

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Объектно-ориентированное программирование»
на тему «Разработка многомодульного приложения на языке Java»

28.05.26
Ходяков
ф.и.о.

Выполнил:
Студент группы 23ВВИ1
Макаров А.С.

Приняли:
к.т.н доцент Юрова О.В.
к.т.н доцент Карамышева Н.С.

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М. А. Митрохин

«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Разработка клиентских приложений

Студенту Макарову Алексею Сергеевичу Группа 23ВВИ1

Тема проекта Разработка многомодульного приложения на языке Java

Исходные данные (технические требования) на проектирование

Разработать игровое приложение "Настольный"

Язык программирования Java, среда разработки NetBeans.

Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. Java Stream API

4. Java Multithreading

5. Сетевое взаимодействие.

Объем работы по курсу

1. Расчетная часть

2. Графическая часть

Диаграммы языка UML:

диаграмма вариантов использования;

диаграмма классов;

диаграмма деятельности;

диаграмма развёртывания;

диаграмма последовательности.

3. Экспериментальная часть

Тестирование многомодульного приложения

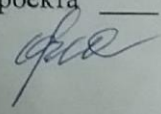
Срок выполнения проекта по разделам

1	
2	
3	
4	
5	
6	
7	
8	

Дата выдачи задания "14" февраля

Дата защиты проекта " " "

Руководитель Юрова Ольга Викторовна



Задание получил "05" 03

2025г.

Студент Макаров Алексей Сергеевич



Содержание

Введение.....	7
1 Постановка задачи.....	8
2 Теоретическая часть задания	10
2.1 Язык программирования Java	10
2.2 Среда разработки Apache NetBeans IDE	11
2.3 Игра «Тетрис».....	11
2.4 Используемые технологии и структура проекта	12
3 Описание алгоритма работы	12
3.1 Общая логика приложения.....	12
3.2 Игровой цикл	14
3.3 Генерация фигур и обработка линий	14
3.4 Работа многопоточности	15
3.5 Сетевое взаимодействие	16
4 Описание программы.....	20
4.1 Класс GameBoard.....	22
4.2 Класс Tetromino и перечисление TetrominoType	22
4.3 Класс TetrisFrame	23
4.4 Классы NetworkDialog и NetworkManager.....	24
4.5 Класс GameStatePacket.....	24
5 Руководство пользователя.....	25
6 Тестирование	29
Заключение	31
Список литературы	32
Приложение А. Листинг программы.....	33
Приложение Б. UML-диаграммы	59

Введение

На сегодняшний момент язык Java является одним из наиболее распространенных языков программирования для разработки учебных, прикладных и сетевых приложений. Его применение в курсовой работе обосновано тем, что язык сочетает объектно-ориентированную модель, развитые стандартные библиотеки и возможность запуска программ на разных операционных системах без изменения исходного кода.

Ключевой особенностью Java является выполнение программного кода виртуальной машиной JVM. Исходный код сначала компилируется в байт-код, после чего этот байт-код выполняется в среде выполнения Java. Такой подход обеспечивает переносимость приложения между Windows, Linux и macOS при наличии установленной JRE или JDK.

В данной курсовой работе разработано многомодульное приложение «Тетрис» с графическим интерфейсом пользователя. Программа поддерживает одиночный режим, локальный мультиплеер на одном компьютере и сетевой режим взаимодействия между двумя игроками. В процессе разработки были использованы Java Collections Framework, Java Stream API, механизмы обработки исключений, библиотека Swing, таймеры пользовательского интерфейса и сетевое взаимодействие через TCP/IP сокет.

Отдельное внимание уделено соответствию структуры отчета программному коду. Поэтому в основной части пояснительной записки добавлены UML-диаграммы, отражающие варианты использования, классы, последовательность взаимодействия объектов, алгоритм работы и развертывание приложения. Дублирование диаграмм в приложении оставлено для удобства просмотра и печати.

1 Постановка задачи

Целью курсовой работы является разработка многомодульного приложения «Тетрис» на языке Java с графическим интерфейсом пользователя и несколькими режимами игры. Программа должна демонстрировать применение принципов объектно-ориентированного программирования, разделение ответственности между классами, работу с коллекциями, потоками, событиями интерфейса и сетевыми подключениями.

1. Для достижения цели необходимо выполнить следующие задачи:
создать игровое поле размером 10 на 20 клеток и реализовать хранение состояния каждой клетки;
2. реализовать набор фигур тетриса, их случайную генерацию, перемещение, поворот и фиксацию на поле;
3. обеспечить подсчет очков, учет очищенных линий и повышение уровня сложности;
4. разработать графический интерфейс с отображением текущего поля, следующей фигуры, счета, уровня и статистики соперника;
5. реализовать одиночный режим, локальный мультиплеер и сетевое взаимодействие между игроками;
6. применить многопоточность для обработки сетевых соединений без блокировки интерфейса;
7. использовать коллекции Java, Stream API и механизм исключений для построения устойчивой логики приложения;
8. провести тестирование основных функций программы.
9. Диаграмма вариантов использования показывает, какие действия доступны пользователю и второму игроку, а также какие внутренние варианты запуска используются при выборе сетевого режима.

Диаграмма к разделу 1

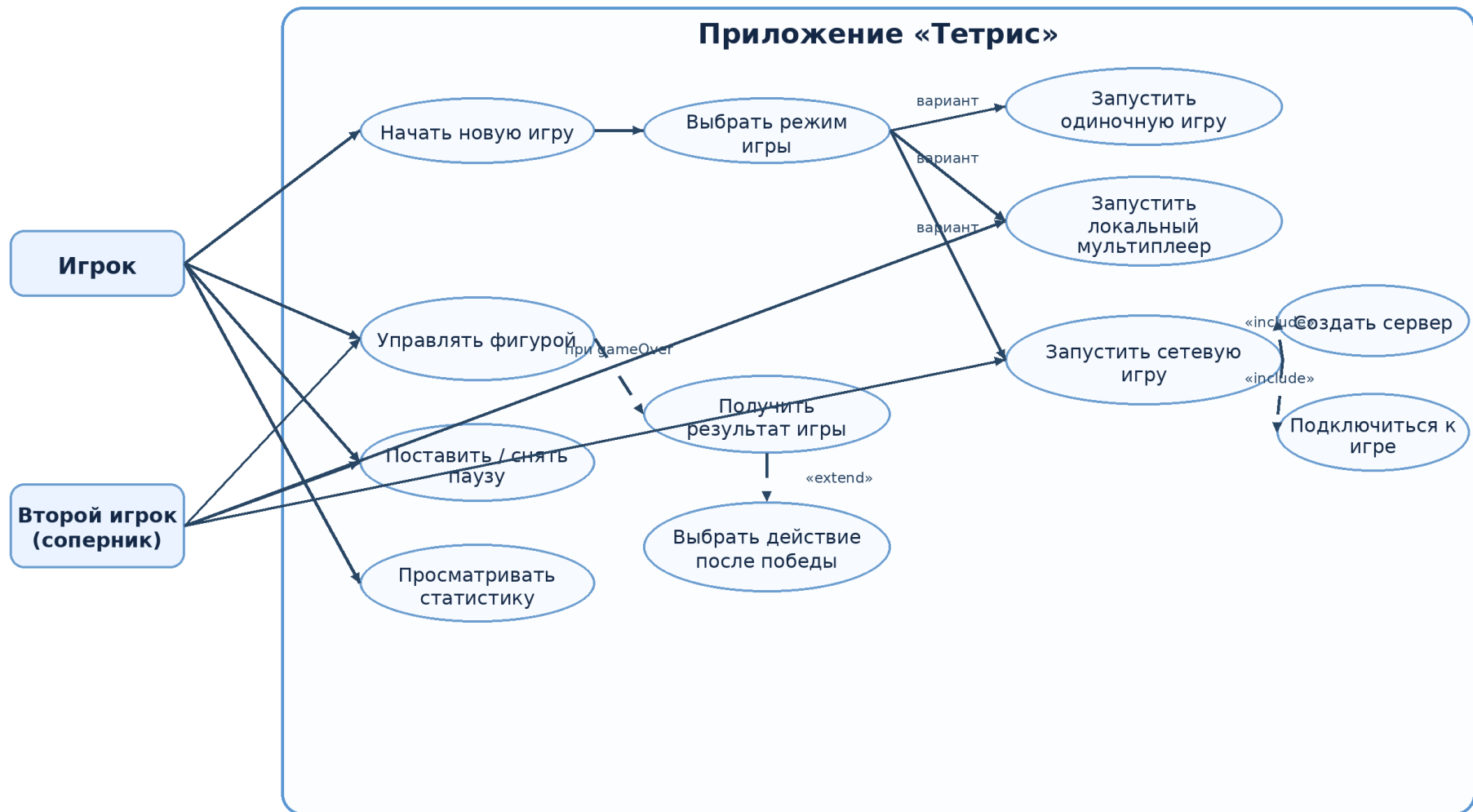


Рисунок 1 - UML-диаграмма вариантов использования приложения

2 Теоретическая часть задания

2.1 Язык программирования Java

Java представляет собой объектно-ориентированный язык программирования с синтаксисом, близким к C, C++ и C#. Основу проектирования на Java составляют классы и объекты. Класс описывает данные и поведение объекта, а объект является экземпляром класса, который используется во время выполнения программы.

В данной работе объектно-ориентированный подход выражен в разделении приложения на несколько логических частей. Игровое поле реализовано в классе GameBoard, фигуры описаны классом Tetromino и перечислением TetrominoType, графический интерфейс сосредоточен в TetrisFrame, а сетевой обмен вынесен в NetworkManager и GameStatePacket. Такое разделение упрощает сопровождение программы, так как изменение одной части не требует переписывания всей системы.

Java поддерживает инкапсуляцию, наследование, полиморфизм и обработку исключительных ситуаций. Инкапсуляция используется для защиты внутреннего состояния классов, например поле grid класса GameBoard не изменяется напрямую извне. Исключение GameOverException применяется для фиксации завершения партии, когда новая фигура не может быть размещена на игровом поле.

Стандартная библиотека Java предоставляет готовые средства для создания интерфейса, работы с коллекциями, потоками ввода-вывода, сетью и многопоточностью. Это позволило выполнить проект без подключения внешних библиотек и оставить программу достаточно простой для анализа в рамках курсовой работы.

2.2 Среда разработки Apache NetBeans IDE

Apache NetBeans IDE является интегрированной средой разработки, предназначенной для создания, компиляции, запуска и отладки Java-приложений. В рамках курсовой работы среда использовалась для организации проекта, редактирования исходных файлов, сборки программы и проверки работы графического интерфейса.

NetBeans удобна для разработки Swing-приложений, так как предоставляет визуальный редактор форм и автоматически формирует часть кода, связанную с размещением элементов интерфейса. В проекте это особенно важно для класса TetrisFrame, который содержит игровую панель, панель статистики, блок отображения следующей фигуры и кнопки управления.

Среда разработки также позволяет быстро находить ошибки компиляции, переходить между классами и проверять структуру пакетов. В проекте классы разделены по пакетам: `kursovaya.core` содержит игровую логику, `kursovaya.gui` содержит графический интерфейс, а `kursovaya.network` отвечает за сетевую часть приложения.

2.3 Игра «Тетрис»

Тетрис является логической игрой, в которой пользователь управляет падающими фигурами, состоящими из четырех квадратных блоков. Игрок перемещает фигуры влево и вправо, вращает их и старается заполнить горизонтальные линии игрового поля. Когда линия полностью заполнена, она удаляется, а блоки выше сдвигаются вниз.

Основная сложность игры состоит в том, что скорость падения фигур постепенно увеличивается, а пространство на поле ограничено. При неправильном размещении фигур свободное место быстро заканчивается. Партия завершается, если новая фигура не может появиться в верхней части поля.

В разработанном приложении реализованы стандартные фигуры I, O, T, S, Z, J и L. Для генерации фигур используется алгоритм 7-Bag: все семь

типов складываются в набор, перемешиваются, после чего выдаются по одному. Когда набор заканчивается, он заполняется и перемешивается снова. Благодаря этому фигуры появляются равномернее, чем при обычном случайном выборе.

2.4 Используемые технологии и структура проекта

Проект состоит из нескольких модулей, сгруппированных по назначению. Такой подход позволяет отделить правила игры от интерфейса и сетевой логики. В результате код проще читать, тестировать и расширять.

Пакет / класс	Назначение
kursovaya.Kursovaya	Точка входа в приложение, запуск TetrisFrame.
kursovaya.core	Игровая логика: поле, клетки, фигуры, счет, уровень, исключение завершения игры.
kursovaya.gui	Окна интерфейса, обработка клавиатуры, отрисовка поля, диалоги сетевой игры.
kursovaya.network	Создание TCP/IP соединения, отправка и прием объектов GameStatePacket.
Java Swing	Построение графического интерфейса и обработка событий пользователя.
Java Collections Framework	Хранение поля, набора фигур, списка наблюдателей и очереди 7-Bag.
ExecutorService	Запуск фоновых задач приема сетевых сообщений.

В качестве основной архитектурной идеи используется разделение на модель, представление и сетевой обмен. GameBoard отвечает за состояние игры, TetrisFrame отображает это состояние пользователю, а NetworkManager передает состояние между игроками в сетевом режиме.

3 Описание алгоритма работы

3.1 Общая логика приложения

После запуска программы создается главное окно TetrisFrame. В окне формируется игровая панель, блок статистики, область просмотра следующей фигуры и кнопки управления. При создании нового экземпляра GameBoard выполняется инициализация сетки размером 10 на 20 клеток, установка начального счета, уровня и количества очищенных линий.

Игрок выбирает режим работы: одиночный, локальный или сетевой. В одиночном режиме создается одно окно игры. В локальном режиме

создаются два окна TetrisFrame, каждому игроку назначается свой набор клавиш управления. В сетевом режиме один пользователь может создать сервер, а второй подключиться к нему по адресу и порту.

Общая логика работы показана на диаграмме деятельности. Она отражает запуск режима, инициализацию поля, обработку ввода, проверку допустимости движения, фиксацию фигуры, очистку линий и завершение партии.

