МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВПО «Омский государственный технический университет»

Кафедра ***Автоматизированные системы обработки информации и управления***

Специальность **220100.62 – «Системный анализ и управление»**

**КУРСОВОЙ ПРОЕКТ**

на тему:

Реализация клеточного автомата Инь-Ян \_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по дисциплине «Объектно-ориентированное программирование на JAVA»

Студент Антипин Игорь Юрьевич группы ИВТ-233

(Ф.И.О. полностью)

Пояснительная записка

#### Шифр проекта (работы)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель проекта

Никонов В.А. (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата)

Разработал студент

Антипин И.Ю. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата)

Омск 2025

# **АННОТАЦИЯ**

Пояснительная записка к курсовому проекту: 18 рис., 5 источников, 1 прил., 30 стр.

КОМПЬЮТЕРНАЯ ИГРА, ЛОГИЧЕСКАЯ ИГРА НА РАЗВИТИЕ, JAVA, SPRING FRAMEWORK, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, МОДУЛЬНОСТЬ, ТЕСТИРОВАНИЕ, ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Цель работы – проектирование и реализация компьютерной логической игры «2048» с возможностью выбора уровня сложности.

Разработана архитектура клиент-серверного приложения, описаны алгоритмы управления игровым полем, перемещения и слияния плиток, отслеживания состояний игры. На основе разработанных схем реализовано приложение с графическим интерфейсом, возможностью выбора уровня сложности (целевое значение плитки: 16, 32, 64, 128, 256, 512, 1024, 2048), меню настроек, информацией об авторе и системой отслеживания завершения игры.

Реализованное приложение позволяет пользователю взаимодействовать с игровым полем, наблюдать за случайным появлением новых плиток, выполнять ходы, объединять числовые значения и достигать целевого значения для победы. Приложение также корректно обрабатывает ситуацию проигрыша при отсутствии допустимых ходов.

Программа разработана на языке программирования Java с использованием фреймворка Spring Framework , что позволило организовать управление зависимостями, обеспечить модульность, расширяемость и тестируемость кода. Интерфейс пользователя реализован с применением технологий Spring Boot Web, что обеспечивает удобство взаимодействия через браузер.

Федеральное государственное бюджетное образовательное учреждение высшего  
 профессионального образования «Омский государственный технический университет» (ОмГТУ)

|  |  |
| --- | --- |
| Кафедра  «Автоматизированные системы обработки информации и управления» | «УТВЕРЖДАЮ»  Зав. кафедрой АСОИУ, д-р техн.наук, проф.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Никонов  « \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_г. |

# **ЗАДАНИЕ**

на курсовое проектирование

по дисциплине «Объектно-ориентированное программирование на JAVA»

**Студент**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, **группа**: \_\_\_\_\_\_\_\_\_\_\_\_\_

**Тема**: Разработка игры 2048

**Краткая постановка задачи**.

Игра «2048» представляет собой одномерную стратегию на двумерном поле, где игрок управляет перемещением числовых плиток с целью их слияния и достижения максимального значения. Игровое поле состоит из фиксированного количества ячеек (обычно 4×4), каждая из которых может содержать число, равное степени двойки. Новые плитки появляются случайным образом, а их значение зависит от текущего состояния игрового поля.

Правила игры определены таким образом, чтобы обеспечить баланс между случайностью и логическим планированием, заставляя игрока продумывать последовательность ходов для эффективного управления пространством и ростом значений:

1. Появление новых плиток . После каждого хода на свободной ячейке появляется новая плитка со значением 2 или 4, вероятность появления которых определяется случайным генератором.
2. Перемещение плиток . Все плитки могут двигаться одновременно в одну из четырёх сторон (вверх, вниз, влево, вправо). При этом плитки скользят до упора, пока не достигнут края поля или другой плитки.
3. Слияние одинаковых плиток . Если две плитки одного значения сталкиваются при движении, они объединяются в одну плитку с суммарным значением (например, 2 + 2 = 4). За одно ходовое действие одна и та же плитка может участвовать только в одном слиянии.
4. Условие победы . Игра считается выигранной, когда на поле появляется плитка с заданным целевым значением: 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 — в зависимости от выбранного уровня сложности.
5. Условие проигрыша . Игра заканчивается поражением, если на поле нет ни одного допустимого хода: все ячейки заполнены, и невозможно выполнить ни одно перемещение или слияние.

Начальные конфигурации формируются случайно (с помощью генератора случайных чисел для заданных вероятностей появления 2 и 4). Эволюция игрового процесса, порождаемая начальной ситуацией, может быть исследована только путём её пошагового воспроизведения. Однако для каждого уровня сложности (определяющего целевую плитку) можно отследить статистику результатов после множества попыток и говорить о том, какова вероятность достижения цели или раннего завершения игры в зависимости от тактики игрока и случайных факторов.

Выбор уровня сложности влияет на продолжительность партии и степень требуемой стратегической подготовки: более высокие уровни требуют большего количества успешных слияний и точного контроля над расположением плиток на поле.

**Обязательные составляющие пояснительной записки**:

 введение (актуальность темы, цель разработки системы, назначение, решаемые задачи, краткое описание содержания пояснительной записки);

 постановка задачи курсового проектирования (цели и задачи разрабатываемой системы, входная информация, нормативно-справочная информация, выходная информация, перечень функций, обеспечивающих достижение целей);

 теоретический анализ (необходимый теоретический материал, обзор известных подходов к решению подобных задач, обзор существующих аналогов системы);

 описание алгоритмов (общие положения, ввод информации, описание алгоритмов, структура БД, получение выходной информации);

 описание разработанной системы (обоснование и выбор технического обеспечения, аспекты реализации, назначение и область применения, руководство пользователя);

 результаты тестирования (описание методики тестирования, результаты тестов и их анализ, демонстрация достоинств использования системы, полный перечень недостатков, недоработок и отрицательных особенностей системы);

 заключение (основные результаты, использованные методы, оценка полноты решений поставленных задач, значимость работы, область применения полученных результатов);

 список использованных источников (со ссылками в тексте отчёта).

