

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление
подготовки

09.03.01 «Информатика и вычислительная
техника»

Профиль

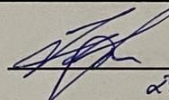
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

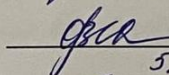
на тему

**КРОССПЛАТФОРМЕННОЕ КЛИЕНТ-
СЕРВЕРНОЕ ПРИЛОЖЕНИЕ**

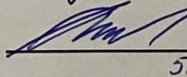
Студент
(подпись, дата)

 Филатова Д.С.
29.05.2023

Руководитель
(подпись, дата)

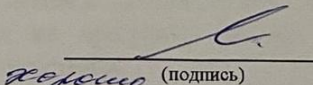
 Юрова О.В.
5.06.2023

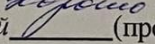
Нормоконтролёр
(подпись, дата)

 Бычков А.С.
5.06.2023

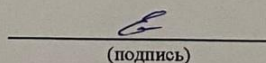
Работа допущена к защите (протокол №14 заседания кафедры от) 2.06.2023г.

Заведующий кафедрой

 М.А. Митрохин
(подпись)

Работа защищена с отметкой  (протокол заседания ГЭК от 14.06.2023 №15)

Секретарь ГЭК

 Гурин Е.И.
(подпись)

Пенза, 2023

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой
М.А. Митрохин

«26» 04 2023 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Филатова Дарья Сергеевна гр.19ВВП2 факультета вычислительной техники.

2. Тема работы: «Кроссплатформенное клиент-серверное приложение»

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы к.т.н., Юрова О.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать игровое приложение, которое поддерживает игру для двух пользователей через локальную вычислительную сеть, имеет понятно интуитивный интерфейс. У игроков должна быть возможность делать ход из всех возможных вариантов. Так же программа должна определить победителя и уведомить об этом каждому участнику.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области, выбрать язык программирования для приложения и среду разработки, провести сравнительный анализ протоколов передачи данных, изучить и понять нюансы разработки кроссплатформенных приложений.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	26.04.23 - 29.04.23	<i>фис</i>
2.	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	15 %	30.04.23 - 04.05.23	<i>фис</i>
3.	Разработка клиентской части	20 %	05.05.23 - 09.05.23	<i>фис</i>
4.	Разработка серверной части	25 %	10.05.23 - 21.05.23	<i>фис</i>
5.	Тестирование и отладка	15 %	22.05.23 - 24.05.23	<i>фис</i>
6.	Выполнение задания по БЖД	5 %	25.05.23 - 27.05.23	<i>фис</i>
7.	Оформление выпускной работы	10 %	28.05.23 - 31.05.23	<i>фис</i>

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы *фис* Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент *Филатова* Филатова Д.С.
(подпись, дата)

Работу к защите допустить
Декан факультета *13.06.2023* Фионова Л.Р.
(подпись, дата)

Реферат

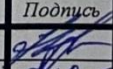
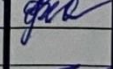
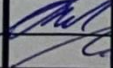
Бакалаврская работа содержит 73 л., 18 рис., 4 источн. и 3 прил.
КРОССПЛАТФОРМЕННОСТЬ, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ,
СОКЕТЫ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, ПАТТЕРНЫ

Цель работы – разработать кроссплатформенное приложение, вида клиент-сервер, с возможностью запускать игру «Домино» для двух игроков по локальной сети.

Объектами разработки являются:

- сервер;
- клиент.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана серверная часть приложения, которая принимает информацию от игрока, передает ее обратно, ведет подсчет ходов, разработано API для взаимодействия серверной и клиентской части приложения, разработана клиентская часть приложения с графическим интерфейсом позволяющая взаимодействовать с серверной частью.

					ПГУ1.090301.37.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата				
Разраб		Филатова Д.С.			Кроссплатформенное клиент-серверное приложение Пояснительная записка	Литера	Лист	Листов
Пров		Юрова О.В.				у	2	73
Н. Контр.		Бычков А.С.			ФВТ, каф. ВТ Гр. 19ВВП2			
Утв		Митрахин М.А.						

Реферат

Бакалаврская работа содержит 76 л., 18 рис., 4 источн. и 7 прил.

КРОССПЛАТФОРМЕННОСТЬ, КЛИЕНТ-СЕРВЕРНОЕ
ПРИЛОЖЕНИЕ, СОКЕТЫ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС,
ПАТТЕРНЫ

Цель работы – разработать кроссплатформенное приложение, вида клиент-сервер, с возможностью запускать игру «Домино» для двух игроков по локальной сети.

Объектами разработки являются:

–Сервер;

–Клиент.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана серверная часть приложения, которая принимает информацию от игрока, передает ее обратно, ведет подсчет ходов, разработано API для взаимодействия серверной и клиентской части приложения, разработана клиентская часть приложения с графическим интерфейсом позволяющая взаимодействовать с серверной частью.

					ПГУ1.090301.37.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата				
Разраб		Филатова Д.С.			Кроссплатформенное клиент-серверное приложение Пояснительная записка	Литера	Лист	Листов
Пров		Юрова О.В.				у	2	75
Н. Контр.		Бычков А.С.				ФВТ, каф. ВТ Гр. 19ВВП2		
Утв		Митрохин М.А.						

Содержание

Реферат	3
Перечень обозначений и сокращений.....	7
Введение.....	8
1 Кроссплатформенное игровое приложение	9
1.1 Анализ предметной области и постановка задачи	10
1.2 Анализ аналогов.....	12
1.3 Постановка задачи	17
1.4 Выбор инструментов разработки	18
1.5 Описание программы	25
1.6 Реализация	27
1.6.1 Общая часть.....	27
1.6.2 Серверная часть	30
1.6.3 Клиентская часть.....	30
1.6.4 Рендеринг.....	32
1.7 Описание способа организации пользовательского.....	35
интерфейса.....	35
1.8 Описание результатов работы программы.....	36
2 Охрана труда.....	40
2.1 Требования к организации рабочего места с ПК	40
2.2 Требования к освещению помещений и рабочих мест.....	41
Заключение	44
Список используемых источников	45
Приложение А- Листинг программы.....	46
Приложение Б.- UML-диаграммы	64
Приложение В- Презентация	72

Перечень обозначений и сокращений

ОС – операционная система

ЭВМ – Электронно-вычислительная машина

API (Application Programming Interface) – программный интерфейс приложения.

AWT (Abstract Window Toolkit) – исходная платформо-независимая оконная библиотека графического интерфейса языка *Java*.

IDE (integrated development environment) – среда разработки.

GUI (graphical user interface) – графический интерфейс пользователя.

TCP (Transmission Control Protocol) – протокол передачи данных.

UNIX (Uniplexed Information and Computing Service) – единый информационно-вычислительный сервис.

Введение

Эра компьютерных игр стала зарождаться уже в 19 веке, как только человечество изобрело первые электронно-вычислительные машины (ЭВМ).

Научные открытия и новинки вычислительной техники развивались благодаря игровой и развлекательной индустрии. Раньше считалось, что другие области предлагают большинство новинок для области развлечений. Но оказалось, что игры очень весомо помогают в развитии масштабных проектов (многие из них начались с разработки игровых приложений).

Если анализировать прошлое индустрии игр, то мы видим, что какие то приложения получают известность и становятся популярными, а другие тормозят финансовое благополучие игровой отрасли. Или, например, появляется множество похожих игр, и они не выдерживают конкуренции у потребителей. Так же, в связи с появлением более современных компьютеров, развитием компьютерных технологий, игры стали намного интереснее и сложнее, что должно привести к развитию технологического процесса.

Создание развлекательных приложений и игр – перспективные направления в программировании. Поэтому тем, кто только начинает изучать процесс разработки игр и программирования, стоит обратить внимание на *Java*. *Java* является довольно простым языком, с помощью которого можно создавать уникальные перспективные проекты.

Язык *Java* является базовым языком для множества сетевых приложений, а так же является мировым стандартом для написания встроенных и мобильных приложений, игр, и корпоративного программного обеспечения. Благодаря кроссплатформенности быстро распространился среди программистов. *Java* вошел в ТОП-5 языков программирования в разных рейтингах, разных изданий и статистик. Специалисты, при разработке приложений, все чаще стали прибегать к данному языку, который позволяет эффективно разрабатывать, тестировать, внедрять и

использовать программное обеспечение.

1 Кроссплатформенное игровое приложение

1.1 Анализ предметной области и постановка задачи

Игры являются не только развлечением, но еще и двигателем прогресса, ведь они являются одной из причин стремительного роста производительности компьютеров, а также причиной для развития 3D технологий, нейронных сетей, алгоритмики.

Разнообразие игр огромно: от настольных до видеоигр, от шутеров до головоломок, от аркад до стратегий и многих других. Игры занимают важное место в досуге во многих странах мира, а в некоторых из них они признаются национальным спортом и даже профессией.

Критерии классификации компьютерных игр могут быть различными:

- Жанр: игровой процесс задает направление жанра или совокупности нескольких жанров;
- Режим игры: может быть одиночным или многопользовательским;
- Визуализация: игры могут быть графически оформленными или обыкновенными текстовыми, трехмерными и двухмерными, и даже звуковыми (когда вместо визуализации процесса используются звуки);
- Платформозависимость: какие платформы поддерживают запуск игры. Игры могут быть и платформонезависимыми или кроссплатформенными.

Сейчас есть возможность написать игровое приложение на более чем достаточном количестве языков. Но можно выделить и составить топ языков для программирования игр:

- Язык C++ считается наиболее практичным и более распространенным вариантом для создания игровых приложений и не только. Большим плюсом является огромная библиотека уже с готовыми решениями, которые можно использовать для каких либо задач. Но этот язык

подойдет уже более опытным программистам и разработчикам, ведь понадобится много времени на его изучения: все его функции и возможности. Приложения, написанные на C++ можно адаптировать под любую операционную систему и платформу, будь то компьютер, планшет или смартфон. Так же большим плюсом данного языка является многозадачность и быстрая обработка большого количества операций, что дает разработчику практически безграничные возможности.

- C#. Так же замечательно подходит для игровых приложений за счет совместимости с движком *Unity*. В отличие от C+, синтаксис C# наиболее интуитивно понятен для программиста. А огромное количество уже готовых библиотек ускорит его работу, ведь вместо полного написания программы, можно брать какие-либо уже готовые скрипты;
- Java. Наилучшее решение для разработки игровых приложений. Хотя он и не поддерживает такой известный движок, как *Unity*, он совместим с — *ngine, jMonkey, LibGDX* и др.. На них можно создать игры абсолютно на любой девайс, которым мы пользуемся повседневно. А так же доступно множество готовых библиотек. Так же плюс данного языка — его простота в изучении. Нужно лишь разобраться и понять что такое объектно-ориентированное программирование;
- JavaScript. По простоте обучения так же легок, как и Java. Так же имеются библиотеки. Отличие в том, что JavaScript подходит только для написания простых немногомерных браузерных игр или игр на смартфон;
- PHP. Данный язык используют для написания кода, отвечающего за мультиплеер и за базы данных. То есть полноценную игру на нем не

создать. Зато в этом языке легко разобраться, стоит лишь понять основы.

1.2 Анализ аналогов

Язык *Java* стоял у истоков начала развлекательной индустрии, потому многие, знакомые нам с детства, игровые приложения были написаны именно на нем. В 21 веке появляется все больше игр разнообразных жанров, таких как: квесты, боевики или экшены, стратегии или симуляции. Но начало было положено самыми обычными, всем известными карточными играми, игрой крестики-нолики и др..

Пятнашки.

Данную игру можно отнести к жанру головоломки. Размер игрового поля 4x4, то есть больше количества игровых костяшек, которых всего 15. Интерфейс игры представлен на рисунок 1. 1.

Суть головоломки в том, чтобы игрок выставил все кости, пронумерованные от 1 до 15, в правильном порядке. Для их передвижения используется пустая «пятнашка». Передвигаться по полю могут все кости.

Теперь, когда все цифры встали на свои места, в пустой «пятнашке» отображается «галочка», которая уведомляет игрока о его победе; представлено на рисунок 1. 2. Так же появляется надпись, с помощью которой мы можем начать игру заново. Либо просто выйти из игры.

В данной игре мы можем только закончить игру, начать заново или выйти из нее. Минусом будет, отсутствие ранговой системы, а так же уровней на усложнение.

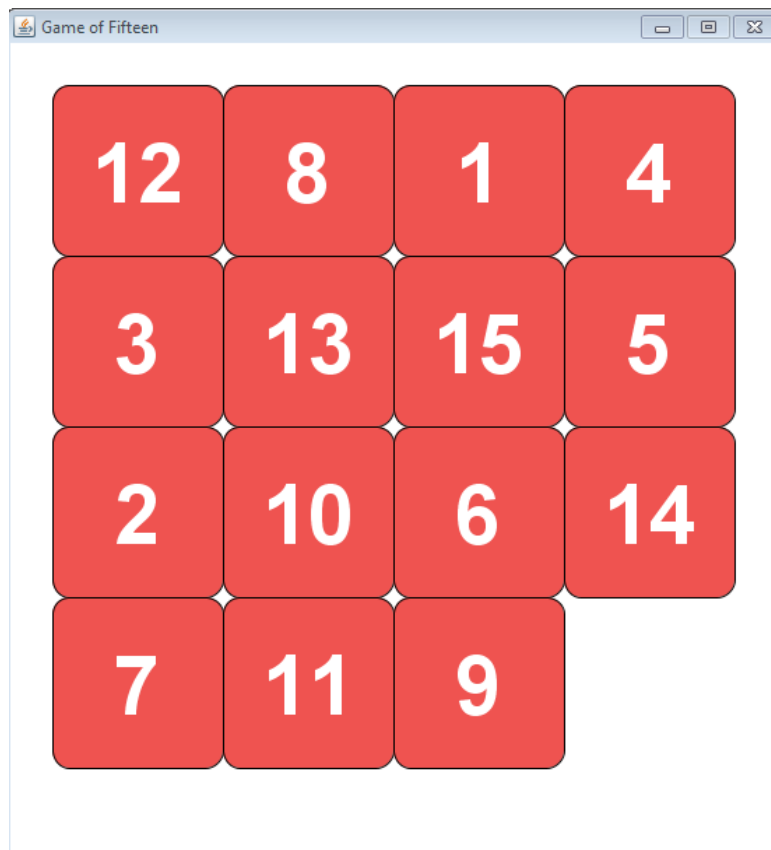


рисунок 1. 1 Игровое поле «Пятнашки»

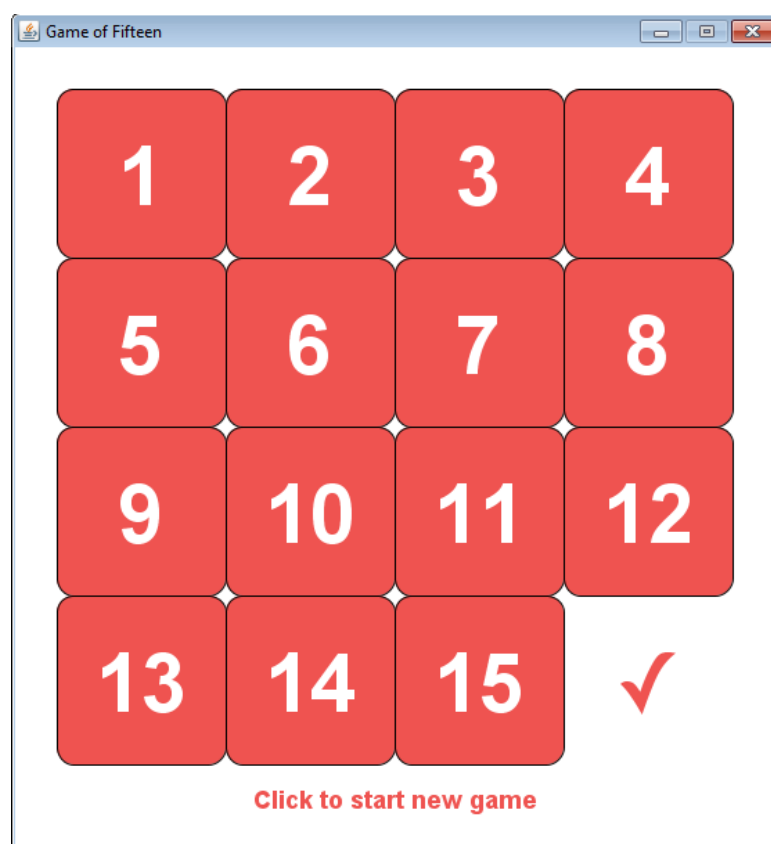


рисунок 1. 2 Победа игрока

Крестики-нолики.

Рассмотрим самый примитивный вариант реализации игры крестики-нолики. Нам предоставлено изначально пустое расчерченное поле 3x3. Он представлен на рисунок 1. 3. Первый ход, по правилам игры, всегда делается значком «крестик».