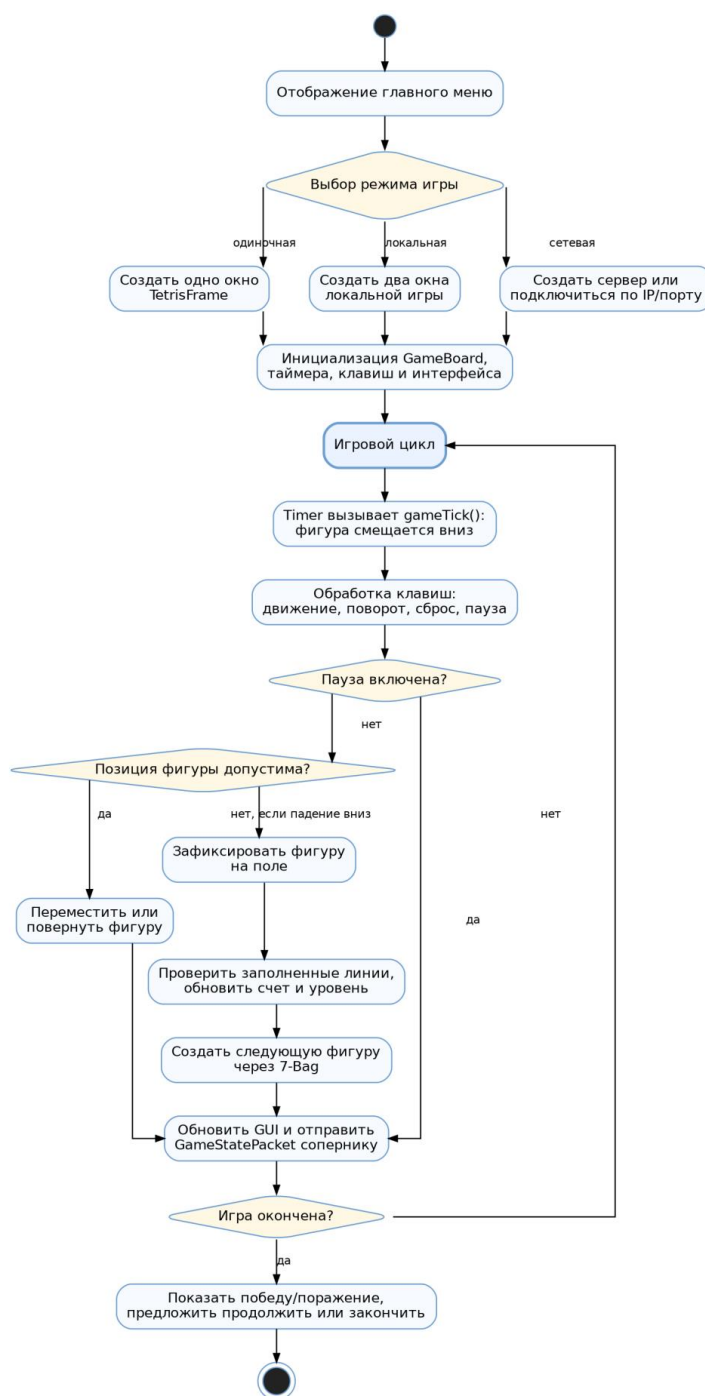


Рисунок 2 - UML-диаграмма деятельности игрового процесса

3.2 Игровой цикл

Игровой цикл реализован с помощью `javax.swing.Timer`. Таймер срабатывает через заданный промежуток времени и вызывает метод `gameTick()` класса `TetrisFrame`. Если игра не находится на паузе, метод обращается к `GameBoard` и пытается переместить текущую фигуру на одну клетку вниз.

Если перемещение вниз допустимо, координаты фигуры изменяются, после чего интерфейс перерисовывается. Если перемещение невозможно, фигура считается достигшей препятствия или нижней границы поля. В этом случае она фиксируется на сетке, затем вызывается проверка заполненных линий и создается новая фигура.

Пользователь может влиять на игровой цикл с клавиатуры. Обработчик `keyPressed` вызывает `movePiece()`, `rotatePiece()` или `dropPiece()`. При нажатии клавиши паузы меняется значение переменной `paused`. В локальном и сетевом режимах состояние паузы дополнительно передается второму игроку, чтобы интерфейсы оставались согласованными.

3.3 Генерация фигур и обработка линий

Фигуры создаются в методе `createRandomPiece()`. Перед созданием проверяется очередь `pieceBag`. Если очередь пуста, метод `refillBagIfNeeded()` формирует список всех значений перечисления `TetrominoType`, перемешивает его и добавляет элементы в очередь. Далее из очереди извлекается следующий тип фигуры.

При появлении новой фигуры вызывается `spawnPiece()`. Если следующая фигура уже подготовлена, она становится текущей, а вместо нее формируется новая следующая фигура. Такой подход позволяет заранее показывать пользователю, какая фигура появится после текущей.

После фиксации фигуры на поле вызывается метод `clearLines()`. Он проходит по строкам снизу вверх и проверяет, заняты ли все клетки. Если строка заполнена, она удаляется, а все строки выше сдвигаются вниз. При

этом проверка этой же позиции повторяется, так как после сдвига на ее место могла попасть другая заполненная строка.

Количество очищенных линий за ход	Начисление очков
1 линия	100 * текущий уровень
2 линии	300 * текущий уровень
3 линии	500 * текущий уровень
4 линии	800 * текущий уровень

Уровень рассчитывается на основе общего количества очищенных линий. В программе используется правило: каждые 10 линий повышают уровень на единицу. Это влияет на отображаемую статистику и позволяет расширить проект за счет изменения скорости таймера при росте уровня.

3.4 Работа многопоточности

В приложении присутствуют несколько источников событий. Графический интерфейс работает в потоке обработки событий Swing. Таймер `javax.swing.Timer` также вызывает обработчики в этом потоке, поэтому операции изменения интерфейса выполняются безопасно для компонентов Swing.

Сетевое взаимодействие не должно блокировать интерфейс. Для этого в `NetworkManager` используется `ExecutorService`. Метод `hostGame()` запускает ожидание подключения клиента в отдельной задаче. После подключения создаются объектные потоки ввода-вывода, устанавливается признак `connected` и запускается фоновое прослушивание входящих `GameStatePacket`.

При получении пакета обработчик `stateListener` передает данные в `NetworkDialog` и `TetrisFrame`. Обновление элементов интерфейса выполняется через `SwingUtilities.invokeLater()`, что исключает прямое изменение Swing-компонентов из фонового потока.

3.5 Сетевое взаимодействие

Сетевой режим построен по клиент-серверной схеме. Один игрок нажимает кнопку создания сервера и ожидает подключения на указанном порту. Второй игрок вводит адрес сервера и подключается к нему как клиент. После установки соединения стороны обмениваются сериализуемыми объектами GameStatePacket.

Пакет состояния содержит имя игрока, счет, уровень, количество очищенных линий, признак завершения игры, признак победителя, состояние паузы и при необходимости состояние игрового поля. Такой набор данных достаточен для отображения статистики соперника и реакции на завершение партии.

Диаграмма последовательности показывает основной обмен между объектами во время игры. В ней отдельно выделен цикл, который повторяется до завершения партии.

Диаграмма к разделу 3.5

UML-диаграмма последовательности игрового процесса

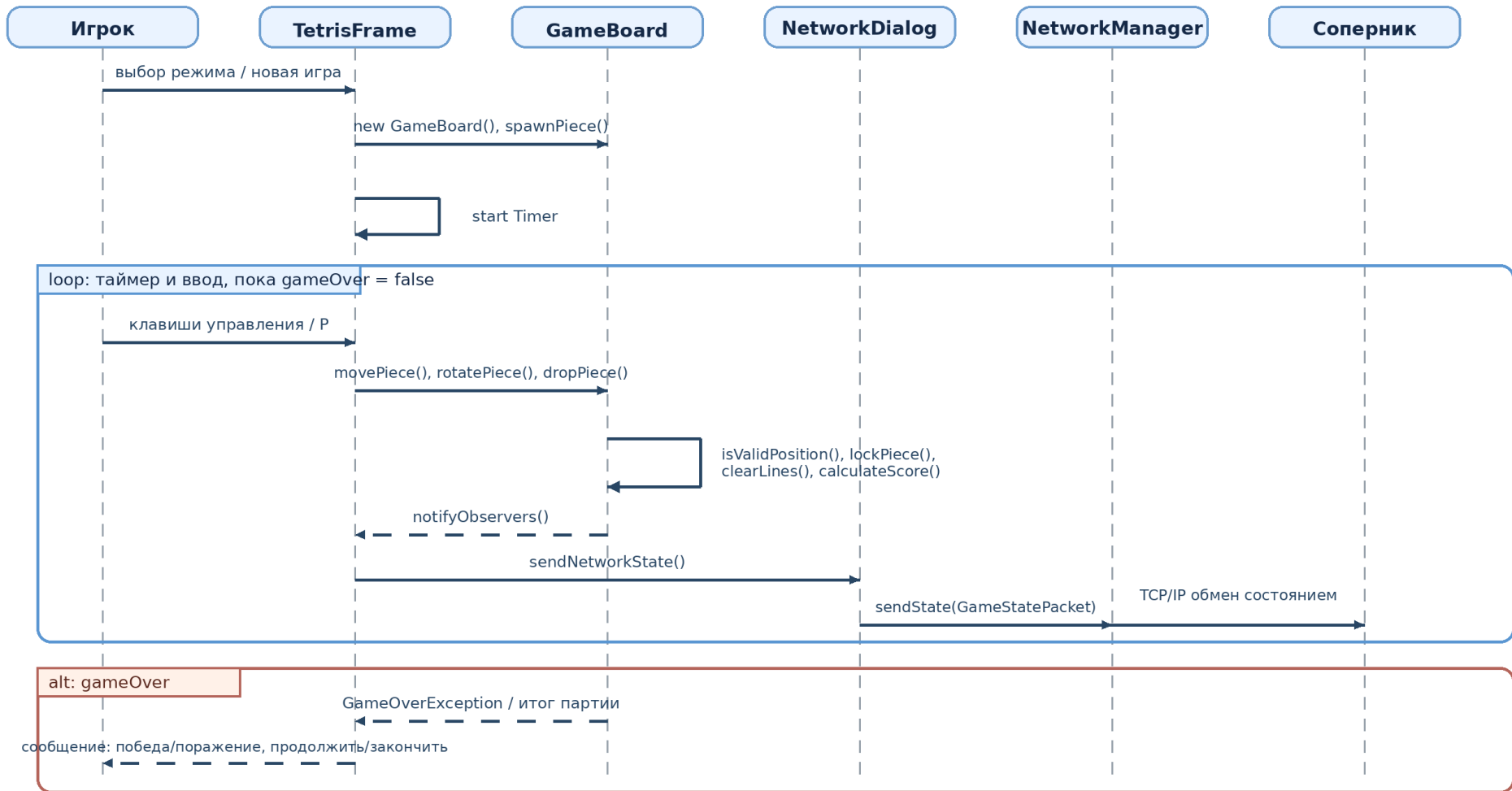


Рисунок 3 - UML-диаграмма последовательности игрового процесса

Развертывание приложения зависит от выбранного режима. В локальном режиме оба окна находятся на одном компьютере и взаимодействуют через ссылки между экземплярами TetrisFrame. В сетевом режиме каждый игрок запускает собственный экземпляр приложения, а связь между ними выполняется через TCP/IP.

Диаграмма к разделу 3.5

UML-диаграмма развертывания

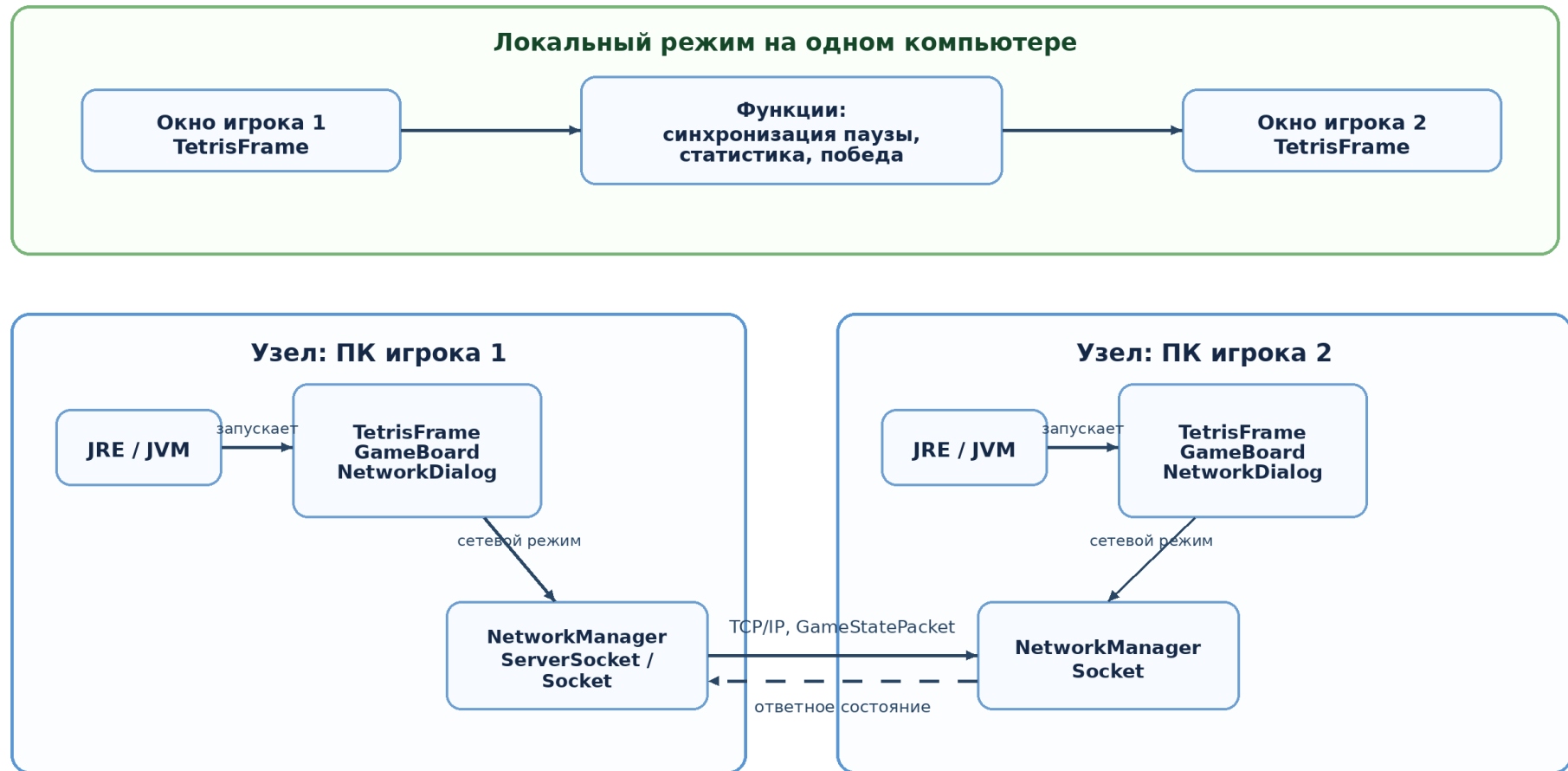


Рисунок 4 - UML-диаграмма развертывания приложения

4 Описание программы

Программа построена как набор связанных классов, каждый из которых отвечает за определенную часть функциональности. Такой подход соответствует принципу единственной ответственности: игровая логика не смешивается с кодом сетевого обмена, а интерфейс не содержит внутренней реализации фигур.

Общая структура классов приложения показана на UML-диаграмме. В диаграмму включены основные поля, методы и связи между классами, которые присутствуют в исходном коде проекта.

