая координата у  
 JLabel statusbar;  
 Shape curPiece;  
 Tetrominoe[] board;  
  
 public Board(Tetris parent) {  
  
 initBoard(parent);  
 }  
  
 // инициализация доски  
 void initBoard(Tetris parent) {  
 setFocusable(true);  
 setBorder(BorderFactory.createLineBorder(Color.pink, 4));  
 timer = new Timer();  
 timer.scheduleAtFixedRate(new ScheduleTask(),  
 INITIAL\_DELAY, PERIOD\_INTERVAL);  
 curPiece = new Shape();  
 statusbar = parent.getStatusBar();  
 board = new Tetrominoe[BOARD\_WIDTH \* BOARD\_HEIGHT];  
 addKeyListener(new TAdapter());  
 clearBoard();  
 }  
  
 int squareWidth() {  
 return (int) getSize().getWidth() / BOARD\_WIDTH;  
 }  
  
 int squareHeight() {  
 return (int) getSize().getHeight() / BOARD\_HEIGHT;  
 }  
  
 Tetrominoe shapeAt(int x, int y) {  
 return board[(y \* BOARD\_WIDTH) + x];  
 }  
  
 public void start() {  
  
 isStarted = true;  
 clearBoard();  
 newPiece();  
 }  
  
 void pause() {  
  
 if (!isStarted) {  
 return;  
 }  
  
 isPaused = !isPaused;  
  
 if (isPaused) {  
  
 statusbar.setText("Paused");  
 } else {  
  
 statusbar.setText(String.valueOf(numLinesRemoved));  
 }  
 }  
  
 void doDrawing(Graphics g) {  
  
 Dimension size = getSize();  
 int boardTop = (int) size.getHeight() - BOARD\_HEIGHT \* squareHeight();  
 for (int i = 0; i < BOARD\_HEIGHT; ++i)  
 {  
 for (int j = 0; j < BOARD\_WIDTH; ++j)  
 {  
 Tetrominoe shape = shapeAt(j, BOARD\_HEIGHT - i - 1);  
 if (shape != Tetrominoe.NoShape) {  
 drawSquare(g, j \* squareWidth(), boardTop + i \* squareHeight(), shape);  
 }  
 }  
 }  
  
 if (curPiece.getShape() != Tetrominoe.NoShape) {  
  
 for (int i = 0; i < 4; ++i) {  
  
 int x = curX + curPiece.x(i); // получить координату x куска  
 int y = curY - curPiece.y(i); // получить координату y куска  
 drawSquare(g, x \* squareWidth(),  
 boardTop + (BOARD\_HEIGHT - y - 1) \* squareHeight(),  
 curPiece.getShape()); // нарисовать фигуру на доске  
 }  
 }  
 }  
  
 //квадрат  
 @Override  
 public void paintComponent(Graphics g) {  
  
 super.paintComponent(g);  
 doDrawing(g);  
 }  
  
 //новые координаты куска при падении  
 void dropDown() throws IOException, InterruptedException {  
  
 int newY = curY; // установить новую координату y на текущую координату y  
  
 while (newY > 0) {  
  
 if (!tryMove(curPiece, curX, newY - 1)) { // если фигура не может двигаться  
  
 break; // break the loop  
 }  
  
 --newY;  
 }  
  
 pieceDropped(); // вызываем метод падения куска  
 }  
  
 void oneLineDown() throws IOException, InterruptedException {  
  
 if (!tryMove(curPiece, curX, curY - 1)) {  
  
 pieceDropped();  
 }  
 }  
  
 void clearBoard() {  
  
 for (int i = 0; i < BOARD\_HEIGHT \* BOARD\_WIDTH; ++i) {  
 board[i] = Tetrominoe.NoShape;  
 }  
 }  
  
 void pieceDropped() throws IOException, InterruptedException {  
  
 for (int i = 0; i < 4; ++i) {  
  
 int x = curX + curPiece.x(i);  
 int y = curY - curPiece.y(i);  
 board[(y \* BOARD\_WIDTH) + x] = curPiece.getShape();  
 }  
  
 removeFullLines(); // удалить строку  
  
 if (!isFallingFinished) { // если фигура не закончила падение  
 newPiece(); // создать новый  
 }  
 }  
 void newPiece() {  
  
 curPiece.setRandomShape(); // установить в случайную форму  
 curX = BOARD\_WIDTH / 2 + 1; // устанавливает координату x фигуры в середину доски  
 curY = BOARD\_HEIGHT - 1 + curPiece.minY();  
  
 if (!tryMove(curPiece, curX, curY)) { // если фигура не может двигаться в новые координаты  
 curPiece.setShape(Tetrominoe.NoShape); //устанавливает форму куска на no shape  
 timer.cancel();  
 isStarted = false;  
 statusbar.setText("GAME OVER!");  
 }  
 }  
  