Наши игроки поэтапно делают ходы: «крестик» или «нолик». Игра продолжается пока один из игроков не составил выигрышную комбинацию или игра пришла к ситуации «ничья». После выигрыша игрокам высвечивается сообщение о победе. Смотреть рисунок 1. 4.

После раунда, игроки могут либо продолжить игру и сыграть новый раунд, либо закончить ее полностью.

У данного аналога довольно простой и интуитивный интерфейс. Минусом же является отсутствие подсчета выигрышей. Так же стоило бы добавить возможность сдаться игроку на любом этапе игры.

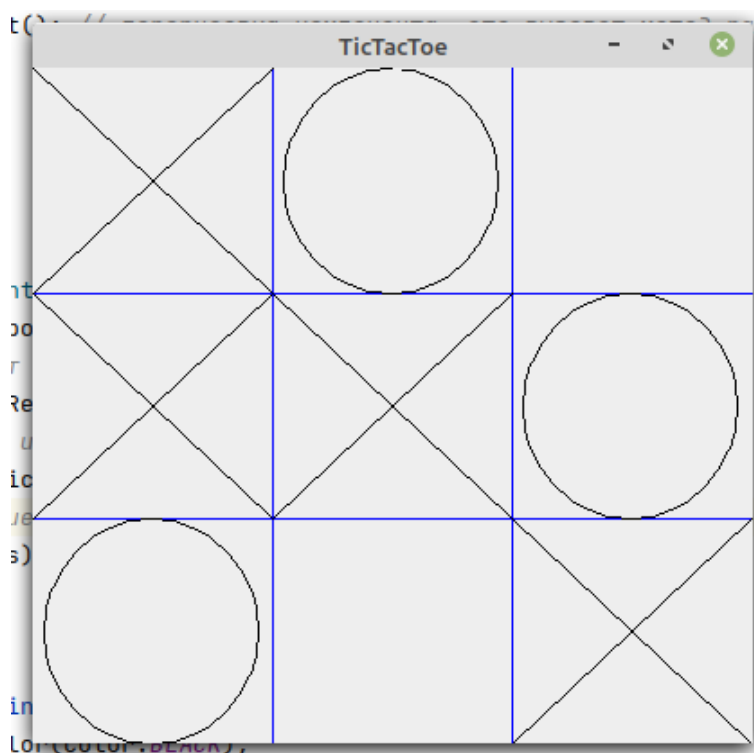


рисунок 1. 3 Игровое поле «Крестики-нолики»

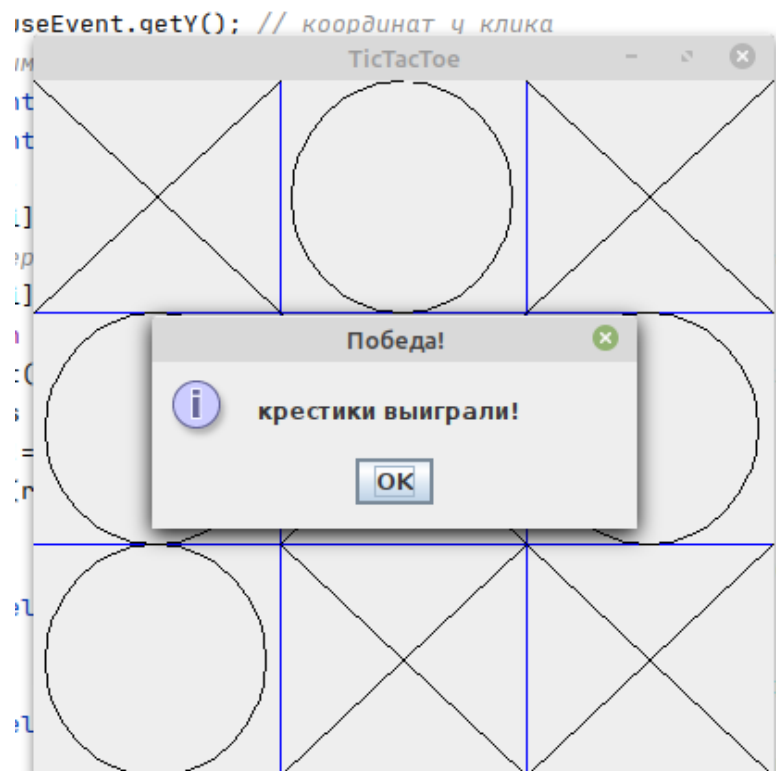


рисунок 1. 4 Сообщение о выигрыше «Крестиков»

Игра 2048.

Еще одна игра из серий головоломки. Игровое поле имеет форму квадрата 4x4. Смотреть рисунок 1. 5. Целью игры является получение плитки номинала «2048».

В самом начале появляются две плитки номиналом «2». Далее с помощью кнопок «вверх», «вниз», «влево», «вправо» игрок сдвигает все плитки в одно из направлений. При этом две плитки одного номинала складываются друг с другом, а на поле выскакивают новые плитки. Если при нажатии кнопки местоположение плиток или их номинал не изменится, то ход не совершается. Если в одной строчке или в одном столбце находится более двух плиток одного номинала, то при сбрасывании они начинают соединяться с той стороны, в которую были направлены. Так продолжается, пока на поле не остается пустых ячеек. В этом случае игра заканчивается и игрок может начать сначала.

Снизу поля идет подсчет баллов, набранных игроком. Так же игрок

может закончить игру. Представлено на рисунок 1. 6.



рисунок 1. 5 Игровое поле «2048»

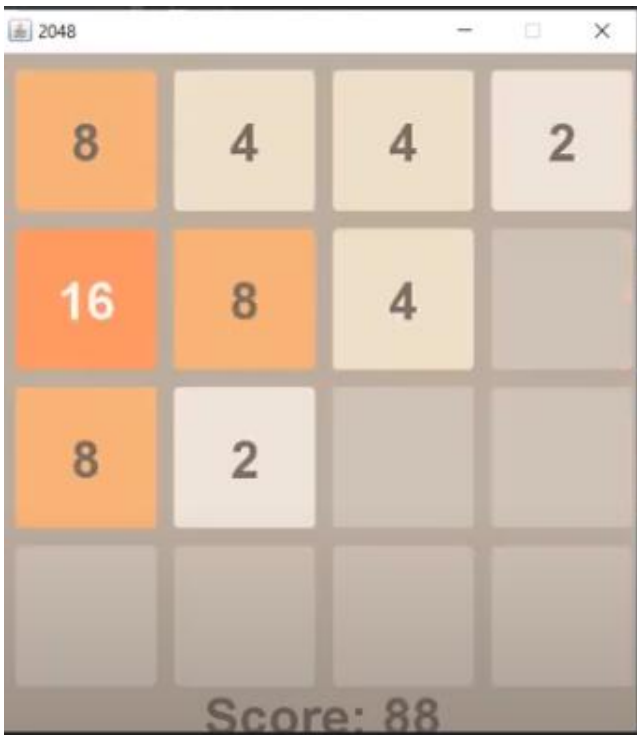


рисунок 1. 6 Подсчет баллов

1.3 Постановка задачи

Разработать систему многомодульных программ, клиент-серверной архитектуры, позволяющую играть в домино по локальной вычислительной сети.

Функции сервера: прием информации от пользователя, ее передача обратно, просчет ходов, выдача костяшек игрокам.

Функции клиента: графический интерфейс пользователя, работа с игровым состоянием, сетевое взаимодействие с сервером.

Операционная система: *Windows*;

Среда разработки: *IntelliJ IDEA*;

Язык программирования *Java*.

Используемые технологии:

Java Collections Framework

Механизм обработки исключительных ситуаций

Java Stream API

Java Multithreading

Сетевое взаимодействие

Основной функционал программы представлен на диаграмме вариантов использования (рисунок 1.7).

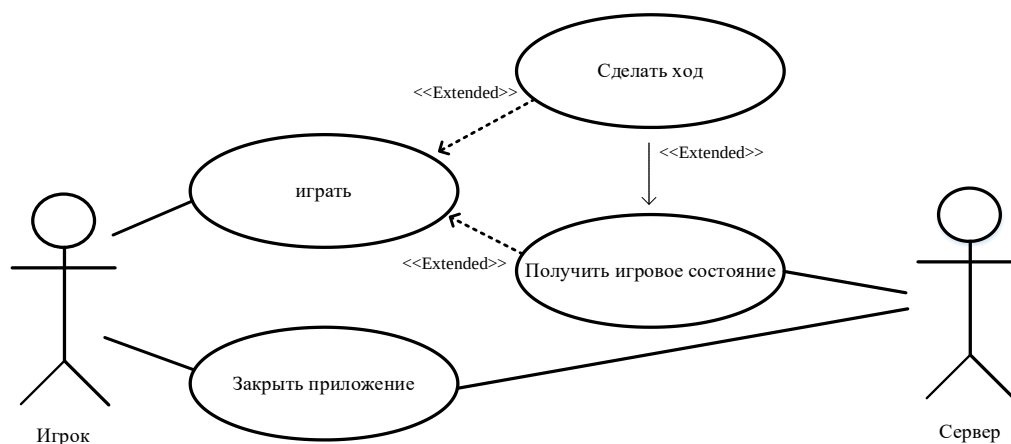


рисунок 1. 7 Диаграмма вариантов использования

1.4 Выбор инструментов разработки

В качестве среды разработки было выбрано *IntelliJ IDEA*. Данная среда лучше всего подходит для разработки приложений на *Java*. Это наиболее адаптированная и функциональная *IDE*, включающая в себя множество функций и инструментов. Все это упрощает задачу разработчику.

Плюсы:

- Сильные возможности настройки
- Поддерживает различные языки программирования
- Встроенная поддержка контроля версий
- Сильная поддержка плагинов и интеграции
- Мощный компилятор
- Система сборки Gradle

Минусы:

- Высокая стоимость конечной версии
- Тяжело загружает системные ресурсы
- Крутая кривая обучения для новых разработчиков

Кроссплатформенность: *Linux, Mac OS, Windows*

Одна из важных вещей в *IDE* – это ее визуализация. Интерфейс *IntelliJ IDEA* достаточно понятен и прост, что увеличивает производительность разработчика. Интерфейс представлен на рисунок 1. 8.[2]

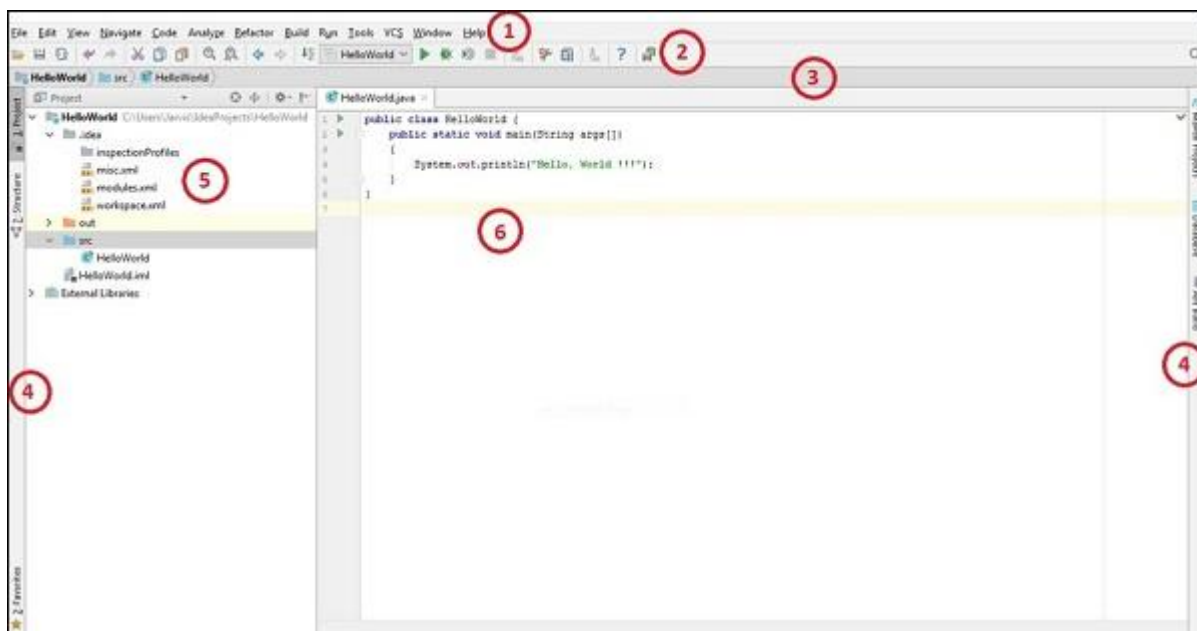


рисунок 1. 8 Интерфейс *IntelliJ IDEA*

Панель меню – Здесь нам представлены параметры, которыми мы можем воспользоваться для создания новых проектов и других важных действий, связанных с проектами, таких как рефакторинг кода, сборки, запуск, отладка, параметры управления версиями и так далее.

Панель инструментов – на панели инструментов находятся ярлыки для компиляции, отладки и запуска параметров, которые можно настроить по нашим требованиям.

Панель навигации – Панель навигации помогает адаптироваться внутри проекта. Эта функция становится полезной при увеличении базы кода.

Вкладка «Инструменты» – вкладка «Инструменты» отображается с обеих сторон главного окна, для получения доступа к важным инструментам, таким как базы данных; *Maven / Ant* строит и так далее.

Перспектива проекта – Окно перспективы проекта показывает различные элементы проектов, такие как пакеты, модули, классы, внешние библиотеки и так далее.

Окно редактора – это место, где разработчик проводит большую часть своего времени. Окно редактора позволяет редактировать / писать код с подсветкой синтаксиса и другими полезными функциями.

В нижней части главного окна находится строка состояния, в которой отображаются некоторые дополнительные атрибуты файла – такие как его формат и тип кодировки. Он также предоставляет возможность переключать атрибут только для чтения текущего файла. Вы также можете управлять уровнем контроля здесь.

Пользовательский интерфейс реализован с помощью *Swing (Java)*.

Swing API – это набор расширяемых компонентов графического интерфейса, облегчающий жизнь разработчика для создания приложений на основе *JAVA Front End / GUI*. Он построен на основе *API*-интерфейса *AWT* и выступает в качестве замены *API*-интерфейса *AWT*, поскольку имеет почти все элементы управления, соответствующие элементам управления *AWT*.

Особенности *Swing*

- у компонентов *Swing* отсутствует зависимость от *API* собственной ОС. Происходит это потому что, элементы управления отображаются, используя только код *Java*, а не как вызовы ОС.
- *Swing* предлагает разработчикам обширный набор различных элементов управления. Например, слайдеры или палитры, элементы управления таблицами и др..
- Широкие возможности настройки – так как видоизменение абсолютно не связано с внутренним представлением, то элементы управления можно легко редактировать.
- Сменный внешний вид – графический интерфейс на основе *SWING* Внешний вид приложения может изменяться во время

выполнения в зависимости от доступных значений.

SWING – Управление

Каждый пользовательский интерфейс рассматривает следующие три основных аспекта:

- Элементы пользовательского интерфейса. Это основные визуальные элементы, с которыми пользователь в конечном итоге видит и взаимодействует. *GWT* предоставляет огромный список широко используемых и распространенных элементов от базовых до сложных.[2]
- Макеты. Они определяют, как элементы пользовательского интерфейса должны быть организованы на экране, и обеспечивают окончательный внешний вид графического интерфейса пользователя.
- Поведение. Это события, которые происходят, когда пользователь взаимодействует с элементами пользовательского интерфейса.

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

В *Java* сокет обеспечивает механизм связи между двумя компьютерами, использующими *TCP*. Клиентская программа создает сокет на своем конце связи и пытается подключить этот сокет к серверу.

Когда соединение установлено, сервер создает объект сокета на своем конце связи. Клиент и сервер теперь могут общаться, записывая и считывая данные с сокета.[3]

Класс *java.net.Socket* представляет собой сокет, а класс *java.net.ServerSocket* предоставляет механизм серверной программы для прослушивания клиентов и установления соединений с ними.

При установлении соединения *TCP* между двумя компьютерами с использованием сокетов, выполняются следующие этапы:

- Сервер создает экземпляр объекта *ServerSocket*, определяющий, по какому номеру порта должна происходить связь.
- Сервер вызывает метод *accept()* класса *ServerSocket*. Этот метод ожидает, пока клиент не подключится к серверу по указанному порту.
- По завершению ожидания сервера клиент создает экземпляр объекта сокета, указывая имя сервера и номер порта подключения.
- Конструктор класса *Socket* осуществляет попытку подключить клиента к указанному серверу и номеру порта. Если связь установлена, у клиента теперь есть объект *Socket*, способный связываться с сервером.
- На стороне сервера метод *accept()* возвращает ссылку к новому сокету на сервере, который подключен к клиентскому сокету.

После того, как соединения установлены, связь может происходить с использованием потоков входных/выходных данных. Каждый сокет имеет и *OutputStream* (поток выходных данных), и *InputStream* (поток входных данных). *OutputStream* клиента подключен к *InputStream* сервера, а *InputStream* клиента подключен к *OutputStream* сервера.