Диаграмма к разделу 4

UML-диаграмма классов приложения «Тетрис»

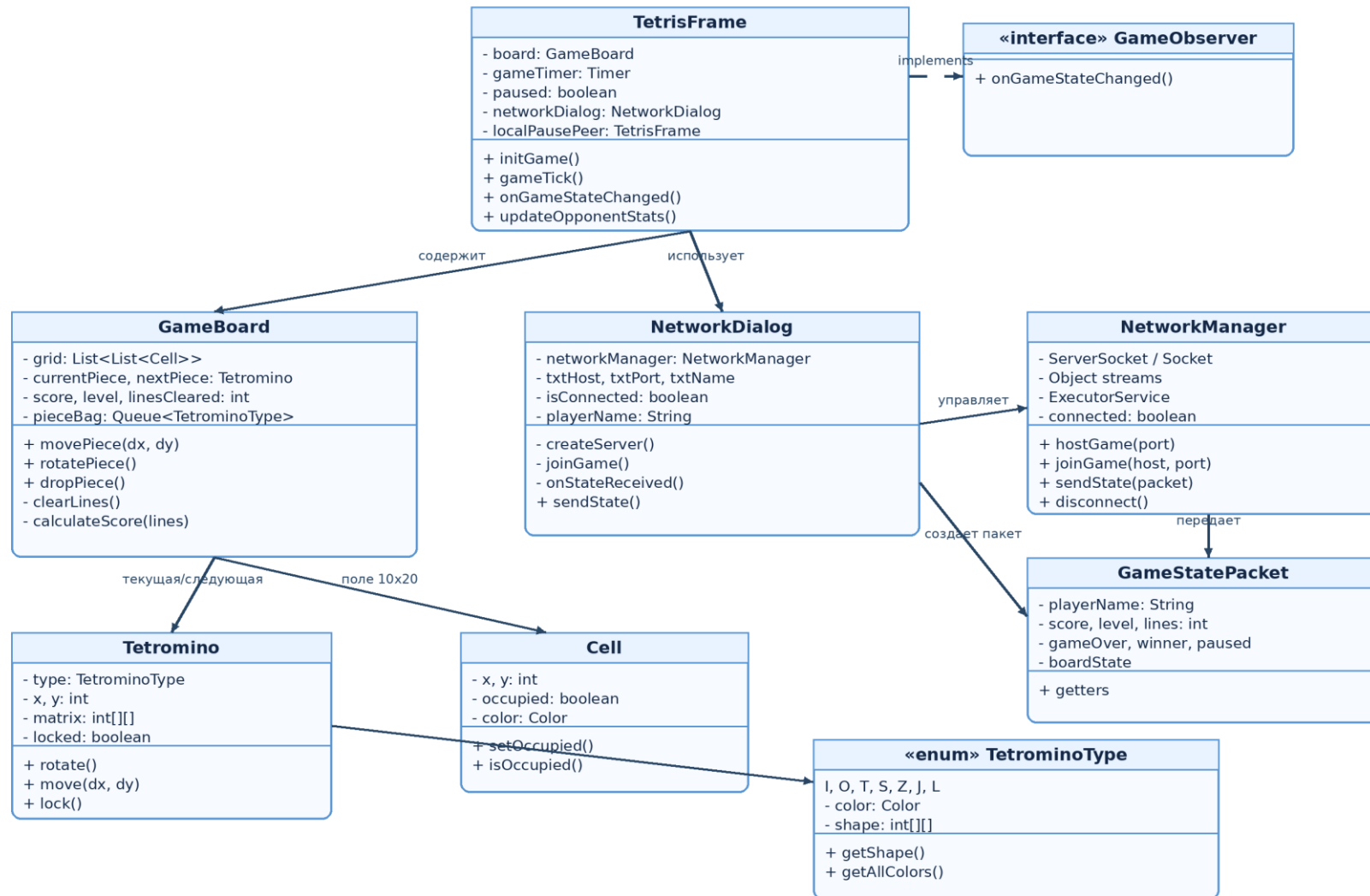


Рисунок 5 – UML-диаграмма классов приложения

4.1 Класс GameBoard

Класс GameBoard является центральным классом игровой логики. В нем хранится сетка игрового поля `grid`, текущая фигура `currentPiece`, следующая фигура `nextPiece`, счет `score`, уровень `level`, количество очищенных линий `linesCleared` и признак завершения игры `gameOver`.

Поле представлено коллекцией `List<List<Cell>>`, что позволяет обращаться к каждой клетке по координатам `x` и `y`. Каждая клетка хранит признак занятости и цвет блока. При перемещении фигуры GameBoard не меняет поле сразу, а сначала проверяет, допустима ли новая позиция. Это предотвращает выход фигуры за границы поля и наложение на уже занятые клетки.

Метод `movePiece(int dx, int dy)` выполняет перемещение на заданное смещение. Если позиция допустима, фигура сдвигается. Если фигура не может опуститься вниз, она фиксируется методом `lockPiece()`. После фиксации вызывается `clearLines()`, затем `spawnPiece()`. Если новая фигура не помещается на поле, создается исключение `GameOverException`.

Также класс GameBoard реализует механизм наблюдателей. Список `observers` содержит объекты, которые должны получать уведомления при изменении состояния игры. `TetrisFrame` подписывается на эти уведомления и обновляет интерфейс после изменений.

4.2 Класс Tetromino и перечисление TetrominoType

Класс Tetromino описывает конкретную игровую фигуру. Он хранит тип фигуры, координаты верхнего левого угла, матрицу формы и признак фиксации. Матрица содержит нули и единицы: ноль означает пустую клетку, единица - блок фигуры.

Поворот фигуры выполняется методом `rotate()`. Метод создает новую матрицу и переносит значения из исходной матрицы так, чтобы получить поворот на 90 градусов. После поворота GameBoard проверяет, можно ли оставить фигуру в новой позиции. Если поворот невозможен, выполняются

попытки небольшого сдвига влево, вправо или вниз. Если ни один вариант не подходит, матрица и координаты откатываются к старому состоянию.

Перечисление TetrominoType содержит семь стандартных типов фигур: I, O, T, S, Z, J и L. Для каждого типа задан цвет и исходная матрица формы. Метод getShape() возвращает копию матрицы, чтобы внешний код не мог случайно изменить шаблон фигуры.

4.3 Класс TetrisFrame

Класс TetrisFrame отвечает за графический интерфейс приложения. Он наследуется от JFrame и реализует интерфейс GameObserver, поэтому может получать уведомления об изменении состояния GameBoard. В классе создаются панели игрового поля, статистики, следующей фигуры и управления.

Обработка управления выполняется через KeyListener и KeyEventDispatcher. Это позволяет принимать нажатия клавиш даже тогда, когда фокус находится не строго на игровой панели. Для разных игроков в локальном мультиплеере могут использоваться разные клавиши управления.

Отрисовка поля выполняется методами drawGamePanel(), drawNextPiece() и drawCell(). Сначала очищается фон, затем рисуются занятые клетки поля, после этого поверх них отображается текущая падающая фигура. Отдельная панель показывает следующую фигуру, что делает игровой процесс понятнее.

TetrisFrame также обрабатывает завершение игры. Если партия закончилась, таймер останавливается, кнопка паузы отключается, а пользователю выводится сообщение. В режиме локального мультиплеера второму окну передается информация о победе и предлагается продолжить игру или завершить ее.

4.4 Классы `NetworkDialog` и `NetworkManager`

`NetworkDialog` является диалоговым окном, через которое пользователь выбирает сетевой режим: создание сервера или подключение к существующей игре. В окне вводятся имя игрока, IP-адрес и порт. После успешного подключения диалог скрывается, а игра продолжается в основном окне.

`NetworkManager` выполняет непосредственную работу с сетью. При создании сервера используется `ServerSocket`, который ожидает подключение клиента. При подключении к игре используется `Socket`. После установления соединения создаются `ObjectOutputStream` и `ObjectInputStream` для передачи сериализуемых объектов.

Для приема пакетов используется отдельная задача `ExecutorService`. Благодаря этому ожидание данных из сети не останавливает графический интерфейс. При разрыве соединения флаг `connected` становится `false`, а ресурсы закрываются методом `disconnect()`.

4.5 Класс `GameStatePacket`

`GameStatePacket` является контейнером данных, который передается между игроками в сетевом режиме. Класс реализует `Serializable`, что позволяет передавать его через `ObjectOutputStream` и читать через `ObjectInputStream`.

В пакет входят имя игрока, счет, уровень, количество очищенных линий, признаки `gameOver`, `winner` и `paused`, а также поле `boardState`. Наличие признака `paused` важно, потому что в коде реализована передача состояния паузы. Благодаря этому второй игрок может видеть согласованное состояние партии.

Класс не содержит сложной логики и состоит преимущественно из полей, конструктора и методов доступа. Такое решение удобно для сетевого обмена: одна сторона формирует объект состояния, отправляет его, а другая сторона читает значения и обновляет интерфейс.

5 Руководство пользователя

Для запуска приложения необходимо иметь установленную среду выполнения Java. После запуска файла с главным классом Kursovaya открывается окно игры «Тетрис». В левой части окна отображается игровое поле, в правой части - счет, уровень, количество очищенных линий, следующая фигура и статистика соперника.

Действие	Клавиша / способ выполнения
Переместить фигуру влево	Стрелка влево или назначенная клавиша игрока
Переместить фигуру вправо	Стрелка вправо или назначенная клавиша игрока
Ускорить падение	Стрелка вниз
Повернуть фигуру	Стрелка вверх
Мгновенно сбросить фигуру	Пробел
Пауза / продолжить	Клавиша Р или кнопка «Пауза»
Начать новую игру	Кнопка «Новая игра»
Открыть сетевой режим	Кнопка «Сетевая игра»

В одиночном режиме пользователь играет один. В локальном мультиплеере открываются два окна, каждое из которых соответствует отдельному игроку. В сетевом режиме один игрок создает сервер, а второй подключается к нему по IP-адресу и порту. Во время игры пользователь может наблюдать счет, уровень и состояние соперника.