 // проверить, может ли фигура двигаться  
 boolean tryMove(Shape newPiece, int newX, int newY) {  
  
 for (int i = 0; i < 4; ++i) {  
  
 int x = newX + newPiece.x(i);  
 int y = newY - newPiece.y(i);  
  
 if (x < 0 || x >= BOARD\_WIDTH || y < 0 || y >= BOARD\_HEIGHT) { // если координаты за пределами доски  
 return false;  
 }  
  
 if (shapeAt(x, y) != Tetrominoe.NoShape) {  
 return false;  
 }  
 }  
  
 curPiece = newPiece; // установить текущую инф на новую инф  
 curX = newX;  
 curY = newY;  
  
 repaint(); // перекрасить доску с новыми координатами фигуры  
  
 return true;  
 }  
 void removeFullLines() throws IOException, InterruptedException {  
  
 int numFullLines = 0; // устанавливаем количество полных строк равным 0  
  
 for (int i = BOARD\_HEIGHT - 1; i >= 0; --i) {  
 boolean lineIsFull = true; // установить строку заполнена в true  
  
 for (int j = 0; j < BOARD\_WIDTH; ++j) {  
  
 if (shapeAt(j, i) == Tetrominoe.NoShape) {  
  
 lineIsFull = false;  
 break;  
 }  
 }  
  
 if (lineIsFull) {  
 ++numFullLines;  
 for (int k = i; k < BOARD\_HEIGHT - 1; ++k) {  
 for (int j = 0; j < BOARD\_WIDTH; ++j) {  
 board[(k \* BOARD\_WIDTH) + j] = shapeAt(j, k + 1); // установить форму по координатам на форму по координатам ниже  
 }  
 }  
 }  
 }  
  
 if (numFullLines > 0) { // если количество полных строк больше 0  
 connect connect = new connect();  
 connect.runServer(numFullLines);  
 numLinesRemoved += connect.getResult();  
 //numLinesRemoved += numFullLines; // увеличить количество удаленных строк на количество полных строк  
 statusbar.setText("Score: "+String.valueOf(numLinesRemoved)); // установить очки на количество удаленных строк  
 isFallingFinished = true;  
 curPiece.setShape(Tetrominoe.NoShape);  
 repaint();  
  
 }  
 }  
  
 void drawSquare(Graphics g, int x, int y,  
 Tetrominoe shape) {  
  
 Color colors[] = {  
 new Color(0, 0, 0), new Color(200, 100, 100),  
 new Color(100, 200, 100), new Color(100, 100, 200),  
 new Color(200, 200, 100), new Color(200, 100, 200),  
 new Color(100, 200, 200), new Color(200, 100, 0),  
  
 };  
 Color color = colors[shape.ordinal()];  
 g.setColor(color);  
 g.fillRect(x + 1, y + 1, squareWidth() - 2, squareHeight() - 2);  
 g.setColor(color.brighter());  
 g.drawLine(x, y + squareHeight() - 1, x, y);  
 g.drawLine(x, y, x + squareWidth() - 1, y);  
  
 g.setColor(color.darker());  
 g.drawLine(x + 1, y + squareHeight() - 1, x + squareWidth() - 1, y + squareHeight() - 1);  
 g.drawLine(x + squareWidth() - 1, y + squareHeight() - 1, x + squareWidth() - 1, y + 1);  
  
 }  
  
 void doGameCycle() throws IOException, InterruptedException {  
 update();  
 repaint();  
 }  
  
 void update() throws IOException, InterruptedException {  
  
 if (isPaused) {  
 return;  
 }  
  
 if (isFallingFinished) {  
  
 isFallingFinished = false;  
 newPiece();  
 } else {  
 oneLineDown(); // переместить фигуру вниз на одну строку  
 }  
 }  
  
 class TAdapter extends KeyAdapter {  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
  
 if (!isStarted || curPiece.getShape() == Tetrominoe.NoShape) { //если игра не началась или фигура не имеет формы  
 return; // return  
 }  
  
 int keycode = e.getKeyCode();  
 if (keycode == KeyEvent.VK\_ENTER) {  
 pause();  
 return;  
 }  
  
