

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Игровое приложение «Карточная игра по мотивам Heroes 3»

БГУИР КП 1-40 01 01 003 ПЗ

Студент: гр. 151003 Барановский Р.А.

Руководитель: Красковский П. Н.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и
радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ

(подпись)
Лапицкая Н.В. 2023г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Барановскому Роману Алексеевичу

1. Тема работы Игровое приложение «Карточная игра по мотивам Heroes 3»

2. Срок сдачи студентом законченной работы 16.05.2023г.

3. Исходные данные к работе Среда программирования Intelij IDEA. Библиотека для создания графического интерфейса swing. Язык программирования java. Наличие графической реализации интерфейса игры. Возможность работы в режиме сервера и клиента. Возможность сетевой игры между двумя игроками. Возможность игрока составить свою колоду.

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение

1 Анализ литературных источников

2 Постановка задачи

3 Разработка программного средства

4 Тестирование и проверка работоспособности программного средства

5 Руководство по использованию программного средства

Заключение

Список использованной литературы

Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

Схема алгоритма в формате A1

6. Консультант по курсовой работе Красковский П. Н. _____

7. Дата выдачи задания 27.02.2023г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

Раздел 1. Введение к 15.03.2023г. – 10 % готовности работы;

Раздел 2 к 30.03.2023г. – 30% готовности работы;

Раздел 3 к 10.04.2023г. – 60% готовности работы;

Разделы 4, 5, Заключение к 01.05.2023 – 90 % готовности работы;

Оформление пояснительной записки и графического материала к 15.05.2023 – 100 % готовности работы.

Защита курсового проекта с 15.05.2023г. по 16.05.2023г.

РУКОВОДИТЕЛЬ _____ Красковский П. Н.
(подпись)

Задание принял к исполнению _____ Барановский Р.А. 27.02.2023г.
(дата и подпись студента)

СОДЕРЖАНИЕ

Введение.....	6
1 анализ литературных источников.....	7
1.1 Анализ существующих аналогов.....	7
1.1.1 «Hearthstone».....	7
1.1.2 «Гвинт».....	8
1.1.3 «Kards».....	10
1.2 Анализ методов и способов разработки	11
1.2.1 Используемые библиотеки и технологии	11
1.2.2 Используемые структуры данных	11
2 Постановка задачи.....	12
2.1 Назначение разработки.....	12
2.2 Перечень функциональных требований	13
2.3 Структура программы.....	14
2.4 Состав и параметры технических и программных средств	15
3 Разработка программного средства.....	16
3.1 Описание алгоритмов решения задачи	16
3.2 Структура типов	18
3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90	22
3.3.1 Схема алгоритма GameController.startGame.....	22
3.3.2 Схема алгоритма GameConroller.nextTurn.....	25
3.3.3 Схема алгоритма Instance.setupServer	27
3.3.4 Схема алгоритма Instance.setupClient.....	29
3.3.5 Схема алгоритма Instance.run.....	31
4 тестирование и проверка работоспособности программного средства.....	33
4.1 Взаимодействие с главным меню	33
4.1.1 Тест 1	33
4.1.2 Тест 2.....	33
4.1.3 Тест 3.....	34
4.2 Взаимодействие с мастером колод.....	35
4.2.1 Тест 4.....	35
4.2.2 Тест 5.....	35
4.2.3 Тест 6.....	36
4.2.4 Тест 7.....	37
4.2.5 Тест 8.....	38
4.2.6 Тест 9.....	38
4.3 Взаимодействие с меню мультиплеера.....	39
4.3.1 Тест 10.....	39
4.3.2 Тест 11.....	39
4.3.3 Тест 12.....	40
4.4 Игровой процесс.....	41
4.4.1 Тест 13.....	41

4.4.2 Тест 14.....	41
4.4.3 Тест 15.....	42
4.4.4 Тест 16.....	43
4.4.5 Тест 17.....	44
5 Руководство по использованию программного средства	45
5.1 Мастер колод	45
5.2 Установление соединения	45
5.3 Игра.....	45
5.4 Завершение игры	45
Заключение	46
Список использованной литературы.....	47
Приложение А	48

ВВЕДЕНИЕ

Данная работа посвящена созданию карточной компьютерной игры по мотивам игры «Heroes of might and magic 3». Компьютерные игры являются неизменными спутниками развития человечества с самого начала развития вычислительной техники. Компьютерные игры могут создаваться на основе книг и фильмов, с чистого листа. Их влияние огромно, что может доказываться, например, тем, что с 2011 компьютерные игры официально признаны в США отдельным видом искусства. Компьютерные игры оказали столь огромное влияние на общество, что в информационных технологиях отмечена устойчивая тенденция к геймификации для неигрового прикладного программного обеспечения. Углубляясь в тему, на сегодняшний день огромную популярность завоевали коллекционные карточные игры. Суть данного типа игр сводится к созданию собственной карточной коллекции, которая пополняется путем покупок комплектов карт или получения их в качестве награды. Матчи между игроками сводятся к разыгрыванию карт с задачей первым свести очки здоровья оппонента к нулю. Данный тип игр завоевал сердца миллионов людей по всему миру благодаря своей увлекательности, вариативности и возможности собрать свою собственную колоду. Огромное количество игр данного жанра регулярно получают награды в самых разных номинациях. Цель данного курсового проекта состоит в создании увлекательной карточной игры, где у игрока будет возможность сразиться с оппонентом по сети, собрать свою собственную колоду на основе имеющихся в игре карт, изучить и опробовать различные игровые механики. В качестве основы для создания данной карточной игры выступила всемирно известная и народно любимая игра 1999 года выпуска «Heroes of might and magic 3». В качестве игровых карт выступают существа и заклинания из «Heroes of might and magic 3» с различными способностями и свойствами. Игровой интерфейс также выдержан в духе «Heroes of might and magic 3».

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

1.1 Анализ существующих аналогов

Современные коллекционные карточные игры представляют из себя сложные проекты с огромным количеством возможностей, карт и игровых механик. В множестве из них также выходят регулярные обновления, в которых меняется не только внутренняя составляющая, но и карты, используемые в игре. Это позволяет разнообразить игровой процесс и сделать игру более увлекательной.

1.1.1 «Hearthstone»

«Hearthstone» – наиболее известная на сегодняшний день коллекционная карточная игра, созданная компанией Blizzard Entertainment по мотивам вселенной warcraft. Hearthstone имеет огромную аудиторию, так, по статистике, в 2021 году в день в нее играли в среднем от 200000 до 270000 игроков.

Достоинства игры:

- отличная графическое, музыкальное сопровождение;
- регулярные обновления;
- разнообразие различных игровых режимов.

В свою очередь к недостаткам можно отнести:

- не слишком хорошую систему баланса.



Рисунок 1.1 – «Hearthstone»



Рисунок 1.2 – «Hearthstone»

1.1.2 «Гвинт»

«Гвинт» – игра, созданная польской студией CD Project RED по мотивам вселенной «Ведьмака», созданной писателем Анджеем Сапковским. «Гвинт» является развитием мини-игры из ролевой компьютерной игры «Ведьмак 3: Дикая охота».

Достоинства игры:

- отличное графическое сопровождение;
- оригинальность механики.

В свою очередь к недостаткам можно отнести:

- низкий темп игры.



Рисунок 1.3 – «Гвинт»



Рисунок 1.4 – «Гвинт»

1.1.3 «Kards»

«Kards» – коллекционная карточная игра по мотивам второй мировой войны, созданная исландской студией 1939 Games, была выпущена в 2020 году. Игра получила множество положительных отзывов, хоть и является относительно небольшим проектом, в steam игру скачало более 250000 игроков.

Достоинства игры:

- необычные механики;
- множество справочной информации о картах.

В свою очередь к недостаткам можно отнести:

- редкая частота обновлений.