Конструкторы класса *ServerSocket*

Класс *ServerSocket* имеет четыре конструктора:

- *public ServerSocket(int port) throws IOException* .Попытки создания серверного сокета, связанного с указанным портом. Исключение происходит, если порт уже связан другим приложением.

- *public ServerSocket(int port, int backlog) throws IOException* . Как и в предыдущем конструкторе, параметр *backlog* указывает, сколько входящих клиентов нужно сохранить в очереди ожидания.
- *public ServerSocket(int port, int backlog, InetAddress address) throws IOException* . Как и в предыдущем конструкторе, параметр *InetAddress* указывает локальный *IP*-адрес для осуществления привязки. *InetAddress* используется в серверах, которые могут иметь несколько *IP*-адресов, что позволяет серверу указывать *IP*-адрес приема запросов клиентов.
- *public ServerSocket() throws IOException*. Создает непривязанный сокет сервера. При использовании этого конструктора используйте метод привязки (), когда будете готовы привязать сокет сервера.

Методы класса *ServerSocket*

Ниже приведены некоторые из распространенных методов в *Java* класса *ServerSocket*.

public int getLocalPort() - возвращает порт, который прослушивает сокет сервера. Этот метод полезен, если вы передали 0 в качестве номера порта в конструкторе и позволили серверу найти порт.

public Socket accept() throws IOException – ожидает входящего клиента. Этот метод блокируется до тех пор, пока клиент не подключится к серверу на указанном порту или не истечет время ожидания сокета, при условии, что значение времени ожидания было установлено с помощью метода *setSoTimeout()*. В противном случае этот метод блокируется на неопределенный срок.

public void setSoTimeout(int timeout) - устанавливает значение времени ожидания клиента сокетом сервера во время *accept()*.

public void bind (хост *SocketAddress*, *int backlog*) - привязывает сокет к указанному серверу и порту в объекте *SocketAddress*. Используйте этот метод, если вы создали *ServerSocket* с помощью конструктора без аргументов.

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

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

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

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

1.5 Описание программы

Данная программа состоит из клиентской и серверной части. Каждый новый процесс запущенный пользователем идентичен, и “думает” что он клиент. Каждый процесс сначала пытается подключиться к серверу и, если это не удастся, считается, что сервер не запущен и данный процесс его запускает, по сути, создавая отдельную нить. Очевидно, что этот процесс, просто должен запуститься первым.

Таким образом, у нас получается два способа взаимодействия с игровыми данными – сервером: напрямую, если процесс – инициатор сервера, и с помощью сокетов, если процесс – не инициатор. Чтобы реализовать это был применен паттерн Стратегия.

Основным замыслом паттерна является следующая ситуация. Вы определяете семейство алгоритмов, инкапсулируете каждый и делаете их взаимозаменяемыми. Стратегия позволяет алгоритму варьироваться независимо от клиентов, которые используют его. [4]

Практические примеры применения шаблона «Стратегия»

Сортировка (*sorting*): мы хотим отсортировать эти числа, но мы не знаем, будем ли мы использовать *BrickSort*, *BubbleSort* или какую-либо другую сортировку. Например, у вас есть веб-сайт, на котором страница отображает элементы в зависимости от популярности. Однако «Популярным» может быть много вещей (большинство просмотров, большинство подписчиков, дата создания, большая активность, наименьшее количество комментариев). В случае, если руководство еще не знает точно, как сделать заказ, и может захотеть поэкспериментировать с различными заказами на более поздний срок, вы создаете интерфейс (*IOrderAlgorithm* или что-то еще) с методом заказа и позволяете объекту *Orderer* делегировать порядок конкретной реализации интерфейса *IOrderAlgorithm*. Вы можете создать «*CommentOrderer*», «*ActivityOrderer*» и т. д. и просто отключить их при появлении новых требований.

Обработки очереди из разнородных объектов (*queue processing and saving data*): Примером может быть прокси-система накапливающая объекты от разных источников данных, тогда после извлечения из очереди объекта и последующее его сохранение определяется стратегией выбора, на основе свойств этого объекта.

Валидация (*validation*). Нам нужно проверять элементы в соответствии с «Неким правилом», но пока не ясно, каким будет это правило, и мы можем подумать о новых. [4]

Аутентификации (*authentication*): выбор стратегии аутентификации между схемами *Basic*, *Digest*, *OpenID*, *OAuth*.

Схема паттерна приведена на рисунок 1. 9.

В нашем случае *Context* – “клиент-сервер” или “клиент-клиент”, *ConcreteStrategy1* – прямой доступ к данным игры, *ConcreteStrategy2* – сетевой.

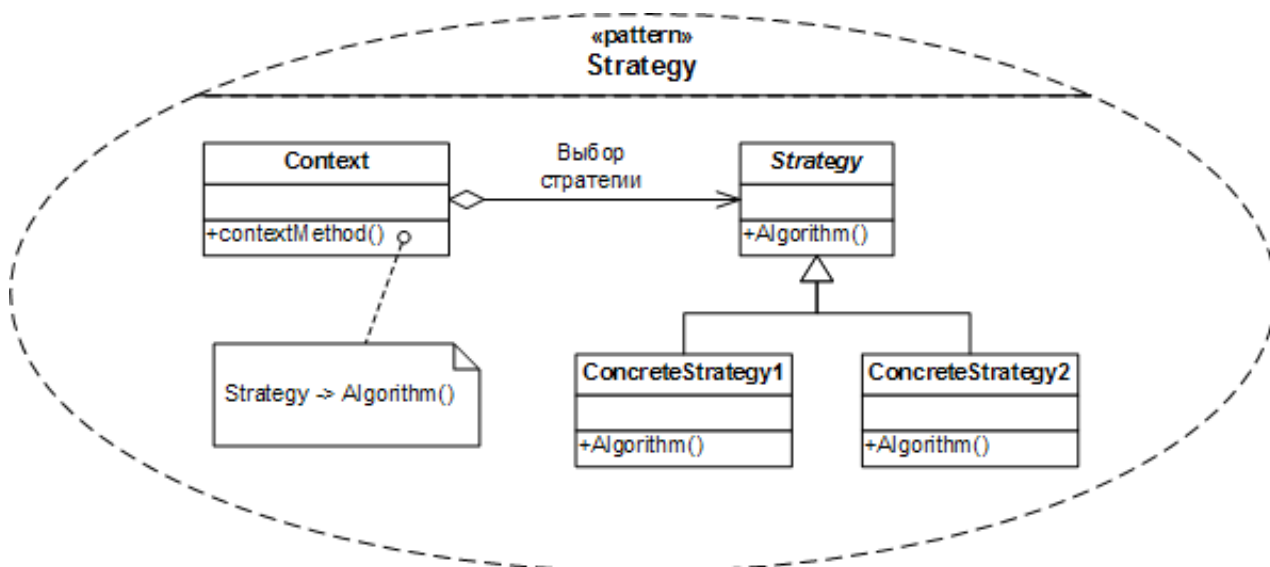


рисунок 1. 9 Схема паттерна «Стратегия»

1.6 Реализация

1.6.1 Общая часть

GameProхy.java – абстрактный класс, определяющий функции доступа к игровым данным.

Абстрактные методы:

acceptStep() – отправить игровое состояние на сервер и проинформировать, об окончании хода

getData() – получить игровое состояние от сервера

createPlayer() – запросить создание игрока.

Статические методы:

setInstatnce() – инициализировать конкретную стратегию.

getInstance() – получить инициализированную стратегию.

GameState.java – Перечисления возможных состояний игры.

PlayerData.java – класс данных игрока.

Методы:

getId() – получить *id* игрока

getHand() – получить список плиток находящихся на руках у игрока

getSteps() – получить все возможные ходы, которые может сделать игрок

Tile.java – класс плитки домино. Вес плитки – это порядок чисел на их фронтальной стороне. Таким образом класс имеет всего два поля типа *int* – *w1,w2* (Порядок первого числа веса, порядок второго числа веса (далее просто число плитки))

TilesTree.java – класс реализующий дерево плиток выложенных на столе.

Методы:

getLeafsWithWeight() – получить список листьев соответствующие одному числу плитки.

Вложенная структура *Leaf* – лист дерева.

Корень дерева – первая выложенная на стол плитка (определяется игрой при старте).

Узлы дерева – вложенный класс *Node*. Данный класс использует фабричный метод создания своих экземпляров.

Типы создаваемых узлов перечислены в *enumNode.Kind*. От типа зависит количество дочерних узлов у конкретного узла.

```
public enum Kind{
    /*Неверный*/
    INVALID,
    /*Костяшка*/
    BONE,
    /*Дубль*/
    DOUBLET,
    /*Корневой дубль*/
    ROOT_DOUBLET,
    /*Корневая костяшка*/
    ROOT_BONE,
    /*Перевернутый дубль*/
    FLIPPED_DOUBLET
}
```

GameData.java – структура представляющая игрового состояние. Поля класса:

```
/*Состояние игры*/
public GameState state;

/*Дерево плиток*/
public TilesTree tiles_tree;

/*Количество плиток на базаре*/
public int stock_size;

/*ID игрока, который сейчай ходит*/
public int player_turn_id;

/*Список данных свех игроков*/
public ArrayList<PlayerData> players_data;

/*ID победителя*/
public int winner_id = -1;
```

Game.java – класс реализующий основную логику игры. Является классом одиночкой (*Singleton*), то есть у процесса может быть только один экземпляр этого класса. Все публичные методы статические, и синхронизированы между собой с помощью конструкции *synchronized(Game.class){//...}*.

Методы:

initStock() – инициализировать базар с костяшками.

shuffleStock() – перемешать базар.

giveStepsToPlayer() – выдать костяшки игроку, если у него нет ходов

start() – начать игру. Вызывается, когда количество игроков равно заданному.

Статические методы:

createPlayer() – создать нового игрока. Возвращает идентификатор игрока.

getData() – запросить игровое состояние.

acceptStep() – принять ход игрока – новое состояние игры.

GameRequest.java – абстрактный класс реализующий отправку клиентом запроса, и исполнения этого запроса сервером.

Абстрактные методы:

doRequest() – обработать запрос

Реализации данного абстрактного класса соответствуют методам класса *GameProxy*, а именно:

GameRequestAcceptStep

GameRequestCreatePlayer

GameRequestGetData

GameResponse.java – абстрактный класс – результат, выполнения запроса *GameRequest*. Имеет одно поле типа *boolean* - статус исполнения запроса.

Реализации данного абстрактного класса две:

GameResponseDefault – содержит только статус исполнения запроса.

GameResponseValue<T> - шаблонный класс содержит данные соответствующие параметру шаблона.

1.6.2 Серверная часть

Server.java – класс нити сервера. Он инициализирует сервер и последующие нити класса *ServerThread*, которые непосредственно взаимодействуют с удаленным клиентом.

ServerThread.java – класс нити обработчика запросов клиента. Входной и выходной потоки данных имеют типы *ObjectInputStream* и *ObjectOutputStream*.

Метод *run()*:

```
@Override
public void run() {
    super.run();
    while(true){
        try{
            //чтение данных от клиента
            var request = (GameRequest) in.readObject();
            //исполнения запроса, получения результата
            GameResponse response = request.doRequest();
            //запись результата
            out.reset();
            out.writeObject(response);
        } catch (IOException | ClassNotFoundException) {
            e.printStackTrace();
        }
    }
}
```

ServerGameProxy.java – реализация серверной стратегии получения взаимодействия с игрой.

1.6.3 Клиентская часть

Main.java – основной класс программы. Наследуется от *JFrame*.

Инициализирует окно игры, задает стратегию получения игровых данных, инициализирует глобальные переменные.

Client.java – класс реализующий функция клиента, а именно – получение игровых данных от сервера и их получение.

Метод *sendRequest()*:

```
GameResponse sendRequest(GameRequest request){
    GameResponse response = null;
    try {
        this.out.writeObject(request);
        response = (GameResponse)this.in.readObject();
    }catch (Exception e){
        e.printStackTrace();
    }
    return response;
}
```

ClientGameProxy.java – реализация серверной стратегии получения взаимодействия с игрой.

ClientData.java – глобальные переменные клиента.

Поля:

```
// ID игрока-клиента
publicstaticintplayer_id;
// Вид игрока
publicstaticPlayerViewplayer_view;
// Данные игрока
publicstaticPlayerplayer= newPlayer();

// Ходы игрока
publicstaticArrayList<PlayerData.Step>steps;
// Список фантомных плиток. Нужны для отрисовки полупрозрачных плиток
// на столе, и для совершения хода.
publicstaticArrayList<PseudoTile>pseudo_tiles;

// Локальное игровое состояние
publicstaticGameDatagame_data= null;

// Сцена
publicstaticScenescene= newScene();

// Состояние клиента
publicstaticClientStateclient_state;
```

1.6.4 Рендеринг

Отрисовка сцены пользователя происходит на основе метода *paint* класса *JPanel*. В программе объявлен интерфейс *Drawable*. Классы, которые его реализуют обязаны реализовать метод *draw*, который производит некоторую отрисовку в графическом контексте *Graphics*.

```
public interface Drawable {  
    void draw(Graphics g);  
}
```

Scene.java – класс сцены, основной метод – *paint*.

```
@Override  
public void paint(Graphics g) {  
    super.paint(g);  
    Graphics2D g2d = (Graphics2D)g;  
    g2d.clearRect(0,0,getWidth(), getHeight());  
    if (is_ready){  
        ClientData.player_view.draw(g2d);  
    }  
    else{  
        g2d.drawString("Waiting for connection",10,20);  
    }  
}
```

Методы:

setReady() – перевести сцену в состояние готовности выводить контент.

addStepsFromHandComponents() – добавить к *JPanel*-родителю сцены компоненты отвечающие за обработку событий мыши в области плиток руки пользователя.

removeStepsFromHandComponents() – удалить вышеперечисленные компоненты из сцены.

addStepsFromTreeComponents() – добавить к *JPanel*-родителю сцены компоненты отвечающие за обработку событий мыши в области плиток стола.

removeFromTreeComponents() – удалить вышеперечисленные

компоненты из сцены.

PlayerView.java – абстрактный класс вида пользователя. Наивысший класс в иерархии реализаторов интерфейса *Drawable*. Рисует всю информацию на экране. Имеет абстрактный метод *update*, который обновляет рисующуюся информацию.

Поля класса:

```
protected final TreeModel tree_model;  
protected final PlayerHandModel[] player_hand_models;  
protected final StockModel stock_model;
```

PlayerView2.java – дочерний класс *PlayerView* реализующий метод *update* для двух игроков.

PlayerHandModel.java– *Drawable* класс, реализующий отрисовку плиток в руке игрока (как основного, так и оппонентов).

StockModel.java– *Drawable* класс, реализующий отрисовку базара и количества костяшек в нем.

TreeModel.java – самый тяжеловесный *Drawable* класс, реализующий отрисовку дерева плиток на столе.

Алгоритм рисования примерно такой:

Производится прямой обход в глубину по всему дереву. Позиция корневой плитки – всегда центр. Позиция каждой следующей плитки рассчитывается на основе позиции ее родителя, после чего она добавляется в *ArrayList<TileModel>*. Важно, что обход дерева при отрисовки производится точно также, как и обход дерева при поиске листьев для ходов, т.к. после обработки дерева получается структура данных куча (*heap*). А при поиске листьев для каждого найденного хода сохраняется так называемый *heapindex* родителя данного листа. Таким образом, можно легко отрисовать фантомные плитки, зная *heapindex* родителя, по сути

являющегося индексом в массиве *ArrayList*.

TileModel.java – *Drawable* класс, реализующий отрисовку отдельной плитки. Имеет также дочерний класс *TileSquareModel* который рисует отдельный квадрат плитки, по сути выбирая нужную позицию на спрайте.

Отрисовка спрайтов представлена на рисунок 1. 10.