 if (isPaused) {  
 return;  
 }  
  
 switch (keycode) {  
 case KeyEvent.VK\_LEFT:  
 tryMove(curPiece, curX - 1, curY);  
 break;  
  
 case KeyEvent.VK\_RIGHT:  
 tryMove(curPiece, curX + 1, curY);  
 break;  
  
 case KeyEvent.VK\_DOWN:  
 tryMove(curPiece.rotateRight(), curX, curY);  
 break;  
  
 case KeyEvent.VK\_UP:  
 tryMove(curPiece.rotateLeft(), curX, curY);  
 break;  
  
 case KeyEvent.VK\_SPACE:  
 try {  
 dropDown();  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 } catch (InterruptedException ex) {  
 throw new RuntimeException(ex);  
 }  
 break;  
  
 case KeyEvent.VK\_D:  
 try {  
 oneLineDown();  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 } catch (InterruptedException ex) {  
 throw new RuntimeException(ex);  
 }  
 break;  
 }  
 }  
 }  
  
 class ScheduleTask extends TimerTask {  
  
 @Override  
 public void run() {  
 try {  
 doGameCycle();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
}  
  
// main  
class Tetris extends JFrame {  
  
 //static final long serialVersionUID = 1L;  
 JLabel statusbar;  
  
 public Tetris() {  
 initUI();  
 }  
 void initUI() {  
 JPanel panel = new JPanel();  
 panel.setBackground(new Color(0XF5EBE0));  
 statusbar = new JLabel("Score: 0");  
 statusbar.setFont(new Font("MV Boli", Font.ROMAN\_BASELINE, 30));  
 panel.add(statusbar, BorderLayout.NORTH);  
  
 Board board = new Board(this);  
 add(panel, BorderLayout.NORTH);  
 add(board);  
 board.setBackground(new Color(0Xf0e2d3));  
 board.start();  
  
 setTitle("Тетрис");  
 setSize(400, 600);  
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
 setResizable(false);  
 setLocationRelativeTo(null);  
 }  
 public JLabel getStatusBar() {  
 return statusbar;  
 }  
 // run  
 public static void main(String[] args) {  
 EventQueue.invokeLater(() -> {  
 Tetris game = new Tetris();  
 game.setVisible(true);  
 });  
 }  
}

**Main.java(serv)**

import java.io.IOException;  
public class Main {  
 public Main() {  
 }  
 public static void main(String[] args) throws IOException, InterruptedException {  
 Server server = new Server();  
 System.out.println("Сервер запущен");  
 while (true) {  
 System.out.println("-----------Ч--М--С--Мс--------Ожидаю...-------------------------------");  
 server.runi();  
 }  
 }  
}

**Connect.java**

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

public class connect {

private int lines = 0;

private double score = 0;

private static int Result;

public int getLines(){

return lines;

}

public connect()

{

this.Result = connect.Result;

}

public double getscore()

{

return score;

}

public void setResult(int result) {

Result = result;

}

public int getResult(){

return Result;

}

public void SendData(int score) throws IOException {

DatagramSocket socket = new DatagramSocket();

byte[] data = (score +", "+ getscore() + " = Score").getBytes();

InetAddress serverAddress = InetAddress.getByName("127.0.0.1");

int serverPort = 12345;

DatagramPacket packet = new DatagramPacket(data, data.length, serverAddress, serverPort);

socket.send(packet);

socket.close();

}

public int ReadData() throws IOException {

DatagramSocket socket = new DatagramSocket(12346);

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

String response = new String(message);

socket.close();

return Integer.valueOf(response);

}

public void runServer(int score) throws IOException, InterruptedException {

SendData(score);

setResult(ReadData());

}

}

**Server.java**import java.io.IOException;

import java.net.\*;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Server {

int score;

double randata;

private String[] stlArray;

public void runi() throws IOException {

ReceiveRequest();

Date date = new Date();

SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter.format(date)+": Данные приняты");

Date date2 = new Date();

SimpleDateFormat formatter2 = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter2.format(date2)+": Счётчик сработал");

score = Integer.valueOf(stlArray[0]);

//randata =Double.valueOf(stlArray[1]);

Calculate();

System.out.println(score +": Счётчик сработал");

SendResponce(score);

Date date1 = new Date();

SimpleDateFormat formatter1 = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter1.format(date1)+": Данные отправлены");

}

private void Calculate() {

Thread t = new Thread();

int n = score;

final int[] res = {0};

Runnable task = new Runnable() {

public void run() {

int localResult = 0;

switch (n)