Рисунок 1.5 – «Kards»

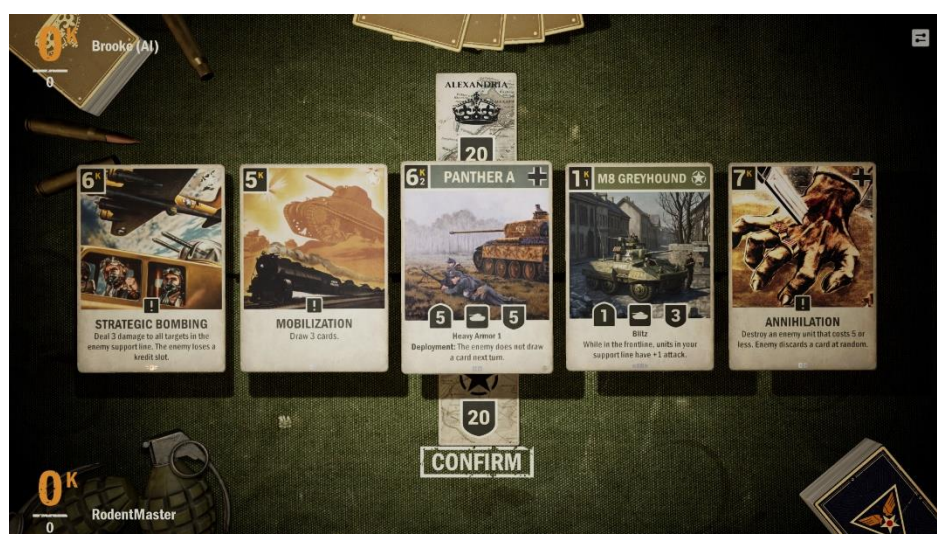


Рисунок 1.6 – «Kards»

1.2 Анализ методов и способов разработки

1.2.1 Используемые библиотеки и технологии

Предполагается, что разрабатываемая игра будет обладать графикой с возможностью взаимодействия с пользователем. Для этих целей будет использована сторонняя библиотека Swing.

Swing – библиотека для создания графического интерфейса на языке java. ПО было разработано компанией Sun Microsystems. Оно содержит ряд графических компонентов (англ. Swing Widgets), таких как кнопки, поля ввода, таблицы и другие.

Для построения архитектуры взаимодействия клиента и сервера будут использованы сокеты. Сокет - это интерфейс между вашим приложением и внешним миром: через сокет вы можете отправлять и получать данные. Сокет обеспечивает конечную точку сетевого соединения. Поэтому любой сетевой программе, скорее всего, придется иметь дело с сокетами, они являются центральным элементом сетевого взаимодействия.

Существует несколько видов сокетов, каждый из которых предоставляет определенные функции. В данной работе используются идущие в комплекте JDK Socket и ServerSocket.

1.2.2 Используемые структуры данных

В данном проекте в качестве основной структуры данных будет использован ArrayList.

Структура ArrayList – структура данных в языке java, предназначенная для хранения множества значений. Данная структура данных основана на массиве и является его усовершенствованной версией, так как динамична. Использование данной структуры облегчает работу за счет удаления необходимости вручную изменять размер массива, удалять элементы из него, вставлять элементы в массив. Структура ArrayList представлена на рисунке 1.7.

ArrayList Integer Object Type :

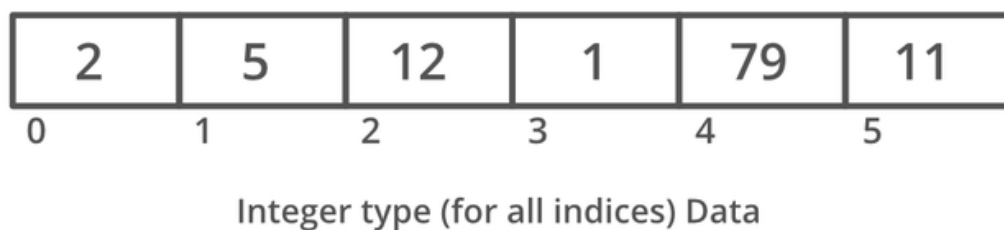


Рисунок 1.7 – ArrayList

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Назначение разработки

Назначением проектирования является разработка игры «Карточная игра по мотивам Heroes 3». На основании произведенного обзора существующих аналогов, выявленных преимуществ и недостатков данных игр, сделан вывод, что для решения поставленной цели необходимо выполнить следующие задачи:

- проектирование архитектуры игры;
- проектирование графического сопровождения;
- разработка алгоритмов взаимодействия игрока с картами;
- разработка алгоритмов взаимодействия объектов между собой;
- разработка алгоритма инициализации сервера;
- разработка алгоритма инициализации клиента;
- разработка алгоритмов обмена данными между пользователями;
- разработка алгоритмов синхронизации действий игроков;
- разработка алгоритмов отображения сопровождающей информации;
- тестирование приложения.

2.2 Перечень функциональных требований

Целью разработки игры «Карточная игра по мотивам Heroes 3» является объединение основных достоинств рассмотренных существующих аналогов, а также компенсация недостатков этих игр. В результате разработки необходимо предоставить реализацию следующих функций:

- подключение клиента к серверу;
- инициализация сервера и подключенного клиента;
- взаимодействие между картами обоих клиентов;
- взаимодействие между картами и героями клиентов
- отображение здоровья героя;
- завершение игры при уменьшении количества здоровья до нуля.

2.3 Структура программы

Основные модули программы:

- Main.java – из данного модуля производится запуск игры;
- SpriteHandler.java – загружает в игру все изображения
- Hero.java – содержит информацию о герое и производимые им и над ним действия
- PlayArea.java – обеспечивает логику размещения существ, находящихся у конкретного игрока на поле
- Minion.java – абстрактный класс для создания существ
- Card.java – абстрактный класс для создания карт
- MainMenuWindow.java – графический интерфейс главного меню
- Instance.java – класс, отвечающий за сетевое взаимодействие сервера и клиента

2.4 Состав и параметры технических и программных средств

Игра «Карточная игра по мотивам Heroes 3» должна функционировать на персональных компьютерах со следующими характеристиками:

- Операционная система Windows 10;
- RAM: 2 GB;
- Пространство на диске: 1 GB;
- Процессор: минимальное требование - Pentium 2 266 МГц;
- Монитор;
- Мышь;
- Клавиатура.

В данном разделе указаны минимальные технические требования для запуска игры. Для эксплуатации в реальных условиях могут потребоваться более мощные технические средства. Разработанная игра должна корректно функционировать на более мощном оборудовании.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Описание алгоритмов решения задачи

Таблица 1 – Описание алгоритмов решения задачи

№ п.п.	Наименование алгоритма	Назначение алгоритма	Формальные параметры	Предлагаемый тип реализации
1	Main.main()	Служит отправной точкой выполнения программы. Создает окно главного меню. Производит загрузку спрайтов игры. Использует алгоритмы: initialize(), setBackgroundImage(), MainMenuWindow()		Процедура
2	GameController.startGame(isServer)	Служит для запуска игры. Отправляет колоду противнику. Тасует колоды. Игроки вытягивают начальные карты. Использует алгоритмы: sendDeck(), shuffle(), draw(), dispose()	isServer – true, если метод вызвал сервер	Процедура
3	GameController.nextTurn()	Служит для передачи хода. Вызывает методы класса Hero для выполнения действий, связанных с окончанием хода одного игрока и началом хода другого. Использует алгоритмы: onTurnEnd(), onTurnStart(),		Процедура

Продолжение таблицы 1 – Описание алгоритмов решения задачи

4	Instance. setupServer()	Выполняет настройку сервера. Создает сокет игрока, выполняющего роль сервера, и ожидает подключения клиента.		Процедура
5	Instance. setupClient()	Выполняет настройку клиента. Создает сокет игрока, выполняющего роль клиента, и подключается к серверу.		Процедура
6	Instance. sendMessage (message)	Отправляет строку, сформированную определенным образом, другому пользователю	message – отправляемая строка	Процедура
7	Instance.run()	В данном алгоритме реализован цикл, обрабатывающий сообщения, пока открыт сокет. Работает аналогично и для клиента, и для сервера.		Процедура
8	Instance. parseMessage (message)	Обрабатывает присланную другим пользователем строку и выполняет действия, основываясь на полученном	message – полученная строка	Процедура
9	Board.render()	Отрисовывает основную часть игры(поле, руку, портреты, надписи). Использует алгоритмы: renderHeroes(), renderMinions(), renderPlayerHand(), renderEnemyHand()		Процедура
10	InputHandler. getMinionAt (x, y)	Возвращает миньона на поле с координатами { x, y }	x, y – координаты миньона	Функция. Возвращае мое значение - миньон

3.2 Структура типов

Таблица 2 – Структура типов программы

Элементы данных	Рекомендуемый тип	Назначение
Instance	<pre> class Instance{ public: Hero hero; boolean isServer; boolean isSyncedRandom; boolean isReceivedDeck; int port; boolean isConnected; String connectionAddress; Random random; ServerSocket serverSocket; Socket socket; InputStreamReader input; PrintStream output; BufferedReader reader; Instance instance; private: int randSeed package-private: connected public: closeConnections(); setupServer(); setupClient(); sendMessage(String message) sendDeck(ArrayList <Card> deck); parseMessage(String message); </pre>	<p>Класс, определяющий клиента или сервер;</p> <p>Поля данных:</p> <ul style="list-style-type: none"> hero – герой игрока; isServer – является ли сервером; isSyncedRandom – синхронизирован ли рандом; isReceivedDeck – была ли получена колода противника port – порт isConnected – установлено ли соединение сервера и клиента connectionAddress – IP-адрес сервера; random – рандом на основе randSeed; serverSocket – сокет сервера; socket – сокет клиента; input – поток для получения данных, который передается reader; output – поток для отправки данных; reader – для получения данных из потока, при этом BufferedReader имеет буфер, что делает чтение быстрее;