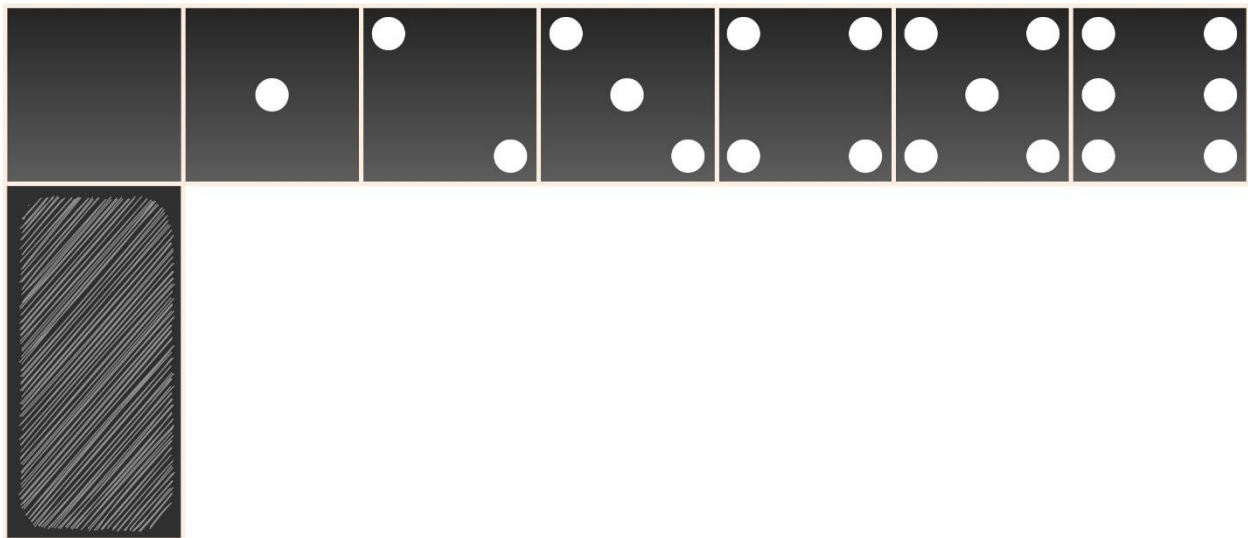


рисунок 1. 10 Отрисовка спрайтов

1.7 Описание способа организации пользовательского интерфейса

Пользовательский интерфейс данного приложения крайне минималистичен. Основное окно программы – *JFrame* библиотеки *Swing*. Оно имеет один дочерний компонент *JPanel* в котором происходит отрисовка всех компонентов игры, а именно, плитки в руках игроков, стол с выложенными плитками и базар, с количеством доступных плиток. Пользователь может выбирать нужные костяшки для хода из руки, и, если ходов выбранной костяшкой больше одного, то выбирает предложенный ход на игральном столе. Обработка событий мыши происходит за счет переопределения класса *JComponent* с имплементацией интерфейсов нужных обработчиков событий.

1.8 Описание результатов работы программы

Первый игрок запустился. Смотреть рисунок 1. 11.

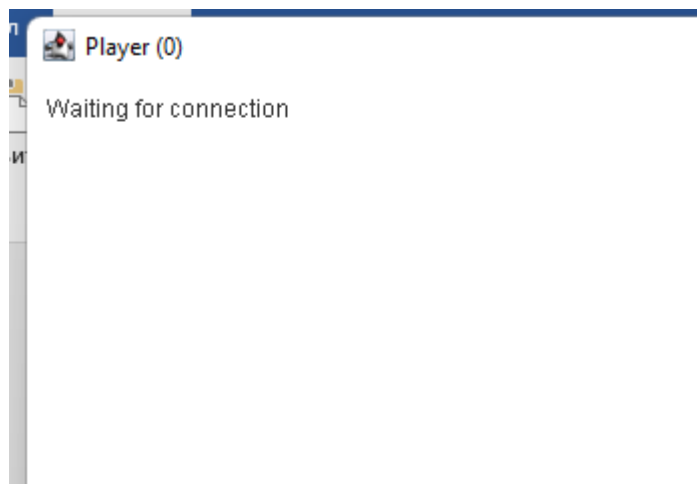


рисунок 1. 11 Ожидания подключения игроков.

Второй игрок запустился. Смотреть рисунок 1. 12.

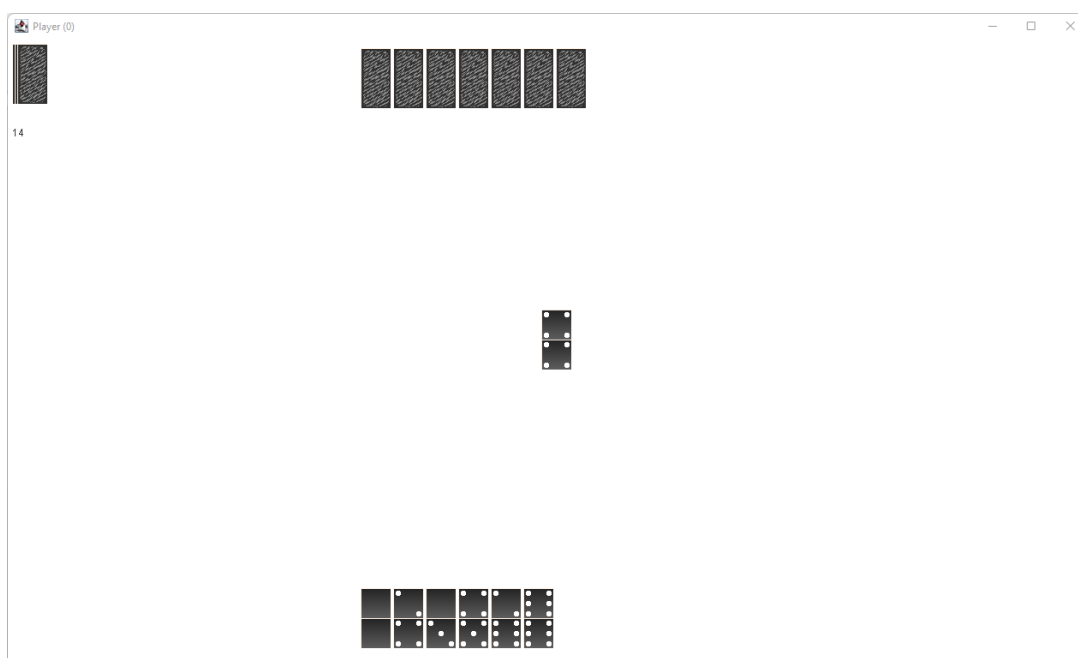


рисунок 1. 12 Все игроки на месте

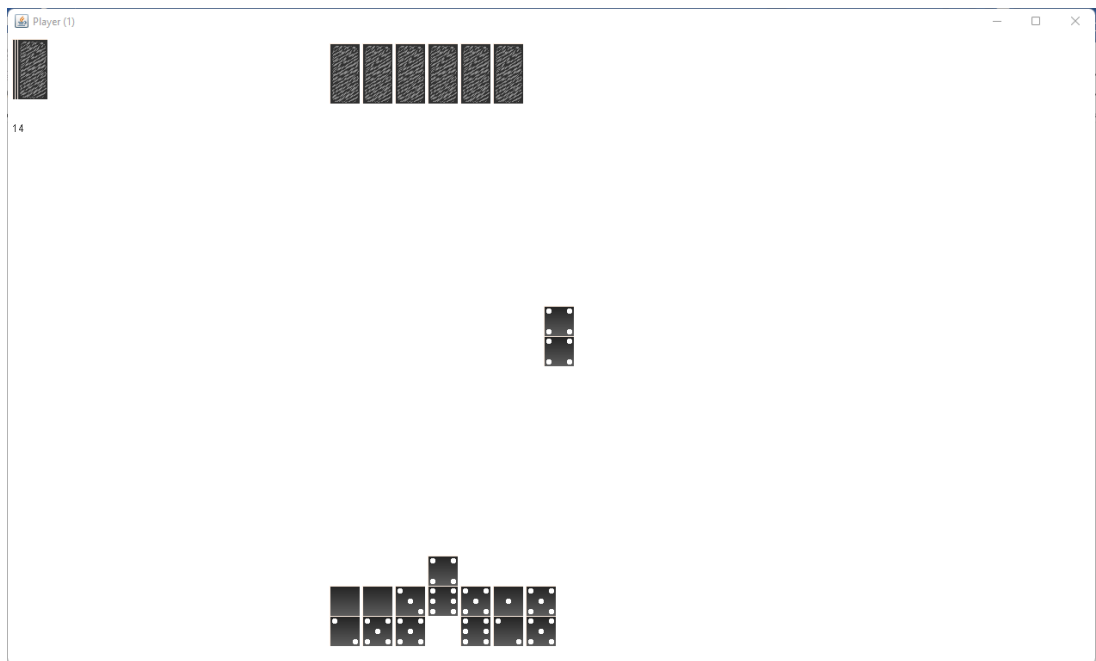


рисунок 1. 13 Игрок готов сделать ход из предложенных вариантов

Ходит второй игрок (“*Player 1*”). На рисунок 1. 14 видим, какой плиткой может походить игроки (такие плитки рисуются выше остальных в руке). Наведя на данную плитку, видим варианты ее расположения на столе. Кликаем на плитку в руке и на вариант со стола. Ход сделан.

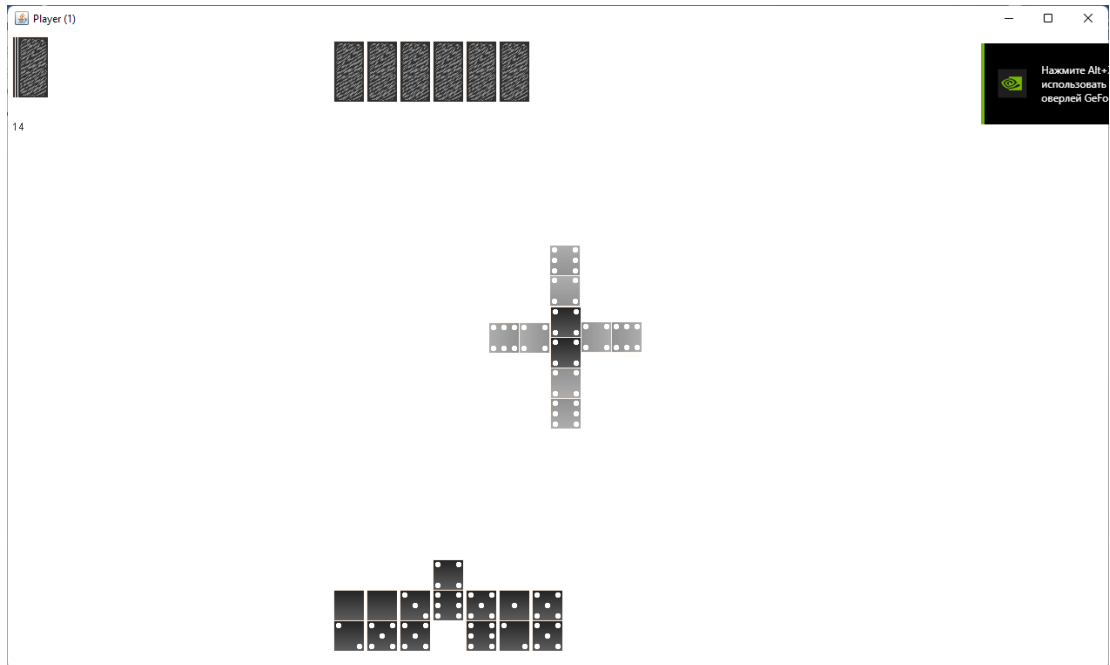


рисунок 1. 14 Выбор из конкретных ходов на поле

Дубль можно положить либо лицевой стороной, либо обратной. Смотреть рисунок 1. 15.

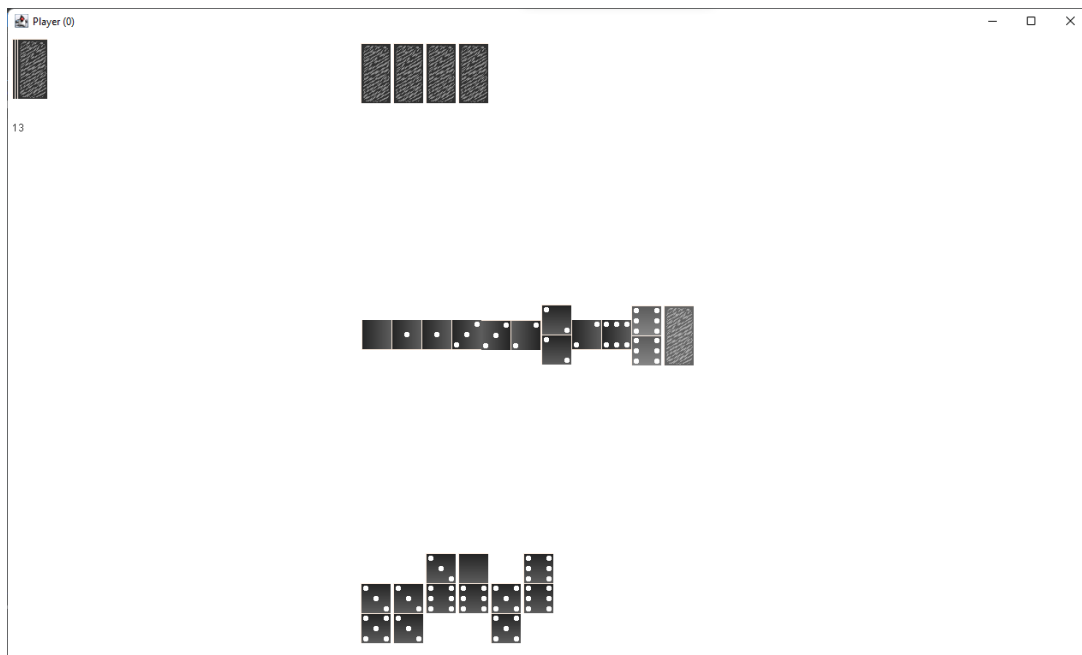


рисунок 1. 15 Варианты расстановки дубля

Ни у одного из игроков нет хода – «рыба». Смотреть рисунок 1. 16.

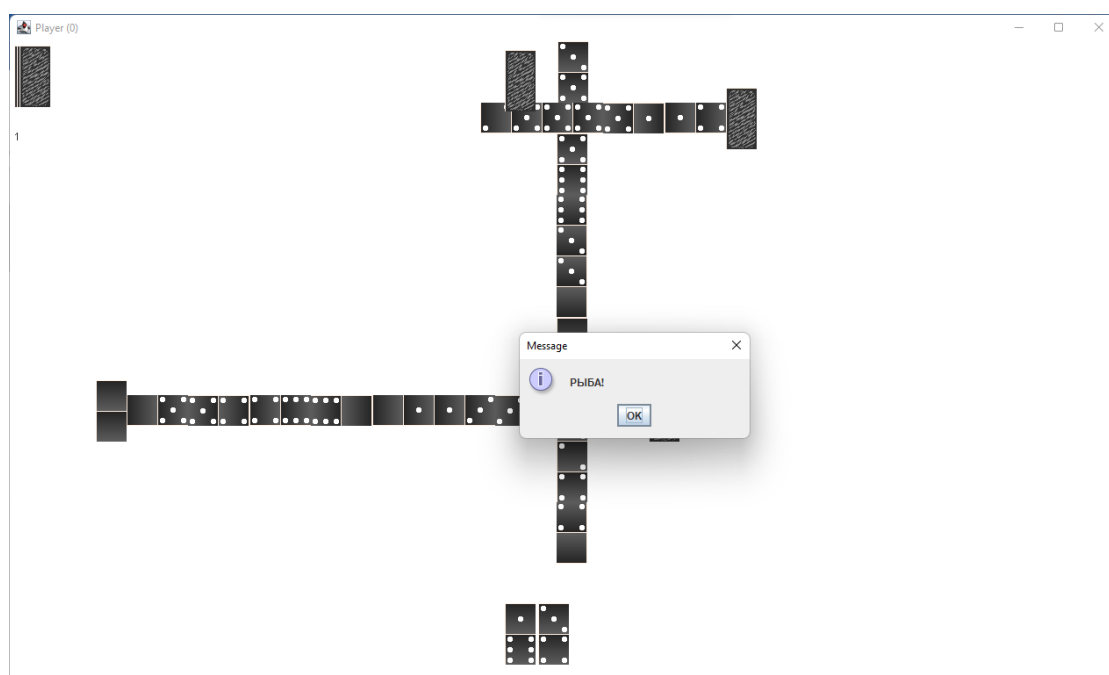


рисунок 1. 16 «Рыба»

“*Player 1*” выложил все плитки из своей руки раньше и победил. Виды от лиц обоих игроков представлены на рисунок 1. 17 и рисунок 1. 18.

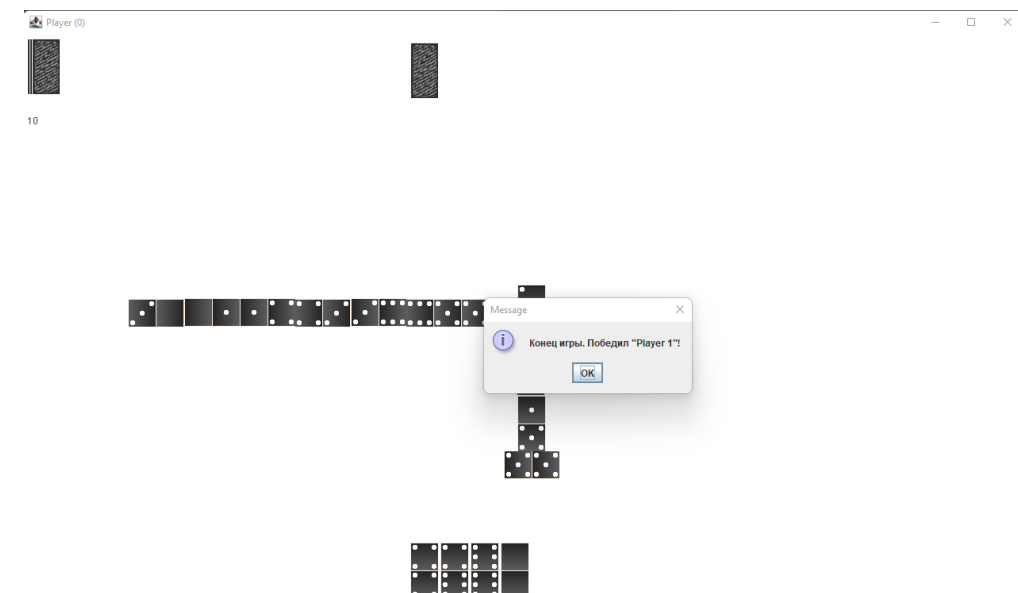


рисунок 1. 17 Победил первый игрок. Вид от лица нулевого.

