МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева (Самарский университет)»

Институт информатики и кибернетики

Кафедра программных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «Объектная распределенная обработка»

по теме «Разработка клиент-серверного приложения «Игра «Тетрис» с использованием технологии servlets (JSP, JSF)»

Выполнил:
Обучающийся группы № 6403-020302D _____ М.А. Мананников

Проверил:
Руководитель работы,
доцент кафедры программных систем,
доцент, к.т.н. _____ О.А. Гордеева

Дата защиты _____

Оценка _____

Самара 2025

ЗАДАНИЕ

Написать распределённое клиент-серверное приложение, используя технологию servlets (JSP, JSF). Приложение реализует игру «Тетрис» со следующими правилами:

- в игре присутствует один игрок, который заполняет игровое поле фигурами;
- в игре есть базовый набор фигур, размер поля также можно задавать;
- серверное приложение предлагает игроку последовательно фигура из набора, игрок должен расположить на поле снизу вверх;
- игрок может управлять фигурами до момента размещения их на поле (вращать, двигать);
- игра заканчивается, когда поле полностью заполнено фигурами снизу доверху, при этом баллы начисляются с учетом возможных пустых ячеек на поле.

РЕФЕРАТ

Пояснительная записка 28 с, 10 рисунков, 1 таблица, 9 источников, 1 приложение.

ТЕТРИС, ИГРА, ТЕТРАМИНО, ИГРОВОЕ ПОЛЕ, ФИГУРА, JAVA SERVER PAGES, SERVLETS, ИГРОК.

Во время курсовой работы разработаны алгоритмы и соответствующая им программа игры «Тетрис», с помощью которой пользователь сможет выкладывать в ряд фигуры для заполнения горизонтальных линий. Серверное приложение обрабатывает запросы от клиентской части и возвращает сгенерированную случайно фигуру из списка базовых фигур тетрамино. Клиентское приложение формирует запрос в виде, требуемом серверным приложением, обрабатывает соответствующим образом полученный ответ и отвечает за визуализацию игры.

Программа написана на языке Java в среде разработки IntelliJ IDEA Ultimate 2025.3.4 и функционирует под управлением операционной системы Windows 7 и выше.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

«Тетрис» – это головоломка, построенная на использовании геометрических фигур, состоящих из четырёх квадратов. В 1984 году ее придумал инженер-компьютерщик Алексей Пажитнов, в том же году игра была представлена общественности. В 1985 году Алексей Пажитнов вместе с Дмитрием Павловским написали первоначальную версию игры на языке Паскаль для компьютера «Электроника-60», а чуть позже шестнадцатилетний школьник Вадим Герасимов переписал эту игру для IBM PC. В последующие годы «Тетрис» во множестве различных версий был портирован на великое множество устройств, включая всевозможные компьютеры и игровые консоли, а также такие устройства, как графические калькуляторы, мобильные телефоны, медиаплееры, карманные персональные компьютеры и – в качестве «пасхального яйца» – устройства, вовсе не предназначенные для воспроизведения медиаконтента, такие, как паяльник [1].

Во время курсовой работы необходимо написать распределённое клиент-серверное приложение игра «Тетрис». Разработка системы будет производиться с использованием технологии RAD (Rapid Application Development), которая поддерживается методологией структурного проектирования и включает элементы объектно-ориентированного проектирования и анализа предметной области [2].

При проектировании системы будут использоваться методология ООАП (Object-Oriented Analysis/Design), в основу которой положена объектно-ориентированная методология представления предметной области в виде объектов, являющихся экземплярами соответствующих классов, и язык моделирования UML (Unified Modeling Language), который является стандартным инструментом для разработки «чертежей» программного обеспечения [3].

1 Описание и формализация предметной области

Под предметной областью (application domain) принято понимать ту часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения некоторой задачи [4].

1.1 Описание предметной области

«Тетрис» – это игра-головоломка, придуманная и написанная Алексеем Пажитновым, на тот момент советским программистом, работавшим в Вычислительном центре Академии наук СССР. Там он работал над искусственным интеллектом и распознаванием речи [5].

Идею он взял не из воздуха – основой для будущего творения послужила такая головоломка, как Pentomino Puzzle, которую придумал американский математик Соломон Голомба. Суть игры проста и понятна: из нескольких геометрических фигур собрать одну большую. На рисунке 1 представлен пример физической копии игры «Pentomino Puzzle».

Рисунок 1 – Игра «Pentomino Puzzle»

В общем, пентамино вместе с другими похожими головоломками Алексей Пажитнов и использовал в качестве основы для обкатки своих идей и создания будущей игры, но уже не абы где, а на компьютере – «Электронике-60», используя язык программирования Pascal. Задумка была в том, чтобы игра, во-первых, происходила в реальном времени, а во-вторых, фигурки должны были переворачиваться вокруг собственного центра тяжести, но техника того времени не позволяла реализовать такие геймплейные инновации, потому Алексей принял мудрое решение сократить фигуры до четырех — получились тетрамино [5]. На рисунке 2 показан пример пример игры «Тетрис».

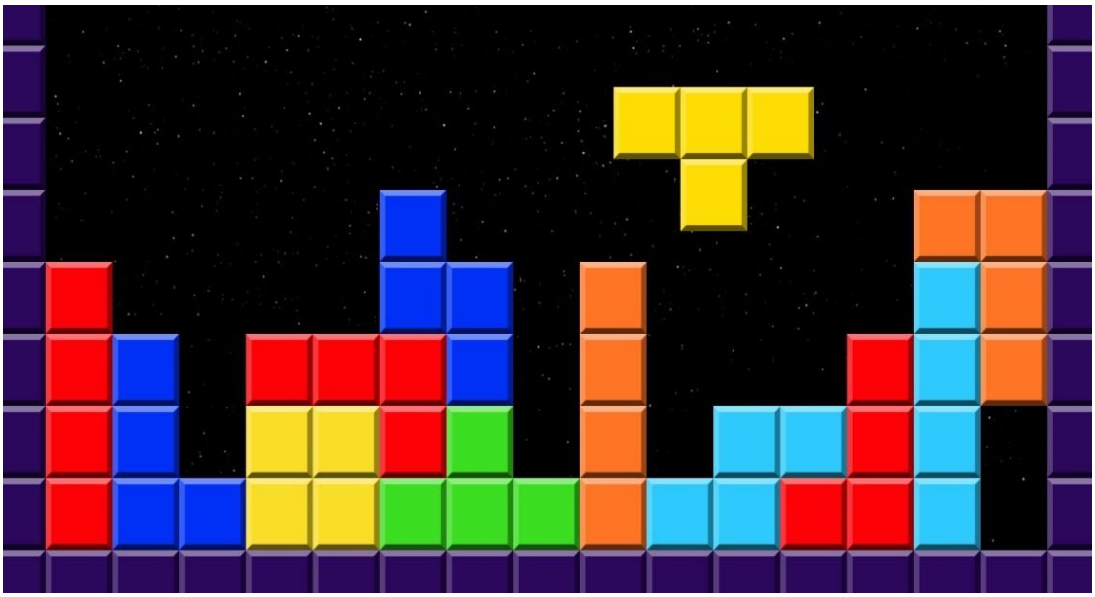

Рисунок 2 – Тетрис

1.2 Постановка задачи

Во время курсовой работы необходимо написать распределённое клиент-серверное приложение «Игра «Тетрис».