Продолжение таблицы 2 – Структура типов программы

	displayCard(Card card); run();	instance – текущий instance Методы класса: closeConnections() – закрывает сокет; setupServer() – настраивает сервер; setupClient() – настраивает клиента; sendMessage(String message) – отправляет сообщение определенного формата другому пользователю; sendDeck(ArrayList <Card> deck) – отправляет колоду в виде текста другому пользователю; parseMessage(String message) – парсит сообщение, пришедшее от другого пользователя.
Card	public: Integer WIDTH; Integer HEIGHT; double fontScale; String name; CardType cardType; boolean isTargeted; boolean isDamagesHeroes; int immediateDamage; Minion toSummon; String cardText; int cost; BufferedImage sprite; HeroClass cardClass; Hero owner;	Абстрактный класс, представляющий общую структуру любой карты в игре; Поля данных: WIDTH – ширина карты в пкс; HEIGHT – высота карты в пкс; fontScale – используется для простого изменения размера шрифта карт; String name – имя карты;

Продолжение таблицы 2 - Структура типов программы

	<p>public:</p> <p>render(Graphics2D graphics2D, int x, int y, boolean override);</p> <p>renderCardText(Graphics2D graphics2D, int x, int y);</p> <p>getCoordinateX();</p> <p>getCoordinateY();</p> <p>cast(Minion target);</p> <p>castOnHero(Hero target);</p> <p>canAfford();</p> <p>defaultMinionSummon();</p> <p>notifyPhantom(Minion targetMinion, Hero targetHero);</p> <p>setHero(Hero hero);</p> <p>getOwner();</p> <p>getCardList();</p>	<p>cardType – тип карты(миньон или заклинание);</p> <p>isTargeted – при использовании необходимо ли выбирать цель;</p> <p>isDamagesHeroes – при выборе цели можно ли наносить урон героям;</p> <p>immediateDamage – урон цели при выходе;</p> <p>toSummon – миньон, призывающийся при использовании карты;</p> <p>cardText – текст карты;</p> <p>cost – цена использования карты;</p> <p>sprite – изображение карты;</p> <p>cardClass – класс карты;</p> <p>owner – герой-владелец карты;</p> <p>Методы класса:</p> <p>render(Graphics2D graphics2D, int x, int y, boolean override) – отрисовывает карту;</p> <p>renderCardText Graphics2D graphics2D, int x, int y) – отрисовывает текст карты;</p> <p>getCoordinateX() – возвращает X координату карты;</p> <p>getCoordinateY() – возвращает Y координату карты;</p>
--	--	---

Продолжение таблицы 2 – Структура типов программы

		<p>cast(Minion target) – разыграть карту(если есть цель – это миньон);</p> <p>castOnHero(Hero target) – разыграть карту(если есть цель – это герой);</p> <p>canAfford() – возвращает true, если герой-владелец карты может позволить ее разыграть по стоимости;</p> <p>defaultMinionSummon() – стандартный призыв миньона при разыгрывании карты;</p> <p>notiphyPhantom(Minion targetMinion, Hero targetHero) – формирует сообщение и отправляет противнику;</p> <p>setHero(Hero hero) – назначает героя-владельца карты;</p> <p>getOwner() – возвращает героя-владельца;</p> <p>getCardList() – возвращает все карты, доступные в игре.</p>
--	--	--

3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90

3.3.1 Схема алгоритма GameController.startGame

Схема алгоритма запуска игры. Порядок тасования колод важен, так как после синхронизации рандома, имея randSeed, сервер и клиент будут по отдельности генерировать одинаковые псевдорандомные значения. И для того, чтобы клиент мог знать о порядке карт в колоде сервера, а сервер о порядке карт в колоде клиента, необходимо, чтобы псевдорандомные значения, определяющие то, как будут перетасованы колоды, совпадали. Это необходимо, так как при взаимодействии сервера и клиента передаются индексы карт, миньонов и т.д., и подразумевается, что сторона-получатель сообщения знает, какая карта, миньон и т.д. находится под соответствующим индексом.



Рисунок 3.1 – Схема алгоритма startGame (часть 1)



Рисунок 3.1 – Схема алгоритма `startGame` (часть 2)

3.3.2 Схема алгоритма GameController.nextTurn

Схема алгоритма передачи хода.

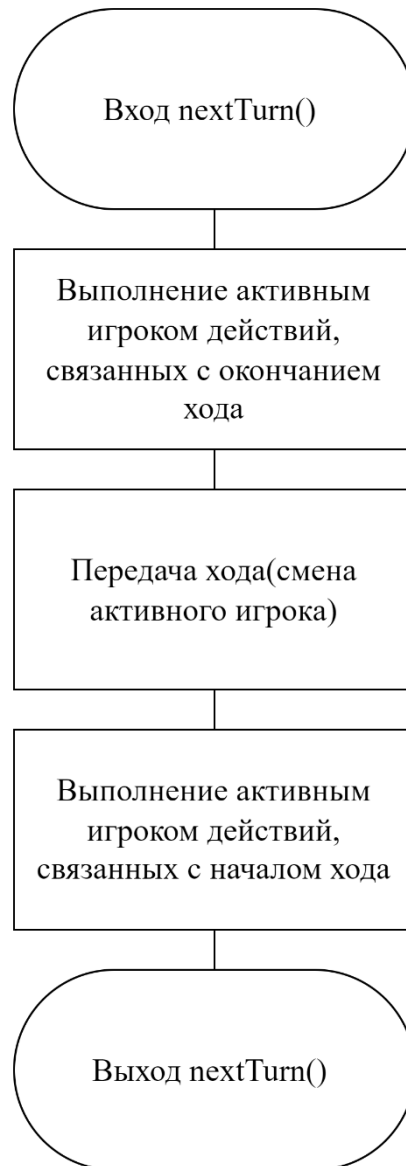


Рисунок 3.2 – Схема алгоритма nextTurn

3.3.3 Схема алгоритма Instance.setupServer

Схема алгоритма настройки сервера перед началом сетевого взаимодействия.

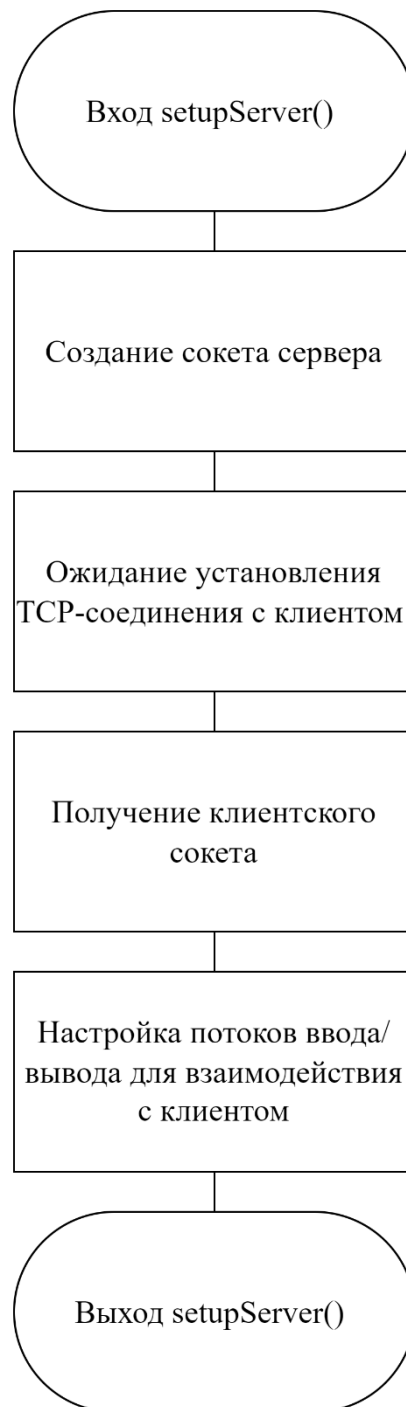


Рисунок 3.3 – Схема алгоритма `setupServer`

3.3.4 Схема алгоритма Instance.setupClient

Схема алгоритма настройки клиента перед началом сетевого взаимодействия.