Дата выдачи задания « \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_г.

Дата сдачи курсового проекта « \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_г.

Преподаватель: ст. преп. кафедры АСОИУ **/** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **/**

(подпись, дата) (Фамилия И.О.)

Задание принял к исполнению студент **/** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **/**

(подпись, дата) (Фамилия И.О.)

# **Введение**

Представьте, что вы сидите за столом с листком бумаги, на котором нарисована сетка 4x4, и у вас есть несколько кубиков с числами. Ваша задача — двигать их по полю, соединять одинаковые и мечтать, чтобы из скромной двойки выросла заветная плитка с числом 2048. Звучит просто, но за этой незамысловатой идеей скрывается настоящая головоломка, которая заставляет думать стратегически. Именно так работает игра «2048», созданная в 2014 году Габриэле Чирулли. Эта игра завоевала миллионы сердец благодаря своей простоте и глубине.

В этой работея я не просто воссоздал классическую игру «2048», но и добавил ей изюминку — возможность выбирать уровень сложности, где цель варьируется от 16 до 8192. Я использовал Java, чтобы сделать код надёжным и понятным, а Spring Boot помог организовать приложение так, чтобы оно работало как часы в браузере. Эта работа — не просто код, а пример того, как можно соединить логику, дизайн и современные технологии, чтобы создать что-то увлекательное. Давайте разберем, как это все устроено, и попробуем понять, почему такие проекты — отличный способ научиться программировать.

**Оглавление**

[**АННОТАЦИЯ** 2](#_Toc199232291)

[**ЗАДАНИЕ** 3](#_Toc199232292)

[**Введение** 5](#_Toc199232293)

[**Основная часть** 7](#_Toc199232294)

[**1. Идея и механика игры «2048»** 7](#_Toc199232295)

[**2. Архитектура приложения** 7](#_Toc199232296)

[**2.1. Backend: Java и Spring Boot** 7](#_Toc199232297)

[**2. Архитектура приложения** 9](#_Toc199232298)

[**2.1. Backend: Java и Spring Boot** 9](#_Toc199232299)

[**2.2. Frontend: HTML, CSS и немного JavaScript** 10](#_Toc199232300)

[**3. Реализация игровой логики** 10](#_Toc199232301)

[**4. Выбор уровня сложности** 11](#_Toc199232302)

[**5. Использование ООП и паттернов** 12](#_Toc199232303)

[**Заключение** 14](#_Toc199232304)

[**Список источников** 15](#_Toc199232305)

[**Приложение** 16](#_Toc199232306)

[**Демонстрация программы** 28](#_Toc199232307)

# **Основная часть**

## **1. Идея и механика игры «2048»**

Игра «2048» — это как танец чисел на квадратном поле. У вас есть сетка 4x4, где появляются плитки с числами 2 или 4. Вы двигаете их влево, вправо, вверх или вниз. Если две плитки с одинаковым числом сталкиваются, они сливаются в одну с удвоенным значением, а ваш счет растет. Цель — добраться до плитки с числом, заданным уровнем сложности (например, 2048 на восьмом уровне). Но есть подвох: поле заполняется, и если некуда двигаться, игра заканчивается.

Создание такой игры — это как построить маленький мир, где вы контролируете правила. Нужно продумать, как хранить поле, как обрабатывать движения, как проверять победу или поражение. Это учит вас думать наперед и разбивать задачу на части.

Схема алгоритма игры представлена на рисунке 1.

## **2. Архитектура приложения**

Эта версия «2048» — это не просто код, а полноценное клиент-серверное приложение. Использовались Java и Spring Boot, чтобы разделить логику игры и интерфейс. Backend отвечает за игровую логику, а frontend — за отображение в браузере. Это похоже на ресторан: кухня (Java) готовит блюдо, а официант (HTML/CSS/JavaScript) подает его клиенту.

### **2.1. Backend: Java и Spring Boot**

Java — как надежный фундамент дома. Он был выбран ее за строгую типизацию и поддержку объектно-ориентированного программирования (ООП). Spring Boot упростил настройку сервера и управление запросами. Вот как был организован код:

* GameState (одиночка): Хранит игровое поле, счет, цель и уровень сложности. Одиночка гарантирует, что состояние игры всегда одно и то же.
* Tile: Класс для плитки, просто хранит значение, но делает код чище.