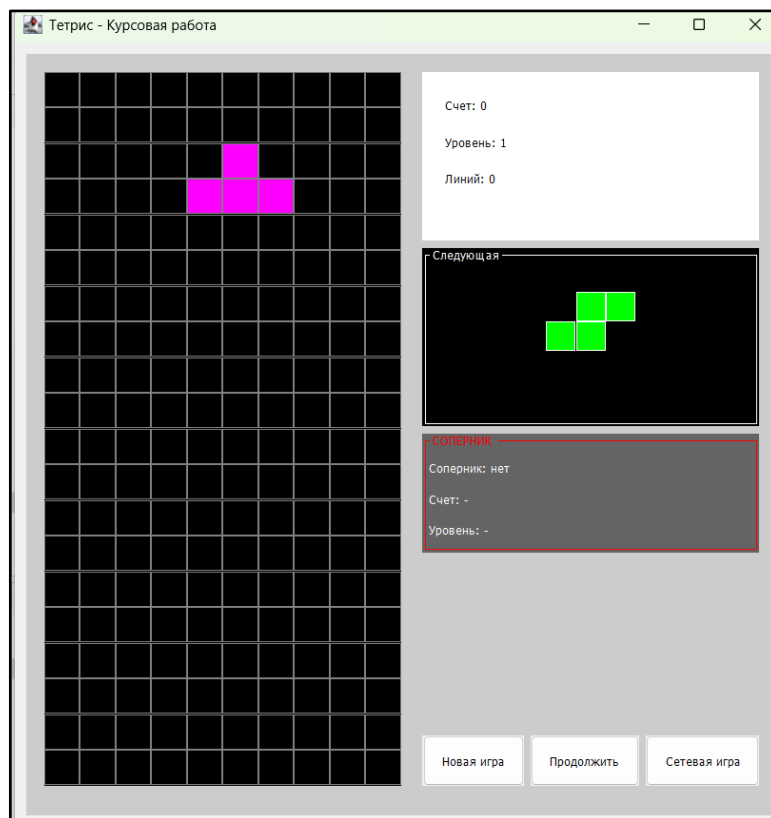


Рисунок 6 - Главное окно приложения

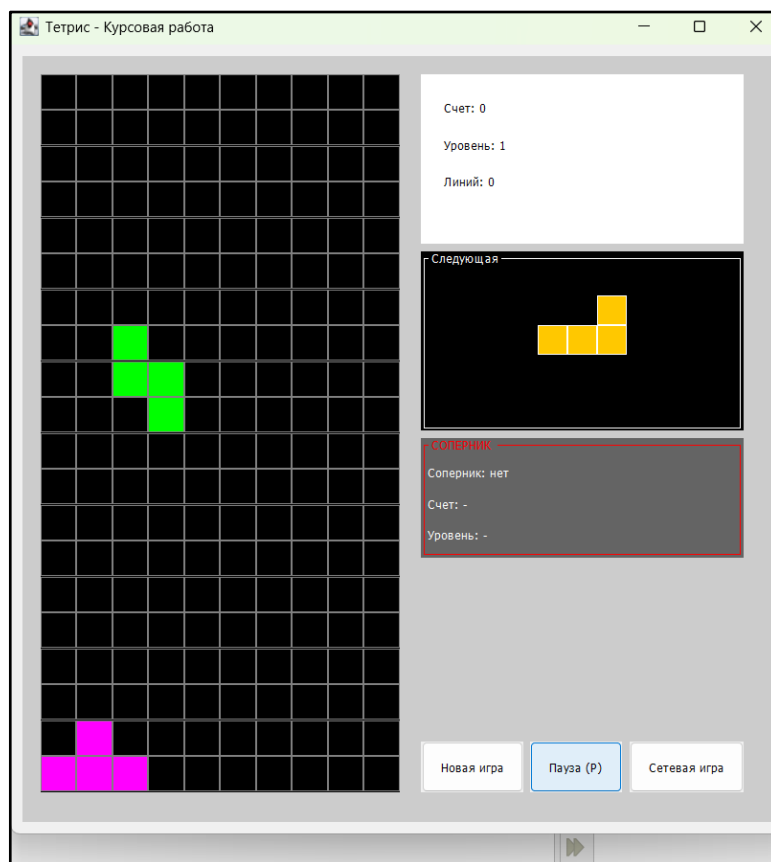


Рисунок 7 - Процесс игры

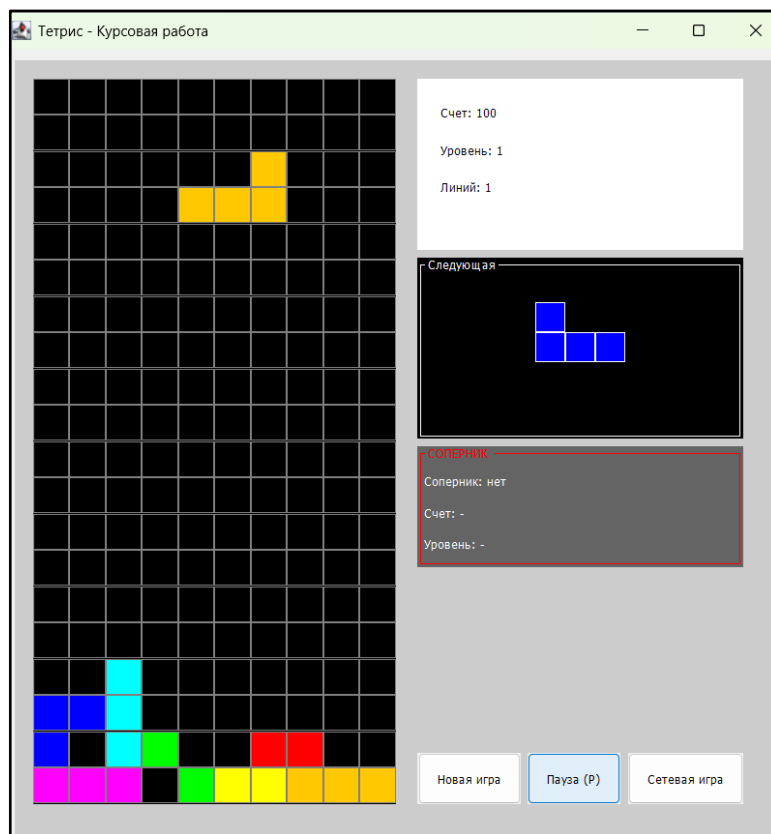


Рисунок 8 - Момент закрытия линии

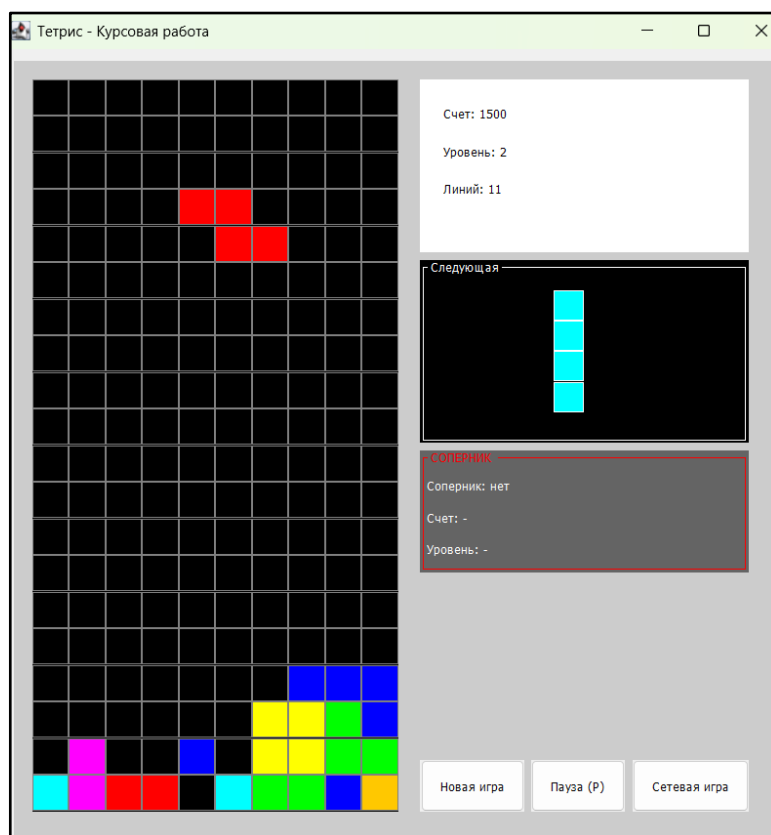


Рисунок 9 - Повышение уровня

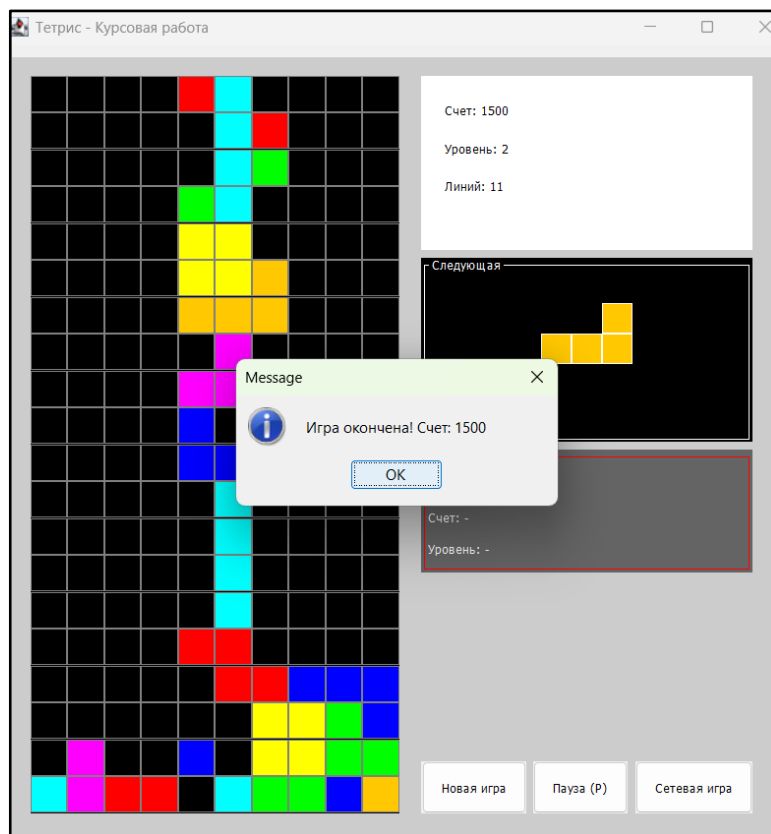


Рисунок 10 - Окончание игры

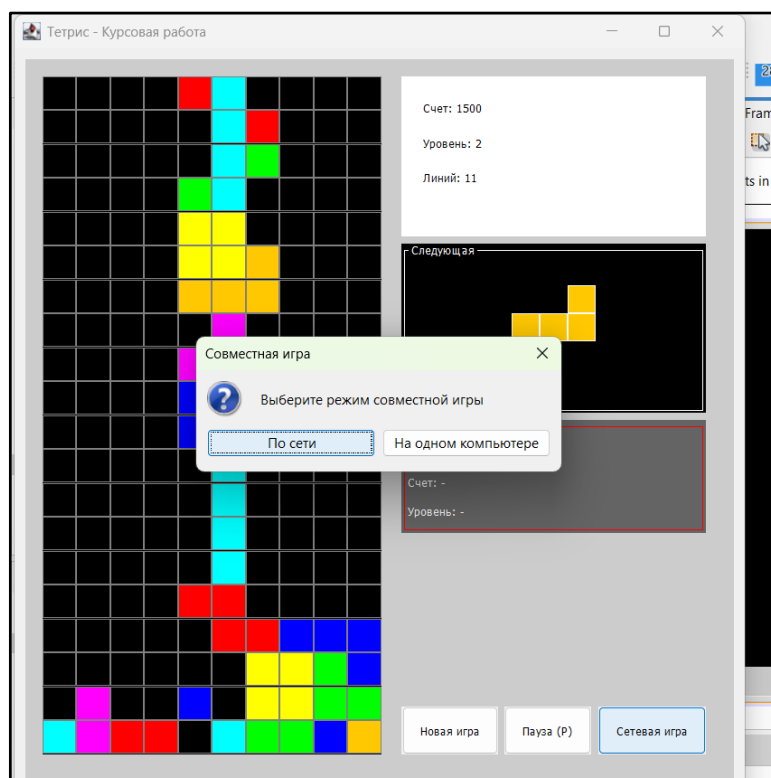


Рисунок 11 - Выбор режима совместной игры

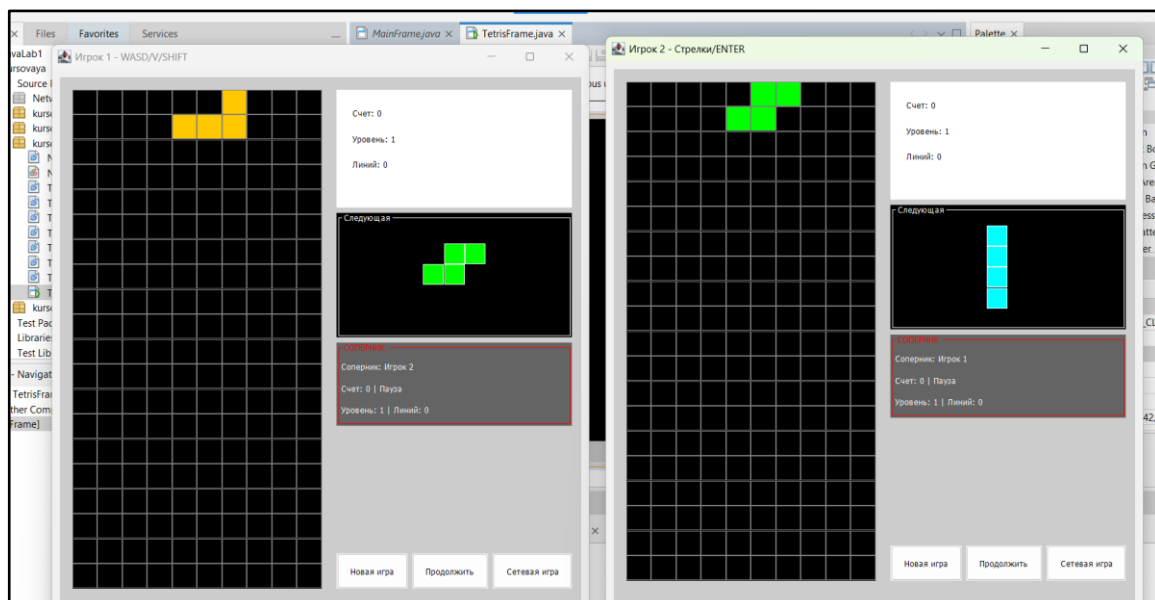


Рисунок 12 - Локальный мультиплеер

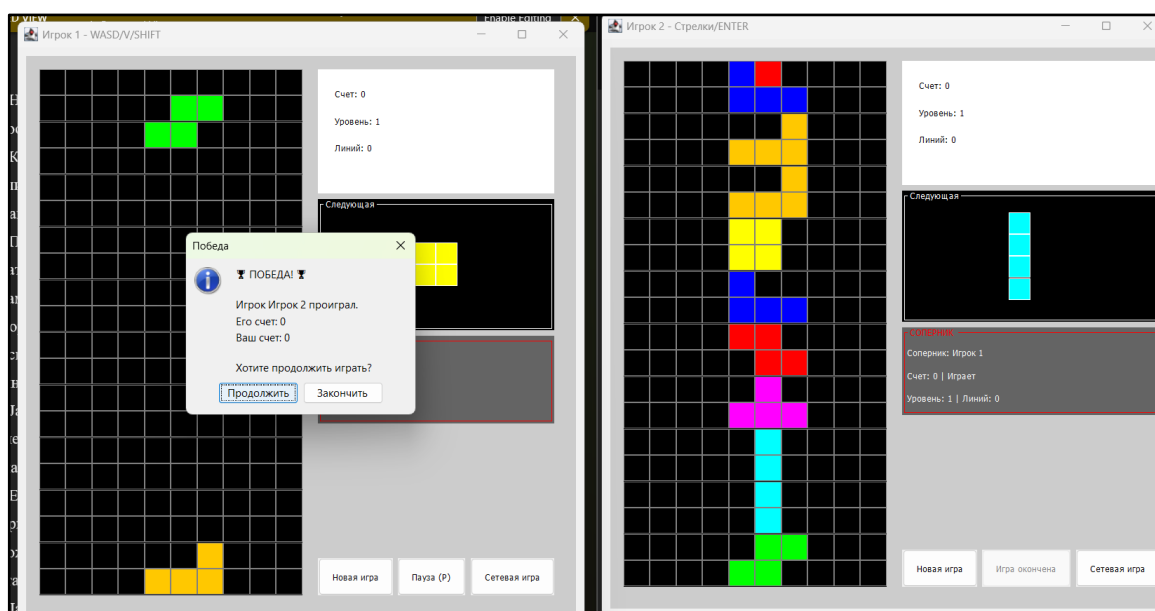


Рисунок 13 - Сообщение о победе

6 Тестирование

Тестирование проводилось вручную путем запуска приложения и проверки основных пользовательских сценариев. Проверялись одиночный режим, локальный мультиплеер, сетевое подключение, обработка клавиш, начисление очков, повышение уровня и завершение партии.

№	Проверка	Ожидаемый результат	Результат
1	Запуск приложения	Открывается главное окно игры	Успешно
2	Создание новой игры	Поле очищается, счет равен 0, уровень равен 1	Успешно
3	Перемещение фигуры	Фигура двигается влево, вправо и вниз без выхода за границы	Успешно

4	Поворот фигуры	Фигура поворачивается, при столкновении выполняется откат или сдвиг	Успешно
5	Мгновенный сброс фигуры	Фигура падает до препятствия и фиксируется	Успешно
6	Очистка линии	Заполненная строка удаляется, счет увеличивается	Успешно
7	Повышение уровня	После 10 линий уровень становится выше	Успешно
8	Пауза	Таймер останавливается, повторное нажатие продолжает игру	Успешно
9	Локальный мультиплеер	Открываются два независимых окна, статистика обновляется	Успешно
10	Сетевая игра	Игроки подключаются через сокет и обмениваются состоянием	Успешно
11	Отключение клиента	Соединение закрывается без аварийного завершения интерфейса	Успешно
12	Окончание игры	Появляется сообщение о проигрыше или победе	Успешно

По результатам проверки основные требования к приложению выполнены. Ошибки, связанные с блокировкой интерфейса при ожидании сетевого подключения, устранены за счет использования `ExecutorService`. Ошибки вращения фигуры у границ поля обрабатываются проверкой допустимой позиции и откатом состояния при невозможности поворота.

Заключение

В результате выполнения курсовой работы было разработано многомодульное приложение «Тетрис» на языке Java. Программа имеет графический интерфейс, поддерживает одиночный режим, локальный мультиплеер и сетевое взаимодействие между двумя игроками.

В ходе разработки были применены принципы объектно-ориентированного программирования: инкапсуляция, разделение ответственности между классами, использование интерфейса наблюдателя и перечислений. Для хранения данных применялись коллекции Java, для обработки фигур использовались матрицы, для генерации фигур - очередь 7-Bag.

Отдельно реализовано сетевое взаимодействие через TCP/IP сокет. Обмен между игроками выполняется объектами GameStatePacket, а прием данных вынесен в отдельную задачу, что позволяет сохранить отзывчивость интерфейса. Проведенное тестирование подтвердило работоспособность основных функций программы.

Также в пояснительную записку добавлены UML-диаграммы в основной части и приложении. Диаграммы приведены в соответствие с исходным кодом: в диаграмме последовательности отражен цикл игрового процесса, диаграмма развертывания уточнена для локального и сетевого режимов, а диаграмма вариантов использования переработана с сохранением логики работы приложения.