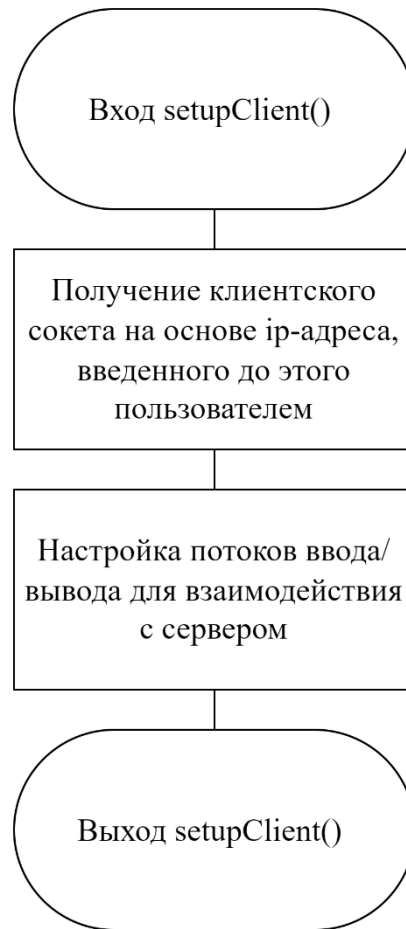


Рисунок 3.4 – Схема алгоритма `setupClient`

3.3.5 Схема алгоритма Instance.run

Схема алгоритма работы потока сервера или клиента. Следует обратить внимание, что работа сервера и клиента по большому счету идентична, за исключением стадии подготовки к игре. Также стоит отметить, что функция отправки сообщения реализована также в классе Instance и вызывается в других методах, которые и формируют отправляемое сообщение.

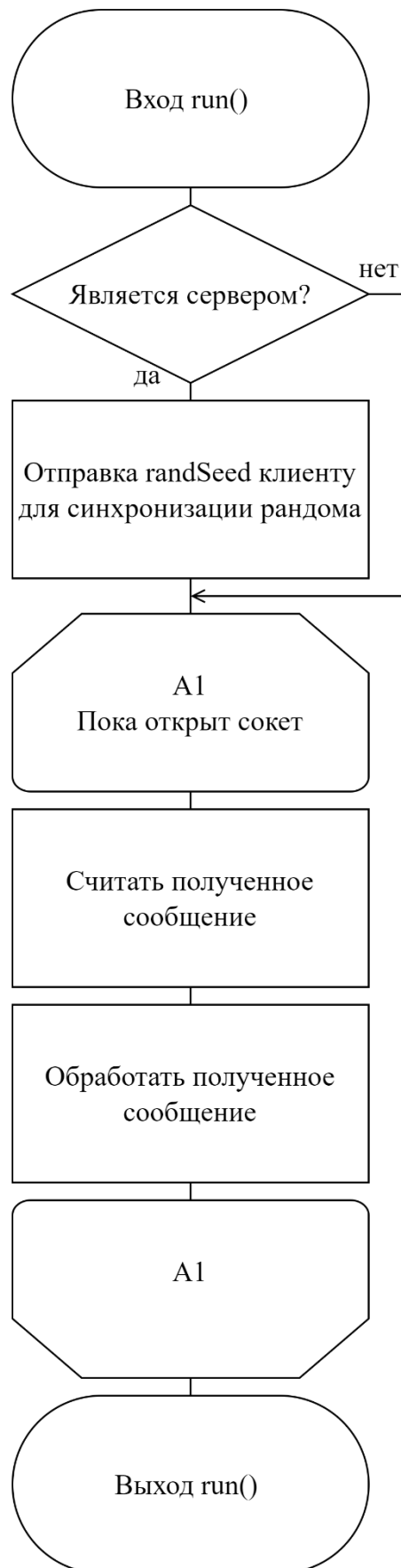


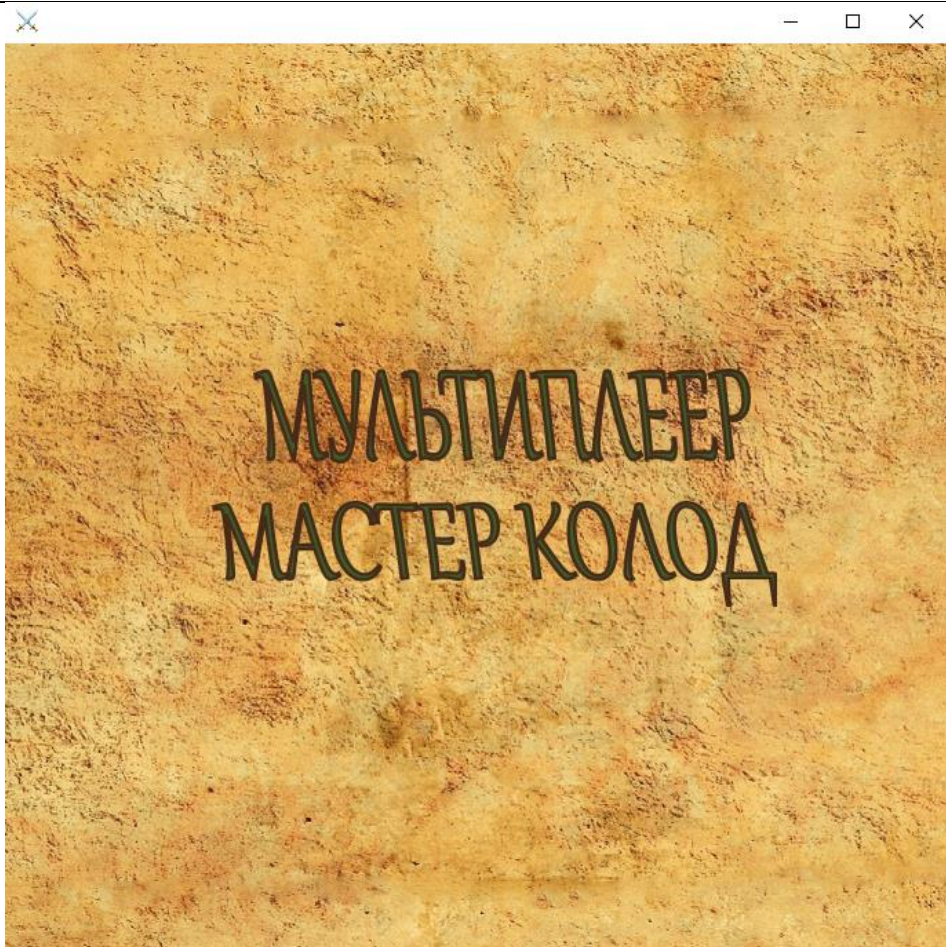
Рисунок 3.5 – Схема алгоритма run

4 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

4.1 Взаимодействие с главным меню

4.1.1 Тест 1

Таблица 3 – Тест1

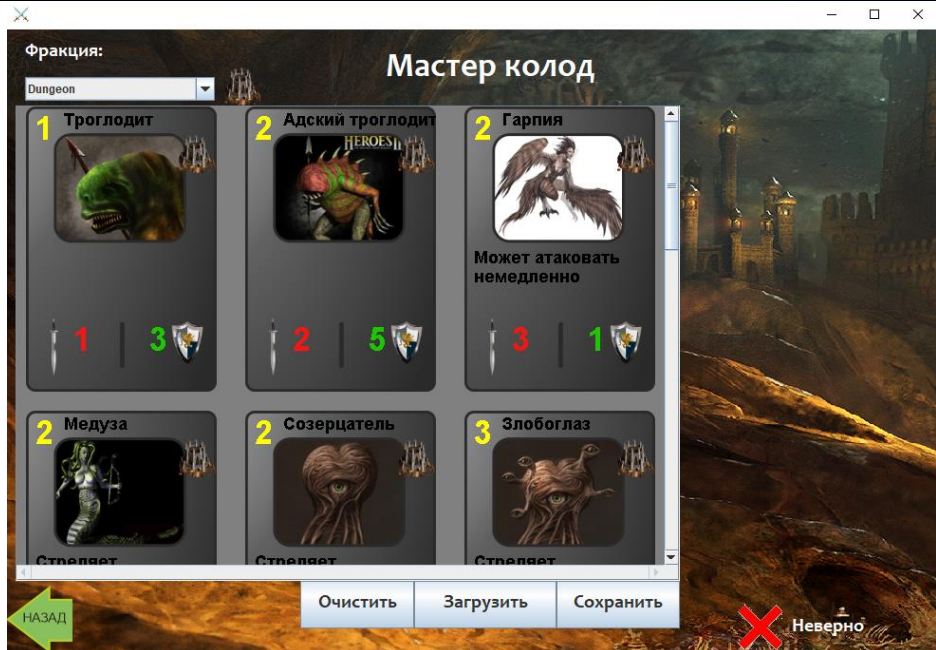
Тестовая ситуация:	Проверка корректности отображения главного меню при запуске программы
Исходный набор данных:	Запуск программы
Ожидаемый результат:	Корректное открытие главного меню
Полученный результат:	