Правила игры:

- в игре будет три уровня, от которых зависит размер игрового поля;
- игра начинается после нажатия кнопки «Играть». В процессе игры игроку будут выпадать случайные фигуры из набора 7 фигур, называемых тетрамино;

− задача игрока, используя клавиши для движения и поворота фигур, располагать их таким образом, чтобы заполнить горизонтальную линию игрового поля, после заполнения которой линия становится пустой и происходит начисление очков;

− игра заканчивается, когда новая случайно сгенерированная фигура не может поместится в игровое поле. После завершения игры выводится сообщение об окончании игры, а в случае установления нового рекорда соответствующее сообщение.

Входные параметры:

− ввод имени пользователя;

− выбор уровня сложности;

− движение фигуры пользователем.

Выходные параметры:

− вывод на экран пользователя начального экрана;

− вывод на экран пользователя основного экрана для игры;

− вывод на экран пользователя таблицы рекордов.

Ограничения:

− минимальное количество очков – 0 очков;

− минимальная длина имени пользователя – 3 символа;

− максимальная длина имени пользователя – 10 символов;

− минимальный размер игрового поля – 5x10 ячеек;

− максимальный размер игрового поля – 13x20 ячеек;

− количество уровней сложности – 3.

2 Проектирование системы

2.1 Описание используемых технологий

Сервлет (servlet) – это класс, который расширяет функциональность класса HttpServlet и запускается внутри контейнера сервлетов [6].

Сервлет размещается на сервере, однако, чтобы сервер мог использовать сервлет для обработки запросов, сервер должен поддерживать движок или контейнер сервлетов.

Для каждого сервлета движок сервлетов создает только одну копию. Вне зависимости от того, сколько запросов будет отправлено сервлету, все запросы будут обрабатываться только одной копией сервлета. Объект сервлета создается либо при запуске движка сервлетов, либо когда сервлет получает первый запрос. Затем для каждого запроса запускается поток, который обращается к объекту сервлета [6].

Java Server Pages (JSP) представляет технологию, которая позволяет создавать динамические веб-страницы. Изначально JSP (вместе с сервлетами) на заре развития Java EE являлись доминирующим подходом к веб-разработке на языке Java. И хотя в настоящее время они уступило свое место другой технологии – JSF, тем не менее, JSP продолжают широко использоваться.

2.2 Структурная схема системы

На рисунке 3 приведена структурная схема разрабатываемой системы, разделяется клиентскую и серверную часть. Взаимодействие между ними осуществляется посредством протокола HTTP.

В состав клиентской части входит:

− подсистема взаимодействия с сервером, которая осуществляет установку соединения с сервером, формирование и отправку запросов;

− подсистема визуализации, которая отображает поле игрока и результаты действий пользователя на экран.

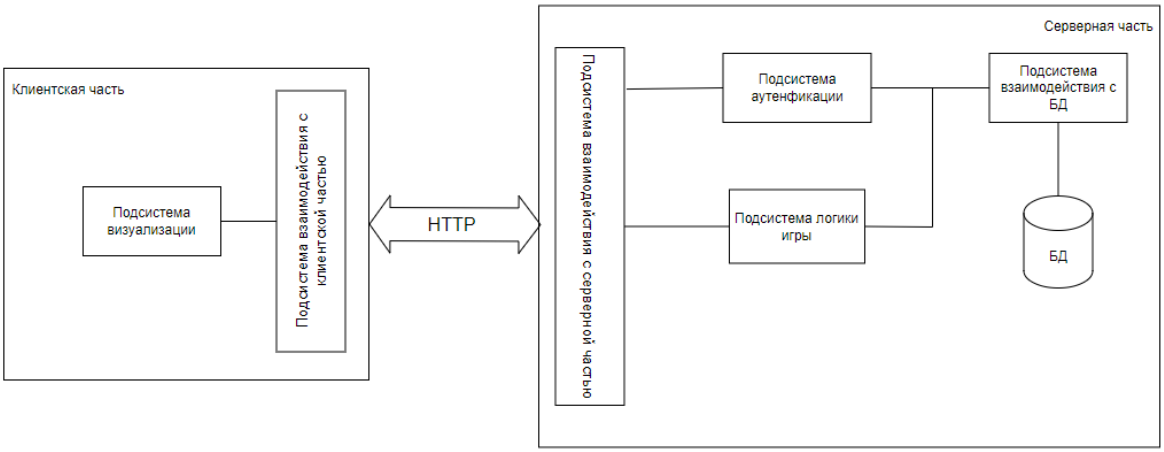

Рисунок 3 – Структурная схема системы

В состав серверной части входит:

- подсистема взаимодействия с клиентом, которая осуществляет приём данных с клиента и передачу их на обработку;
- подсистема аутентификации, которая обеспечивает пользователям вход в систему;
- подсистема логики игры, которая отвечает за процесс ведения игры;
- подсистема взаимодействия с базой данных, которая отвечает за взаимодействие системы с базой данных.

2.3 Разработка информационно-логического проекта системы

В процессе разработки информационно-логического проекта системы проводится структурное проектирование, которое охватывает анализ требований, моделирование данных и процессов, а также определение функциональной и логической структуры системы [8].

Целью информационно-логического проектирования является создание детальной модели системы, которая отражает ее функциональные возможности, основные процессы, взаимодействия между компонентами и пользовательские требования. Использование информационно-логического подхода в проектировании позволяет создать наглядное представление системы, и выявить потенциальные проблемы на ранних стадиях разработки

2.3.1 Язык UML

Для специфицирования (построения точных, недвусмысленных и полных моделей) системы и ее документирования используется унифицированный язык моделирования UML.

UML был разработан для создания унифицированного подхода к моделированию, который бы охватывал разнообразные потребности разработчиков и помогал им в создании, поддержке и модернизации информационных систем [8]. UML часто применяется на этапах анализа и проектирования в процессе разработки программного обеспечения и помогает эффективно визуализировать структуру, поведение и взаимодействие компонентов системы.

2.3.2 Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. На ней изображаются отношения между актерами и вариантами использования [4].

На рисунке 4 приведена диаграмма вариантов использования для разрабатываемой системы.