Список литературы

1. Wielenga G. Beginning NetBeans IDE: For Java Developers. - Apress, 2015.
2. Савитч У. Язык Java. Курс программирования. - М.: Вильямс, 2015. - 928 с.
3. Niemeyer P., Knudsen J. Learning Java. - O'Reilly Media, Inc., 2005.
4. Документация Java Platform, Standard Edition. Oracle.
5. Stack Overflow: вопросы и ответы по программированию.
<https://stackoverflow.com/>.
6. Форум программистов и системных администраторов Киберфорум.
<http://www.cyberforum.ru/>.

Приложение А. Листинг программы

kursovaya/Kursovaya.java

```
package kursovaya;

import kursovaya.gui.TetrisFrame;

public class Kursovaya {
    public static void main(String[] args) {
        TetrisFrame.main(args);
    }
}
```

kursovaya/core/Cell.java

```
package kursovaya.core;

import java.awt.Color;

public class Cell {
    private int x, y;
    private boolean occupied;
    private Color color;

    public Cell(int x, int y) {
        this.x = x;
        this.y = y;
        this.occupied = false;
        this.color = Color.BLACK;
    }

    public void setOccupied(boolean occupied, Color color) {
        this.occupied = occupied;
        this.color = color;
    }

    public int getX() { return x; }
    public int getY() { return y; }
    public void setY(int y) { this.y = y; }
    public boolean isOccupied() { return occupied; }
    public Color getColor() { return color; }
}
```

kursovaya/core/GameBoard.java

```
package kursovaya.core;

import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class GameBoard {
    public static final int WIDTH = 10;
    public static final int HEIGHT = 20;

    private final List<List<Cell>> grid;
    private Tetromino currentPiece;
    private Tetromino nextPiece;
    private int score;
    private int level;
```

```

private int linesCleared;
private boolean gameOver;
private final List<GameObserver> observers = new ArrayList<>();
private final Queue<TetrominoType> pieceBag = new LinkedList<>();

public GameBoard() {
    this.grid = new ArrayList<>();
    this.score = 0;
    this.level = 1;
    this.linesCleared = 0;
    this.gameOver = false;
    initializeGrid();
    spawnPiece();
}

private void initializeGrid() {
    for (int y = 0; y < HEIGHT; y++) {
        List<Cell> row = new ArrayList<>();
        for (int x = 0; x < WIDTH; x++) {
            row.add(new Cell(x, y));
        }
        grid.add(row);
    }
}

public void spawnPiece() {
    if (nextPiece == null) {
        nextPiece = createRandomPiece();
    }
    currentPiece = nextPiece;
    nextPiece = createRandomPiece();

    if (!isValidPosition(currentPiece, 0, 0)) {
        gameOver = true;
        notifyObservers();
        throw new GameOverException("Игра окончена! Счет: " + score);
    }
    notifyObservers();
}

private Tetromino createRandomPiece() {
    refillBagIfNeeded();
    return new Tetromino(pieceBag.poll(), WIDTH / 2 - 1, 0);
}

// 7-bag randomizer как в классическом современном Tetris.
// Каждая фигура появляется ровно один раз за цикл,
// после чего набор перемешивается заново.
private void refillBagIfNeeded() {
    if (!pieceBag.isEmpty()) {
        return;
    }

    List<TetrominoType> shuffled = new
ArrayList<>(Arrays.asList(TetrominoType.values()));
    Collections.shuffle(shuffled);
    pieceBag.addAll(shuffled);
}

public boolean movePiece(int dx, int dy) {
    if (gameOver || currentPiece == null) return false;
    if (isValidPosition(currentPiece, dx, dy)) {

```

```

        currentPiece.move(dx, dy);
        notifyObservers();
        return true;
    }
    if (dy > 0) {
        lockPiece();
    }
    return false;
}

public void rotatePiece() {
    if (gameOver || currentPiece == null) return;

    // Сохраняем текущее состояние
    int oldX = currentPiece.getX();
    int oldY = currentPiece.getY();
    int[][] oldMatrix = currentPiece.getMatrix();

    // Пробуем повернуть
    currentPiece.rotate();

    // Если позиция валидна – оставляем как есть
    if (isValidPosition(currentPiece, 0, 0)) {
        notifyObservers();
        return;
    }

    // Пробуем сдвинуть вправо (стена слева)
    if (isValidPosition(currentPiece, 1, 0)) {
        currentPiece.move(1, 0);
        notifyObservers();
        return;
    }

    // Пробуем сдвинуть влево (стена справа)
    if (isValidPosition(currentPiece, -1, 0)) {
        currentPiece.move(-1, 0);
        notifyObservers();
        return;
    }

    // Пробуем сдвинуть вправо на 2 клетки (для I-фигуры)
    if (isValidPosition(currentPiece, 2, 0)) {
        currentPiece.move(2, 0);
        notifyObservers();
        return;
    }

    // Пробуем сдвинуть влево на 2 клетки (для I-фигуры)
    if (isValidPosition(currentPiece, -2, 0)) {
        currentPiece.move(-2, 0);
        notifyObservers();
        return;
    }

    // Пробуем сдвинуть вниз (потолок)
    if (isValidPosition(currentPiece, 0, 1)) {
        currentPiece.move(0, 1);
        notifyObservers();
        return;
    }
}

```

```

        // Ничего не помогло – откатываем поворот
        currentPiece.move(oldX - currentPiece.getX(), oldY -
currentPiece.getY());
        currentPiece.setMatrix(oldMatrix);
        notifyObservers();
    }

    private void lockPiece() {
        currentPiece.lock();
        int[][] matrix = currentPiece.getMatrix();
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] != 0) {
                    int boardX = currentPiece.getX() + j;
                    int boardY = currentPiece.getY() + i;
                    if (boardY >= 0) {
                        grid.get(boardY).get(boardX).setOccupied(true,
currentPiece.getType().getColor());
                    }
                }
            }
        }
        clearLines();
        spawnPiece();
    }

    private void clearLines() {
        int cleared = 0;

        // Проверяем строки снизу вверх как в оригинальном Tetris
        for (int y = HEIGHT - 1; y >= 0; y--) {
            boolean full = true;

            for (int x = 0; x < WIDTH; x++) {
                if (!grid.get(y).get(x).isOccupied()) {
                    full = false;
                    break;
                }
            }

            // Если линия заполнена – удаляем и сдвигаем всё вниз
            if (full) {
                cleared++;

                for (int row = y; row > 0; row--) {
                    for (int col = 0; col < WIDTH; col++) {
                        Cell above = grid.get(row - 1).get(col);
                        grid.get(row).get(col)
                            .setOccupied(above.isOccupied(), above.getColor());
                    }
                }

                // Очищаем верхнюю строку
                for (int col = 0; col < WIDTH; col++) {
                    grid.get(0).get(col)
                        .setOccupied(false, java.awt.Color.BLACK);
                }

                // Повторно проверяем эту же строку после сдвига
                y++;
            }
        }
    }
}

```

```

        if (cleared > 0) {
            linesCleared += cleared;
            score += calculateScore(cleared);

            // Уровень растёт каждые 10 линий
            level = Math.max(1, (linesCleared / 10) + 1);

            notifyObservers();
        }
    }

    private boolean isLineFull(List<Cell> row) {
        return row.stream().allMatch(Cell::isOccupied);
    }

    private int calculateScore(int lines) {
        switch (lines) {
            case 1: return 100 * level;
            case 2: return 300 * level;
            case 3: return 500 * level;
            case 4: return 800 * level;
            default: return 0;
        }
    }

    private boolean isValidPosition(Tetromino piece, int dx, int dy) {
        int[][] matrix = piece.getMatrix();
        int newX = piece.getX() + dx;
        int newY = piece.getY() + dy;

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] != 0) {
                    int boardX = newX + j;
                    int boardY = newY + i;
                    if (boardX < 0 || boardX >= WIDTH || boardY >= HEIGHT)
                        return false;
                    if (boardY >= 0 &&
                        grid.get(boardY).get(boardX).isOccupied()) return false;
                }
            }
        }
        return true;
    }

    public void dropPiece() {
        while (movePiece(0, 1)) {}
    }

    public void addObserver(GameObserver observer) {
        observers.add(observer);
    }

    private void notifyObservers() {
        observers.forEach(GameObserver::onGameStateChanged);
    }

    public List<List<Cell>> getGrid() { return grid; }
    public Tetromino getCurrentPiece() { return currentPiece; }
    public Tetromino getNextPiece() { return nextPiece; }
    public int getScore() { return score; }

```

```

        public int getLevel() { return level; }
        public int getLinesCleared() { return linesCleared; }
        public boolean isGameOver() { return gameOver; }
    }

```

kursovaya/core/GameObserver.java

```

package kursovaya.core;

public interface GameObserver {
    void onGameStateChanged();
}

```

kursovaya/core/GameOverException.java

```

package kursovaya.core;

public class GameOverException extends RuntimeException {
    public GameOverException(String message) {
        super(message);
    }
}

```

kursovaya/core/Tetromino.java

```

package kursovaya.core;

import java.util.Arrays;

public class Tetromino {
    private TetrominoType type;
    private int x, y;
    private int[][] matrix;
    private boolean locked;

    public Tetromino(TetrominoType type, int startX, int startY) {
        this.type = type;
        this.x = startX;
        this.y = startY;
        this.matrix = deepCopy(type.getShape());
        this.locked = false;
    }

    public void rotate() {
        if (locked) throw new IllegalStateException("Фигура уже зафиксирована");
        int n = matrix.length;
        int[][] rotated = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                rotated[j][n - 1 - i] = matrix[i][j];
            }
        }
        matrix = rotated;
    }

    public void move(int dx, int dy) {
        if (locked) throw new IllegalStateException("Фигура уже зафиксирована");
        this.x += dx;
    }
}

```

```

        this.y += dy;
    }

    public void lock() { this.locked = true; }
    public boolean isLocked() { return locked; }

    public int getX() { return x; }
    public int getY() { return y; }
    public int[][] getMatrix() { return deepCopy(matrix); }
    public void setMatrix(int[][] matrix) { this.matrix = deepCopy(matrix); }
    public TetrominoType getType() { return type; }

    private int[][] deepCopy(int[][] src) {
        return Arrays.stream(src).map(int[]::clone).toArray(int[][]::new);
    }
}

```

kursovaya/core/TetrominoType.java

```

package kursovaya.core;

import java.awt.Color;
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public enum TetrominoType {
    I(Color.CYAN, new int[][]{{0,1,0,0}, {0,1,0,0}, {0,1,0,0}, {0,1,0,0}}),
    O(Color.YELLOW, new int[][]{{1,1}, {1,1}}),
    T(Color.MAGENTA, new int[][]{{0,1,0}, {1,1,1}, {0,0,0}}),
    S(Color.GREEN, new int[][]{{0,1,1}, {1,1,0}, {0,0,0}}),
    Z(Color.RED, new int[][]{{1,1,0}, {0,1,1}, {0,0,0}}),
    J(Color.BLUE, new int[][]{{1,0,0}, {1,1,1}, {0,0,0}}),
    L(Color.ORANGE, new int[][]{{0,0,1}, {1,1,1}, {0,0,0}});

    private final Color color;
    private final int[][] shape;
    private static final List<TetrominoType> VALUES = Arrays.asList(values());
    private static final Random RANDOM = new Random();

    TetrominoType(Color color, int[][] shape) {
        this.color = color;
        this.shape = shape;
    }

    public Color getColor() { return color; }
    public int[][] getShape() {
        return Arrays.stream(shape).map(int[]::clone).toArray(int[][]::new);
    }

    public static TetrominoType random() {
        return VALUES.get(RANDOM.nextInt(VALUES.size()));
    }

    public static List<Color> getAllColors() {
        return Arrays.stream(values())
            .map(TetrominoType::getColor)
            .collect(Collectors.toList());
    }
}

```

kursovaya/gui/NetworkDialog.java

```
package kursovaya.gui;

import kursovaya.network.NetworkManager;
import kursovaya.network.GameStatePacket;
import javax.swing.*;
import java.awt.*;
import java.io.IOException;

public class NetworkDialog extends JDialog {

    private NetworkManager networkManager;
    private JTextField txtHost;
    private JTextField txtPort;
    private JTextField txtName;
    private JLabel lblStatus;
    private JButton btnHost;
    private JButton btnJoin;
    private Timer sendTimer;
    private boolean isConnected = false;
    private String playerName = "Игрок";
    private boolean isServer = false;

    public NetworkDialog(JFrame parent) {
        super(parent, "Сетевая игра", true);
        setSize(400, 300);
        setLocationRelativeTo(parent);
        setDefaultCloseOperation(JDialog.HIDE_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel panel = new JPanel(new GridLayout(6, 2, 10, 10));
        panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        txtName = new JTextField("Игрок" + (System.currentTimeMillis() %
1000));
        txtHost = new JTextField("localhost");
        txtPort = new JTextField("8080");
        lblStatus = new JLabel("Статус: не подключено");

        panel.add(new JLabel("Ваше имя:"));
        panel.add(txtName);
        panel.add(new JLabel("IP адрес:"));
        panel.add(txtHost);
        panel.add(new JLabel("Порт:"));
        panel.add(txtPort);
        panel.add(new JLabel("Статус:"));
        panel.add(lblStatus);

        btnHost = new JButton("Создать сервер");
        btnJoin = new JButton("Подключиться");

        btnHost.addActionListener(e -> createServer());
        btnJoin.addActionListener(e -> joinGame());

        panel.add(btnHost);
        panel.add(btnJoin);

        add(panel, BorderLayout.CENTER);
    }

    private void createServer() {
```

```

try {
    playerName = txtName.getText();
    isServer = true;
    int port = Integer.parseInt(txtPort.getText());
    networkManager = new NetworkManager();

    // Устанавливаем слушатель ДО запуска сервера
    networkManager.setStateListener(this::onStateReceived);

    networkManager.hostGame(port);

    lblStatus.setText("Сервер запущен на порту " + port);
    btnHost.setEnabled(false);
    btnJoin.setEnabled(false);

    new Thread(() -> {
        try {
            int attempts = 0;
            while (!networkManager.isConnected() && attempts < 60) {
                Thread.sleep(500);
                attempts++;
            }

            if (networkManager.isConnected()) {
                isConnected = true;
                SwingUtilities.invokeLater(() -> {
                    lblStatus.setText("Клиент подключился!");
                    JOptionPane.showMessageDialog(this,
                        "Клиент подключился!\nЗакройте это окно – игра
продолжится.");
                    setVisible(false);
                });
            } else {
                SwingUtilities.invokeLater(() -> {
                    lblStatus.setText("Никто не подключился");
                });
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }).start();

} catch (Exception e) {
    lblStatus.setText("Ошибка: " + e.getMessage());
}

}

private void joinGame() {
    try {
        playerName = txtName.getText();
        isServer = false;
        String host = txtHost.getText();
        int port = Integer.parseInt(txtPort.getText());
        networkManager = new NetworkManager();

        // Устанавливаем слушатель ДО подключения
        networkManager.setStateListener(this::onStateReceived);

        networkManager.joinGame(host, port);

        isConnected = true;
        lblStatus.setText("Подключено!");
    }
}