{

case 1:

localResult +=100;

break;

case 2:

localResult +=300;

break;

case 3:

localResult +=700;

break;

case 4:

localResult +=1500;

break;

}

synchronized(this) {

res[0] = localResult;

}

}

};

t = new Thread(task);

t.start();

try {

t.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

score = res[0];

}

private void SendResponce(int responseData) throws IOException {

DatagramSocket socket = new DatagramSocket();

byte[] data = String.valueOf(responseData).getBytes();

InetAddress clientAddress = InetAddress.getByName("127.0.0.1");

int clientPort = 12346;

DatagramPacket packet = new DatagramPacket(data, data.length, clientAddress, clientPort);

socket.send(packet);

socket.close();

}

public void ReceiveRequest() throws IOException, IOException {

DatagramSocket socket = new DatagramSocket(12345);

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

String requestData = new String(message);

socket.close();

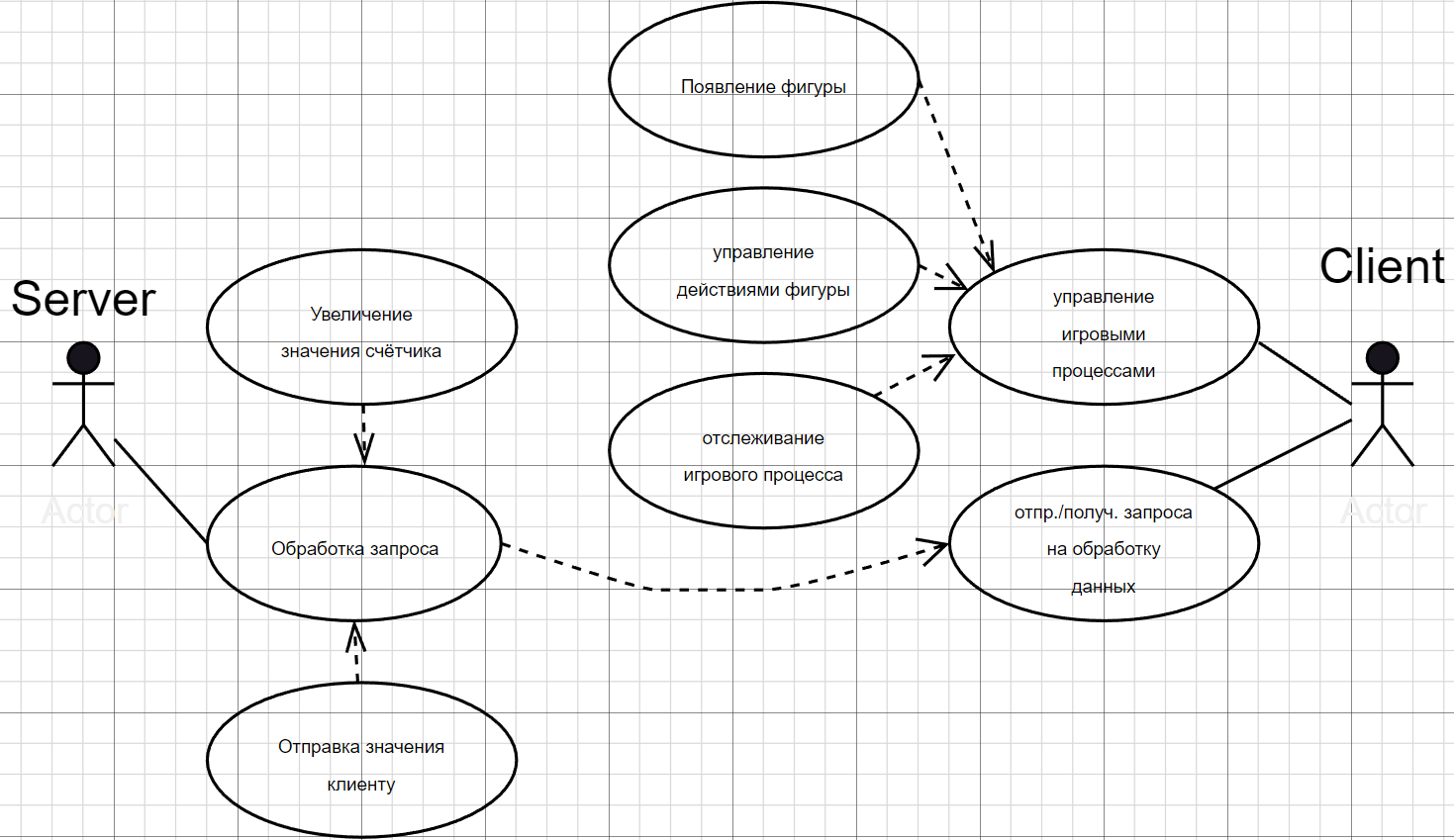
stlArray = requestData.split(",");