Перед началом игрок обязан выбрать размер поля и ввести имя, после чего он будет идентифицирован в системе. В процессе игры он может управлять движением фигуры: поворачивать её, перемещать по горизонтали и ускорять её падение. Игрок может увидеть сообщение об итоге игры, после чего начать новую игру. Также игрок может посмотреть рейтинг игроков в системе.

2.3.3 Диаграмма последовательности

Диаграмма последовательности (sequence diagram) – своеобразный временной график «жизни» всей совокупности объектов, связанных между

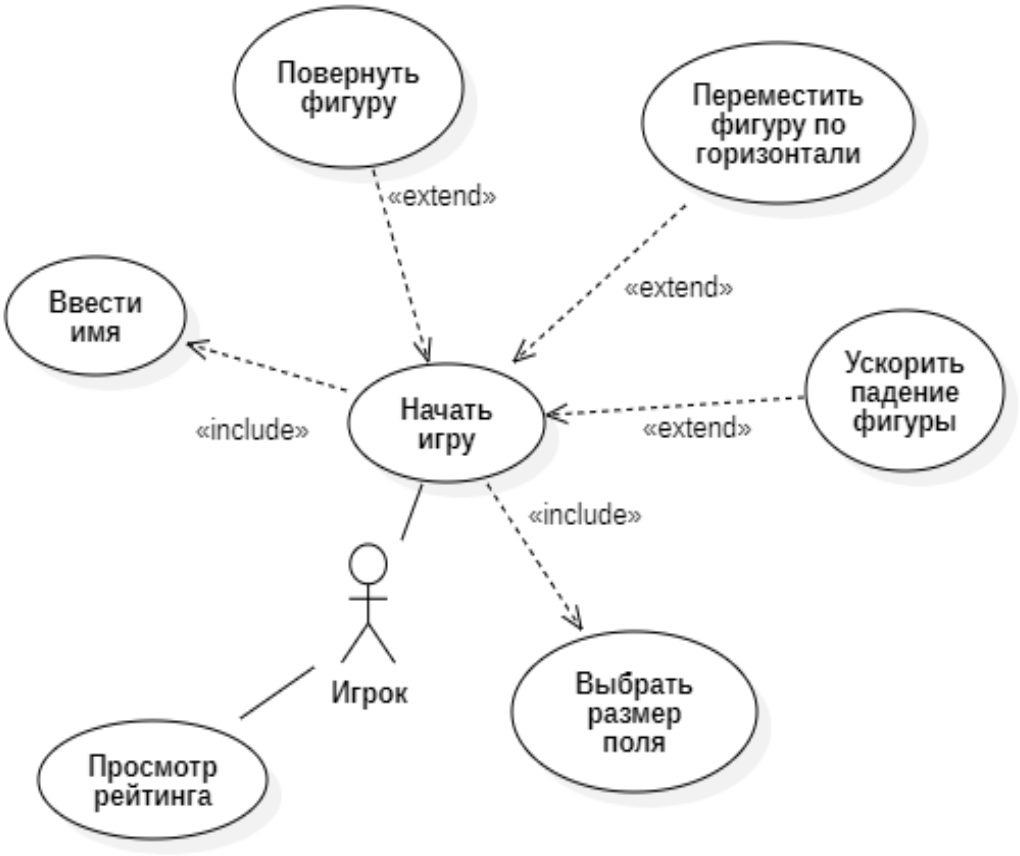

Рисунок 4 – Диаграмма вариантов использования

собой для реализации варианта использования программной системы, достижения бизнес-цели или выполнения какой-либо задачи [4].

На рисунке 5 представлена диаграмма последовательности для варианта использования «Начало новой игры».

2.3.4 Диаграмма деятельности

На рисунке 6 приведена диаграмма деятельности системы. Система открывает начальный экран, в котором пользователь вводит имя и выбирает размер игрового поля. Также игрок может открыть таблицу рейтинга с просмотром лучших результатов для каждого уровня. После ввода данных система проверяет введенные данные на корректность и затем открывает основной экран игры.