```

```

        btnHost.setEnabled(false);
        btnJoin.setEnabled(false);

        JOptionPane.showMessageDialog(this,
            "Подключено!\nЗакройте это окно – игра продолжится.");
        setVisible(false);
    } catch (IOException e) {
        lblStatus.setText("Ошибка: " + e.getMessage());
    }
}

private void onStateReceived(GameStatePacket packet) {
    SwingUtilities.invokeLater(() -> {
        TetrisFrame frame = (TetrisFrame) getParent();

        // Обновляем статистику соперника и синхронизируем паузу
        String status = packet.isGameOver() ? "Проиграл" :
(packet.isPaused() ? "Пауза" : "Играет");
        frame.updateOpponentStats(
            packet.getPlayerName(),
            packet.getScore(),
            packet.getLevel(),
            packet.getLinesCleared(),
            status
        );
        frame.applyNetworkPause(packet.isPaused());

        // Проверяем условия победы/поражения
        if (packet.isGameOver()) {
            if (!packet.isWinner()) {
                // Соперник проиграл – мы победили!
                frame.opponentGameOver(packet.getPlayerName(), true);
            }
        }
    });
}

public void sendState(int score, int level, int linesCleared, boolean
gameOver, boolean paused) {
    if (networkManager != null && networkManager.isConnected()) {
        boolean winner = !gameOver;
        GameStatePacket packet = new GameStatePacket(
            playerName, score, level, linesCleared, gameOver, winner,
            paused, null
        );
        networkManager.sendState(packet);
    }
}

public boolean isConnected() {
    return isConnected && networkManager != null &&
networkManager.isConnected();
}

public String getPlayerName() { return playerName; }
public NetworkManager getNetworkManager() { return networkManager; }
public boolean isServer() { return isServer; }

public void realDispose() {
    if (sendTimer != null) sendTimer.stop();
    if (networkManager != null) networkManager.disconnect();
}

```

```

        super.dispose();
    }
}

```

kursovaya/gui/TetrisFrame.java

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
 to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit
 this template
 */
package kursovaya.gui;

import kursovaya.core.GameBoard;
import kursovaya.core.GameObserver;
import kursovaya.core.GameOverException;
import kursovaya.core.Cell;
import kursovaya.core.Tetromino;
import javax.swing.SwingUtilities;
import javax.swing.JOptionPane;

/**
 *
 * @author alex
 */
public class TetrisFrame extends javax.swing.JFrame implements GameObserver {

    private static final java.util.logging.Logger logger =
        java.util.logging.Logger.getLogger(TetrisFrame.class.getName());

    private GameBoard board;
    private javax.swing.Timer gameTimer;
    private volatile boolean paused = false;
    private static final int CELL_SIZE = 30;

    private NetworkDialog networkDialog;
    private javax.swing.Timer networkSendTimer;
    private TetrisFrame localPausePeer;
    private String playerName = "Игрок";
    private boolean localGameOverHandled = false;
    private boolean localVictoryDecisionShown = false;

    private int leftKey = java.awt.event.KeyEvent.VK_LEFT;
    private int rightKey = java.awt.event.KeyEvent.VK_RIGHT;
    private int downKey = java.awt.event.KeyEvent.VK_DOWN;
    private int rotateKey = java.awt.event.KeyEvent.VK_UP;
    private int dropKey = java.awt.event.KeyEvent.VK_SPACE;

    private java.awt.KeyEventDispatcher globalKeyDispatcher;

    /**
     * Creates new form TetrisFrame
     */
    public TetrisFrame() {
        initComponents();
        initGame();
    }

    public TetrisFrame(String title, int leftKey, int rightKey, int downKey,
        int rotateKey, int dropKey) {
        this(title, title, leftKey, rightKey, downKey, rotateKey, dropKey);
    }
}

```

```

    public TetrisFrame(String title, String playerName, int leftKey, int
rightKey, int downKey, int rotateKey, int dropKey) {
        this.playerName = playerName;
        this.leftKey = leftKey;
        this.rightKey = rightKey;
        this.downKey = downKey;
        this.rotateKey = rotateKey;
        this.dropKey = dropKey;
        initComponents();
        setTitle(title);
        initGame();
    }

    private void initGame() {
        paused = false;
        localGameOverHandled = false;
        localVictoryDecisionShown = false;
        btnPause.setEnabled(true);
        btnPause.setText("Пауза (P)");

        if (gameTimer != null) {
            gameTimer.stop();
        }

        for (java.awt.event.KeyListener listener : gamePanel.getKeyListeners())
        {
            gamePanel.removeKeyListener(listener);
        }

        board = new GameBoard();
        board.addObserver(this);

        gamePanel.setFocusable(true);
        gamePanel.requestFocusInWindow();

        gamePanel.addKeyListener(new java.awt.event.KeyAdapter() {
            @Override
            public void keyPressed(java.awt.event.KeyEvent evt) {
                formKeyPressed(evt);
            }
        });

        if (globalKeyDispatcher != null) {
            java.awt.KeyboardFocusManager.getCurrentKeyboardFocusManager()
                .removeKeyEventDispatcher(globalKeyDispatcher);
        }

        globalKeyDispatcher = e -> {
            if (!isVisible()) return false;
            if (e.getID() == java.awt.event.KeyEvent.KEY_PRESSED) {
                int key = e.getKeyCode();
                if (key == leftKey || key == rightKey || key == downKey ||
                    key == rotateKey || key == dropKey ||
                    key == java.awt.event.KeyEvent.VK_P) {
                    formKeyPressed(e);
                    return true;
                }
            }
            return false;
        };
    }

```

```

        java.awt.KeyboardFocusManager.getCurrentKeyboardFocusManager()
            .addKeyEventDispatcher(globalKeyDispatcher);

        gameTimer = new javax.swing.Timer(1000, e -> gameTick());
        gameTimer.start();

        onGameStateChanged();
    }

    private void gameTick() {
        if (!paused) {
            try {
                board.movePiece(0, 1);
                gamePanel.repaint();
            } catch (GameOverException ex) {
                handleGameOver(ex.getMessage());
            }
        }
    }

    private void formKeyPressed(java.awt.event.KeyEvent evt) {
        int key = evt.getKeyCode();

        if (key == java.awt.event.KeyEvent.VK_P) {
            togglePause();
            gamePanel.repaint();
            return;
        }

        if (paused) {
            return;
        }

        try {
            if (key == leftKey) board.movePiece(-1, 0);
            else if (key == rightKey) board.movePiece(1, 0);
            else if (key == downKey) board.movePiece(0, 1);
            else if (key == rotateKey) board.rotatePiece();
            else if (key == dropKey) board.dropPiece();
        } catch (GameOverException ex) {
            handleGameOver(ex.getMessage());
        }
        gamePanel.repaint();
    }

    private void handleGameOver(String message) {
        if (localGameOverHandled) {
            return;
        }

        localGameOverHandled = true;
        if (gameTimer != null) {
            gameTimer.stop();
        }
        btnPause.setEnabled(false);
        btnPause.setText("Игра окончена");

        updateLocalOpponentInfo();
        sendNetworkState(true);

        if (localPausePeer != null) {
            localPausePeer.showLocalVictoryChoice(playerName, board.getScore(),

```

```

board.getLevel(), board.getLinesCleared());
        JOptionPane.showMessageDialog(this,
            "☹️ ПОРАЖЕНИЕ ☹️\n\n" +
            "Вы проиграли.\n" +
            "Ваш счет: " + board.getScore() + "\n" +
            "Линий: " + board.getLinesCleared(),
            "Игра окончена",
            JOptionPane.ERROR_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(this, message);
    }
}

private void showLocalVictoryChoice(String defeatedName, int defeatedScore,
int defeatedLevel, int defeatedLines) {
    if (!SwingUtilities.isEventDispatchThread()) {
        SwingUtilities.invokeLater(() ->
showLocalVictoryChoice(defeatedName, defeatedScore, defeatedLevel,
defeatedLines));
        return;
    }

    updateOpponentStats(defeatedName, defeatedScore, defeatedLevel,
defeatedLines, "Проиграл");

    if (localVictoryDecisionShown || localGameOverHandled || board == null
|| board.isGameOver()) {
        return;
    }

    localVictoryDecisionShown = true;
    Object[] options = {"Продолжить", "Закончить"};
    int choice = JOptionPane.showOptionDialog(this,
        "🏆 ПОБЕДА! 🏆\n\n" +
        "Игрок " + defeatedName + " проиграл.\n" +
        "Его счет: " + defeatedScore + "\n" +
        "Ваш счет: " + board.getScore() + "\n\n" +
        "Хотите продолжить играть?",
        "Победа",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.INFORMATION_MESSAGE,
        null,
        options,
        options[0]);

    if (choice != JOptionPane.YES_OPTION) {
        finishLocalGameAfterVictory();
    }
}

private void finishLocalGameAfterVictory() {
    stopCurrentLocalGame();

    if (localPausePeer != null) {
        localPausePeer.stopCurrentLocalGame();
    }
}

private void stopCurrentLocalGame() {
    if (gameTimer != null) {
        gameTimer.stop();
    }
}

```

```

        paused = false;
        btnPause.setEnabled(false);
        btnPause.setText("Игра окончена");
        gamePanel.repaint();
        updateLocalOpponentInfo();
    }

    private void togglePause() {
        setPaused(!paused, true, true);
    }

    private void setPaused(boolean pause, boolean notifyNetwork, boolean
notifyLocalPeer) {
        if (board != null && board.isGameOver()) {
            return;
        }

        paused = pause;
        btnPause.setText(paused ? "Продолжить" : "Пауза (P)");
        gamePanel.repaint();

        if (notifyLocalPeer && localPausePeer != null) {
            localPausePeer.setPaused(paused, false, false);
        }

        updateLocalOpponentInfo();

        if (notifyNetwork) {
            sendNetworkState();
        }
    }

    public void setLocalPausePeer(TetrisFrame peer) {
        this.localPausePeer = peer;
        updateLocalOpponentInfo();
    }

    private void updateLocalOpponentInfo() {
        if (localPausePeer == null || board == null) {
            return;
        }

        String status;
        if (board.isGameOver() || localGameOverHandled) {
            status = "Проиграл";
        } else if (!btnPause.isEnabled()) {
            status = "Игра окончена";
        } else if (paused) {
            status = "Пауза";
        } else {
            status = "Играет";
        }

        localPausePeer.updateOpponentStats(
            playerName,
            board.getScore(),
            board.getLevel(),
            board.getLinesCleared(),
            status
        );
    }
}

```

```

    public void applyNetworkPause(boolean pause) {
        setPaused(pause, false, false);
    }

    public boolean isPaused() {
        return paused;
    }

    private void sendNetworkState() {
        sendNetworkState(board != null && board.isGameOver());
    }

    private void sendNetworkState(boolean gameOver) {
        if (networkDialog != null && networkDialog.isConnected() && board !=
null) {
            networkDialog.sendState(
                board.getScore(),
                board.getLevel(),
                board.getLinesCleared(),
                gameOver,
                paused
            );
        }
    }

    private void drawGamePanel(java.awt.Graphics g) {
        g.setColor(java.awt.Color.BLACK);
        g.fillRect(0, 0, gamePanel.getWidth(), gamePanel.getHeight());

        for (int y = 0; y < GameBoard.HEIGHT; y++) {
            for (int x = 0; x < GameBoard.WIDTH; x++) {
                Cell cell = board.getGrid().get(y).get(x);
                drawCell(g, x, y, cell.isOccupied() ? cell.getColor() :
java.awt.Color.BLACK);
            }
        }

        Tetromino piece = board.getCurrentPiece();
        if (piece != null) {
            int[][] matrix = piece.getMatrix();
            for (int i = 0; i < matrix.length; i++) {
                for (int j = 0; j < matrix[i].length; j++) {
                    if (matrix[i][j] != 0) {
                        drawCell(g, piece.getX() + j, piece.getY() + i,
piece.getType().getColor());
                    }
                }
            }
        }
    }

    private void drawNextPiece(java.awt.Graphics g) {
        g.setColor(java.awt.Color.BLACK);
        g.fillRect(0, 0, nextPiecePreview.getWidth(),
nextPiecePreview.getHeight());

        Tetromino next = board.getNextPiece();
        if (next == null) return;

        int[][] matrix = next.getMatrix();
        int cellSize = 25;
        int offsetX = (nextPiecePreview.getWidth() - matrix[0].length *

```

```

        cellSize) / 2;
        int offsetY = (nextPiecePreview.getHeight() - matrix.length * cellSize)
        / 2;

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] != 0) {
                    g.setColor(next.getType().getColor());
                    g.fillRect(offsetX + j * cellSize, offsetY + i * cellSize,
cellSize - 1, cellSize - 1);
                    g.setColor(java.awt.Color.WHITE);
                    g.drawRect(offsetX + j * cellSize, offsetY + i * cellSize,
cellSize - 1, cellSize - 1);
                }
            }
        }

        private void drawCell(java.awt.Graphics g, int x, int y, java.awt.Color
color) {
            int px = x * CELL_SIZE;
            int py = y * CELL_SIZE;
            g.setColor(color);
            g.fillRect(px, py, CELL_SIZE - 1, CELL_SIZE - 1);
            g.setColor(java.awt.Color.GRAY);
            g.drawRect(px, py, CELL_SIZE - 1, CELL_SIZE - 1);
        }

        @Override
        public void onGameStateChanged() {
            SwingUtilities.invokeLater(() -> {
                scoreLabel.setText("Счет: " + board.getScore());
                levelLabel.setText("Уровень: " + board.getLevel());
                linesLabel.setText("Линий: " + board.getLinesCleared());
                nextPiecePreview.repaint();
                gamePanel.repaint();

                int newDelay = Math.max(100, 1000 - (board.getLevel() - 1) * 100);
                gameTimer.setDelay(newDelay);

                sendNetworkState();
                updateLocalOpponentInfo();
            });
        }

        // ===== СЕТЕВЫЕ МЕТОДЫ =====

        private void btnNewGameActionPerformed(java.awt.event.ActionEvent evt) {
            if (networkSendTimer != null) networkSendTimer.stop();
            if (networkDialog != null) {
                networkDialog.realDispose();
                networkDialog = null;
            }

            gameTimer.stop();
            initGame();
            gamePanel.repaint();
        }

        private void btnPauseActionPerformed(java.awt.event.ActionEvent evt) {
            togglePause();
            gamePanel.requestFocusInWindow();
        }

```

```

    }

    private void btnNetworkActionPerformed(java.awt.event.ActionEvent evt) {
        String[] options = {"По сети", "На одном компьютере"};
        int mode = JOptionPane.showOptionDialog(this,
            "Выберите режим совместной игры",
            "Совместная игра",
            JOptionPane.DEFAULT_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,
            options,
            options[0]);

        if (mode == 1) {
            if (gameTimer != null) {
                gameTimer.stop();
            }

            if (networkSendTimer != null) {
                networkSendTimer.stop();
            }

            this.dispose();