}

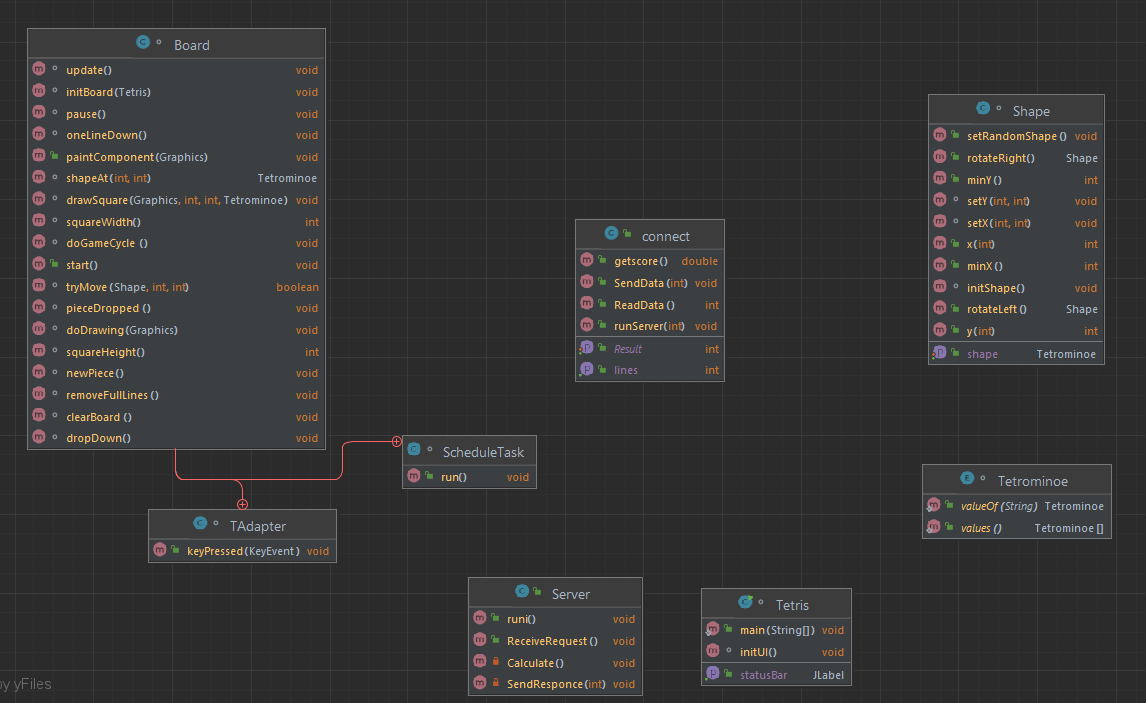
}

# **UML – диаграммы**

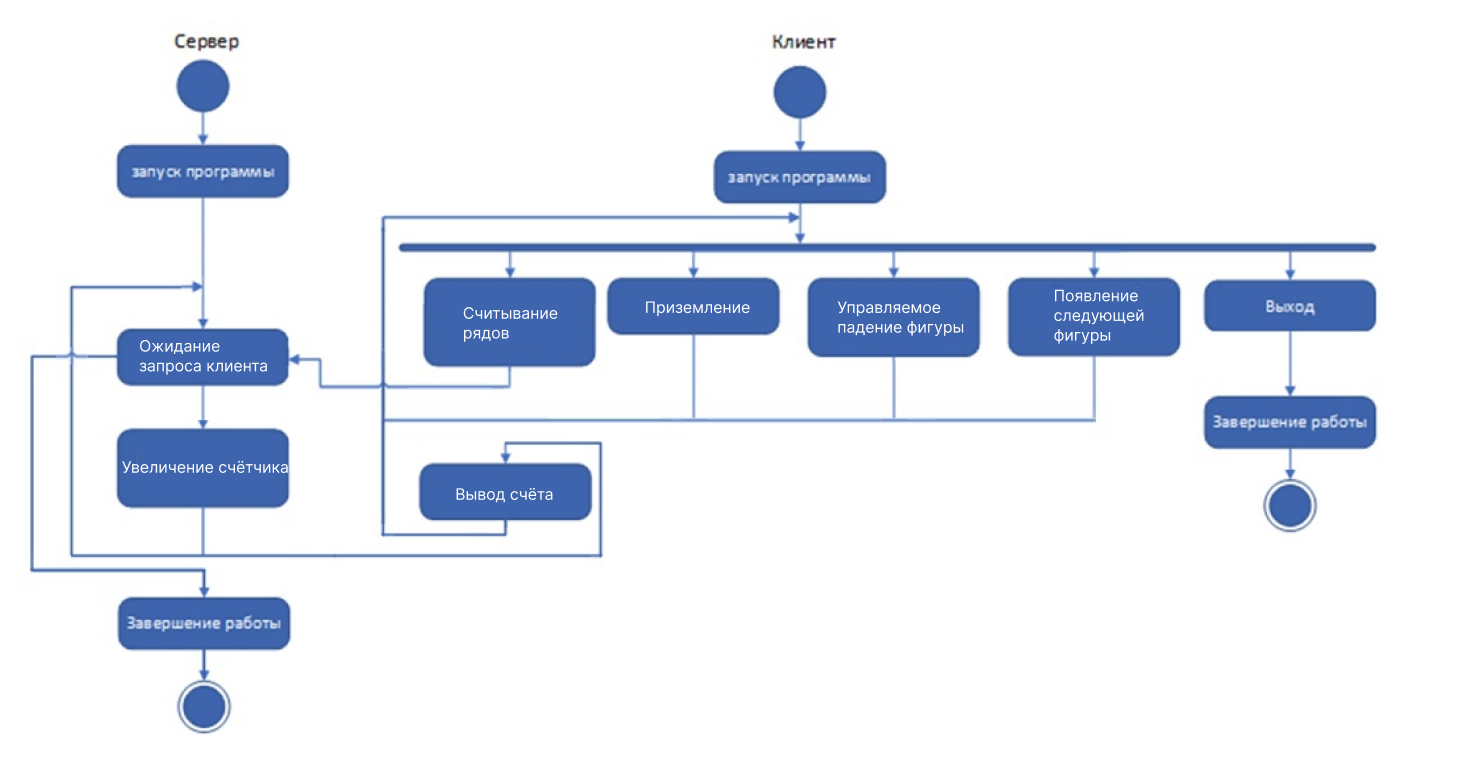
1. диаграмма вариантов использования;