4.1.2 Тест 2

Таблица 4 – Тест 2

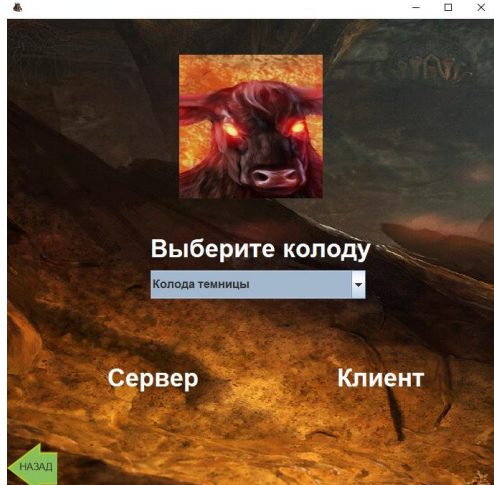
Тестовая ситуация:	Проверка корректности открытия мастера колод
--------------------	--

Продолжение таблицы 4 – Тест 2

Исходный набор данных:	Нажатие на кнопку «Мастер колод»
Ожидаемый результат:	Корректное открытие мастера колод
Полученный результат:	

4.1.3 Тест 3

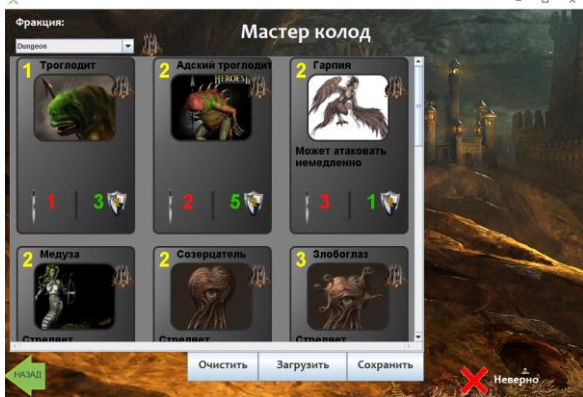
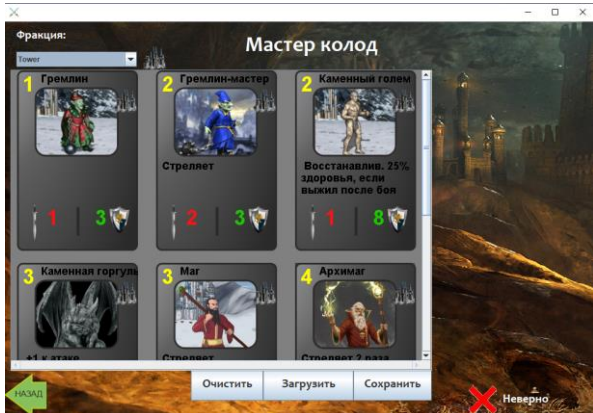
Таблица 5 – Тест 3

Тестовая ситуация:	Проверка корректности открытия мультиплеера
Исходный набор данных:	Нажатие на кнопку «Мультиплеер»
Ожидаемый результат:	Корректное открытие мультиплеера
Полученный результат:	

4.2 Взаимодействие с мастером колод

4.2.1 Тест 4

Таблица 6 – Тест 4

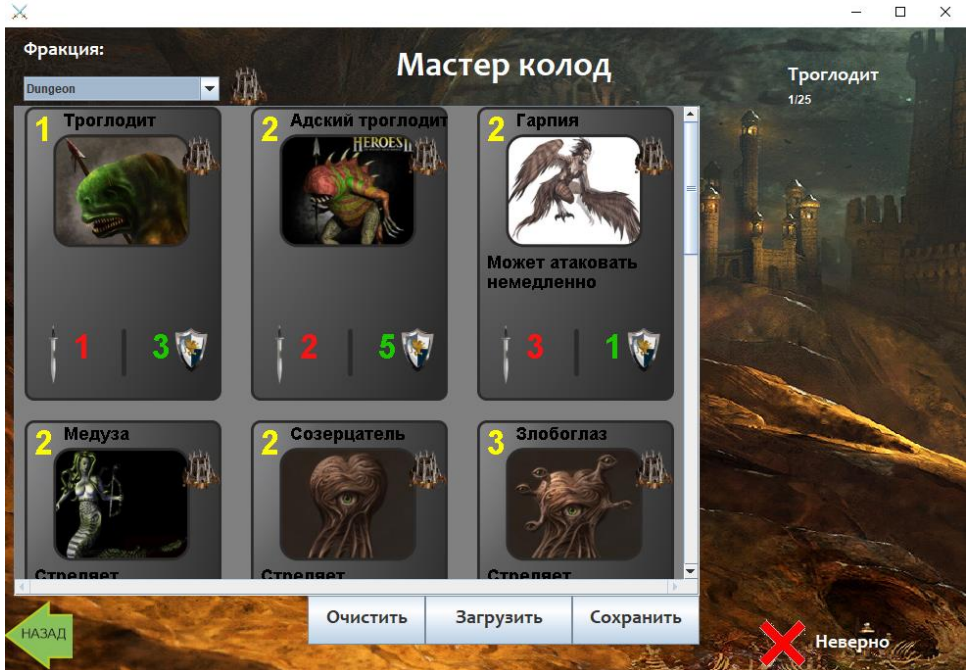
Тестовая ситуация:	Смена фракции
Исходный набор данных:	Нажатие на выпадающий список фракциями
Ожидаемый результат:	Корректная смена текущей фракции
Полученный результат:	<p>До:</p>  <p>После:</p> 

4.2.2 Тест 5

Таблица 7 – Тест 5

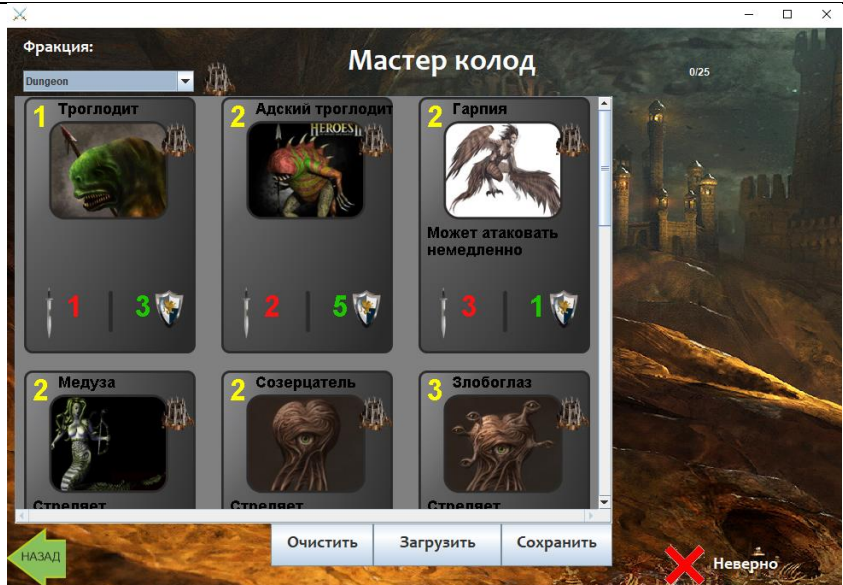
Тестовая ситуация:	Добавление карты в колоду
Исходный набор данных:	Нажатие на карту существа в списке
Ожидаемый результат:	Добавление карты в колоду в правой части экрана