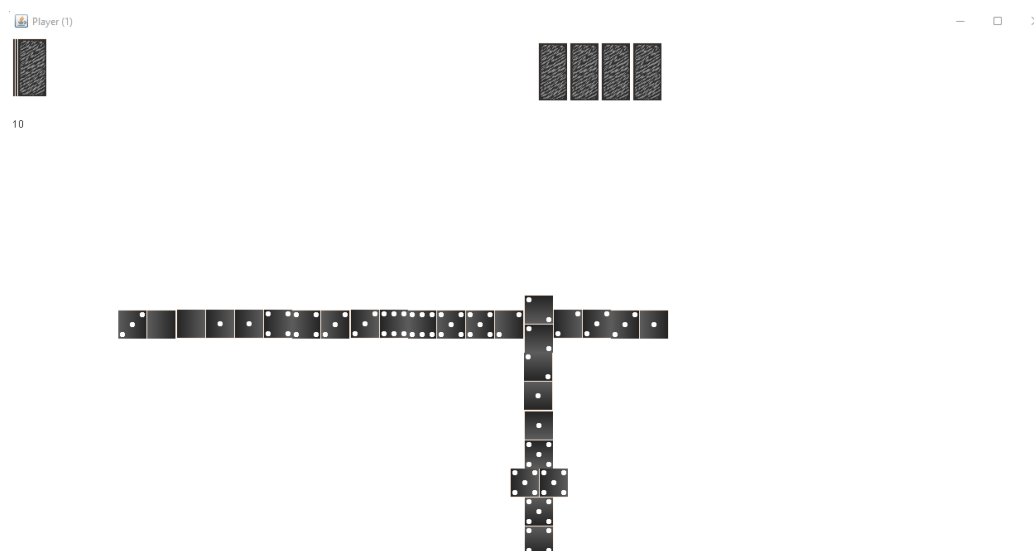


рисунок 1. 18 Победил первый игрок. Вид от лица первого.

2 Охрана труда

2.1 Требования к организации рабочего места с ПК

Во-первых обязательно нужно соблюдать расстояние в 2 метра между столами и не менее 1 метра между мониторами при расстановке рабочих мест в помещении.

Согласно СанПиН 2.2.2/2.4.1340-03, площадь одного рабочего места сотрудника с монитором на базе электроннолучевой трубки (ЭЛТ) должна составлять более 6 квадратных метров, а в культурных и развлекательных учреждениях и с монитором на базе плоских дискретных экранов (жидкокристаллические, плазменные) - 4,5 квадратных метра. При использовании персонального компьютера с монитором перового типа с продолжительностью работы не более четырёх часов в день допускается минимальная площадь 5 квадратных метров на одно рабочее место пользователя старше 18 лет.

Рабочие места с персональным компьютером, необходимые для работы, требующей высокого умственного напряжения, четкой концентрации внимания, предлагается разделять перегородками высотой 1,5-2,0 метра.

Расстояние между монитором и глазами пользователя должно быть 60-

70 сантиметра, но не менее 50 сантиметров с учётом размеров знаков и символов. Расположения клавиатуры должно быть 10-30 сантиметров от края, обращённого к пользователю, или на регулируемой по высоте рабочей поверхности, отделённой от основной столешницы.

Кресло или стул работника должно поддерживать правильную форму тела сотрудника при работе, потому что в противном случае в районе шейно-плечевой области и спины разовьется усталость и возможно развитие искривления позвоночника. Рабочий кресло нужно выбирать, учитывая рост пользователя и характер его работы с компьютером. Рабочий

стул (кресло) должен быть подъемно-поворотным, с возможностью изменения угла наклона спинки и изменением высоты подлокотников. При этом каждая регулировка должна быть независимой, легко осуществляемой и иметь надёжную фиксацию. Поверхности стульев или кресел и других элементов стула желательно должны быть полумягкими, нескользящими, а так же с воздухопроницаемым покрытием.

2.2 Требования к освещению помещений и рабочих мест

Компьютерные залы должны иметь искусственное и естественное освещение. Световой поток из оконного проёма должен падать на рабочее место пользователя с левой стороны и с коэффициентом естественного освещения не ниже 1,2% в зонах со снежным покровом и не ниже 1,5% на остальной территории. Искусственное освещение в помещениях с компьютерами должно быть равномерным.

Освещённость на столах должна быть 300-500 лк. Допускается установка светильников местного освещения для подсветки документов. Местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещённость экрана более 300 лк. Яркость светящихся поверхностей, находящиеся в поле зрения, должна быть не более 200 кд/м². Яркость бликов на экране монитора не должна превышать 40 кд/м². Для их уменьшения стоит повернуть монитор от попадания естественного или местного освещения [7].

Для искусственного освещения помещений с персональными компьютерами следует применять светильники с зеркализированными решётками. Светильники должны располагаться в виде сплошных или прерывистых линий сбоку от рабочих мест параллельно линии зрения пользователя при разном расположении компьютеров. Защитный угол светильников должен быть не менее 40 градусов.

Режим труда и отдыха предполагает определенную длительность непрерывной работы на ПК, а так же перерывы, которые регламентированы

с учётом рабочей смены и характера трудовой деятельности.

Виды трудовой деятельности на ПК разделяются на 3 группы: работа по считыванию информации с экрана, работа по вводу информации, творческая работа на ПК с высоким умственным напряжением или высокой концентрации внимания .

Если в течение рабочей смены пользователь выполняет разные виды работ, то его деятельность относят к той группе работ, на выполнение которой тратится не менее 50% времени рабочей смены.

Регламентированные перерывы, их количество и длительность распределяется в течение рабочей смены в зависимости от категории работы на ПК и продолжительности рабочей смены.

При 8-часовой рабочей смене и работе на ПК регламентированные перерывы следует устанавливать:

- для первой категории работ через 2 часа от начала смены и через 2 часа после обеденного перерыва продолжительностью 15 минут каждый;
- для второй и третьей категории работ — через 1,5- 2,0 часа от начала рабочей смены и через 1,5-2,0 часа после обеденного перерыва продолжительностью 20 минут каждый или продолжительностью 15 минут через каждый час работы.

При 12 часовой рабочей смене регламентированные перерывы должны устанавливаться в первые 8 часов работы аналогично перерывам при 8-часовой рабочей смене, а в течение последних 4 часов работы, независимо от категории и вида работ, каждый час продолжительностью 15 минут. Продолжительность непрерывной работы на ПК без регламентированного перерыва не должна превышать 2 часа. Если работник работает в ночную смену, то регламентированные перерывы увеличиваются на 60 минут, независимо от характера и вида трудовой деятельности.

Чтобы выполнить комплекс упражнений, массаж, гимнастику для пальцев рук, глаз, рекомендуется использовать нерегламентированные перерывы длительностью 1-3 минуты. Они являются наиболее

						Лист
Изм.	Лист	№ докум	Подпись	Дата		40

эффективными. Комплексы упражнений целесообразно менять через 2- 3 недели.

Пользователям ПК, выполняющим работу с высоким уровнем напряженности, показана психологическая разгрузка во время регламентированных перерывов и в конце рабочего дня в специально оборудованных помещениях.

Рекомендуется использовать досуг для пассивного и активного отдыха (занятия на тренажерах, плавание, езда на велосипеде, бег, игра в теннис, футбол, лыжи, аэробика, прогулки по парку, лесу, экскурсии, прослушивание музыки и т.п.). Дважды в год (весной и поздней осенью) рекомендуется проводить курс витаминотерапии в течение месяца. Следует отказаться от курения. Категорически должно быть запрещено курение на рабочих местах и в помещениях с ПК.

						Лист
						41
Изм.	Лист	№ докум	Подпись	Дата		

Заключение

В 21 веке игровая индустрия все больше и больше заполняет жизнь людей, развиваясь во все возможных направлениях. Она привлекает к себе как новых пользователей, так и разработчиков.

Результатом выпускной квалификационной работы является кроссплатформенное игровое приложение с интуитивно понятным пользовательским интерфейсом. Приложение позволяет двум игрокам сыграть одиночную игру по локальной вычислительной сети и определить победителя. Данный проект удовлетворяет большинству поставленных целей.

За время работы с данным проектом были изучены новые способы и механизмы программирования и проектирования систем. Значительно улучшили навык разработки программ на языке *Java* в среде разработки *Unity*. Ближе познакомились с средой разработки *IntelliJ IDEA*.

Список используемых источников

- 1 Берд, Барри *Java* / Барри Берд. - М.: Диалектика / Вильямс, 2013. - 521 с.
- 2 Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубаков – СПб: НИУ ИТМО, 2013. – 248 с.
- 3 *Introduction to Algorithms, Third Edition*, 2009 *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*. – с 151-169.
- 4 Паттерны объектно-ориентированного программирования, Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.— 448 с

Приложение А- Листинг программы

(Обязательное)

Кроссплатформенное клиент-серверное приложение

Листинг программы

Server.java

```
public class Server extends Thread{
    static final int PORT = 2048;

    public void run() {
        ServerSocket serverSocket = null;
        Socket socket = null;

        try {
            serverSocket = new ServerSocket(PORT);
        } catch (IOException e) {
            e.printStackTrace();
        }

        while (true) {
            try {
                socket = serverSocket.accept();
                new ServerThread(socket).start();
            } catch (IOException e) {
                System.out.println("I/O error: " + e);
            }
        }
    }
}
```

ServerThread.java

```
public class ServerThread extends Thread{
    protected Socket socket;
    ObjectInputStream in;
    ObjectOutputStream out;

    public ServerThread(Socket socket) throws IOException {
        this.socket = socket;
        this.out = new ObjectOutputStream(socket.getOutputStream());
        this.in = new ObjectInputStream(socket.getInputStream());
    }

    @Override
    public void run() {
        super.run();
        while (true){
            try {
                var request = (GameRequest)in.readObject();
                GameResponse response = request.doRequest();
                out.reset();
                out.writeObject(response);
            }
        }
    }
}
```

```

        } catch (IOException | ClassNotFoundException e) {}
    }
}

```

TilesTree.java

```

public class TilesTree implements Serializable {
    public Node root;

    public static class Node implements Serializable{
        public Node[] children = null;
        public int[] weights = null;
        private Node(int w1,int w2,int children_count){
            if (w1 != w2){
                this.weights = new int[2];
                this.weights[0] = w1;
                this.weights[1] = w2;
            }else{
                this.weights = new int[1];
                this.weights[0] = w2;
            }

            children = children_count != 0 ? new Node[children_count] :
            null;
        }

        public enum Kind{
            /*Неверный*/
            INVALID,
            /*Костяшка*/
            BONE,
            /*Дубль*/
            DOUBLET,
            /*Корневойдубль*/
            ROOT_DOUBLET,
            /*Корневаякостяшка*/
            ROOT_BONE,
            /*Перевернутыйдубль*/
            FLIPPED_DOUBLET
        }

        public Kind getKind(){
            if (this.children == null){
                return Kind.FLIPPED_DOUBLET;
            }

            if (this.weights.length == 1){
                if (this.children.length == 4){
                    return Kind.ROOT_DOUBLET;
                }else if (this.children.length == 3){
                    return Kind.DOUBLET;
                }
            }
        }
    }
}

```

```

        }else if (this.weights.length == 2){
if (this.children.length == 2){
return Kind.ROOT_BONE;
        }else if (this.children.length == 1){
return Kind.BONE;
        }
    }
return Kind.INVALID;
}

public static void pushNode(PseudoTile pt){
    pt.leaf.parent.children[pt.leaf.child_index] =
createNodeFromPseudoTile(pt);
}

public static Node createNodeFromPseudoTile(PseudoTile pt){
    Node node;
if (!pt.is_flipped){
int w1 = pt.tile.w1, w2 = pt.tile.w2;
if (pt.leaf.parent.weights[0] == pt.tile.w1){
int tmp = w1;

        w1 = w2;
        w2 = tmp;
    }
    node = w1 == w2 ?
createDoublet(w1) :
createBone(w1, w2);
    }else{
        node = createFlippedDoublet(pt.tile.w1);
    }
return node;
}

/**
    * BoneNode has 2 weights, and 1 children
    * @param w1 is the weight by which node is connected
with child.
    * @return TileTree.Node
    */
public static Node createBone(int w1,int w2){
return new Node(w1,w2,1);
}

/**
    * RootDoublet has 1 weight, by which it's connected
with its 3 children
    * @return TileTree.Node
    */
public static Node createDoublet(int w){
return new Node(w,w,3);
}

```

```

/**
    * RootDoublet has 1 weight, by which it's connected
with its 4 children
    * @return TileTree.Node
    */
public static Node createRootDoublet(int w) {
return new Node(w,w,4);
}

/**
    * RootBone has 2 weights, and 2 children
    * @param w1 - is the weight by which node is connected
with children.
    *
    * @return TileTree.Node
    */
public static Node createRootBone(int w1,int w2){
return new Node(w1,w2,2);
}

/**
    * FlippedDoublet has 1 weight, and no children
    * @return TileTree.Node
    */
public static Node createFlippedDoublet(int w) {
return new Node(w,w,0);
}

}

public static class Leaf{
public Node parent;
public int parent_heap_index;
public int child_index;
    Leaf(Node p,int heap_idx,int child_idx){
this.parent = p;
this.child_index = child_idx;
this.parent_heap_index = heap_idx;
    }

}

private Pair<LinkedList<Leaf>,Integer>
getLeafsWithWeightFromNode(int w, Node n, int heap_index){
var list = new LinkedList<Leaf>();
if (n.weights[0] == w && n.children != null){
for (int i = 0; i < n.children.length; i++){
if (n.children[i] == null){
list.add(new Leaf(n,heap_index,i));
}
}
}
heap_index++;
if (n.children != null){

```

```

for (int i = 0; i < n.children.length; i++){
if (n.children[i] != null){
var result =
getLeafsWithWeightFromNode(w,n.children[i],heap_index);
list.addAll(result.first);
heap_index = result.second;
}
}
}
return new Pair<>(list,heap_index);
}
public LinkedList<Leaf> getLeafsWithWeight(int w){
return getLeafsWithWeightFromNode(w,root,0).first;
}
public TilesTree(Node root){
this.root = root;
}
TilesTree(){
this.root = null;
}
}

```

PlayerData.java

```

public class PlayerData implements Serializable {
private int id;
private LinkedList<Tile>hand;
PlayerData(int id, LinkedList<Tile> hand){
this.id = id;
this.hand = hand;
}

public int getId(){return this.id;}

public LinkedList<Tile> getHand(){return hand;}

public int getTilesCountOnHand(){
return hand.size();
}

public static class Step{
public int tile_idx;
public LinkedList<TilesTree.Leaf>leafs;
}

public ArrayList<Step> getSteps(TilesTree tree){
ArrayList<Step> steps = new ArrayList<>();
for (var i = 0; i <this.hand.size(); i++){
var step = new Step();
var tile = this.hand.get(i);

```

```

        step.tile_idx = i;
        step.leafs = tree.getLeafsWithWeight(tile.w1);
if (tile.w1 != tile.w2){

step.leafs.addAll(tree.getLeafsWithWeight(tile.w2));
        }
if (step.leafs.size() > 0){
        steps.add(step);
        }
    }
return steps;
    }
}

```

Game.java

```
package Main.Server;
```

```
import Main.Server.Exceptions.GameTooMuchPlayersException;
import Main.Server.Exceptions.GameExceptionInvalidData;
```

```
import java.util.ArrayList;
import java.util.LinkedList;
```

```
public class Game {
    static class GameSingleton{
public static Game instance = null;
    }

```

```

synchronized private static Game getInstance(){
return GameSingleton.instance = GameSingleton.instance == null ?
new Game() : GameSingleton.instance;
    }

```

```

static final int TILES_COUNT_IN_PLAYER_HAND = 7;
static final int MAX_PLAYERS_COUNT = 2;
static final int MAX_WEIGHT = 6;
static final int TILES_COUNT =
((MAX_WEIGHT+1)*(MAX_WEIGHT+2))/2;

```

```

private GameData data = new GameData();
private final Tile[] stock = new Tile[TILES_COUNT];