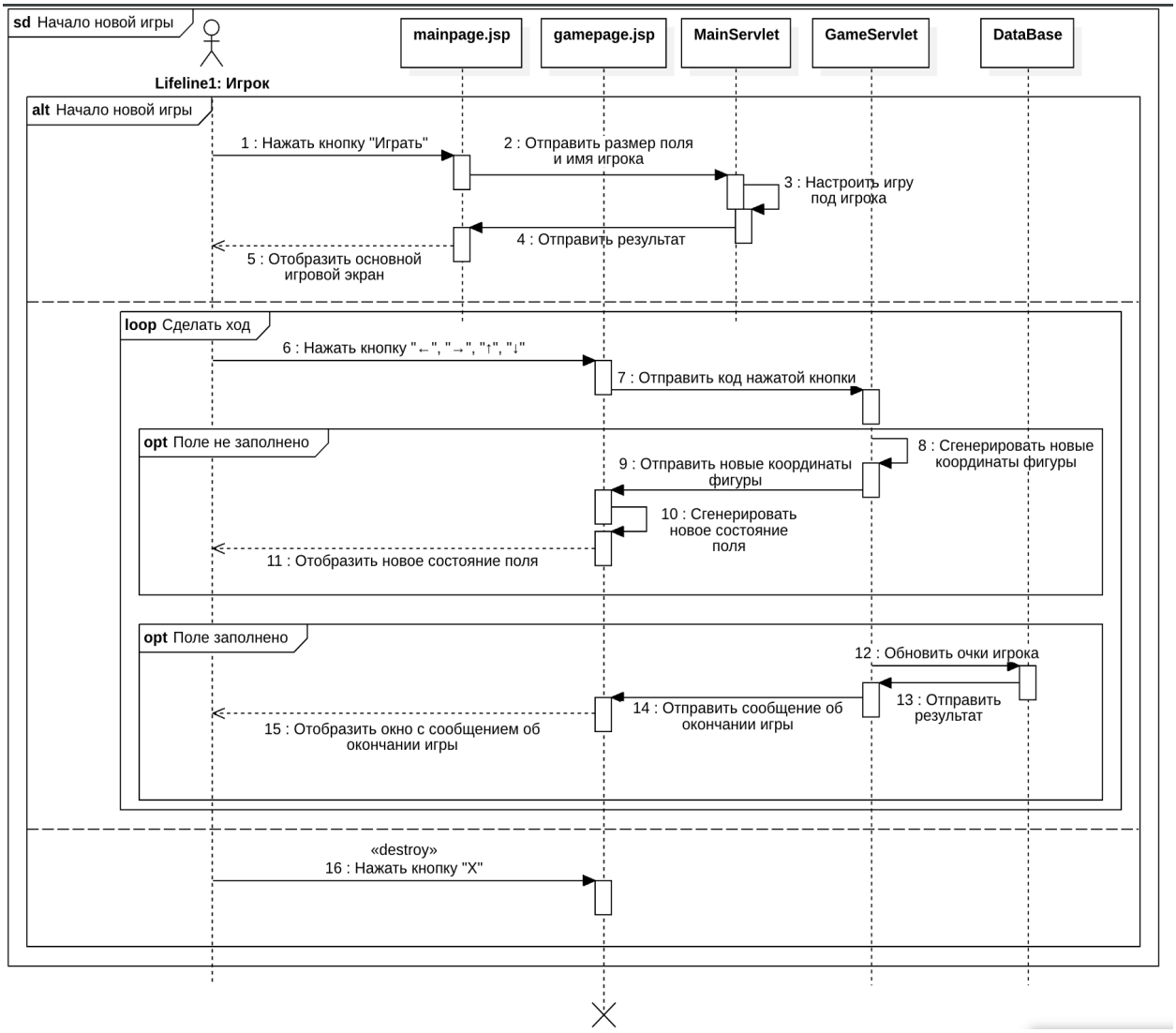

Рисунок 5 – Диаграмма последовательности

Далее происходит генерация случайной фигуры из набора и начинается её падение. В процессе клиент может управлять движением фигуры с помощью кнопок на клавиатуре. При падении фигуры на дно игра проверяет, можно ли удалить линию и поместится ли новая сгенерированная фигура в игровом поле. При удалении линии увеличивается счет игрока на 100 очков. Если поле заполнено, то игра заканчивается и система отображает сообщение об окончании игры, а также количество очков, набранных за игру. В ином случае генерируется новая фигура и игра идет до заполнения поля. Чтобы выйти из системы, пользователю необходимо закрыть страницу в браузере.

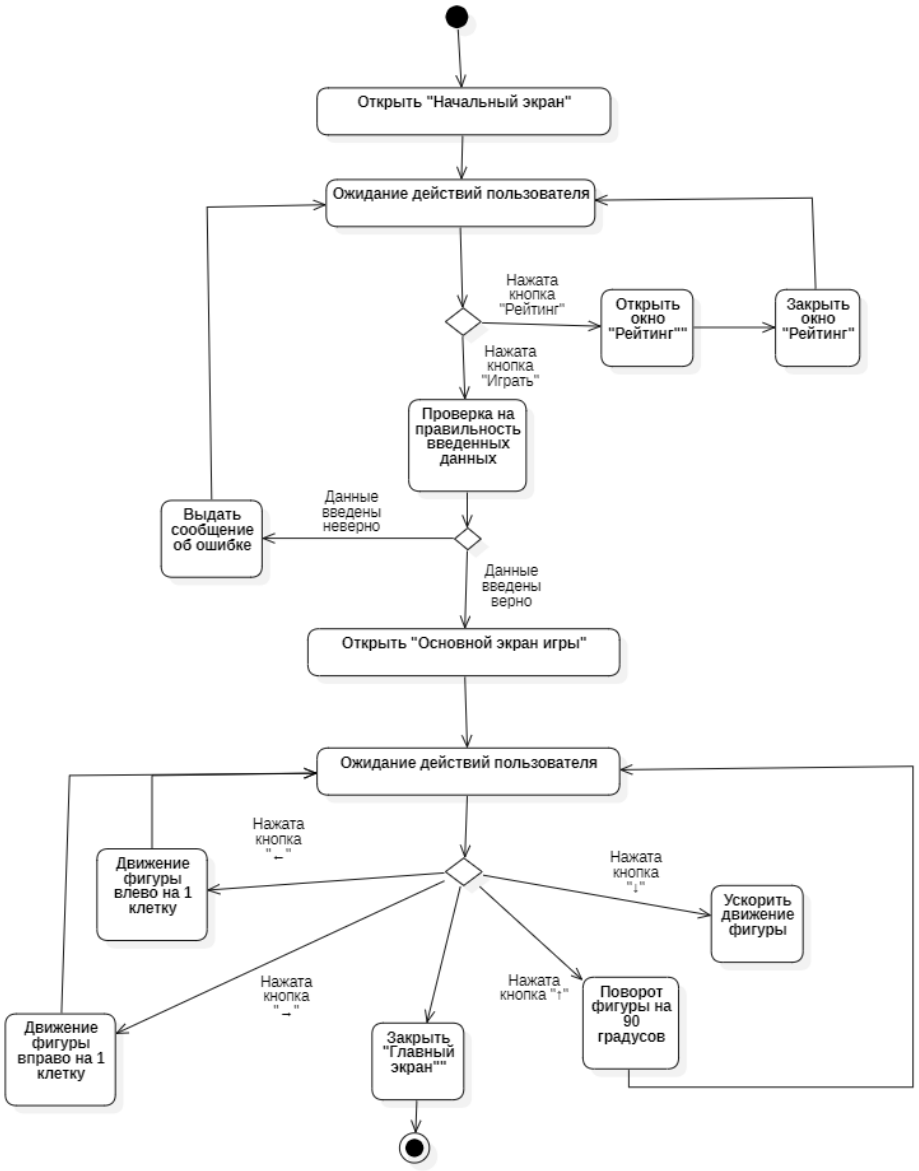

Рисунок 6 – Диаграмма деятельности системы

3 Реализация системы

3.1 Разработка и описание интерфейса пользователя

На рисунке 7 представлена начальный экран игры. На окне отображается название игры, поле для ввода имени, выбор уровня, кнопки «Играть» и «Рейтинг».