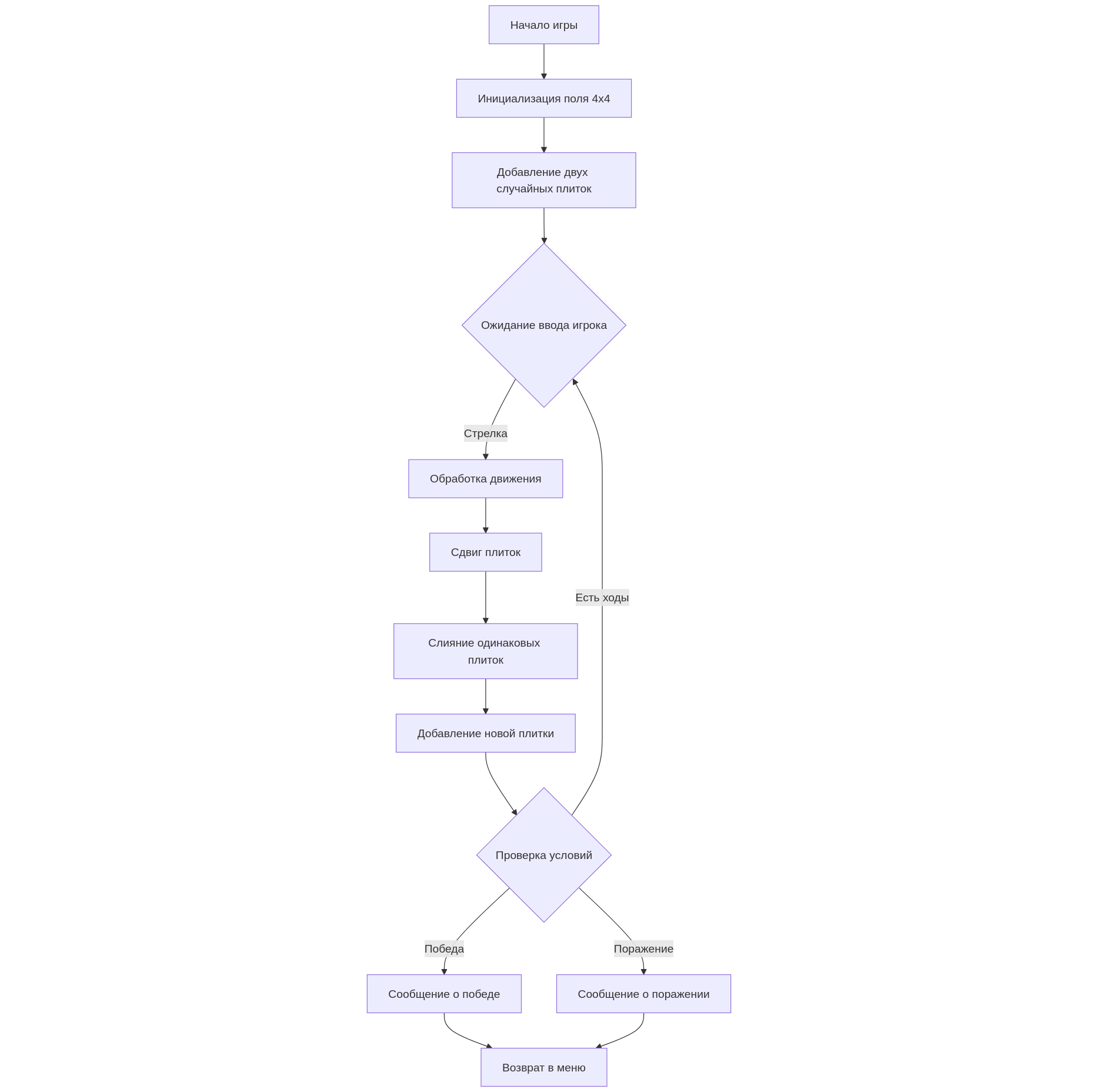


Рис. 1. Схема алгоритма игры

* MoveStrategy: Универсальный класс для обработки движений (влево, вправо, вверх, вниз). Используется параметр направления, чтобы не плодить лишние классы.
* Game2048: Главный класс, связывающий все компоненты. Он принимает ходы и обновляет состояние.
* GameController: REST-контроллер, который принимает запросы от фронтенда (например, «сделать ход вправо»).

## **2. Архитектура приложения**

Наша версия «2048» — это не просто код, а полноценное клиент-серверное приложение. Использовались Java и Spring Boot, чтобы разделить логику игры и интерфейс. Backend отвечает за игровую логику, а frontend — за отображение в браузере. Это похоже на ресторан: кухня (Java) готовит блюдо, а официант (HTML/CSS/JavaScript) подает его клиенту.

## **2.1. Backend: Java и Spring Boot**

Java — как надежный фундамент дома. Она была выбрана за строгую типизацию и поддержку объектно-ориентированного программирования (ООП). Spring Boot упростил настройку сервера и управление запросами. Вот как организован код:

* GameState (одиночка): Хранит игровое поле, счет, цель и уровень сложности. Одиночка гарантирует, что состояние игры всегда одно и то же
* Tile: Класс для плитки, просто хранит значение, но делает код чище
* MoveStrategy: Универсальный класс для обработки движений (влево, вправо, вверх, вниз). Использован параметр направления, чтобы не плодить лишние классы
* Game2048: Главный класс, связывающий все компоненты. Он принимает ходы и обновляет состояние
* GameController: REST-контроллер, который принимает запросы от фронтенда (например, «сделать ход вправо»).

UML-диаграмма классов представлена на рисунке 2.

### **2.2. Frontend: HTML, CSS и немного JavaScript**

Интерфейс — это лицо игры. HTML и CSS отвечают за внешний вид, а минимальный JavaScript отправляет запросы к серверу, когда игрок нажимает стрелки или выбирает сложность.

Когда вы нажимаете стрелку, JavaScript отправляет запрос на /api/move. Сервер обрабатывает ход, обновляет поле и возвращает новое состояние. Фронтенд рисует плитки, меняя их цвета в зависимости от значений.