```

```

private void shuffleStock(){
int i1,i2;
    Tile temp;
for (int i = 0; i <TILES_COUNT; i++){
        i1 = (int) (Math.random()*(100))%(TILES_COUNT-1);
        i2 = (int) (Math.random()*(100))%(TILES_COUNT-1);
        temp = stock[i1];
stock[i1] = stock[i2];
stock[i2] = temp;
}

```

```

    }
}

private void initStock(){
int i = 0;
for (int w2 = 0; w2 <MAX_WEIGHT+1; w2++){
for (int w1 = 0; w1 < w2+1; w1++){
stock[i] = Tile.createBone(w1,w2);
i++;
}
}
shuffleStock();
}

private Game(){
this.data.players_data = new ArrayList<>();
this.data.player_turn_id = -1;
this.data.tiles_tree = new TilesTree();
this.data.stock_size = TILES_COUNT;
this.data.state = GameState.STARTING;
initStock();
}

private LinkedList<Tile> generateHand(){
var hand = new LinkedList<Tile>();
for (int i = 0; i <TILES_COUNT_IN_PLAYER_HAND; i++){
var t = stock[data.stock_size-1];
hand.push(t);
data.stock_size--;
}
return hand;
}

/**
 * @return 0 - if t0 better for start, otherwise - 1
 * */
private static int compareByStartPriority(Tile t0,Tile t1){
//t0 is [0/0] doublet
if (t0.w1 == t0.w2 && t0.w1 == 0){
return 1;
}
//t1 is [0/0] doublet
if (t1.w1 == t1.w2 && t1.w1 == 0){
return 0;
}
//t0 is doublet, t1 is not
if (t0.w1 == t0.w2 && t1.w1 != t1.w2){
return 0;
}
//t1 is doublet, t1 is not
if (t0.w1 != t0.w2 && t1.w1 == t1.w2){
return 1;
}
}

```

```

//t0 and t1 are both doublets
if (t0.w1 == t0.w2 && t1.w1 == t1.w2){
return t0.w1 < t1.w1 ? 0 : 1;
}
//t0 and t1 are both not doublets
if (t0.w2 == t1.w2){
return t0.w1 < t1.w1 ? 0 : 1;
}
return t0.w2 < t1.w2 ? 0 : 1;
}

private StartTile getStartTile(){
    StartTile start_tile = new StartTile(-1,-1);
    Tile best_tile = Tile.createDoublet(0);
    for (int player_idx = 0; player_idx
<data.players_data.size();player_idx++){
var player_hand = data.players_data.get(player_idx).getHand();
for (int tile_idx = 0; tile_idx <
player_hand.size();tile_idx++){
var tile = player_hand.get(tile_idx);
if (compareByStartPriority(best_tile,tile) == 1){
        best_tile = tile;
        start_tile = new
StartTile(tile_idx,player_idx);
    }
}
}
return start_tile;
}

private int getNextPlayerTurn(){
if (this.data.player_turn_id <data.players_data.size() - 1){
return this.data.player_turn_id+1;
}
return 0;
}

private static class StartTile{
public int tile_idx;
public int player_index;
    StartTile(int tile_idx,int player_index){
this.tile_idx = tile_idx;
this.player_index = player_index;
    }
}

private boolean tileCanGiveSteps(Tile tile){
if (this.data.tiles_tree.getLeafsWithWeight(tile.w1).size() !=
0) {
return true;
}
if (tile.w1 == tile.w2){
return false;
}
}

```

```

    }
    return this.data.tiles_tree.getLeafsWithWeight(tile.w1).size()
    != 0;
    }

    private void giveStepsToPlayer(){
    var player = data.players_data.get(data.player_turn_id);
    var steps = player.getSteps(data.tiles_tree);
    if (steps.size() == 0){
    var hand = player.getHand();
    while (data.stock_size != 1){
    var tile = stock[data.stock_size-1];
    data.stock_size--;
    hand.add(tile);
    if (tileCanGiveSteps(tile)){
    break;
    }
    }
    }

    private void start(){
    var start_tile = getStartTile();
    var player_hand =
    data.players_data.get(start_tile.player_index).getHand();
    var tile = player_hand.get(start_tile.tile_idx);
    data.tiles_tree.root = tile.w1 == tile.w2 ?
    TilesTree.Node.createRootDoublet(tile.w1) :
    TilesTree.Node.createRootBone(tile.w1, tile.w2);
    player_hand.remove(start_tile.tile_idx);
    data.player_turn_id = start_tile.player_index;
    data.player_turn_id = getNextPlayerTurn();
    giveStepsToPlayer();
    data.state = GameState.CONTINUE;
    }

    private boolean isCorrectData(GameData data){
    return true;
    }

    private boolean isGameLocked(){
    for (var player : data.players_data){
    if (player.getSteps(data.tiles_tree).size() != 0){
    return false;
    }
    }
    return true;
    }

    private boolean isGameOver(){
    for (var player : data.players_data){
    if (player.getHand().size() == 0){
    data.winner_id = player.getId();

```

```

return true;
    }
}
return false;
}

public static int createPlayer() throws
GameTooMuchPlayersException {
    synchronized (Game.class){
        Game game = getInstance();
        if (game.data.players_data.size() == MAX_PLAYERS_COUNT){
            throw new GameTooMuchPlayersException();
        }
        int id = game.data.players_data.size();
        game.data.players_data.add(new
        PlayerData(id,game.generateHand()));
        if (game.data.players_data.size() == MAX_PLAYERS_COUNT){
            game.start();
        }
        return id;
    }
}

public static GameData getData(){
    synchronized (Game.class){
        return getInstance().data;
    }
}

public static void acceptStep(GameData data) throws
GameExceptionInvalidData {
    Game.setData(data);
    synchronized (Game.class){
        var game = Game.getInstance();
        game.data.player_turn_id = game.getNextPlayerTurn();
        game.giveStepsToPlayer();
        if (game.isGameOver()){
            game.data.state = GameState.IS_OVER;
        }
        if (game.isGameLocked()){
            game.data.state = GameState.LOCKED;
        }
    }
}

public static void setData(GameData data) throws
GameExceptionInvalidData {
    synchronized (Game.class) {
        Game game = getInstance();
        if (game.isCorrectData(data)) {
            game.data = data;
        }
        return;
    }
}

```

```

throw new GameExceptionInvalidData();
    }
}

PlayerHandModel.java

package Main.Rendering;

import Main.Server.PlayerData;

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

public class PlayerHandModel implements Drawable{

    public enum Orientation{
        VERTICAL,
        HORIZONTAL,
    }

    public boolean isMainPlayersHandModel(){
        return is_main_player;
    }

    public StepTileModel[] getStepTiles(){
        return this.step_tiles;
    }

    public void update(PlayerData player, ArrayList<PlayerData.Step>
        steps){
        var hand = player.getHand();
        int steps_count = 0;
        this.step_tiles = null;
        if (steps != null){
            this.step_tiles = new StepTileModel[steps_count = steps.size()];
        }
        this.common_tiles = new TileModel[hand.size()-steps_count];

        int si = 0;
        int ci = 0;
        int i = 0;
        int x,y;
        var orientation = TileModel.Orientation.VERTICAL;
        try{
            for (var it: hand){
                x = position.x + TileModel.DEFAULT_SPACE*i;
                y = position.y;

                if (is_main_player && steps != null){
                    boolean is_step_tile = false;
                    for (var step: steps){
                        var tile = hand.get(step.tile_idx);
                        if (it == tile){

```

```

        is_step_tile = true;
        y = position.y -
TileModel.DEFAULT_SIZE;
this.step_tiles[si] = new StepTileModel(

it,x,y,TileModel.DEFAULT_SIZE,
                                orientation,
                                i
                                );
        si++;
        i++;

break;
    }
}

if (is_step_tile)
continue;
    }
this.common_tiles[ci] = new
TileModel(it,x,y,TileModel.DEFAULT_SIZE,orientation);
if (!is_main_player){
this.common_tiles[ci].hide();
    }
    ci++;
    i++;
    }
} catch (Exception ex){
    ex.printStackTrace();
}
}

@Override
public void draw(Graphics g) {
for (var it : this.common_tiles){
    it.draw(g);
}

if (this.step_tiles == null)
return;
for (var it : this.step_tiles){
    it.draw(g);
}
}

PlayerHandModel(boolean is_main_player, Orientation
orientation){
this.is_main_player = is_main_player;
this.orientation = orientation;
}

private TileModel[] common_tiles;
private StepTileModel[] step_tiles;
private final Orientation orientation;
private final boolean is_main_player;
public Point position;

```

```

}

TileFromTreeComponent.java

package Main.Client;

import java.awt.*;
import java.awt.event.MouseEvent;

public class TileFromTreeComponent extends TileComponent{
    public TileFromTreeComponent(int step_index, int index_in_list,
    Rectangle bounds) {
        super(step_index, index_in_list, bounds);
    }

    @Override
    public void mouseClicked(MouseEvent e) {

        ClientData.player.makeStep(ClientData.pseudo_tiles.get(index_in_
        list), index_in_hand);
        ClientData.scene.repaint();
    }

    @Override
    public void mousePressed(MouseEvent e) {}

    @Override
    public void mouseReleased(MouseEvent e) {}

    @Override
    public void mouseEntered(MouseEvent e) {}

    @Override
    public void mouseExited(MouseEvent e) {}
}

```

```

TileFromHand.java

package Main.Client;

import Main.Rendering.TileModel;
import Main.Server.PseudoTile;

import java.awt.*;
import java.awt.event.MouseEvent;

public class TileFromHandComponent extends TileComponent {

    public TileFromHandComponent(int step_index, int index_in_list,
    Rectangle bounds) {
        super(step_index, index_in_list, bounds);
    }
}

```

```

        @Override
public void mouseClicked(MouseEvent e) {
    if (ClientData.client_state !=
        ClientState.CHOOSSES_TILE_FROM_HAND) {
        return;
    }
    var tree_model = ClientData.player_view.getTreeModel();
        tree_model.removePseudoTiles();
    if (ClientData.pseudo_tiles.size() == 1){

        ClientData.player.makeStep(ClientData.pseudo_tiles.get(0),
            index_in_hand);
            ClientData.scene.repaint();
        return;
    }
    var tree = ClientData.game_data.tiles_tree;

    tree_model.updatePseudoTiles(tree, ClientData.pseudo_tiles, TileMo
        del.Appearance.TRANSPARENT75);
        ClientData.client_state =
        ClientState.CHOOSSES_STEP_VARIANT;
        ClientData.scene.repaint();
    }

    @Override
public void mousePressed(MouseEvent e) {

    }

    @Override
public void mouseReleased(MouseEvent e) {

    }

    @Override
public void mouseEntered(MouseEvent e) {
    if (ClientData.client_state !=
        ClientState.CHOOSSES_TILE_FROM_HAND) {
        return;
    }
    var step = ClientData.steps.get(index_in_list);
    var hand =
        ClientData.game_data.players_data.get(ClientData.player_id).getH
            and();
        ClientData.pseudo_tiles =
        PseudoTile.createPseudoTilesFromStep(step, index_in_hand, hand);

    var tree = ClientData.game_data.tiles_tree;
    var tree_model = ClientData.player_view.getTreeModel();
        tree_model.updatePseudoTiles(
            tree,
            ClientData.pseudo_tiles,
            TileModel.Appearance.TRANSPARENT50
        );
    };

```

```

ClientData.scene.addStepsFromTreeComponents(tree_model.getPseudo
Tiles());
    ClientData.scene.repaint();
}

@Override
public void mouseExited(MouseEvent e) {
    if (ClientData.client_state !=
ClientState.CHOOSSES_TILE_FROM_HAND) {
        return;
    }
    ClientData.pseudo_tiles = null;

ClientData.player_view.getTreeModel().removePseudoTiles();
var tree_model = ClientData.player_view.getTreeModel();
    tree_model.removePseudoTiles();
    ClientData.scene.removeStepsFromTreeComponents();
    ClientData.scene.repaint();
}
}

```

Player.java

```

package Main.Client;

import Main.Server.*;
import Main.Server.Exceptions.GameExceptionInvalidData;

import javax.swing.*;

public class Player {
    interface Notification{
        boolean check(GameData data);
    }

    private static class ClientNotifier extends Thread{
        Notification notification;
        @Override
        public void run() {
            super.run();
            GameData game_data;
            while (true){
                game_data = GameProxy.getInstance().getData();
                if (notification.check(game_data)){
                    break;
                }
            }
            try {
                Thread.sleep(20);
            } catch (InterruptedException e) {}
        }
    }
}

```

```

        ClientData.player.acceptTurn(game_data);
    }
    ClientNotifier(Notification notification){
this.notification = notification;
    }
}

public void makeStep(PseudoTile pt, int tile_index_in_hand){
    TilesTree.Node.pushNode(pt);
    var hand =
ClientData.game_data.players_data.get(ClientData.player_id).getH
and();

    ClientData.player_view.getTreeModel().removePseudoTiles();
    hand.remove(tile_index_in_hand);
    ClientData.steps = null;
    ClientData.scene.removeStepsFromHandComponents();
    ClientData.client_state = ClientState.IDLE;
    ClientData.player_view.update(ClientData.steps);
    try {

        GameProxy.getInstance().acceptStep(ClientData.game_data);
    } catch (GameExceptionInvalidData e) {
    }
    waitForTurn();
}

public void acceptTurn(GameData game_data){
    String message = null;
    if (game_data.state == GameState.IS_OVER){
        message = "Конечигры. Победил \"Player
\"+game_data.winner_id+\"\"!\"";
    }
    if (game_data.state == GameState.LOCKED){
        message = "РЫБА!";
    }
    if (game_data.state == GameState.IS_OVER ||
        game_data.state == GameState.LOCKED){
        JOptionPane.showMessageDialog(null, message);
    }
    try {

        GameProxy.getInstance().acceptStep(ClientData.game_data);
    } catch (GameExceptionInvalidData e) {
    }
    return;
}

    ClientData.client_state = game_data.player_turn_id ==
ClientData.player_id ?
        ClientState.CHOOSSES_TILE_FROM_HAND :
        ClientState.IDLE;
    if (ClientData.client_state ==
ClientState.CHOOSSES_TILE_FROM_HAND){

```

```

        ClientData.steps =
game_data.players_data.get(ClientData.player_id).getSteps(game_d
ata.tiles_tree);
    }
    ClientData.game_data = game_data;
    ClientData.player_view.update(ClientData.steps);
    ClientData.scene.setReady();
    ClientData.scene.repaint(1);
if (ClientData.client_state != ClientState.IDLE){
if (ClientData.steps.size() == 0){
try {

GameProxy.getInstance().acceptStep(ClientData.game_data);
        } catch (GameExceptionInvalidData e) {
        }
        waitForTurn();
    }else{
        ClientData.scene.addStepsFromHandComponents(

ClientData.player_view.getMainPlayerHandModel().getStepTiles()
        );
    }
    }else{
        waitForTurn();
    }
}

public void waitForTurn(){
new ClientNotifier(data -> data.player_turn_id ==
ClientData.player_id).start();
}

public void waitForStart(){
new ClientNotifier(data -> data.state ==
GameState.CONTINUE).start();
}
}

Main.java

package Main.Client;

import Main.Rendering.PlayerView2;
import Main.Server.Exceptions.GameTooMuchPlayersException;
import Main.Server.GameProxy;
import Main.Server.Server;
import Main.Server.ServerGameProxy;

import javax.swing.*;
import java.io.IOException;

public class Main extends JFrame{
public static final int WIDTH = 1280;
public static final int HEIGHT = 720;

```

```

        Main() throws GameToMuchPlayersException {
super();
        GameProxy proxy;
try {
            proxy = new ClientGameProxy(new Client());
        }catch (IOException e){
new Server().start();
            proxy = new ServerGameProxy();
        }
        GameProxy.setInstance(proxy);
        ClientData.player_id =
GameProxy.getInstance().createPlayer();
        ClientData.player_view = new PlayerView2(WIDTH,HEIGHT);
super.setTitle("Player (" +ClientData.player_id+"");
    }

public static void main(String[] args){
try {
var main = new Main();
        main.add(ClientData.scene);
        main.setSize(WIDTH, HEIGHT+50);
        main.setVisible(true);
        main.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ClientData.player.waitForStart();
    }catch (GameToMuchPlayersException e){
        JOptionPane.showMessageDialog(null, "Превышено
максимальное количество игроков");
    }
}
}