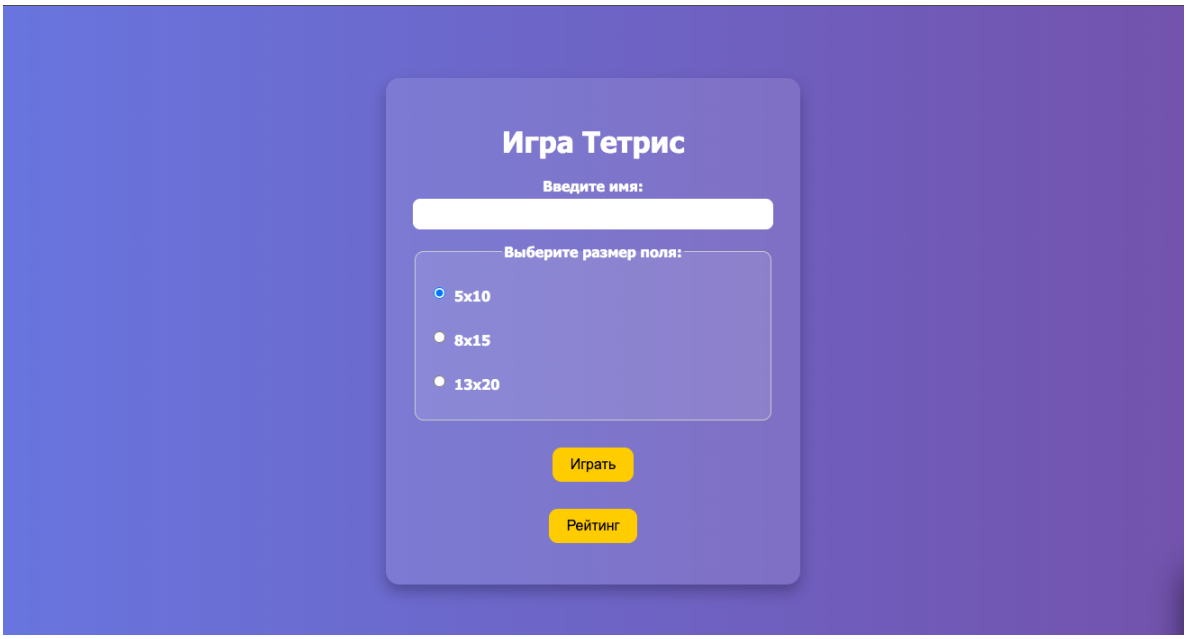

Рисунок 7 – Начальный экран

По нажатию кнопки «Играть», выводится основной экран для игры, пример которого изображен на рисунке 8. На данном экране отображается имя пользователя, количество набранных очков и поле для игры. Пользователь может управлять движением фигуры по нажатию кнопок «←», «→». Фигуры можно поворачивать по нажатию кнопок «↑». Можно увеличивать скорость падения по нажатию кнопки «↓». При заполнении линии поля игроку начисляется 100 очков. При заполнении игрового поля фигурами до верхней линии игрок проигрывает, ему выводится информационное сообщение об игре: сообщение, очки, набранные за игру, и кнопка «Начать заново».

По нажатию кнопки «Рейтинг» отображается окно со статистикой. На ней отображаются лучшие результаты игроков по сложности игры. На рисунке 9 показан пример страницы с рейтингом игроков.