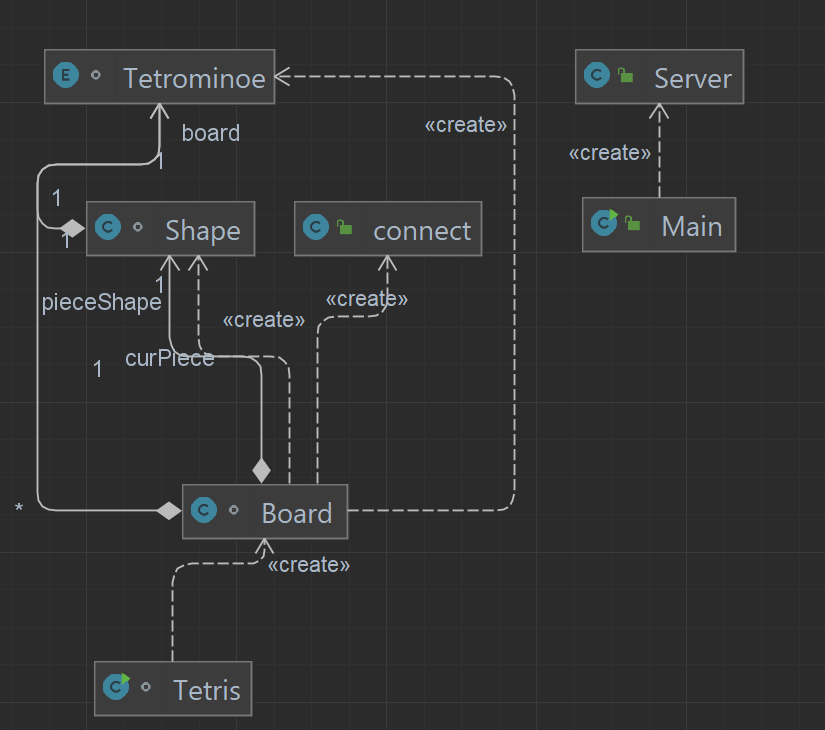
1. диаграмма классов



1. диаграмма деятельности



1. диаграмма развёртывания;



1. диаграмма последовательности.