Продолжение таблицы 7 – Тест 5

Полученный результат:	
-----------------------	--

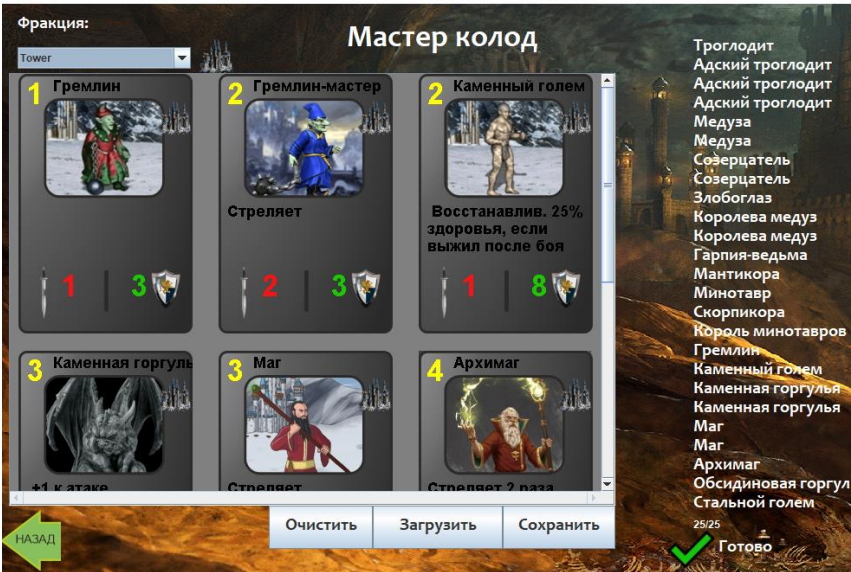
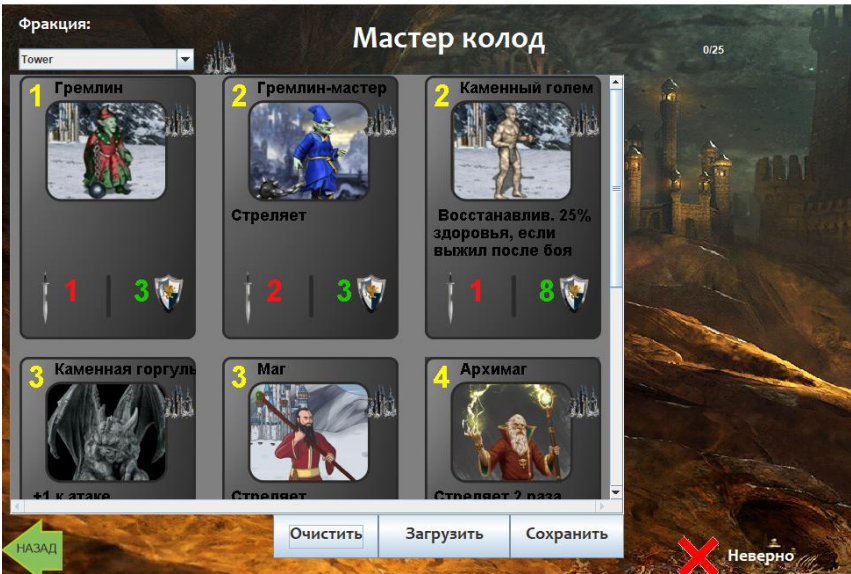
4.2.3 Тест 6

Таблица 8 – Тест 6

Тестовая ситуация:	Удаление карты из колоды
Исходный набор данных:	Нажатие на карту в списке или на название карты в списке карт в колоде
Ожидаемый результат:	Удаление карты из колоды в правой части экрана
Полученный результат:	

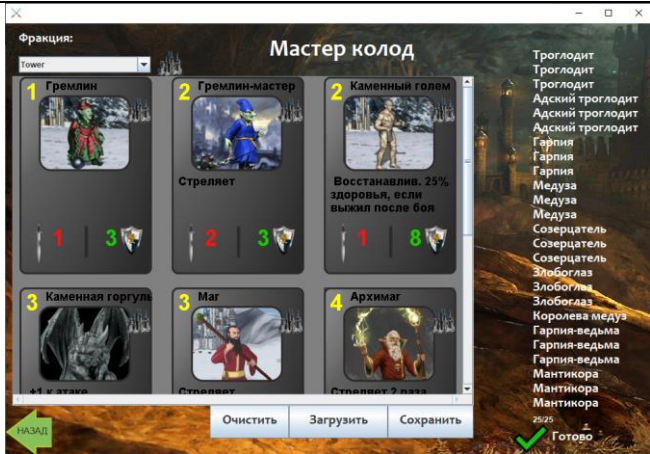
4.2.4 Тест 7

Таблица 9 – Тест 7

Тестовая ситуация:	Проверка работоспособности кнопки «Очистить»
Исходный набор данных:	Нажатие на кнопку «Очистить»
Ожидаемый результат:	Удаление всех карт из колоды
Полученный результат:	<p>До:</p>  <p>После:</p> 

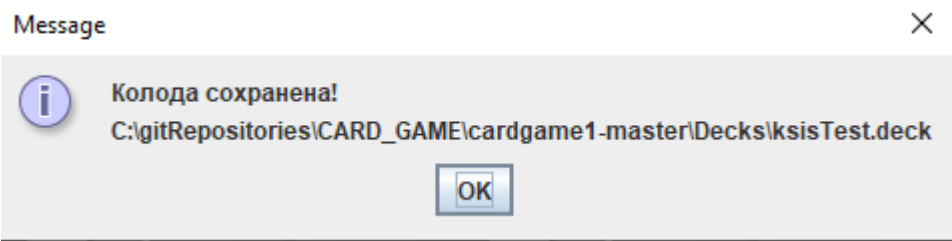
4.2.5 Тест 8

Таблица 10 – Тест 8

Тестовая ситуация:	Проверка работоспособности кнопки «Загрузить»
Исходный набор данных:	Нажатие на кнопку «Загрузить»
Ожидаемый результат:	Загрузка выбранной колоды
Полученный результат:	

4.2.6 Тест 9

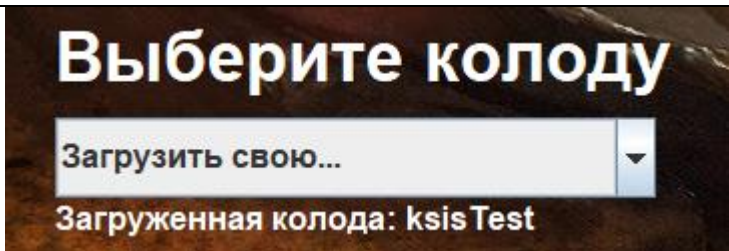
Таблица 11 – Тест 9

Тестовая ситуация:	Проверка работоспособности кнопки «Сохранить»
Исходный набор данных:	Нажатие на кнопку «Сохранить»
Ожидаемый результат:	Сохранение созданной колоды
Полученный результат:	

4.3 Взаимодействие с меню мультиплеера


4.3.1 Тест 10

Таблица 12 – Тест 10

Тестовая ситуация:	Проверка возможности загрузить колоду, созданную в мастере колод
Исходный набор данных:	Выбор в выпадающем списке пункта «Загрузить свою»
Ожидаемый результат:	Загрузка кастомной колоды
Полученный результат:	

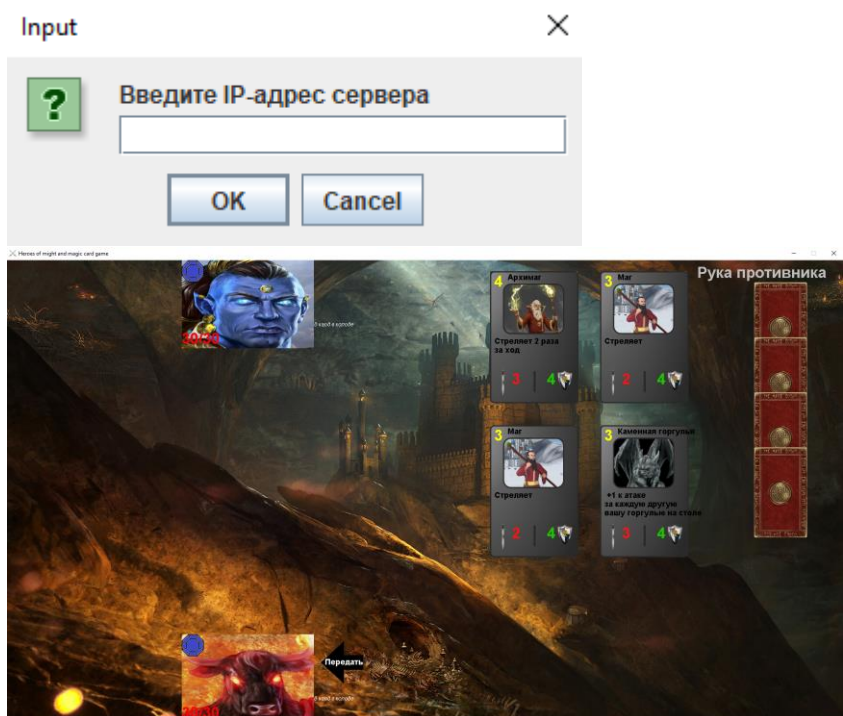
4.3.2 Тест 11

Таблица 13 – Тест 11

Тестовая ситуация:	Проверка корректности запуска сервера
Исходный набор данных:	Нажатие на кнопку «Сервер»
Ожидаемый результат:	Запуск сервера
Полученный результат:	