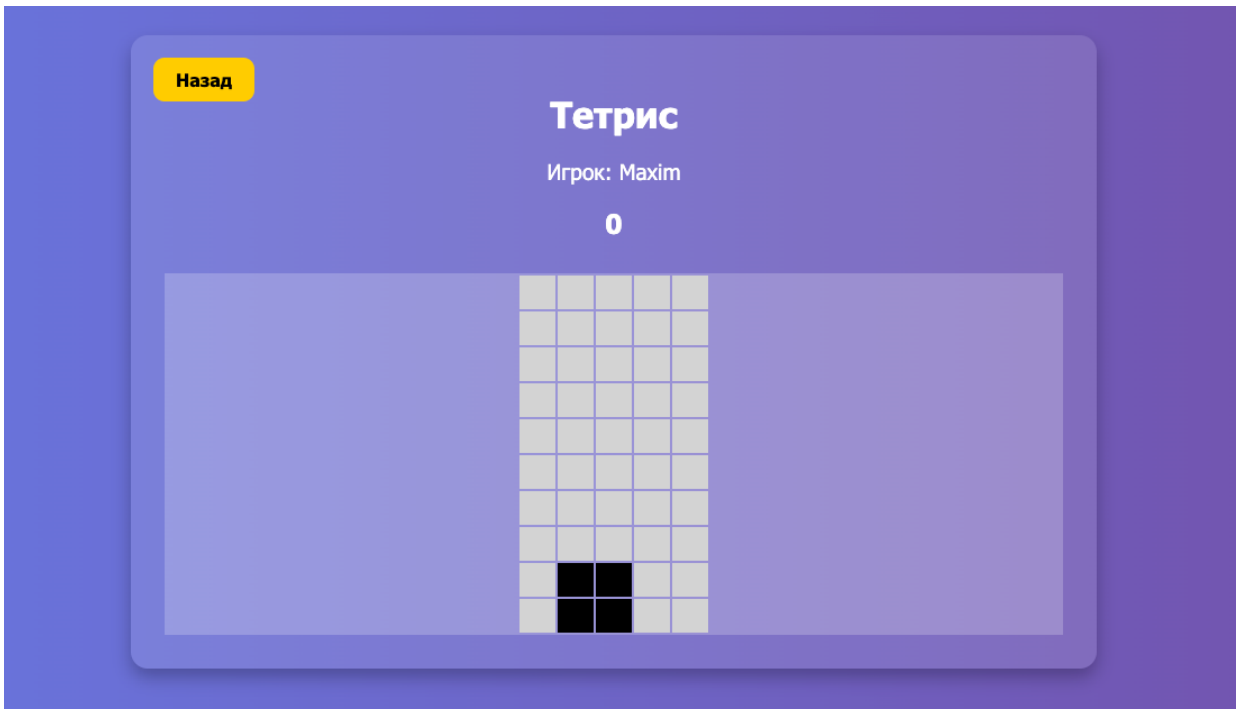

Рисунок 8 – Основной экран

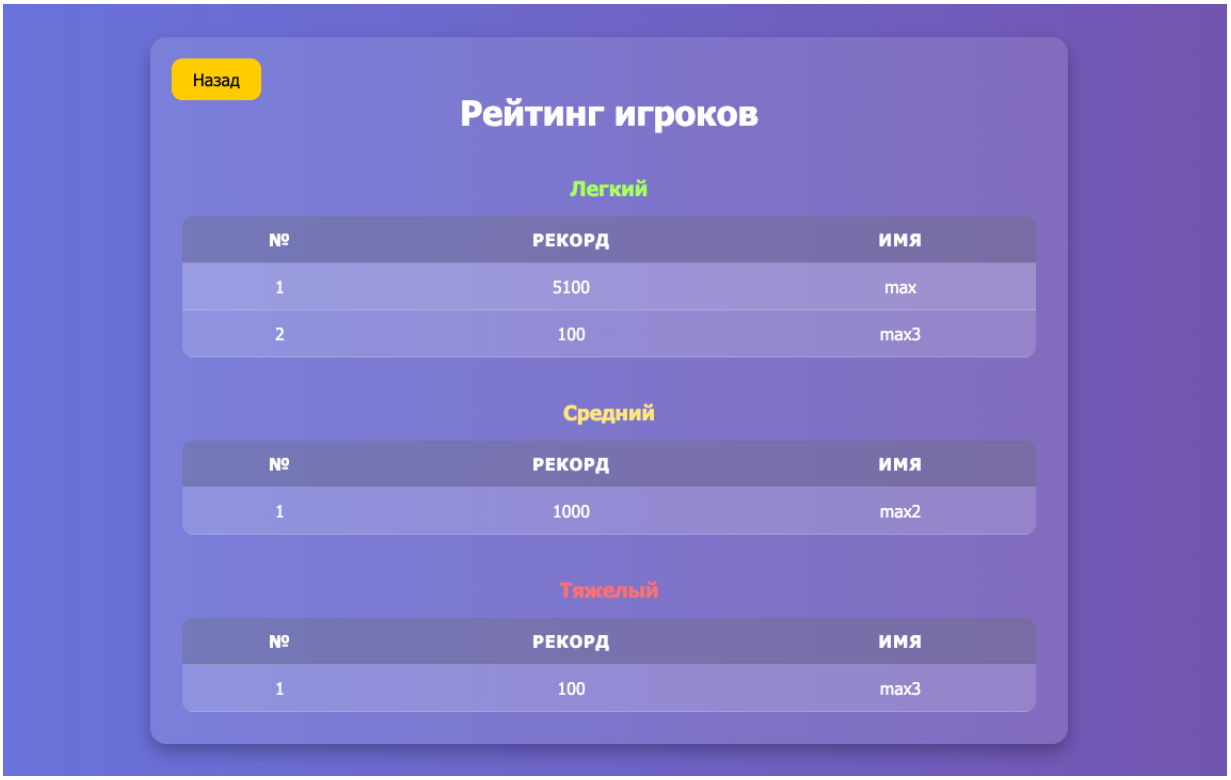

Рисунок 9 – Рейтинг игроков

3.2 Диаграммы реализации

Диаграммы реализации предназначены для отображения состава компилируемых и выполняемых модулей системы, а также связей между

ними. Диаграммы реализации разделяются на два конкретных вида: диаграммы компонентов (component diagrams) и диаграммы развёртывания (deployment diagrams) [9].

3.2.1 Диаграмма компонентов

Диаграмма компонентов описывает особенности физической реализации приложения, определяет архитектуру приложения и устанавливает зависимость между компонентами, в роли которых выступает исполняемый код. Диаграмма компонентов отображает общие зависимости между компонентами. Основными графическими элементами диаграммы являются компоненты, интерфейсы и зависимости между ними [4].

Диаграмма компонентов отображена на рисунке 10. Компоненты, входящие в диаграмму представлены в таблице 1.

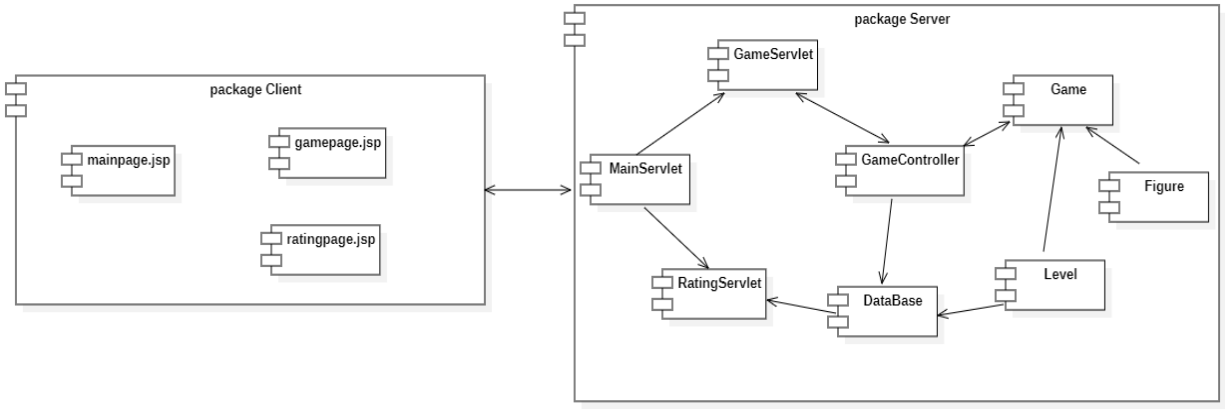

Рисунок 10 – Диаграмма компонентов системы

Таблица 1 – Основные компоненты системы

Название	Назначение
1	2
mainpage.jsp	Отвечает за отображение стартовой страницы и ввода данных
mainpage.jsp	Отвечает за отображение стартовой страницы и ввода данных

Продолжение таблицы 1

1	2
gamepage.jsp	Отвечает за отображение процесса игры и нажатия клавиш пользователем
ratingpage.jsp	Отвечает за отображение таблицы рейтинга
MainServlet	Отвечает за запуск игры
GameServlet	Формирует игровое поле, отображение очков и передачу действий пользователя
RatingServlet	Отвечает за формирование таблицы рейтинга
GameController	Отвечает за обработку действий пользователя и обновление состояния игры
Game	Отвечает за текущее состояние игры
Figure	Хранит информацию о фигуре на поле
Level	Хранит информацию о размере игрового поля

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы было разработано распределённое клиент-серверное приложение «Игра «Тетрис» с использованием технологий servlets (JSP, JSF).

Приведены основные понятия предметной области, рассмотрена история и правила игры «Тетрис». На основе проведённого анализа была сформулирована постановка задачи.

Была описана структурная схема системы, разработаны экранные формы системы. Также был разработан информационно-логический проект по методологии UML, в который вошли диаграммы вариантов использования, деятельности, последовательности, компонентов.

Разработанная система может использоваться пользователями разных возрастов для разнообразия своего досуга.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Тетрис — Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Тетрис (дата обращения: 12.03.2024).

2 RAD-программирование [Электронный ресурс] // Википедия: электронная. энциклопедия. 2001-2017. URL: https://ru.wikipedia.org/ wiki/RAD_(программирование) (дата обращения: 12.03.2024).

3 ООП: понятие и основные принципы программирование [Электронный ресурс] // Studfile: [сайт]. URL: https://studfile.net/preview/ 2910773/page:2/ (дата обращения: 12.03.2024).

4 Нотация и семантика языка UML. [Электронный ресурс] // НАО Интуит: [сайт]. https://intuit.ru/studies/courses/32/32/lecture/1000?page=2 (дата обращения: 12.03.2024).

5 Тетрис: история одной игры. [Электронный ресурс] // Хабр: [сайт]. https://habr.com/ru/companies/timeweb/articles/669676/ (дата обращения: 12.03.2024).

6 Java EE | Как работает сервлет. [Электронный ресурс] // metanit.com: [сайт]. https://metanit.com/java/javaee/4.8.php (дата обращения: 12.03.2024).

7 Java EE | Что такое JSP. Первая страница [Электронный ресурс] // metanit.com: [сайт]. https://metanit.com/java/javaee/3.1.php (дата обращения: 12.03.2024).

8 Что такое UML? [Электронный ресурс]. URL: https://habr.com/ru/articles/458680/ (дата обращения: 31.10.2024).