## **3. Реализация игровой логики**

Логика игры — это сердце проекта. Давайте разберем, как она работает, на примере движения влево:

* Берем строку поля (4 клетки).
* Убираем нули (пустые клетки).
* Если рядом стоят одинаковые числа, сливаем их (например, 2+2=4, добавляем 4 к счету).
* Заполняем строку результатом, добавляя нули в конец.
* Если что-то изменилось, добавляем новую плитку (2 или 4).

Схема алгоритма движения представлена на рисунке 3.

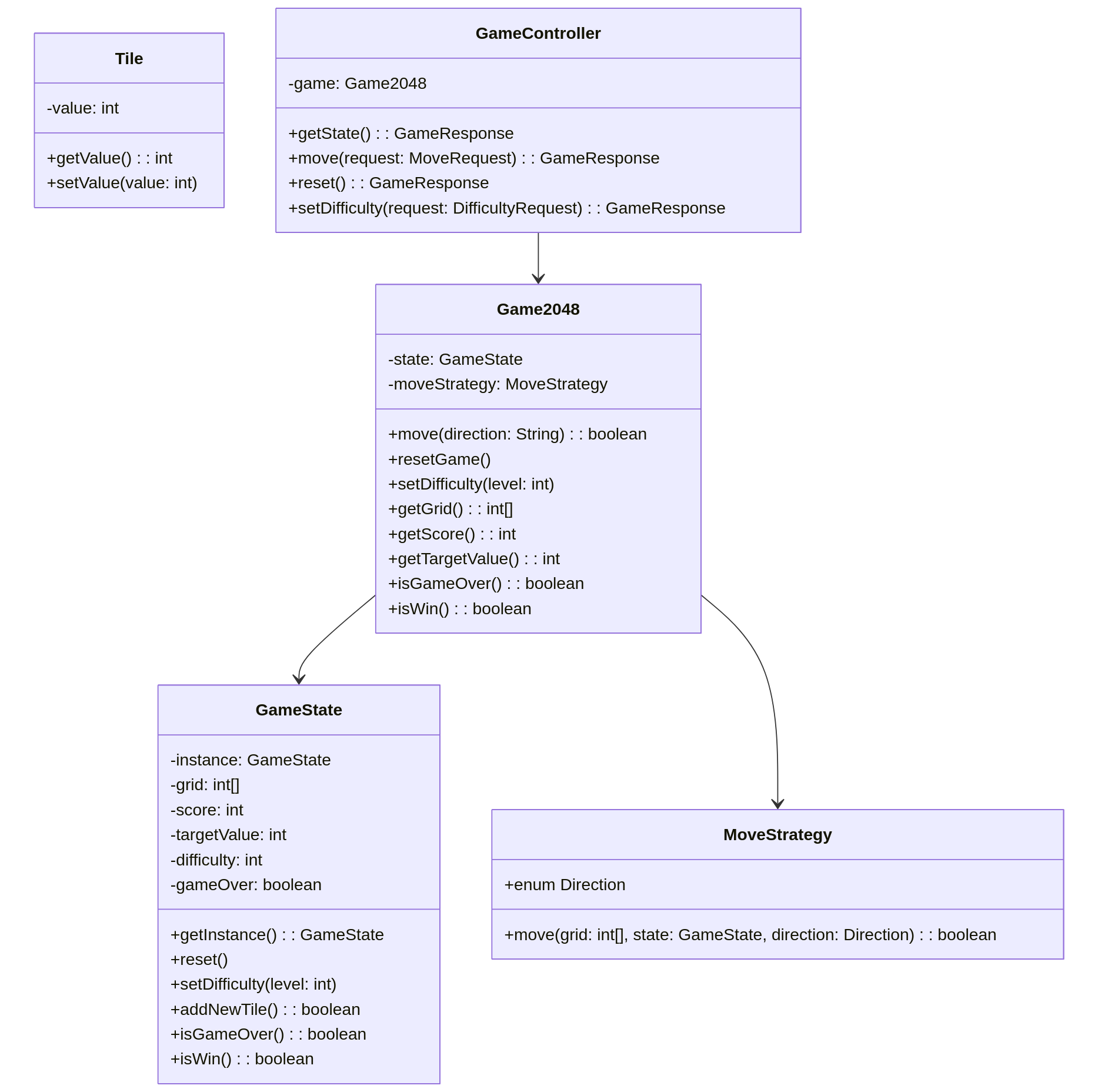


Рис. 2. UML-диаграмма классов

## **4. Выбор уровня сложности**

Была добавлена возможность выбирать уровень сложности (от 1 до 10), где цель — достичь плитки от 16 до 8192. Это делает игру гибкой: новички могут стартовать с легкого уровня, а опытные игроки — бросить себе вызов.

При выборе уровня фронтенд отправляет запрос на /api/difficulty. Сервер пересчитывает цель (2^(3+уровень)) и сбрасывает игру. Игрок видит новую цель в интерфейсе.

Когда игра тестировалась, было замечено, что на высоких уровнях (например, 8192) поле заполняется быстрее, чем ожидаешь. Это заставляет быть осторожнее с каждым ходом, как будто играешь в шахматы с таймером.

## **5. Использование ООП и паттернов**

В программе применены несколько принципов ООП, чтобы код был понятным и легко расширяемым:

* Инкапсуляция: Данные игры (поле, счет) спрятаны в GameState, доступ возможен только через методы.
* Одиночка (Singleton): GameState гарантирует, что состояние игры едино для всего приложения.
* Стратегия (Strategy): MoveStrategy обрабатывает все направления, просто меняя параметр. Это избавило нас от дублирования кода.
* Разделение ответственности: GameController занимается запросами, Game2048 — логикой, а фронтенд — отображением.