```

Приложение Б.- UML-диаграммы

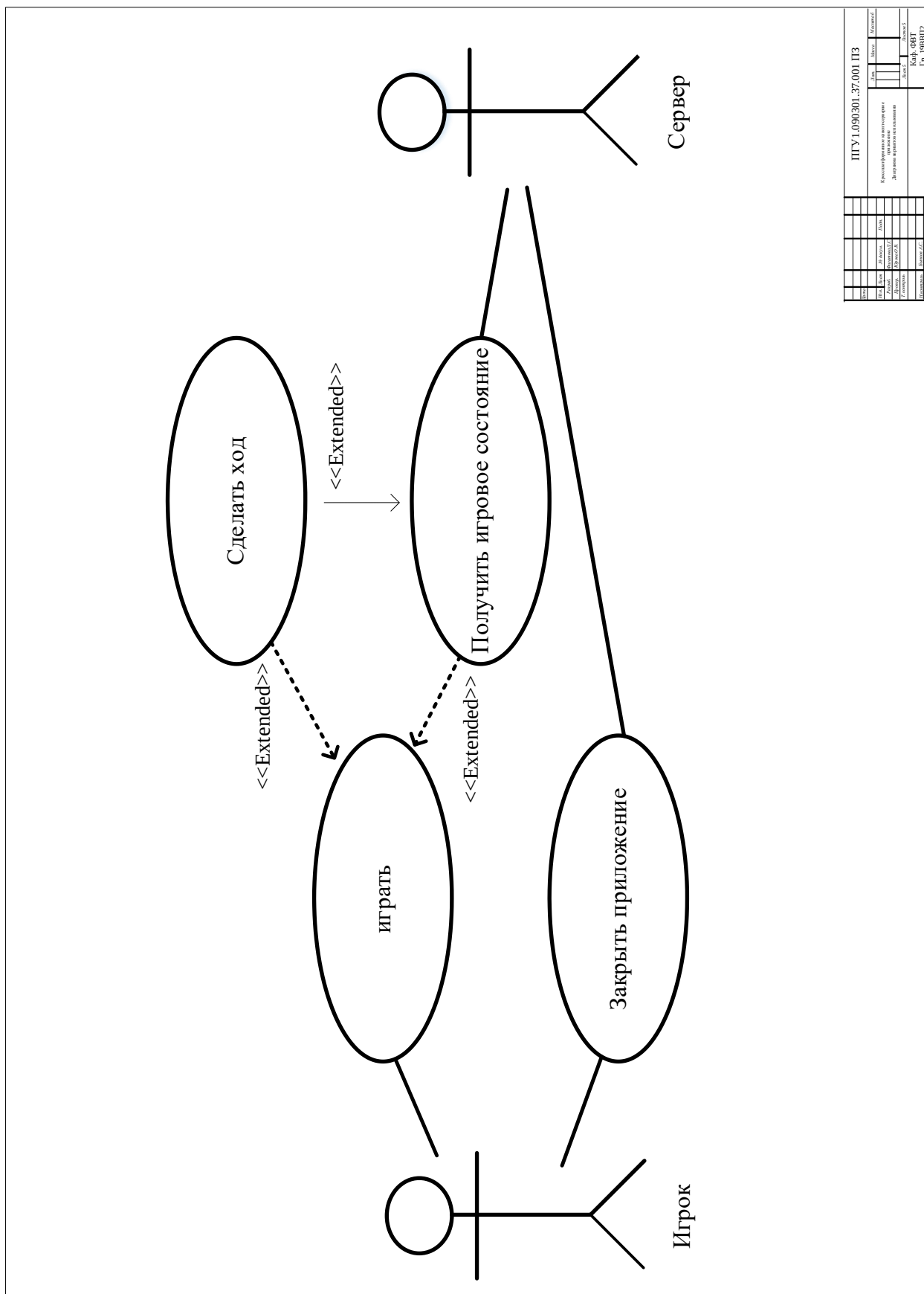
(Обязательное)

Кроссплатформенное клиент-серверное приложение

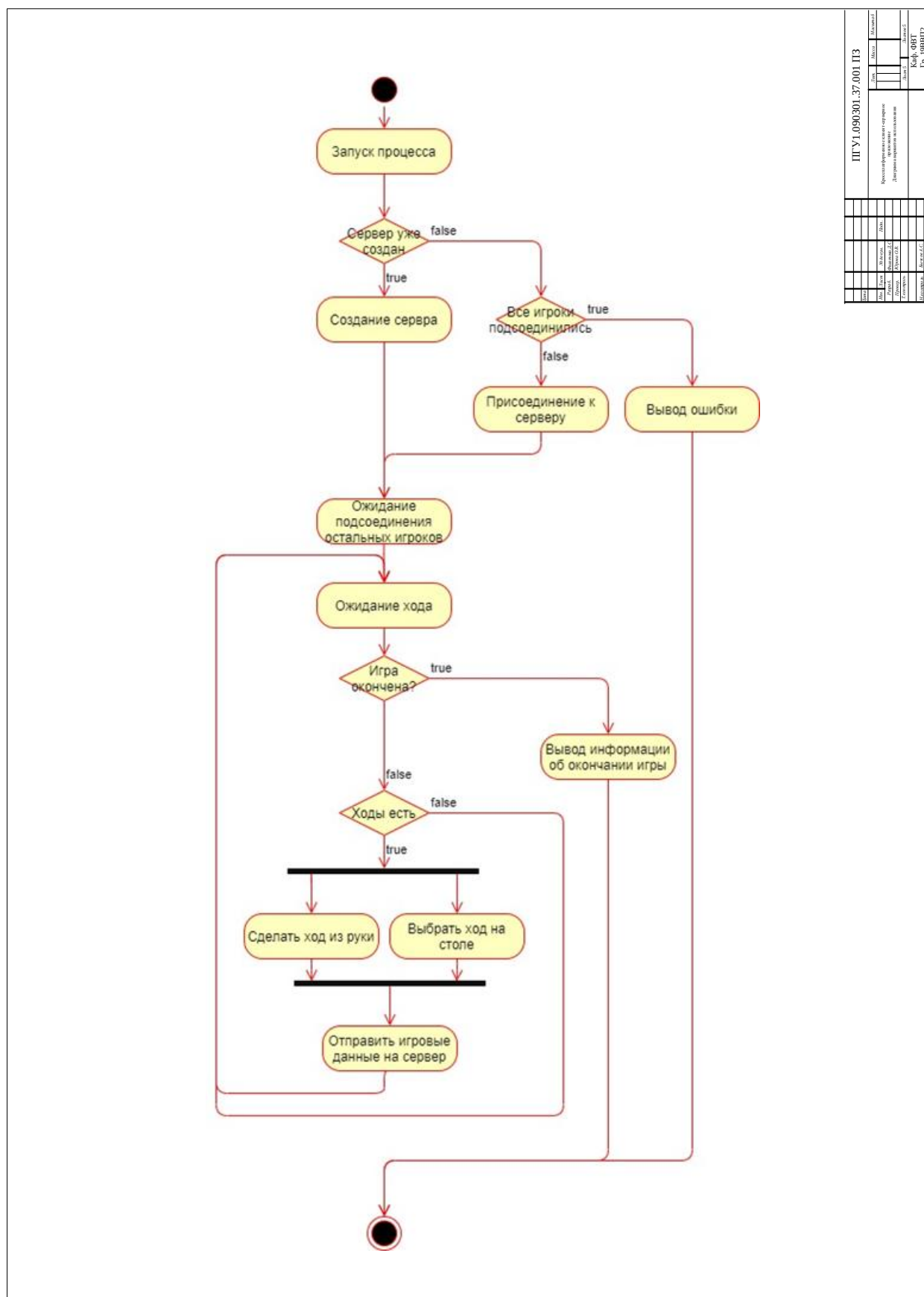
UML-диаграммы

					<i>ПГУ1.090301.37.001 ПЗ</i>	Лист
						62
Изм.	Лист	№ докум	Подпись	Дата		

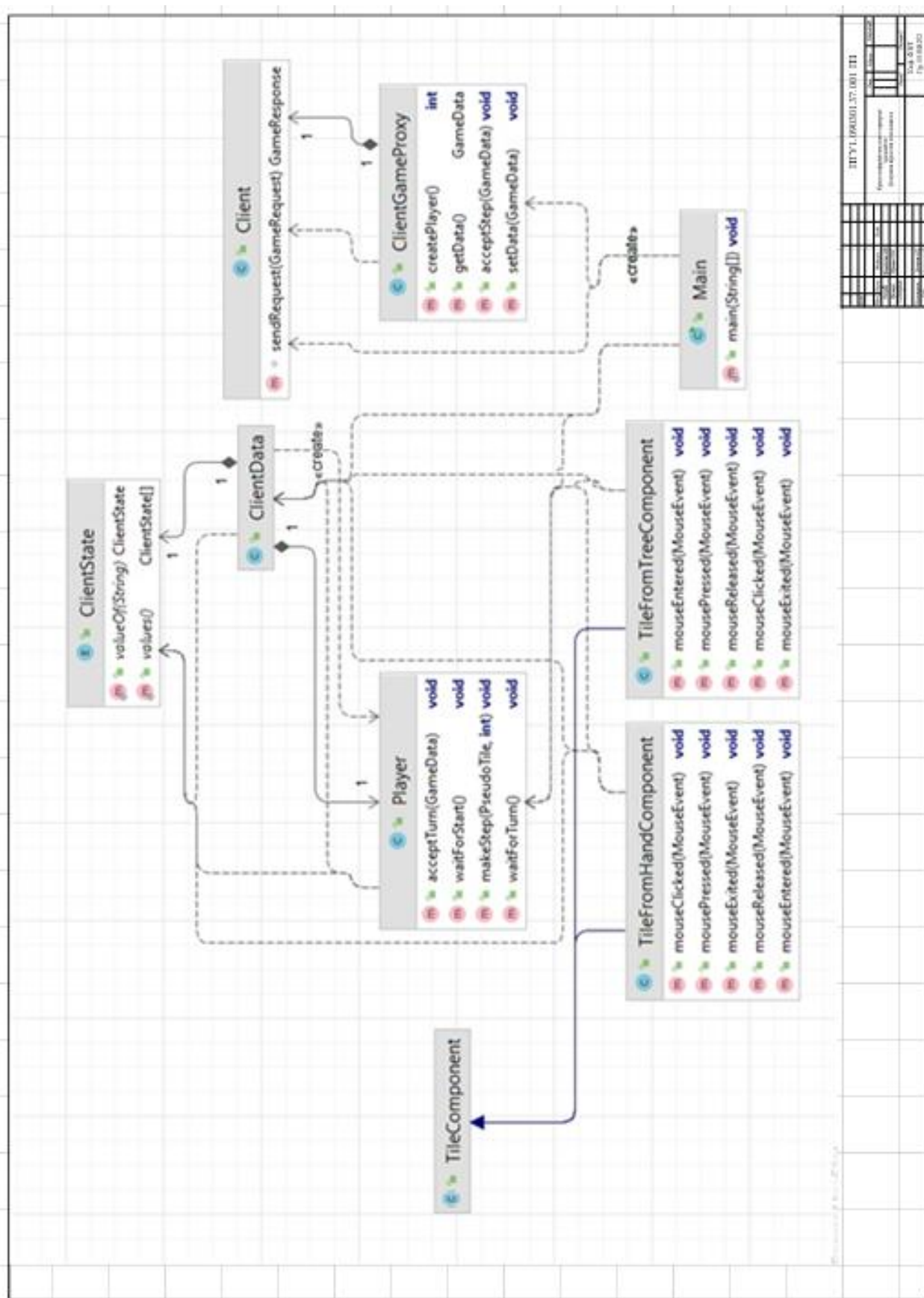
Приложение Б.1. –UML-диаграмма вариантов использования



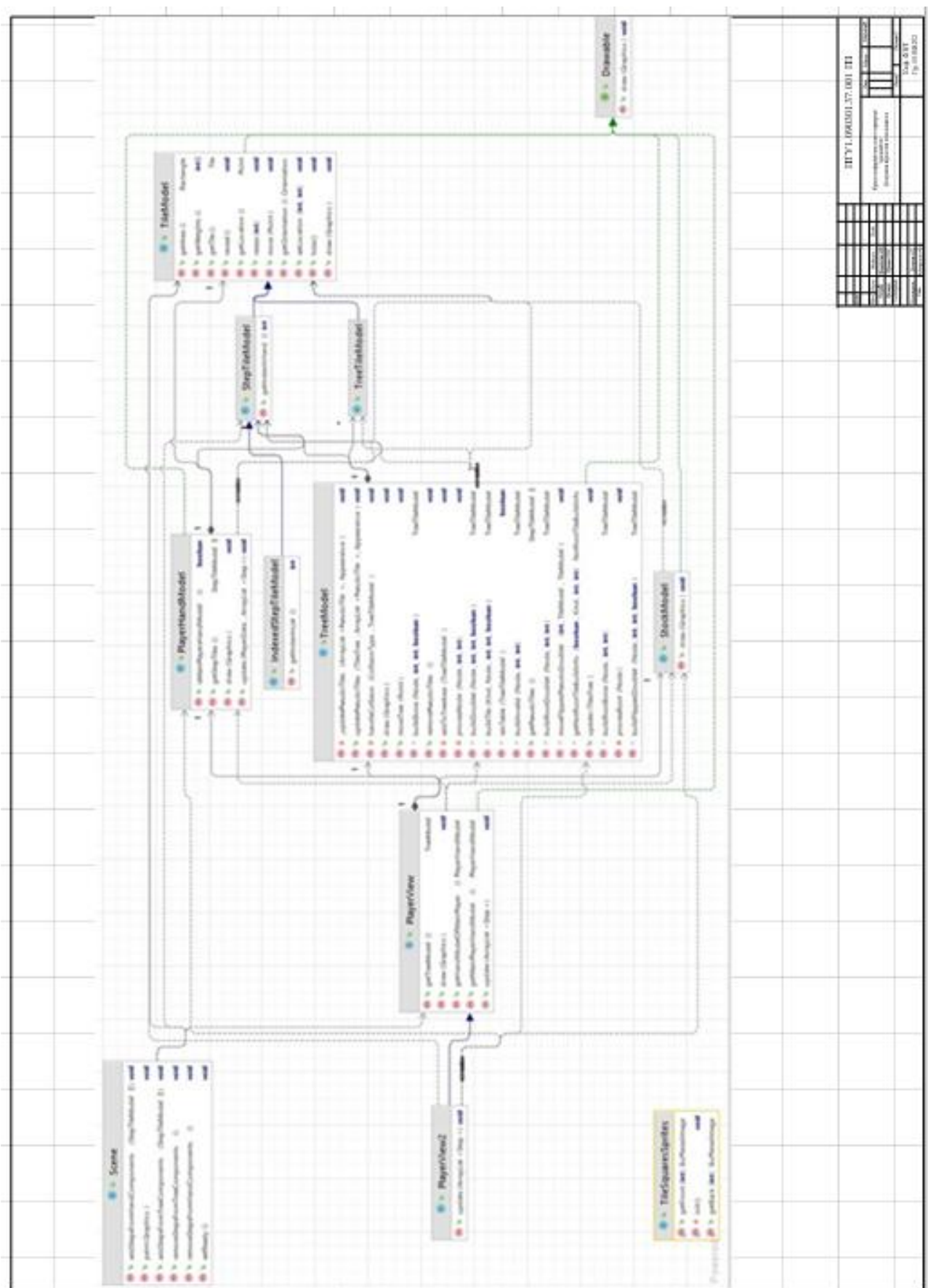
Приложение Б.2. –UML– диаграмма деятельности