9 Диаграммы реализации [Электронный ресурс] // maksakov–sa.ru: [сайт]. URL: http://www.maksakov–sa.ru/ModelUML/DiagrReal/index.html (дата обращения: 26.02.2025).

ПРИЛОЖЕНИЕ А

Листинг программы

DataBase.java

```
package org.example.database;

import org.example.game.Level;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DataBase {
   public void connect() {
      try {
         Class.forName("org.sqlite.JDBC");
      } catch (ClassNotFoundException e) {
         System.err.println("Class.forName: " + e.getMessage());
      }
      try (
            // create a database connection
            Connection connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
            Statement statement = connection.createStatement();
      ) {
         statement.setQueryTimeout(30);  // set timeout to 30 sec.
         statement.executeUpdate("create table records (id integer PRIMARY KEY AUTOINCREMENT,
name string, level string, points string)");
      } catch (SQLException e) {
         System.err.println("Connect SQLite: " + e.getMessage());
      }
   }

   public int getRecord(Level level, String name) {
      try (
            // create a database connection
            Connection connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
            Statement statement = connection.createStatement();
      ) {
         statement.setQueryTimeout(30);  // set timeout to 30 sec.

         ResultSet rs = statement.executeQuery("select points from records where level = '" + level + "' and
name = \" + name + \"' order by points desc limit 1");
         if (rs.next()) {
            int record = Integer.parseInt(rs.getString("points"));
            return record;
         }
      } catch (SQLException e) {
         System.err.println("getRecord SQLite: " + e.getMessage());
      }
      return 0;
   }

   public void setRecord(Level level, String name, int points) {
      try (
            // create a database connection
            Connection connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
            Statement statement = connection.createStatement();
      ) {
         statement.setQueryTimeout(30);  // set timeout to 30 sec.

         ResultSet rs = statement.executeQuery("select id from records where level = '" + level + "' and name
= '" + name + "' order by points desc limit 1");
```

```
                int id = 0;
                if (rs.next()) {
                    id = Integer.parseInt(rs.getString("id"));
                }
                rs.close();
                if (id != 0)
                    statement.executeUpdate("update records set points = " + points + " where id = " + id);
                else
                    statement.executeUpdate("insert into records(name, level, points) values('" + name + "', '" + level
+ "', " + points + ")");
            } catch (SQLException e) {
                System.err.println("setRecord SQLite: " + e.getMessage());
            }
        }

        public List<Record> getAllRecords(Level level) {
            List<Record> records = new ArrayList<>();
            try (
                    // create a database connection
                    Connection connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
                    Statement statement = connection.createStatement();
            ) {
                statement.setQueryTimeout(30);  // set timeout to 30 sec.

                ResultSet rs = statement.executeQuery("select points, name from records where level = '" + level + "'
order by points desc");
                while (rs.next()) {
                    records.add(new Record(rs.getString("points"), rs.getString("name")));
                }
    rs.close();
```

Record.java

```
    package org.example.database;

    public class Record {
        private final String name;
        private final String points;

        public Record(String points, String name) {
            this.name = name;
            this.points = points;   }
        public String getName() { return name;  }
        public String getPoints() { return points; } }
    package org.example.game.figures;

    import java.awt.*;
    import java.util.ArrayList;
    import java.util.Random;

    public abstract class Figure {
        ArrayList<Point[]> states = new ArrayList<Point[]>();
        private int currentState = 0;
        // x - по вертикали, y - по горизонтали
        private int dx = 0;
        private int dy = 0;

        public boolean setFirst(int[][] field) {
            Random random = new Random();
            currentState = random.nextInt(states.size());
            Point[] currentPoints = states.get(currentState);
            return checkField(currentPoints, field);
        }
        public boolean turn90(int[][] field) {
            int width = field[0].length;
```

```
            int height = field.length;
            int newState = currentState + 1;
            if (newState == states.size()) newState = 0;
            Point[] newPoints = getNewPoints(newState, dx, dy);
            for (Point newPoint : newPoints)
                if (newPoint.x < 0 || newPoint.y < 0 || newPoint.x >= height || newPoint.y >= width)
                    return false;
            if (checkField(newPoints, field)) {
                currentState = newState;
                return true;
            } else return false;
        }
        public boolean move(int x, int y, int[][] field) {
            int width = field[0].length;
            int height = field.length;
            int newdx = dx + x;
            int newdy = dy + y;
            Point[] newPoints = getNewPoints(currentState, newdx, newdy);
            for (Point newPoint : newPoints)
                if (newPoint.x < 0 || newPoint.y < 0 || newPoint.x >= height || newPoint.y >= width)
                    return false;
            if (checkField(newPoints, field)) {
                dx = newdx;
                dy = newdy;
                return true;
            } else return false;
        }
        private boolean checkField(Point[] newPoints, int[][] field) {
            boolean isFree = true;
            for (int i = 0; i < field[0].length; i++) {
                if (field[0][i] == 2) {
                    isFree = false;
                    break;
                }
            }
            for (Point newPoint : newPoints)
                if (field[newPoint.x][newPoint.y] == 2) {
                    isFree = false;
                    break;
                }
            return isFree;
        }

        //получения текущего положения
        public Point[] getCurrentPoints() {
            return getPoints(currentState, dx, dy);
        }

        private Point[] getNewPoints(int newState, int newdx, int newdy) {
            return getPoints(newState, newdx, newdy);
        }
        private Point[] getPoints(int currentState, int dx, int dy) {
            Point[] currentPoint = new Point[4];
            for (int i = 0; i < currentPoint.length; i++) {
                currentPoint[i] = new Point(states.get(currentState)[i].x + dx, states.get(currentState)[i].y + dy);
            }
            return currentPoint;
        }
}
```