            TetrisFrame player1 = new TetrisFrame(
                "Игрок 1 – WASD/V/SHIFT",
                "Игрок 1",
                java.awt.event.KeyEvent.VK_A,
                java.awt.event.KeyEvent.VK_D,
                java.awt.event.KeyEvent.VK_S,
                java.awt.event.KeyEvent.VK_V,
                java.awt.event.KeyEvent.VK_SHIFT
            );

            TetrisFrame player2 = new TetrisFrame(
                "Игрок 2 – Стрелки/ENTER",
                "Игрок 2",
                java.awt.event.KeyEvent.VK_LEFT,
                java.awt.event.KeyEvent.VK_RIGHT,
                java.awt.event.KeyEvent.VK_DOWN,
                java.awt.event.KeyEvent.VK_UP,
                java.awt.event.KeyEvent.VK_ENTER
            );

            player1.setLocalPausePeer(player2);
            player2.setLocalPausePeer(player1);

            player1.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);
            player2.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);

            player1.setLocation(100, 100);
            player2.setLocation(950, 100);

            player1.setVisible(true);
            player2.setVisible(true);

            player1.gamePanel.requestFocusInWindow();
            player2.gamePanel.requestFocusInWindow();

            return;
        }
    }

```

```

    }

    if (networkDialog == null || !networkDialog.isConnected()) {
        networkDialog = new NetworkDialog(this);
        networkDialog.setVisible(true);
        startNetworkTimer();
    } else {
        JOptionPane.showMessageDialog(this, "Вы уже подключены к сетевой
игре!");
    }
}

private void startNetworkTimer() {
    networkSendTimer = new javax.swing.Timer(500, e -> {
        sendNetworkState();
    });
    networkSendTimer.start();
}

public void updateOpponentStats(String name, int score, int level) {
    updateOpponentStats(name, score, level, -1, null);
}

public void updateOpponentStats(String name, int score, int level, int
lines, String status) {
    Runnable update = () -> {
        opponentNameLabel.setText("Соперник: " + name);
        opponentScoreLabel.setText("Счет: " + score + (status != null ? " |
" + status : ""));
        opponentLevelLabel.setText("Уровень: " + level + (lines >= 0 ? " |
Линий: " + lines : ""));
    };

    if (SwingUtilities.isEventDispatchThread()) {
        update.run();
    } else {
        SwingUtilities.invokeLater(update);
    }
}

public void opponentGameOver(String opponentName, boolean youWon) {
    SwingUtilities.invokeLater(() -> {
        if (youWon) {
            JOptionPane.showMessageDialog(this,
                "🏆 ПОБЕДА! 🏆\n\n" +
                "Соперник " + opponentName + " проиграл!\n\n" +
                "Ваш счет: " + board.getScore(),
                "Игра окончена",
                JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this,
                "😞 ПОРАЖЕНИЕ 😞\n\n" +
                "Соперник " + opponentName + " победил!",
                "Игра окончена",
                JOptionPane.ERROR_MESSAGE);
        }
        gameTimer.stop();
    });
}

public int getScore() { return board.getScore(); }

```

```

/**
 * This method is called from within the constructor to initialize the
form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN: initComponents
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    gamePanel = new javax.swing.JPanel() {
        @Override
        protected void paintComponent(java.awt.Graphics g) {
            super.paintComponent(g);
            if (board != null) {
                drawGamePanel(g);
            }
        }
    };
    nextPiecePanel = new javax.swing.JPanel();
    scoreLabel = new javax.swing.JLabel();
    levelLabel = new javax.swing.JLabel();
    linesLabel = new javax.swing.JLabel();
    nextPiecePreview = new javax.swing.JPanel() {
        @Override
        protected void paintComponent(java.awt.Graphics g) {
            super.paintComponent(g);
            if (board != null) {
                drawNextPiece(g);
            }
        }
    };
    opponentPanel = new javax.swing.JPanel();
    opponentNameLabel = new javax.swing.JLabel();
    opponentScoreLabel = new javax.swing.JLabel();
    opponentLevelLabel = new javax.swing.JLabel();
    btnNewGame = new javax.swing.JButton();
    btnPause = new javax.swing.JButton();
    btnNetwork = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Тетрис - Курсовая работа");

    jPanel1.setBackground(new java.awt.Color(204, 204, 204));

    gamePanel.setBackground(new java.awt.Color(0, 0, 0));
    gamePanel.setPreferredSize(new java.awt.Dimension(300, 600));
    gamePanel.setLayout(null);

    nextPiecePanel.setBackground(new java.awt.Color(255, 255, 255));

    scoreLabel.setText("Счет: 0");

    levelLabel.setText("Уровень: 1");

    linesLabel.setText("Линий: 0");

    javax.swing.GroupLayout nextPiecePanelLayout = new
javax.swing.GroupLayout(nextPiecePanel);
    nextPiecePanel.setLayout(nextPiecePanelLayout);

```

```

        nextPiecePanelLayout.setHorizontalGroup(

nextPiecePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(nextPiecePanelLayout.createSequentialGroup()
            .addGap(19, 19, 19)

.addGroup(nextPiecePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(levelLabel,
                javax.swing.GroupLayout.DEFAULT_SIZE, 97, Short.MAX_VALUE)
            .addComponent(scoreLabel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addComponent(linesLabel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE))
            .addContainerGap(84, Short.MAX_VALUE))
        );
        nextPiecePanelLayout.setVerticalGroup(

nextPiecePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(nextPiecePanelLayout.createSequentialGroup()
            .addContainerGap(22, Short.MAX_VALUE)
            .addComponent(scoreLabel)
            .addGap(18, 18, 18)
            .addComponent(levelLabel)
            .addGap(18, 18, 18)
            .addComponent(linesLabel)
            .addGap(45, 45, 45))
        );

        nextPiecePreview.setBackground(new java.awt.Color(0, 0, 0));

nextPiecePreview.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.
    BorderFactory.createLineBorder(new java.awt.Color(255, 255, 255)),
    "Следующая", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
    javax.swing.border.TitledBorder.DEFAULT_POSITION, null, new java.awt.Color(255,
    255, 255)));
        nextPiecePreview.setPreferredSize(new java.awt.Dimension(200, 150));
        nextPiecePreview.setLayout(null);

        opponentPanel.setBackground(new java.awt.Color(100, 100, 100));

opponentPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.
    BorderFactory.createLineBorder(new java.awt.Color(255, 0, 0)), "СОПЕРНИК",
    javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
    javax.swing.border.TitledBorder.DEFAULT_POSITION, null, new java.awt.Color(255,
    0, 0)));
        opponentPanel.setPreferredSize(new java.awt.Dimension(200, 100));
        opponentPanel.setLayout(new java.awt.GridLayout(3, 1));

        opponentNameLabel.setText("Соперник: нет данных");
        opponentNameLabel.setForeground(java.awt.Color.WHITE);
        opponentPanel.add(opponentNameLabel);

        opponentScoreLabel.setText("Счет: - | Статус: -");
        opponentScoreLabel.setForeground(java.awt.Color.WHITE);
        opponentPanel.add(opponentScoreLabel);

        opponentLevelLabel.setText("Уровень: - | Линий: -");

```

```

opponentLevelLabel.setForeground(java.awt.Color.WHITE);
opponentPanel.add(opponentLevelLabel);

btnNewGame.setText("Новая игра");
btnNewGame.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnNewGameActionPerformed(evt);
    }
});

btnPause.setText("Пауза (P)");
btnPause.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnPauseActionPerformed(evt);
    }
});

btnNetwork.setText("Сетевая игра");
btnNetwork.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnNetworkActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(jPanel1Layout.createSequentialGroup()
        .addGap(15, 15, 15)
        .addComponent(gamePanel,
            javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
            javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)

        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(nextPiecePanel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addComponent(nextPiecePreview,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addComponent(opponentPanel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(15, 15, 15)
                .addComponent(btnNewGame)

                .addGap(15, 15, 15)
                .addComponent(btnPause)

                .addGap(15, 15, 15)
                .addComponent(btnNetwork)))
        .addGap(25, 25, 25)
    );
jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(jPanel1Layout.createSequentialGroup()
        .addGap(15, 15, 15)
        .addComponent(gamePanel,
            javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
            javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(nextPiecePanel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addComponent(nextPiecePreview,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addComponent(opponentPanel,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(15, 15, 15)
                .addComponent(btnNewGame)

                .addGap(15, 15, 15)
                .addComponent(btnPause)

                .addGap(15, 15, 15)
                .addComponent(btnNetwork)))
        .addGap(25, 25, 25)
    );
jPanel1Layout.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

```

        .addGap(15, 15, 15)

        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(gamePanel,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(nextPiecePanel,
                    javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(nextPiecePreview,
                    javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(opponentPanel,
                    javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE)

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
                    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(btnNewGame,
                        javax.swing.GroupLayout.PREFERRED_SIZE, 42,
                        javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnPause,
                        javax.swing.GroupLayout.PREFERRED_SIZE, 42,
                        javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnNetwork,
                        javax.swing.GroupLayout.PREFERRED_SIZE, 42,
                        javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addContainerGap(25, Short.MAX_VALUE))
    );

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

        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(nextPiecePanel, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(nextPiecePreview, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(opponentPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(btnNewGame, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(btnPause, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(btnNetwork, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap(25, Short.MAX_VALUE)))
                .addContainerGap())
    );
    layout.setVerticalGroup(

        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(nextPiecePanel, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(nextPiecePreview, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(opponentPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(btnNewGame, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(btnPause, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(btnNetwork, javax.swing.GroupLayout.DEFAULT_SIZE,
                            javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap(25, Short.MAX_VALUE)))
                .addContainerGap())
    );

    pack();

```

```

} // </editor-fold> // GEN-END: initComponents

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    try {

        javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        e.printStackTrace();
    }

    java.awt.EventQueue.invokeLater(() -> new
        TetrisFrame().setVisible(true));
}

// Variables declaration - do not modify // GEN-BEGIN: variables
private javax.swing.JButton btnNetwork;
private javax.swing.JButton btnNewGame;
private javax.swing.JButton btnPause;
private javax.swing.JPanel gamePanel;
private javax.swing.JPanel jPanel1;
private javax.swing.JLabel levelLabel;
private javax.swing.JLabel linesLabel;
private javax.swing.JPanel nextPiecePanel;
private javax.swing.JPanel nextPiecePreview;
private javax.swing.JPanel opponentPanel;
private javax.swing.JLabel opponentNameLabel;
private javax.swing.JLabel opponentScoreLabel;
private javax.swing.JLabel opponentLevelLabel;
private javax.swing.JLabel scoreLabel;
// End of variables declaration // GEN-END: variables
}

```

kursovaya/network/GameStatePacket.java

```

package kursovaya.network;

import java.io.Serializable;
import java.util.List;

public class GameStatePacket implements Serializable {
    private static final long serialVersionUID = 1L;

    private String playerName;
    private int score;
    private int level;
    private int linesCleared;
    private boolean gameOver;
    private boolean winner;
    private boolean paused;
    private List<List<Integer>> boardState;

    public GameStatePacket(String playerName, int score, int level, int
        linesCleared,
        boolean gameOver, boolean winner, boolean paused,
        List<List<Integer>> boardState) {
        this.playerName = playerName;
        this.score = score;
    }
}

```

```

        this.level = level;
        this.linesCleared = linesCleared;
        this.gameOver = gameOver;
        this.winner = winner;
        this.paused = paused;
        this.boardState = boardState;
    }

    public String getPlayerName() { return playerName; }
    public int getScore() { return score; }
    public int getLevel() { return level; }
    public int getLinesCleared() { return linesCleared; }
    public boolean isGameOver() { return gameOver; }
    public boolean isWinner() { return winner; }
    public boolean isPaused() { return paused; }
    public List<List<Integer>> getBoardState() { return boardState; }
}

```

kursovaya/network/NetworkManager.java

```

package kursovaya.network;

import java.io.*;
import java.net.*;
import java.util.concurrent.*;
import java.util.function.Consumer;

public class NetworkManager {
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private ObjectOutputStream out;
    private ObjectInputStream in;
    private final ExecutorService executor = Executors.newCachedThreadPool();
    private Consumer<GameStatePacket> stateListener;
    private volatile boolean connected = false;

    public void hostGame(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        executor.submit(() -> {
            try {
                clientSocket = serverSocket.accept();
                setupStreams();
                connected = true;
                startListening();
            } catch (IOException e) {
                System.err.println("Ошибка сервера: " + e.getMessage());
            }
        });
    }

    public void joinGame(String host, int port) throws IOException {
        clientSocket = new Socket(host, port);
        setupStreams();
        connected = true;
        startListening();
    }

    private void setupStreams() throws IOException {
        out = new ObjectOutputStream(clientSocket.getOutputStream());
        in = new ObjectInputStream(clientSocket.getInputStream());
    }
}

```

```

private void startListening() {
    executor.submit(() -> {
        while (connected) {
            try {
                GameStatePacket packet = (GameStatePacket) in.readObject();
                if (stateListener != null) {
                    stateListener.accept(packet);
                }
            } catch (IOException | ClassNotFoundException e) {
                connected = false;
                System.err.println("Соединение разорвано: " +
e.getMessage());
            }
        }
    });
}

public void sendState(GameStatePacket packet) {
    if (out != null) {
        try {
            out.writeObject(packet);
            out.flush();
        } catch (IOException e) {
            System.err.println("Ошибка отправки: " + e.getMessage());
        }
    }
}

public void setStateListener(Consumer<GameStatePacket> listener) {
    this.stateListener = listener;
}

public void disconnect() {
    connected = false;
    try {
        if (out != null) out.close();
        if (in != null) in.close();
        if (clientSocket != null) clientSocket.close();
        if (serverSocket != null) serverSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    executor.shutdown();
}

public boolean isConnected() { return connected; }
}

```

Приложение Б. UML-диаграммы
Приложение Б.1. UML-диаграмма вариантов использования



Рисунок Б.1 - UML-диаграмма вариантов использования

Приложение Б.2. UML-диаграмма классов

UML-диаграмма классов приложения «Тетрис»