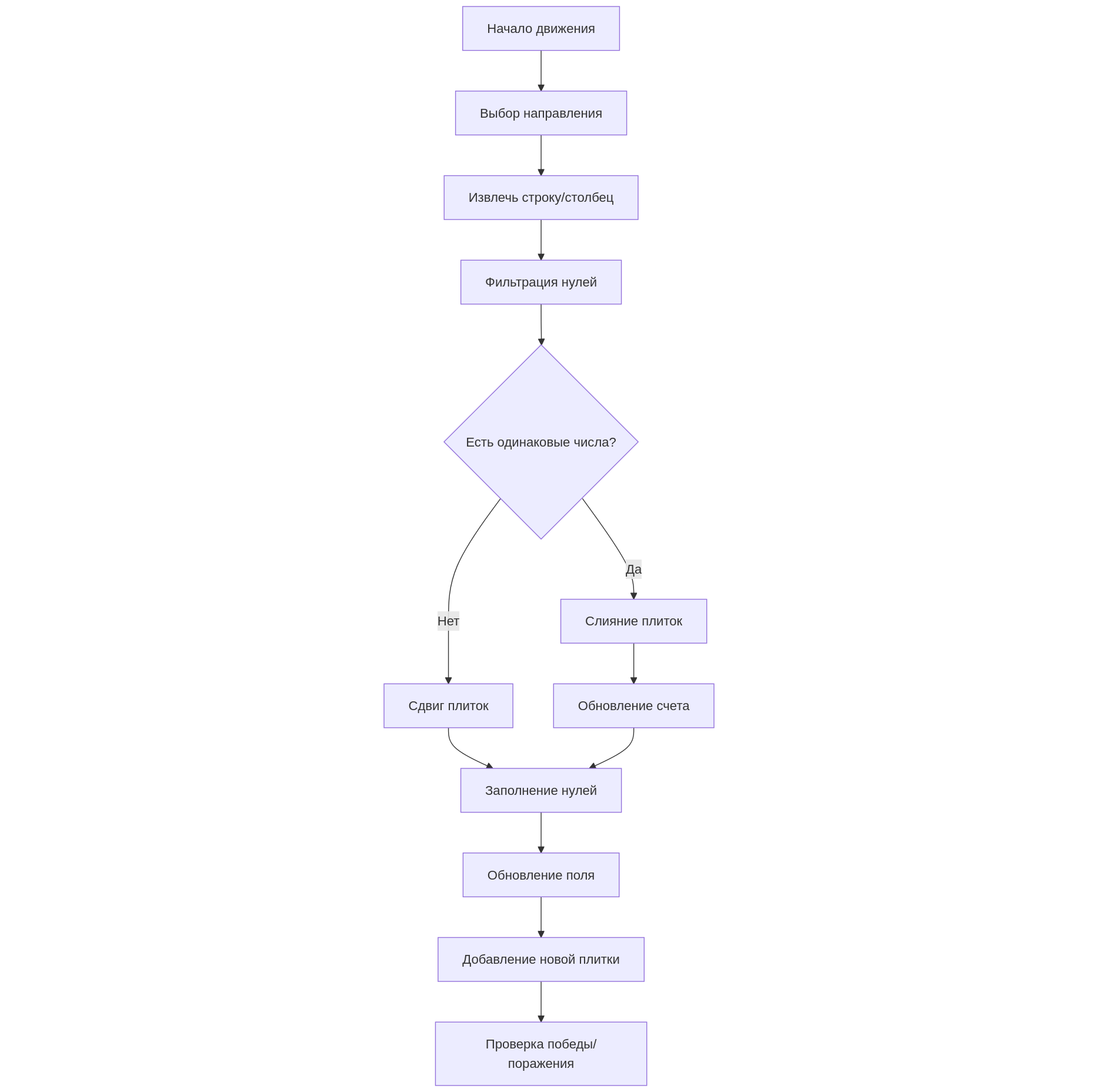


Рис. 3. Схема алгоритма движения

# **Заключение**

Создание игры «2048» на Java — это как собрать пазл, где каждая деталь важна. Нужно было разбивать задачу на части, использовать ООП, чтобы код был чистым, и настроить клиент-серверное приложение, которое работает в браузере. Были добавлены уровни сложности, чтобы каждый мог найти свою игру. Этот проект показывает, как простая идея может стать сложной системой, если подойти к ней с умом.

# **Список источников**

1. Блинов, И. Н. Информатика и программирование : учебник / И. Н. Блинов, В. М. Кириченко. — Москва : Академия, 2020. — 352 с. — ISBN 978-5-4468-0920-1.
2. Кейн, Г. Разработка приложений на Java. Полный курс / Герберт Шилдт. — Санкт-Петербург : BHV-Петербург, 2021. — 848 с. — ISBN 978-5-93118-845-7.
3. Кучерявый, А. А. Основы программирования на Java / А. А. Кучерявый. — Москва : ДМК Пресс, 2019. — 328 с. — ISBN 978-5-97060-720-4.
4. Spring Framework Documentation [Электронный ресурс] : официальная документация. — URL: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/> (дата обращения: 01.04.2025).
5. Thymeleaf Official Documentation [Электронный ресурс]. — URL: <https://www.thymeleaf.org/documentation.html> (дата обращения: 01.04.2025).   
   Официальный сайт игры «2048» [Электронный ресурс]. — URL: https://play2048.co/ (дата обращения: 26.05.2025).
6. Документация Spring Boot [Электронный ресурс]. — URL: https://spring.io/projects/spring-boot (дата обращения: 26.05.2025).
7. Фримен, Э., Фримен, Э., Сьерра, К., Бейтс, Б. Паттерны проектирования / Эрик Фримен, Элизабет Фримен, Кэти Сьерра, Берт Бейтс; пер. с англ. — 2-е изд. — СПб.: Питер, 2011. — 656 с.
8. Руководство по CSS Grid [Электронный ресурс]. — URL: https://css-tricks.com/snippets/css/complete-guide-grid/ (дата обращения: 26.05.2025).