Game.java

```
package org.example.game;

import org.example.database.DataBase;
```

```
import org.example.game.figures.*;

import java.awt.*;
import java.util.Random;

public class Game {
    public String name;
    public Level level;
    int[][] gameField;
    Figure currentFigure;
    int points;
    boolean newRecord = false;

    public int[][] getGameField() {
        return gameField;
    }
    public void setGameField(int[][] gameField) {
        this.gameField = gameField;
        points = 0;
    }
    public boolean makeMove(int x, int y) {
        Point[] point = this.currentFigure.getCurrentPoints();
        if (this.currentFigure.move(x, y, gameField)) {
            redrawFigure(point);
            return true;
        } else {   return false;}}
    private void redrawFigure(Point[] point) {
        for (Point item : point) {
            gameField[item.x][item.y] = EMPTY_CELL;
        }
        point = this.currentFigure.getCurrentPoints();
        for (Point value : point) {
            gameField[value.x][value.y] = MOVING_CELL;
        }
    }
    public boolean generationFigure() {
        int figure = 0;
        Random rand = new Random();
        figure = rand.nextInt(7);
        switch (figure) {
            case 0: {
                this.currentFigure = new IFigure(gameField);
                break;
            }
            case 1: {
                this.currentFigure = new JFigure(gameField);
                break;
            }
            case 2: {
                this.currentFigure = new LFigure(gameField);
                break;
            }
            case 3: {
                this.currentFigure = new QFigure(gameField);
                break;
            }
            case 4: {
                this.currentFigure = new SFigure(gameField);
                break;
            }
            case 5: {
                this.currentFigure = new TFigure(gameField);
                break;
```

```
            }
            case 6: {
                this.currentFigure = new ZFigure(gameField);
                break;
            }
        }
        if (this.currentFigure.setFirst(gameField)) {
            Point[] point = this.currentFigure.getCurrentPoints();
            for (Point value : point) {
                gameField[value.x][value.y] = MOVING_CELL;
            }
            return true;
        } else {
            try {
                setRecord();
            } catch (Exception e) {}
            return false;
        }
    }

    public int checkForDeleteLine() {
        Point[] point = this.currentFigure.getCurrentPoints();
        int deletedLines = 0;
        for (Point value : point) {
            int countOfPlacedCellsInLine = 0;
            for (int j = 0; j < gameField[value.x].length; j++) {
                if (gameField[value.x][j] == PLACED_CELL) {
                    countOfPlacedCellsInLine++;
                }
            }
            if (countOfPlacedCellsInLine == gameField[value.x].length) {
                setCellReadyDelete(value.x);
                deleteLine(value.x);
                points += 100;
                deletedLines++;
            }
        }
        return deletedLines;    }
    public void setCurrentFigureCellsPlaced() {
        Point[] point = this.currentFigure.getCurrentPoints();
        for (Point value : point) {
            gameField[value.x][value.y] = PLACED_CELL;      }    }
    public boolean turn90() {
        Point[] point = this.currentFigure.getCurrentPoints();

        if (this.currentFigure.turn90(gameField)) {
            redrawFigure(point);
            return true;
        } else {
            return false;     }  }
    public boolean isNewRecord() {
        return newRecord;
    }
    public int getCurrentPoint() {
        return points;
    }
    public void setMovedCellsPlaced() {
        for (int i = 0; i < gameField.length; i++) {
            for (int j = 0; j < gameField[i].length; j++) {
                if (gameField[i][j] == READY_DELETE_CELL)
                    gameField[i][j] = PLACED_CELL;
            }
        }
```

```
        }

        private void setCellReadyDelete(int lineNumber) {
            for (int i = 0; i < lineNumber; i++) {
                for (int j = 0; j < gameField[i].length; j++) {
                    if (gameField[i][j] == PLACED_CELL)
                        gameField[i][j] = READY_DELETE_CELL;
                }
            }
        }

        private void deleteLine(int lineNumber) {
            for (int i = lineNumber; i > 0; i--) {
                System.arraycopy(gameField[i - 1], 0, gameField[i], 0, gameField[lineNumber].length);
            }
            for (int i = 0; i < gameField[lineNumber].length; i++) {
                gameField[0][i] = EMPTY_CELL;
            }
        }
        private void setRecord() {
            DataBase dataBase = new DataBase();
            int record = dataBase.getRecord(level, name);
            if (points > record) {
                newRecord = true;
                dataBase.setRecord(level, name, points);
            }
        }
        public static final int EMPTY_CELL = 0;
        static final int MOVING_CELL = 1;
        static final int PLACED_CELL = 2;
        static final int READY_DELETE_CELL = 3;
}
```

Level.java

```
    public enum Level {
        EASY(5, 10),
        MIDDLE(8, 15),
        HARD(13, 20);

        public final int width;
        public final int height;

        Level(int width, int height) {
            this.width = width;
            this.height = height;
        }
}
```

GameServlet.java

```
    import org.example.GameController;
    import org.example.game.Level;

    import javax.servlet.ServletException;
    import javax.servlet.annotation.WebServlet;
    import javax.servlet.http.HttpServlet;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import javax.servlet.http.HttpSession;
    import java.io.IOException;
    import java.io.PrintWriter;
    import java.util.Date;
    import java.util.HashMap;
    import java.util.Map;

    import static org.example.game.Game.EMPTY_CELL;
```

```
@WebServlet("/game")
public class GameServlet extends HttpServlet {

    Map<String, GameController> map = new HashMap<>();

    public GameController getGameController(String sessionId) {
        if (!map.containsKey(sessionId)) {
            GameController gameController = new GameController();
            map.put(sessionId, gameController);
        }
        return map.get(sessionId);
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        HttpSession session = req.getSession();
        GameController gameController = getGameController(session.getId());

        String type = req.getParameter("type");
        resp.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html; charset=UTF-8");

        if (type != null && type.equals("check")) {
            PrintWriter out = resp.getWriter();
            out.println(new Date());
        } else if (type != null && type.equals("tick") && gameController.isGameInitialized()) {
            int[][] currentField = gameController.tickGameAndGetInitField();
            PrintWriter out = resp.getWriter();
            if (currentField == null) {
                out.println(gameController.endGameMessage());
            } else {
                out.println(generateField(currentField) + "_" + gameController.getGamePoints());
            }
        } else {
            String name = req.getParameter("name");
            Level size = Level.valueOf(req.getParameter("size").toUpperCase());
            int[][] initField = gameController.startGameAndGetInitField(size, name);
            String fieldHtml = generateField(initField);
            session.setAttribute("name", name);
            session.setAttribute("field", fieldHtml);
            req.getRequestDispatcher("gamepage.jsp").forward(req, resp);
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws IOException {
        HttpSession session = req.getSession();
        GameController gameController = getGameController(session.getId());
        String requestBody = req.getReader().lines().reduce("", (accumulator, actual) -> accumulator + actual);
        switch (requestBody) {
            case "ArrowLeft":
                gameController.moveLeft();
                break;
            case "ArrowRight":
                gameController.moveRight();
                break;
            case "ArrowDown":
                gameController.moveDown();
                break;
            case "ArrowUp":
```

```
            gameController.turn();
            break;
        default: {
        }
    }
    PrintWriter out = resp.getWriter();
    out.write(generateField(gameController.getGameField()));
}

private String generateField(int[][] field) {
    StringBuilder fieldHtml = new StringBuilder();
    fieldHtml.append("<table>");
    for (int[] row : field) {
        fieldHtml.append("<tr>");
        for (int i : row) {
            if (i == EMPTY_CELL)
                fieldHtml.append("<td width=30 height=30 style=\"background: LightGray\"></td>");
            else
                fieldHtml.append("<td width=30 height=30 style=\"background: black\"></td>");
        }
        fieldHtml.append("</tr>");
    }
    fieldHtml.append("</table>");
    return fieldHtml.toString();
}
```