4.3.3 Тест 12

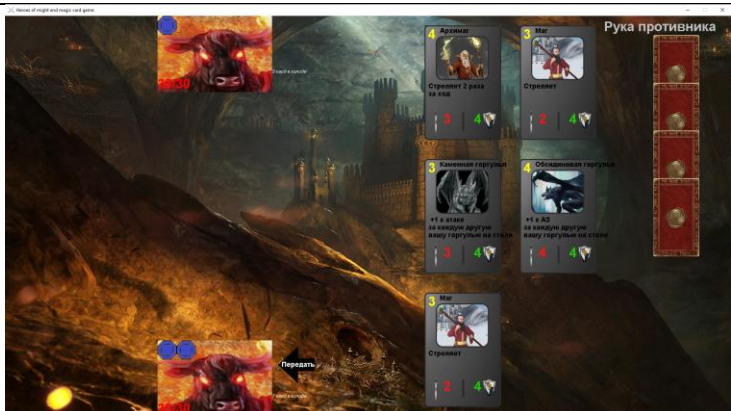
Таблица 14 – Тест 12

Тестовая ситуация:	Проверка корректности подключения клиента
Исходный набор данных:	Нажатие на кнопку «Клиент»
Ожидаемый результат:	Подключение клиента, запуск игры
Полученный результат:	 <p>The screenshot shows a game interface with a connection dialog box in the foreground. The dialog box is titled 'Input' and contains a green question mark icon, the text 'Введите IP-адрес сервера' (Enter server IP address), an input field, and 'OK' and 'Cancel' buttons. In the background, the game board is visible, showing a dark, rocky landscape with a castle. The board includes character portraits (Artemis, Mag, and a Gorgon), action cards (e.g., 'Стреляет 2 раза за ход'), and a 'Рука противника' (Opponent's hand) section on the right. A 'Передать' (Transfer) button is also visible at the bottom.</p>

4.4 Игровой процесс


4.4.1 Тест 13

Таблица 15 – Тест 13

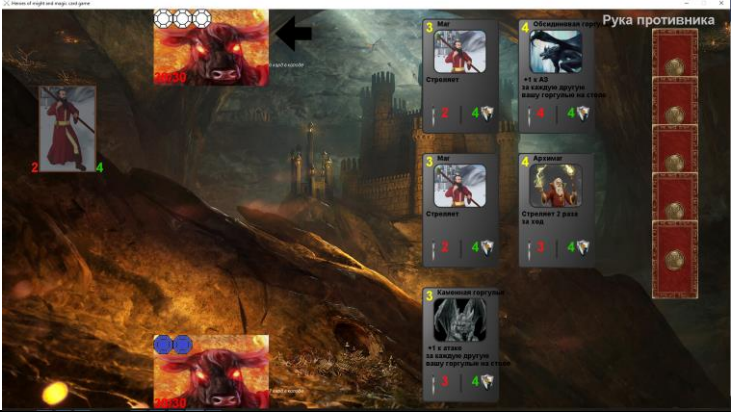
Тестовая ситуация:	Проверка корректности передачи хода и обновление количества маны у игрока, принимающего ход
Исходный набор данных:	Нажатие на кнопку «Передать»
Ожидаемый результат:	Передача хода другому игроку и обновление количества маны
Полученный результат:	 <p>The screenshot shows the Hearthstone game interface during a match. The player's hero, a Demon Hunter, is on the left. The opponent's hero, a Mage, is on the right. The board shows several cards in play, including 'Archaic' and 'Magma'. The 'Pass' button is visible at the bottom center of the board, indicating the player's turn to pass the turn to the opponent.</p>

4.4.2 Тест 14

Таблица 16 – Тест 14

Тестовая ситуация:	Использование карты
Исходный набор данных:	Нажатие на кнопку на нужную карту
Ожидаемый результат:	Появление карты на столе у обоих игроков, обновление количества маны у сыгравшего игрока
Полученный результат:	 <p>The screenshot shows the Hearthstone game interface during a match. The player's hero, a Demon Hunter, is on the left. The opponent's hero, a Mage, is on the right. The board shows several cards in play, including 'Archaic' and 'Magma'. A card is being played from the player's hand onto the board, and the mana cost is being updated.</p>

Продолжение таблицы 16 – Тест 14

Тестовая ситуация:	
--------------------	--

4.4.3 Тест 15

Таблица 17 – Тест 15

Тестовая ситуация:	Атака одним миньоном другого
Исходный набор данных:	Перетягивание курсора мыши с зажатой левой кнопкой с атакующего миньона на атакуемого
Ожидаемый результат:	Вычитание очков урона у обоих миньонов и отображение результата у обоих игроков
Полученный результат:	

4.4.4 Тест 16

Таблица 18 – Тест 16

Тестовая ситуация:	Атака миньоном героя
Исходный набор данных:	Перетягивание курсора мыши с зажатой левой кнопкой с атакующего миньона на героя противника
Ожидаемый результат:	Вычитание очков урона у героя противника и отображение результата у другого игрока
Полученный результат:	

4.4.5 Тест 17

Таблица 19 – Тест 17

Тестовая ситуация:	Проверка работоспособности свойства миньона
Исходный набор данных:	Нахождение миньона на поле.
Ожидаемый результат:	Другие миньона класса «Горгулья» получают прибавку к урону и/или здоровью и результат отображается у другого игрока
Полученный результат:	

5 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

5.1 Мастер колод

В мастере колод у игрока имеется возможность собрать собственную колоду из имеющихся в игре карт. Для добавления карты необходимо нажать правой кнопкой мыши на карту, которую пользователь желает добавить. Для удаления необходимо нажать на имя карты в колоде или же на изображение карты в списке. Для сохранения/загрузки/очистки колоды есть соответствующие кнопки под списком доступных карт.

5.2 Установление соединения

Для установления соединения первому игроку необходимо нажать кнопку «Сервер», на экране появится IP-адрес игрока и порт. Второму игроку необходимо нажать кнопку «Клиент», ввести IP-адрес сервера и нажать кнопку «Ок».

5.3 Игра

После выполнения действий из пункта 5.2 начнется игра. Первым всегда ходит сервер. Во время хода игрок может разыграть карту или атаковать миньоном. В конце хода для передачи его другому игроку необходимо нажать кнопку «Передать» рядом с портретом героя.

5.4 Завершение игры

При желании по нажатию кнопки «Escape» во время игры можно выйти в главное меню и при закрытии приложения соединение будет разорвано. Игра продолжается до тех пор, пока один из игроков не выйдет из нее или же пока здоровье одного из героев игроков не упадет до 0.

ЗАКЛЮЧЕНИЕ

В результате разработки игры «Карточная игра по мотивам Heroes 3» было получено приложение, позволяющее полноценно испытать достоинства и недостатки коллекционных карточных игр.

Данная игра является простой в освоении и интуитивно понятной, так как все функции реализованы с использованием максимально понятных механик.

Для успешного выполнения всех поставленных целей потребовалось ознакомиться со средой разработки IntelliJ Idea. Для создания графического интерфейса требовалось изучить различные аналоги игры в данном жанре. Также потребовалось изучить возможности библиотеки Swing.