# **Приложение**

**src/main/java/com/example/game/Game2048.java**

package com.example.game;

public class Game2048 {

private final GameState state;

private final MoveStrategy moveStrategy;

public Game2048() {

state = GameState.getInstance();

moveStrategy = new MoveStrategy();

}

public boolean move(String direction) {

if (state.isGameOverFlag()) return false;

MoveStrategy.Direction moveDirection;

switch (direction) {

case "ArrowLeft":

moveDirection = MoveStrategy.Direction.LEFT;

break;

case "ArrowRight":

moveDirection = MoveStrategy.Direction.RIGHT;

break;

case "ArrowUp":

moveDirection = MoveStrategy.Direction.UP;

break;

case "ArrowDown":

moveDirection = MoveStrategy.Direction.DOWN;

break;

default:

return false;

}

boolean moved = moveStrategy.move(state.getGrid(), state, moveDirection);

if (moved) {

state.addNewTile();

}

return moved;

}

public void resetGame() {

state.reset();

}

public void setDifficulty(int level) {

state.setDifficulty(level);

}

public int[] getGrid() {

return state.getGrid();

}

public int getScore() {

return state.getScore();

}

public int getTargetValue() {

return state.getTargetValue();

}

public boolean isGameOver() {

return state.isGameOver();

}

public boolean isWin() {

return state.isWin();

}

}

**src/main/java/com/example/game/GameController.java**

package com.example.game;

public class Game2048 {

private final GameState state;

private final MoveStrategy moveStrategy;

public Game2048() {

state = GameState.getInstance();

moveStrategy = new MoveStrategy();

}

public boolean move(String direction) {

if (state.isGameOverFlag()) return false;

MoveStrategy.Direction moveDirection;

switch (direction) {

case "ArrowLeft":

moveDirection = MoveStrategy.Direction.LEFT;

break;

case "ArrowRight":

moveDirection = MoveStrategy.Direction.RIGHT;

break;

case "ArrowUp":

moveDirection = MoveStrategy.Direction.UP;

break;

case "ArrowDown":

moveDirection = MoveStrategy.Direction.DOWN;

break;

default:

return false;

}

boolean moved = moveStrategy.move(state.getGrid(), state, moveDirection);

if (moved) {

state.addNewTile();

}

return moved;

}

public void resetGame() {

state.reset();

}

public void setDifficulty(int level) {

state.setDifficulty(level);

}

public int[] getGrid() {

return state.getGrid();

}

public int getScore() {

return state.getScore();

}

public int getTargetValue() {

return state.getTargetValue();

}

public boolean isGameOver() {

return state.isGameOver();

}

public boolean isWin() {

return state.isWin();

}

}

**src/main/java/com/example/game/GameState.java**

package com.example.game;

public class Game2048 {

private final GameState state;

private final MoveStrategy moveStrategy;

public Game2048() {

state = GameState.getInstance();

moveStrategy = new MoveStrategy();

}

public boolean move(String direction) {

if (state.isGameOverFlag()) return false;

MoveStrategy.Direction moveDirection;

switch (direction) {

case "ArrowLeft":

moveDirection = MoveStrategy.Direction.LEFT;

break;

case "ArrowRight":

moveDirection = MoveStrategy.Direction.RIGHT;

break;

case "ArrowUp":

moveDirection = MoveStrategy.Direction.UP;

break;

case "ArrowDown":

moveDirection = MoveStrategy.Direction.DOWN;

break;

default:

return false;

}

boolean moved = moveStrategy.move(state.getGrid(), state, moveDirection);

if (moved) {

state.addNewTile();

}

return moved;

}

public void resetGame() {

state.reset();

}

public void setDifficulty(int level) {

state.setDifficulty(level);

}

public int[] getGrid() {

return state.getGrid();

}

public int getScore() {

return state.getScore();

}

public int getTargetValue() {

return state.getTargetValue();

}

public boolean isGameOver() {

return state.isGameOver();

}

public boolean isWin() {

return state.isWin();

}

}

**src/main/java/com/example/game/MoveStrategy.java**

package com.example.game;

import java.util.ArrayList;

import java.util.List;

public class MoveStrategy {

public enum Direction { LEFT, RIGHT, UP, DOWN }

public boolean move(int[] grid, GameState state, Direction direction) {

boolean moved = false;

if (direction == Direction.LEFT) {

moved = moveLeft(grid, state);

} else if (direction == Direction.RIGHT) {

moved = moveRight(grid, state);

} else if (direction == Direction.UP) {

moved = moveUp(grid, state);

} else if (direction == Direction.DOWN) {

moved = moveDown(grid, state);

}

return moved;

}

private boolean moveLeft(int[] grid, GameState state) {

boolean moved = false;

for (int row = 0; row < 4; row++) {

int[] line = new int[4];

for (int col = 0; col < 4; col++) {

line[col] = grid[row \* 4 + col];

}

int[] result = processLine(line, state);

for (int col = 0; col < 4; col++) {

int index = row \* 4 + col;

if (grid[index] != result[col]) {

moved = true;

grid[index] = result[col];

}

}

}

return moved;