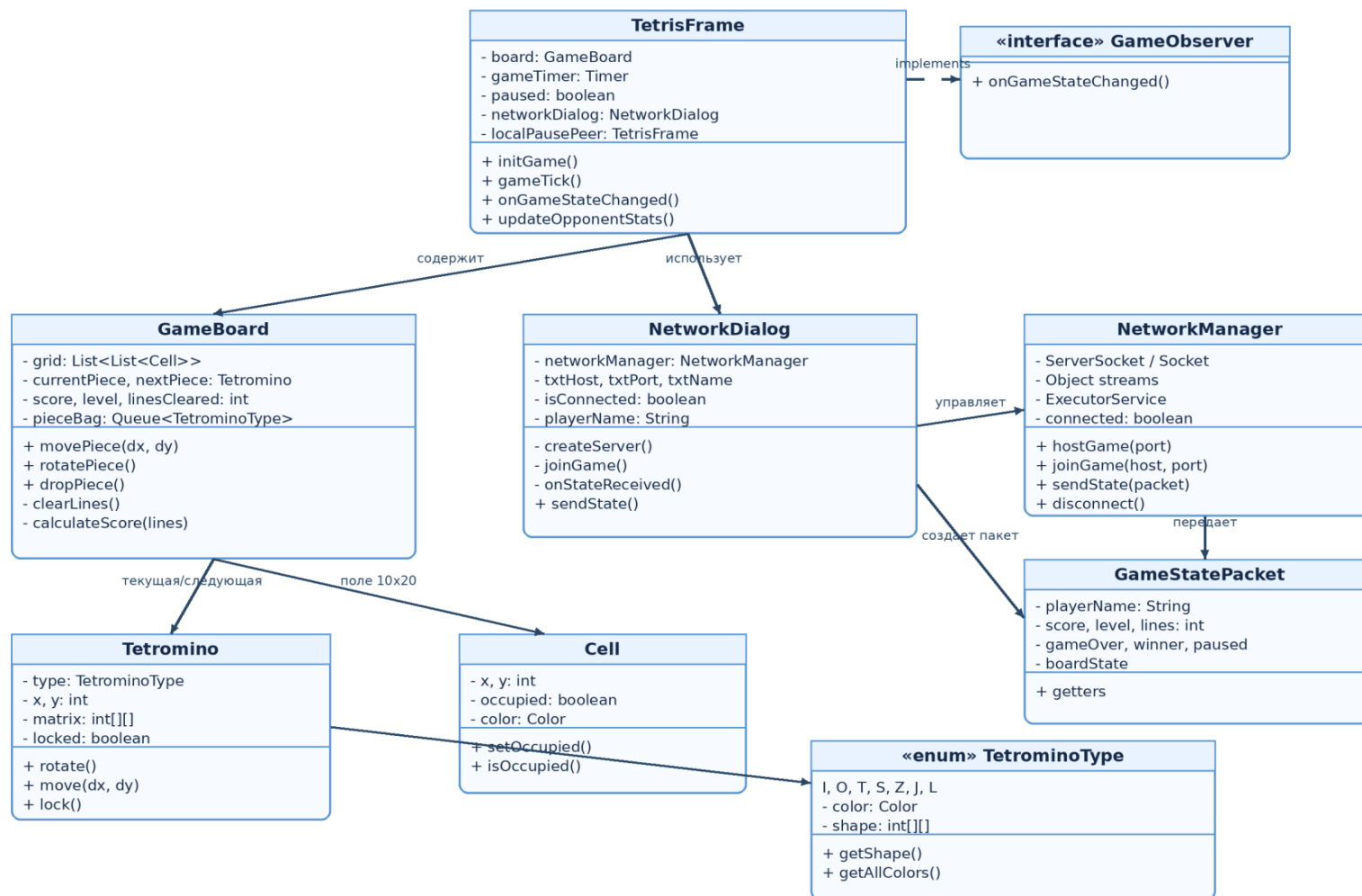


Рисунок Б.2 - UML-диаграмма классов

Приложение Б.3. UML-диаграмма деятельности

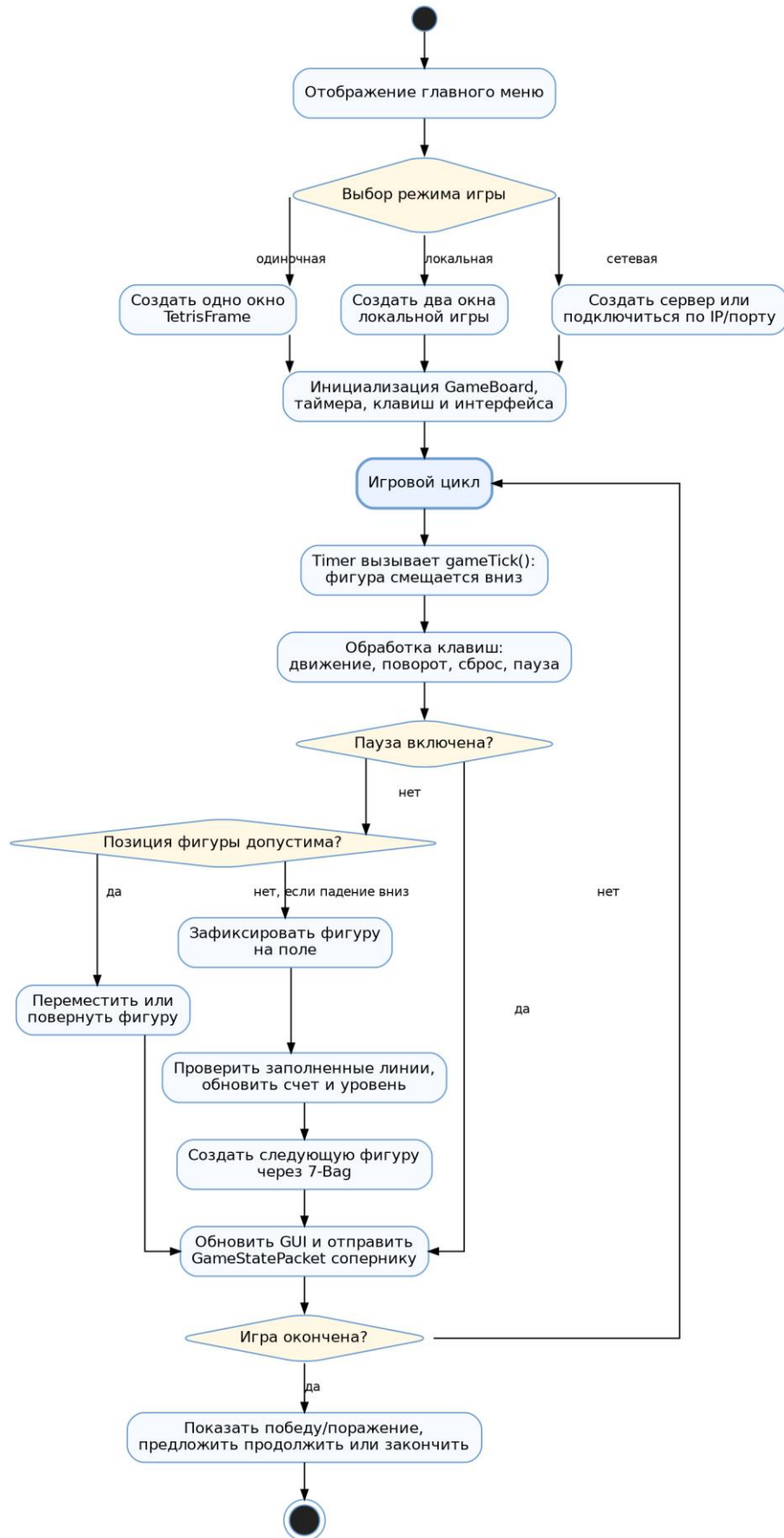


Рисунок Б.3 - UML-диаграмма деятельности

Приложение Б.4. UML-диаграмма последовательности

UML-диаграмма последовательности игрового процесса

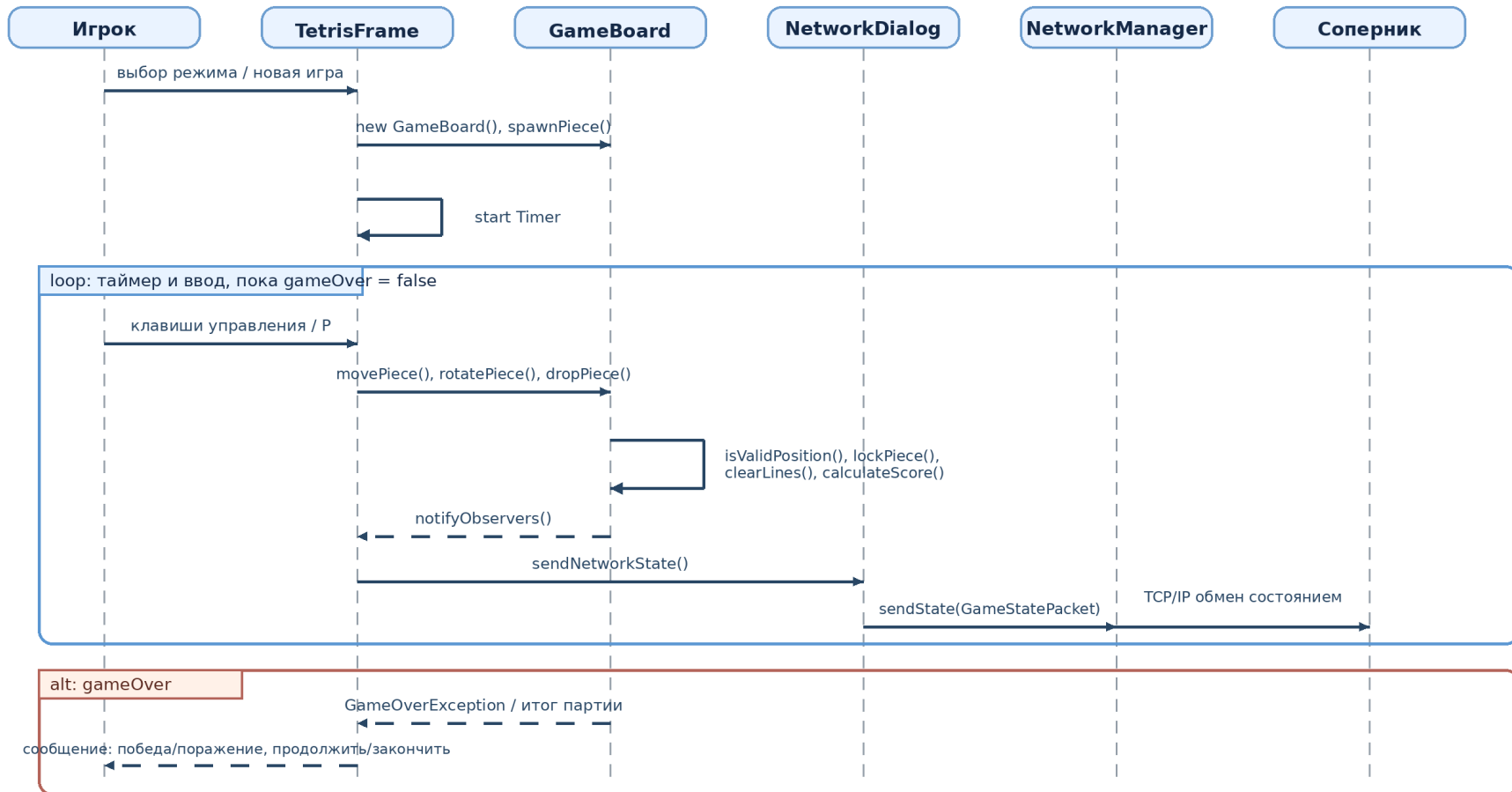


Рисунок Б.4 - UML-диаграмма последовательности

Приложение Б.5. UML-диаграмма развертывания

UML-диаграмма развертывания

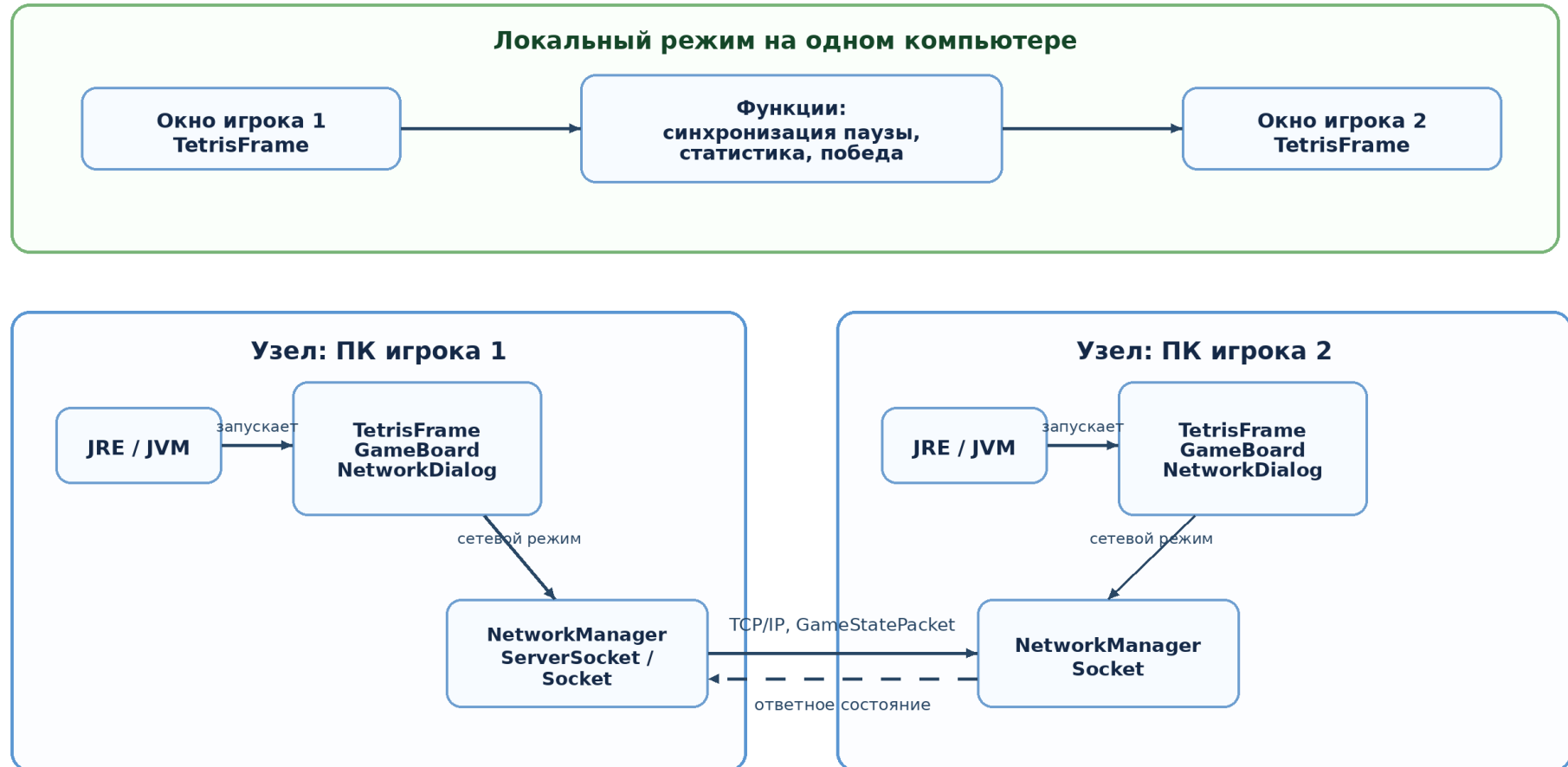


Рисунок Б.5 - UML-диаграмма развертывания