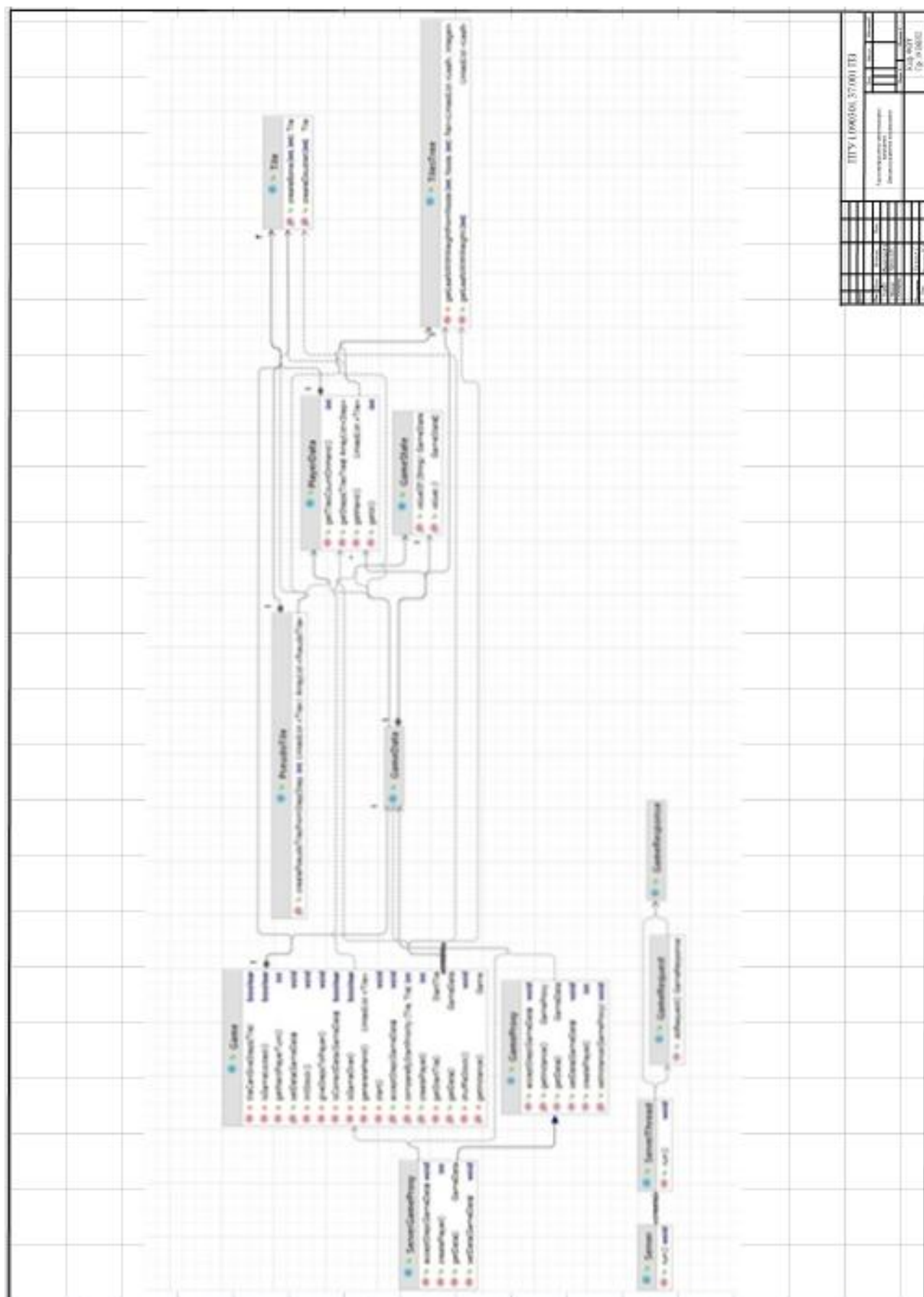
Приложение Б.3. –UML– диаграмма классов клиента



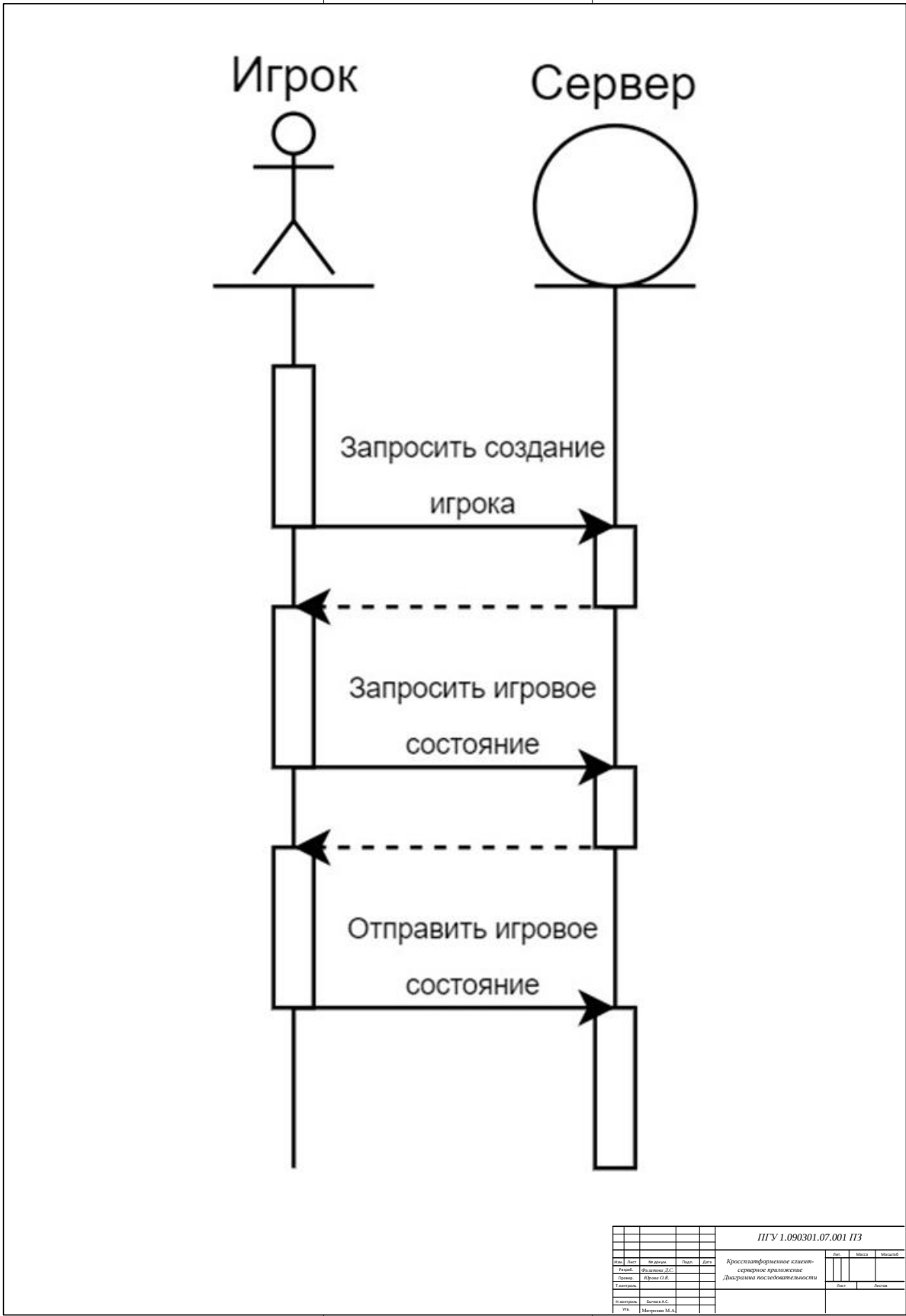
Приложение Б.4. –UML– диаграмма классов сервера



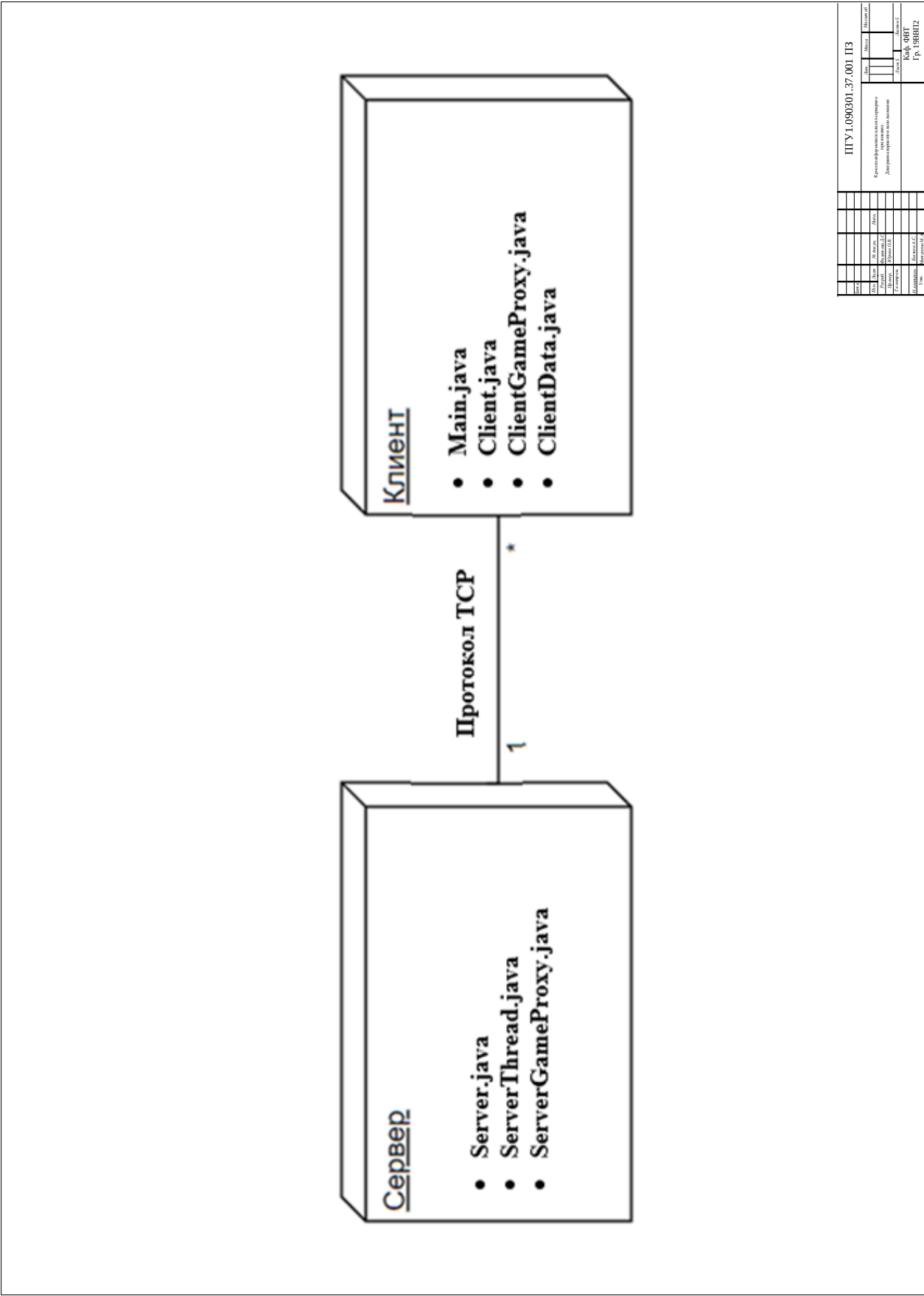
Приложение Б.5. –UML– диаграмма классов модулей рендеринга



Приложение Б.6. –UML– диаграмма последовательностей



Приложение Б.7. –UML– диаграмма развертывания

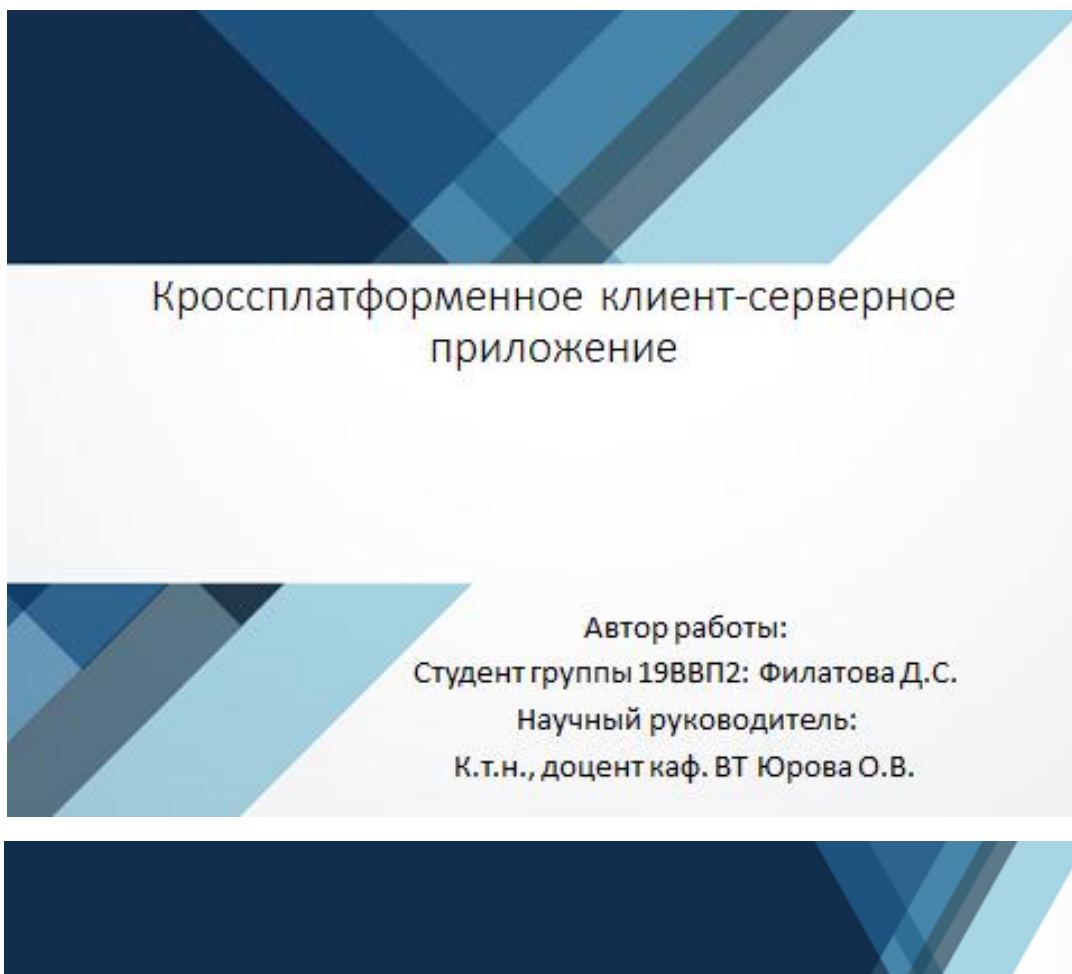


Приложение В- Презентация

(Обязательное)

Кроссплатформенное клиент-серверное приложение

Презентация



Цель работы:

Цель работы – разработать кроссплатформенное приложение, вида клиент-сервер, с возможностью запускать игру «Домино» для двух игроков по локальной сети.

- Функции сервера: прием информации от пользователя, ее передача обратно, просчет ходов, выдача костяшек игрокам.
- Функции клиента: графический интерфейс пользователя, работа с игровым состоянием, сетевое взаимодействие с сервером.

Средства разработки

Для реализации приложения «Домино» были использованы:

- Операционная система: Windows;
- Среда разработки: IntelliJ IDEA;
- Язык программирования Java.
- Используемые технологии:
- Java Collections Framework
- Механизм обработки исключительных ситуаций
- Java Stream API
- Java Multithreading
- Сетевое взаимодействие

Диаграмма использования

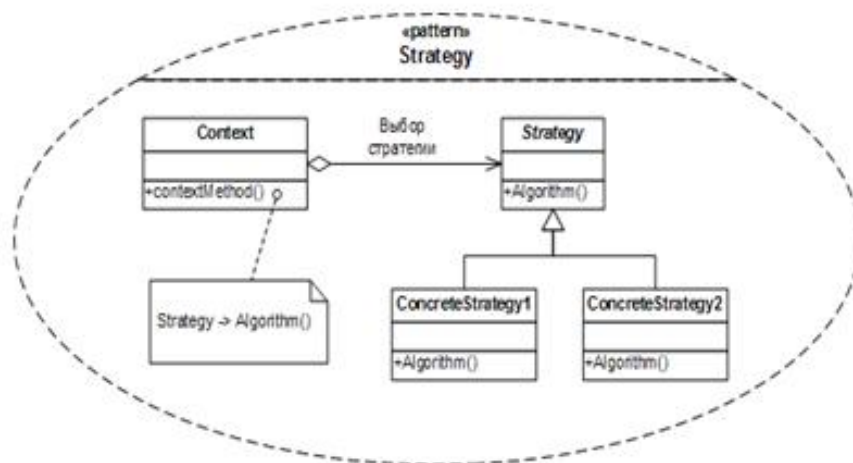


Структура проекта

- Взаимодействие между сервером и клиентом осуществлено с помощью сокетов и паттерна «Стратегия».



Паттерн «Стратегия»





Серверная часть

- *Server.java* – класс нити сервера.
- *ServerThread.java*



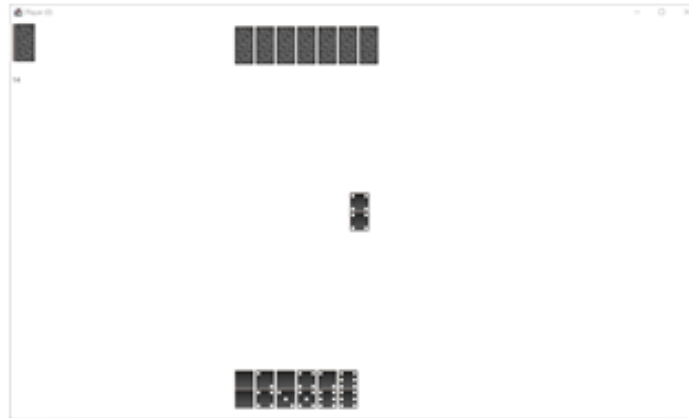
Клиентская часть

Клиентская часть реализована с помощью

- *Main.java* – основной класс программы.
- *Client.java* – класс реализующий функции клиента.

Интерфейс

Основное окно программы – JFrame библиотеки Swing



Заключение

- Результатом выпускной квалификационной работы является кроссплатформенное игровое приложение с интуитивно понятным пользовательским интерфейсом. Приложение позволяет двум игрокам сыграть одиночную игру по локальной вычислительной сети и определить победителя. Данный проект удовлетворяет большинству поставленных целей