}

private boolean moveRight(int[] grid, GameState state) {

boolean moved = false;

for (int row = 0; row < 4; row++) {

int[] line = new int[4];

for (int col = 0; col < 4; col++) {

line[col] = grid[row \* 4 + (3 - col)];

}

int[] result = processLine(line, state);

for (int col = 0; col < 4; col++) {

int index = row \* 4 + (3 - col);

if (grid[index] != result[col]) {

moved = true;

grid[index] = result[col];

}

}

}

return moved;

}

private boolean moveUp(int[] grid, GameState state) {

boolean moved = false;

for (int col = 0; col < 4; col++) {

int[] line = new int[4];

for (int row = 0; row < 4; row++) {

line[row] = grid[row \* 4 + col];

}

int[] result = processLine(line, state);

for (int row = 0; row < 4; row++) {

int index = row \* 4 + col;

if (grid[index] != result[row]) {

moved = true;

grid[index] = result[row];

}

}

}

return moved;

}

private boolean moveDown(int[] grid, GameState state) {

boolean moved = false;

for (int col = 0; col < 4; col++) {

int[] line = new int[4];

for (int row = 0; row < 4; row++) {

line[row] = grid[(3 - row) \* 4 + col];

}

int[] result = processLine(line, state);

for (int row = 0; row < 4; row++) {

int index = (3 - row) \* 4 + col;

if (grid[index] != result[row]) {

moved = true;

grid[index] = result[row];

}

}

}

return moved;

}

private int[] processLine(int[] line, GameState state) {

List<Integer> filtered = new ArrayList<>();

for (int value : line) {

if (value != 0) filtered.add(value);

}

List<Integer> merged = new ArrayList<>();

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

if (i + 1 < filtered.size() && filtered.get(i).equals(filtered.get(i + 1))) {

int newValue = filtered.get(i) \* 2;

merged.add(newValue);

state.setScore(state.getScore() + newValue);

i++;

} else {

merged.add(filtered.get(i));

}

}

int[] result = new int[4];

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

result[i] = merged.get(i);

}

return result;

}

}

**src/main/java/com/example/game/Tile.java**

package com.example.game;

public class Tile {

private int value;

public Tile(int value) {

this.value = value;

}

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

}

**src/main/java/com/example/Application.java**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**src/main/resources/static/index.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Игра 2048</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<!-- Меню выбора сложности -->

<div id="difficulty-menu" class="menu-container">

<h1>2048</h1>

<div class="menu">

<h2>Выберите уровень сложности</h2>

<div class="difficulty-levels">

<button data-level="1" class="difficulty-btn">1 (Цель: 16)</button>

<button data-level="2" class="difficulty-btn">2 (Цель: 32)</button>

<button data-level="3" class="difficulty-btn">3 (Цель: 64)</button>

<button data-level="4" class="difficulty-btn">4 (Цель: 128)</button>

<button data-level="5" class="difficulty-btn">5 (Цель: 256)</button>

<button data-level="6" class="difficulty-btn">6 (Цель: 512)</button>

<button data-level="7" class="difficulty-btn">7 (Цель: 1024)</button>

<button data-level="8" class="difficulty-btn">8 (Цель: 2048)</button>

<button data-level="9" class="difficulty-btn">9 (Цель: 4096)</button>

<button data-level="10" class="difficulty-btn">10 (Цель: 8192)</button>

</div>

</div>

</div>

<!-- Игровой интерфейс -->

<div id="game-container" class="container" style="display: none;">

<h1>2048</h1>

<div class="score-container">

<span>Счёт: </span>

<span id="score">0</span>

</div>

<div class="grid-container">

<div class="grid"></div>

</div>

<button id="new-game">Новая игра</button>

</div>

<script>

class Game2048Client {

constructor() {

this.gridElement = document.querySelector('.grid');

this.scoreElement = document.getElementById('score');

this.gameContainer = document.getElementById('game-container');

this.difficultyMenu = document.getElementById('difficulty-menu');

this.createGridCells();

this.setupEventListeners();

}

createGridCells() {

this.gridElement.innerHTML = '';

for (let i = 0; i < 16; i++) {

const cell = document.createElement('div');

cell.className = 'tile';

this.gridElement.appendChild(cell);

}

this.cells = Array.from(this.gridElement.children);

}

setupEventListeners() {

document.querySelectorAll('.difficulty-btn').forEach(btn => {

btn.addEventListener('click', () => {

const level = parseInt(btn.dataset.level);

fetch('/api/difficulty', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ level })

})

.then(response => response.json())

.then(data => {

this.difficultyMenu.style.display = 'none';

this.gameContainer.style.display = 'block';

this.updateDisplay(data);

});

});

});

document.getElementById('new-game').addEventListener('click', () => {

fetch('/api/reset', { method: 'POST' })

.then(response => response.json())

.then(data => {

this.difficultyMenu.style.display = 'block';

this.gameContainer.style.display = 'none';

this.updateDisplay(data);

});

});

document.addEventListener('keydown', (e) => {

if (['ArrowUp', 'ArrowDown', 'ArrowLeft', 'ArrowRight'].includes(e.key)) {

e.preventDefault();

this.handleKeyPress(e.key);

}

});

}

handleKeyPress(direction) {

fetch('/api/move', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ direction })

})

.then(response => response.json())

.then(data => {

this.updateDisplay(data);

if (data.win) {

setTimeout(() => {

alert(`Вы выиграли! Счёт: ${data.score}`);

this.difficultyMenu.style.display = 'block';

this.gameContainer.style.display = 'none';

}, 100);