Приложение прошло все этапы тестирования, в результате которых были устранены все неполадки. Приложение имеет относительно высокую скорость работы. Возможна дальнейшая доработка при выявлении ошибок.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2004. – 1 ч.
- [2] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2005. – 2 ч.
- [3] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2007. – 3 ч.
- [4] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2013. – 4 ч.
- [5] Серебряная Л.В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л.В. Серебряная, И.М. Марина. – Минск: БГУИР, 2013. – 5 с.
- [6] Вирт Н. Алгоритмы и структуры данных / Н. Вирт. – Москва: Мир 1989. – 90 с.
- [7] Глухова Л.А. Основы алгоритмизации и программирования: учебное пособие / Л.А. Глухова. – Минск: БГУИР, 2006. – 1 ч.
- [8] JavaRush[Электронный ресурс]. – Режим доступа: <https://javarush.com/> – Дата обращения: 10.05.2023.
- [9] JavaOnline[Электронный ресурс]. – Режим доступа: <https://java-online.ru/libs-swing.xhtml> – Дата обращения: 05.05.2023.
- [10] StackOverflow[Электронный ресурс]. – Режим доступа: <https://stackoverflow.com/> – Дата обращения: 12.05.2023.
- [11] Metanit[Электронный ресурс]. – Режим доступа: <https://metanit.com/java/> – Дата обращения: 11.05.2023.
- [12] Сурков Д.А. Сети ЭВМ: лабораторный практикум / Д.А. Сурков, С.В. Коростель, Е.В. Мельникова, И.М. Марина. – Минск: БГУИР, 2006. – 1 ч.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы
Модуль Instance.java

```
package multiplayer;

import cards.Card;
import customDecks.CustomDeck;
import customDecks.HeroClass;
import logic.Board;
import logic.Hero;
import logic.Main;
import logic.Sticker;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.JOptionPane;

public final class Instance implements Runnable {
    public Hero hero;
    public boolean isServer;
    public static volatile boolean isSyncedRandom = false;
    public boolean isReceivedDeck = false;
    public static int port = 444;
    static boolean isConnected = false;
    private static final int randSeed = (int)
                                                (Math.random() * 9999);
    public static String connectionAddress = "localhost";
    public static Random random = new Random(randSeed);
    public ServerSocket serverSocket;
    public Socket socket;
    public InputStreamReader input;
    public PrintStream output;
    public BufferedReader reader;
    public static Instance instance;

    public static void closeConnections() throws
                                                IOException {
        if (instance != null) {
```



```

        if (instance.isServer) {
            instance.serverSocket.close();
        } else {
            instance.socket.close();
        }
    }
    isSyncedRandom = false;
}

//Creates instance to control the given hero,if client,
//automatically tries to connect
public Instance(Hero hero, boolean isServer) throws
                                                    Exception {

    closeConnections();
    instance = this;
    this.hero = hero;
    this.isServer = isServer;
    if (isServer) {
        setupServer();
    } else {
        setupClient();
    }
    Thread t = new Thread(this);
    t.start();
}

public void setupServer() throws Exception {
    serverSocket = new ServerSocket(port);
    int timeoutDuration = 50000; //Time to connect
    new TimeoutController(timeoutDuration,
                           serverSocket);
    socket = serverSocket.accept();

    //Reaching this point means connection is
    //successful
    isConnected = true;
    input = new
        InputStreamReader(socket.getInputStream());
    reader = new BufferedReader(input);
    output = new PrintStream(socket.getOutputStream());
}

public void setupClient() throws Exception {
    socket = new Socket(Instance.connectionAddress,
                       444);

    input = new
        InputStreamReader(socket.getInputStream());
    output = new PrintStream(socket.getOutputStream());
}

```

```

        reader = new BufferedReader(input);
    }

    public synchronized void sendMessage(String message) {
        output.println(message);
    }

    @Override
    public void run() {
        try {
            if (isServer) {
                output.println("randSeed: " + randSeed);
            }
            while (!socket.isClosed()) {
                String message = reader.readLine();
                parseMessage(message);
            }
        } catch (SocketException se) {
            if (se.getMessage().equals("Connection reset"))
            {
                JOptionPane.showMessageDialog(null,
"Соединение разорвано");
                System.exit(0);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public synchronized void sendDeck(ArrayList<Card> deck)
    {
        sendMessage("deckSend");
        for (Card c : deck) {
            sendMessage("card:" + c.name);
        }
        sendMessage("deckEnd");
    }

    /* PLAY COMMANDS FORMAT: actorType-actorIndex-
targetType-targetIndex
    * actorType: what we call to perform an action: c -
card, m - minion
    * actorIndex: index of minion/card: 0-5 cards in hand,
0-3 minion on board
    * targetType: fm(friendly minion), em(enemy minion),
fh(friendly hero), eh(enemy hero)
    * target index: index of target

```

```

    * "end": end turn
    * */
private boolean receivingDeck = false;

private synchronized void parseMessage(String message)
{
    try {
        if (message == null) return;
        if (message.startsWith("heroPortrait")) {
            hero.heroPortraitPath = message.split
                (" ")[1];
            switch (hero.heroPortraitPath) {
                case "Assets\\hero.png" -> hero.picture
                    = HeroClass.Dungeon.getHeroPortrait();
                case "Assets\\towerHero.png" ->
                    hero.picture =
                    HeroClass.Tower.getHeroPortrait();
            }
        }
        if (message.equals("gotRandom")) {
            isSyncedRandom = true;
            return;
        }
        if (message.equals("deckSend")) {
            hero.deck.clear();
            receivingDeck = true;
            return;
        }
        if (message.equals("deckEnd")) {
            receivingDeck = false;
            isReceivedDeck = true;
        }
        if (receivingDeck &&
            message.startsWith("card:")) {
            Card toAdd =
                CustomDeck.getCard(message.substring(5));
            toAdd.setHero(hero);
            hero.deck.add(toAdd);
        }
        if (message.startsWith("randSeed: ")) {
            if (isSyncedRandom) return;
            int seed = Integer.parseInt(message.split
                (" ")[1]);
            random = new Random(seed);
            isSyncedRandom = true;
            sendMessage("gotRandom");
            return;
        }
    }
}

```

```

if (message.equals("end")) {
    Main.wait(300);
    Board.controller.nextTurn();
    return;
}
if (!hero.turn) return; //Messages are not
                        //executed if it's not your turn
String[] contents = message.split(" ");
int actorIndex = Integer.parseInt(contents[1]);
int targetIndex =
    Integer.parseInt(contents[3]);
Card toUse = null;
switch (contents[0]) {
    case "c": //Cast card
        switch (contents[2]) {
            case "fm" -> { //On friendly minion
                toUse =
                    hero.hand.get(actorIndex);
                displayCard(toUse);

                toUse.cast(hero.minions.
                    get(targetIndex));
            }
            case "em" -> { //On enemy minion
                toUse =
                    hero.hand.get(actorIndex);
                displayCard(toUse);

                toUse.cast(hero.opponent.minions.
                    get(targetIndex));
            }
            case "fh" -> { //On friendly hero
                toUse =
                    hero.hand.get(actorIndex);
                displayCard(toUse);
                toUse.castOnHero(hero);
            }
            case "eh" -> { //On enemy hero
                toUse =
                    hero.hand.get(actorIndex);
                displayCard(toUse);

                toUse.castOnHero(hero.opponent);
            }
            case "n" -> {
                toUse =
                    hero.hand.get(actorIndex);
                displayCard(toUse);
            }
        }
    }
}

```

```

        toUse.cast(null);
    }
}
break;
case "m": //Minion attacks
    switch (contents[2]) {
        case "fm" -> //Friendly
            //minion(forbidden)

            hero.minions.get(actorIndex).
            attack(hero.minions.
            get(targetIndex));
        case "em" -> //Enemy minion

            hero.minions.get(actorIndex).
            attack(hero.opponent.minions.
            get(targetIndex));
        case "fh" -> //Friendly
            //hero(forbidden)

            hero.minions.get(actorIndex).
            attack(hero);
        case "eh" -> //Enemy hero

            hero.minions.get(actorIndex).
            attack(hero.opponent);
    }
    break;
}

} catch (Exception e) {
    e.printStackTrace();
}

}

//Used to show user that card is being played
private void displayCard(Card card) {
    new Sticker(card, 1700, 200, 1600);
    Main.wait(1600);
}
}

```

Модуль GameController.java

```
package cardgame;

import GUI.MultiplayerFrame;
import Multiplayer.Instance;

import java.util.logging.Level;
import java.util.logging.Logger;

public class GameController {
    public Hero activeHero;
    public Hero inactiveHero;
    public Board board;

    public GameController(Board board) {
        this.board = board;
    }

    //Actions to do on turn switch
    public void nextTurn() {
        activeHero.onTurnEnd();
        Hero temp = activeHero;
        activeHero = inactiveHero;
        inactiveHero = temp;
        activeHero.onTurnStart();
    }

    public void startGame() {

        Board.topHero.shuffle();
        Board.botHero.shuffle();
        for (int i = 0; i < 4; i++) {
            Board.topHero.draw();
            Board.botHero.draw();
        }
        activeHero = Board.botHero;
        activeHero.turn = true;
        inactiveHero = Board.topHero;
        inactiveHero.turn = false;
    }

    //Starts the game for multiplayer
    public void startGame(boolean isServer) {
        if (MultiplayerFrame.mainMultiplayerFrame != null)
            MultiplayerFrame.mainMultiplayerFrame.dispose();
        Instance.instance.sendDeck(Board.playerHero.deck);
    }
}
```

```

        while (!Instance.isSyncedRandom ||
!Instance.instance.isReceivedDeck) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        if (isServer) {
            Board.botHero.shuffle();
            Board.topHero.shuffle();
        } else {
            Board.topHero.shuffle();
            Board.botHero.shuffle();
        }

        for (int i = 0; i < 4; i++) {
            Board.topHero.draw();
            Board.botHero.draw();
        }
        if (isServer) {
            activeHero = Board.playerHero;
            activeHero.turn = true;
            inactiveHero = Board.enemyHero;
            inactiveHero.turn = false;
        } else {
            activeHero = Board.enemyHero;
            activeHero.turn = true;
            inactiveHero = Board.playerHero;
            inactiveHero.turn = false;
        }
        activeHero.maxResource = 1;
        activeHero.resource = 1;
        inactiveHero.resource = 1;
        inactiveHero.maxResource = 1;
    }
}

```