} else if (data.gameOver) {

setTimeout(() => {

alert(`Игра окончена! Ваш счёт: ${data.score}`);

this.difficultyMenu.style.display = 'block';

this.gameContainer.style.display = 'none';

}, 100);

}

});

}

updateDisplay(data) {

data.grid.forEach((value, index) => {

const cell = this.cells[index];

cell.textContent = value || '';

cell.className = 'tile';

if (value > 0) {

cell.classList.add(`tile-${value}`);

}

});

this.scoreElement.textContent = data.score;

}

}

new Game2048Client();

</script>

</body>

</html>

**src/main/resources/static/style.css**

/\* Общие стили \*/

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

background-color: #faf8ef;

font-family: Arial, sans-serif;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

}

/\* Стили для меню выбора сложности \*/

.menu-container {

text-align: center;

background: #faf8ef;

padding: 20px;

border-radius: 10px;

}

.menu {

background: #bbada0;

padding: 20px;

border-radius: 8px;

margin: 20px auto;

max-width: 500px;

}

.menu h2 {

color: #776e65;

margin-bottom: 20px;

}

.difficulty-levels {

display: grid;

grid-template-columns: repeat(2, 1fr);

gap: 10px;

}

.difficulty-btn {

background: #8f7a66;

color: #f9f6f2;

border: none;

padding: 15px;

font-size: 16px;

border-radius: 4px;

cursor: pointer;

transition: background 0.2s;

}

.difficulty-btn:hover {

background: #9f8b77;

}

/\* Остальные стили (из вашего CSS) \*/

.container {

text-align: center;

}

h1 {

color: #776e65;

font-size: 48px;

margin-bottom: 20px;

}

.score-container {

font-size: 24px;

color: #776e65;

margin-bottom: 20px;

}

.grid-container {

background: #bbada0;

padding: 15px;

border-radius: 8px;

margin-bottom: 20px;

}

.grid {

display: grid;

grid-template-columns: repeat(4, 1fr);

grid-gap: 15px;

background: #bbada0;

padding: 15px;

border-radius: 8px;

width: 500px;

height: 500px;

}

.tile {

background: #cdc1b4;

border-radius: 4px;

font-size: 36px;

font-weight: bold;

display: flex;

justify-content: center;

align-items: center;

color: #776e65;

transition: all 0.15s ease;

}

/\* Стили для плиток с числами \*/

.tile-2 { background: #eee4da; }

.tile-4 { background: #ede0c8; }

.tile-8 { background: #f2b179; color: #f9f6f2; }

.tile-16 { background: #f59563; color: #f9f6f2; }

.tile-32 { background: #f67c5f; color: #f9f6f2; }

.tile-64 { background: #f65e3b; color: #f9f6f2; }

.tile-128 { background: #edcf72; color: #f9f6f2; font-size: 32px; }

.tile-256 { background: #edcc61; color: #f9f6f2; font-size: 32px; }

.tile-512 { background: #edc850; color: #f9f6f2; font-size: 32px; }

.tile-1024 { background: #edc53f; color: #f9f6f2; font-size: 28px; }

.tile-2048 { background: #edc22e; color: #f9f6f2; font-size: 28px; }

.tile-4096 { background: #3c3a32; color: #f9f6f2; font-size: 28px; }

.tile-8192 { background: #000000; color: #f9f6f2; font-size: 24px; }

#new-game {

background: #8f7a66;

color: #f9f6f2;

border: none;

padding: 10px 20px;

font-size: 18px;

border-radius: 4px;

cursor: pointer;

}

#new-game:hover {

opacity: 0.9;

}

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>game2048</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<java.version>17</java.version>

<spring.boot.version>3.1.5</spring.boot.version>

</properties>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.5</version>

<relativePath/>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

# **Демонстрация программы**

Скриншоты работы игры представлены на рисунках 4, 5.



Рис. 4. Меню выбора сложности игры

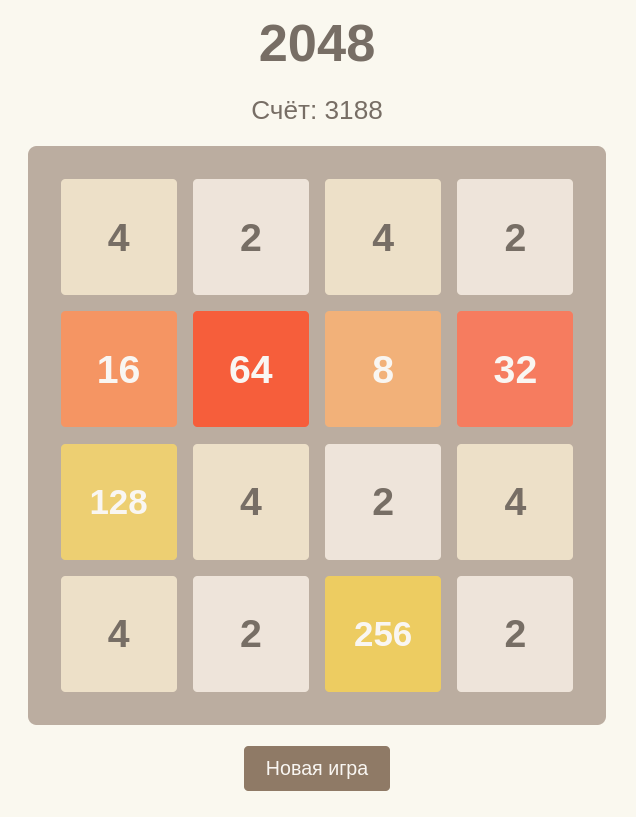


Рис. 5. Демонстрация игры