

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

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина «Основы алгоритмизации и программирования»

«К защите допустить»  
Руководитель курсового проекта  
ассистент кафедры информатики  
\_\_\_\_\_ М.В. Ганусевич  
\_\_\_\_.\_\_\_\_.2024

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовому проекту  
на тему:  
**«СОЗДАНИЕ ТЕКСТОВОЙ RPG (АНАЛОГ DUNGEONS & DRAGONS)»**

БГУИР КП 6-05 0612 02 12 ПЗ

Выполнил студент группы 353502  
КОЗЕЛЛО Владислав  
Александрович

\_\_\_\_\_  
(подпись студента)

Курсовой проект представлен на  
проверку \_\_\_\_\_.2024

\_\_\_\_\_  
(подпись студента)

Минск 2024

# СОДЕРЖАНИЕ

Введение.....	5
1 Обзор существующих аналогов.....	6
1.1 Обзор жанра RPG и его особенностей.....	6
1.2 Анализ игры “Dungeons & Dragons” .....	8
2 Архитектура игры "Dungeons & Data" .....	11
2.1 Описание общей структуры игры .....	11
2.2 Компоненты, обеспечивающие работоспособность программы .....	11
2.2.1 Модель .....	11
2.2.2 Контроллер .....	13
2.2.3 Представление.....	13
3 Теоретическое обоснование разработки программного продукта.....	15
3.1 Обоснование необходимости разработки.....	15
3.2 Технологии и инструменты .....	16
3.2.1 Язык программирования .....	16
3.2.2 Библиотеки и фреймворки .....	18
4 Проектирование функциональных возможностей программы .....	20
4.1 Общий анализ алгоритмов .....	20
4.2 Алгоритм управления системой боя .....	20
4.3 Алгоритм выборов и последствий.....	22
4.4 Алгоритм взаимодействия с неигровыми персонажами.....	23
4.5 Алгоритм обработки событий .....	25
4.6 Алгоритм прогрессии персонажей .....	26
Заключение .....	27
Список использованных источников .....	28
Приложение А (обязательное) Листинг программного кода .....	29
ПРИЛОЖЕНИЕ Б (обязательное) Функциональная схема программы.....	38
ПРИЛОЖЕНИЕ В (обязательное) Блок-схема алгоритма, реализующего управление системой боя .....	39
ПРИЛОЖЕНИЕ Г (обязательное) Блок-схема алгоритма, реализующего механизм выборов и последствий .....	40
ПРИЛОЖЕНИЕ Д (обязательное) Блок-схема алгоритма, реализующего поиск в глубину .....	41
ПРИЛОЖЕНИЕ Е (обязательное) Блок-схема алгоритма, реализующего обработку событий.....	42
ПРИЛОЖЕНИЕ Ж (обязательное) Блок-схема алгоритма, реализующего прогрессию персонажа .....	43

## ВВЕДЕНИЕ

В современном мире компьютерные игры являются неотъемлемой частью развлекательной культуры, привлекая к себе миллионы людей. В рамках её ролевые игры (RPG) занимают особое место, предлагая уникальный игровой опыт, позволяющий игрокам погрузиться в виртуальные миры и взаимодействовать с ними.

Целью данного курсового проекта является разработка текстовой RPG, основой которой является настольная ролевая игра "Dungeons & Dragons", предоставляющая игрокам возможность исследовать фантастический мир, вступать в бои с монстрами, развивать своего персонажа и вступать в диалоги с неигровыми персонажами. Формат RPG будет текстовый, чтобы уделить особое внимание глубине игрового сюжета, а также стимулированию воображения игроков, как и в оригинальной настольной игре.

Выбор системы "Dungeons & Dragons" в качестве основы обусловлен рядом причин, которые делают её особенно привлекательной для разработки проекта. Наиболее важной особенностью являются правила, которые отличаются гибкостью и расширяемостью, позволяя разработчику адаптировать их под свои нужды и внести собственные изменения, чтобы создать уникальный игровой опыт.

Для достижения поставленной цели предусмотрено следующее:

1 Анализ существующих RPG, изучение правил и механик системы "Dungeons & Dragons".

2 Проектирование игрового мира, определяя основные его характеристики и особенности.

3 Проектирование алгоритмов и механик, реализующих возможности программы.

4 Разработка контента для обеспечения максимального погружения в игровой мир, включая диалоги и различные описания.

5 Тестирование и анализ результатов курсовой работы.

В результате выполнения данного курсового проекта ожидается создание полноценной игровой среды, которая будет в состоянии захватить внимание игроков и предоставить им уникальный опыт.

Пояснительная записка оформлена в соответствии с СТП 01-2017 [1].

# 1 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

## 1.1 Обзор жанра RPG и его особенностей

Ролевая игра (RPG, от английского "Role-Playing Game") – знаменитый жанр компьютерных и настольных игр, где основой игрового процесса является отыгрыш определенной роли. Игрок берет под контроль персонажа, который обладает некоторым набором стандартных характеристик и улучшает их по ходу игры [2].

Многие компьютерные RPG берут своё начало в настольных ролевых играх (НРИ), позаимствовав оттуда во многом терминологию и игровую механику. В НРИ игроки взаимодействуют воображаемыми персонажами в созданном ведущим (мастером игры) мире. Обычно игра проходит за столом, отсюда и название. Мир, в котором происходит игра, история, персонажи и сюжет создаются мастером, который руководит ходом игры и предоставляет игрокам различные сценарии и задания.

Настольные ролевые игры могут быть разнообразными по жанру и тематике. От классических фэнтези и научной фантастики до детективов, приключений в стиле пиратов или пост-апокалипсиса – выбор огромен. Кроме того, некоторые из них предлагают различные подходы к игровому процессу, например, более сфокусированные на отыгрыше роли или на тактических сражениях.

Основой НРИ являются правила, которые определяют, как происходит взаимодействие игроков с окружающим миром, как решаются конфликты, и как развиваются персонажи. Правила могут быть простыми или сложными, зависит от конкретной игры, но в любом случае, важнее всего – это воображение игроков [3].

Компьютерные ролевые игры – это те же настольные, только роль мастера игры выполняет аппаратное обеспечение вашего компьютера, а воображение игроков чаще всего заменяется на графику. Одни из первых ролевых видеоигр появились в середине 1970-х и представляли из себя текстовые RPG, то есть вся информация, что получал игрок, была в виде текста. Однако несмотря на столь давний срок появления, они актуальны и по сей день. В настоящее время не составляет труда найти текстовые игры, украшенные привлекательными статичными или анимированными изображениями (рисунок 1.1) [4].

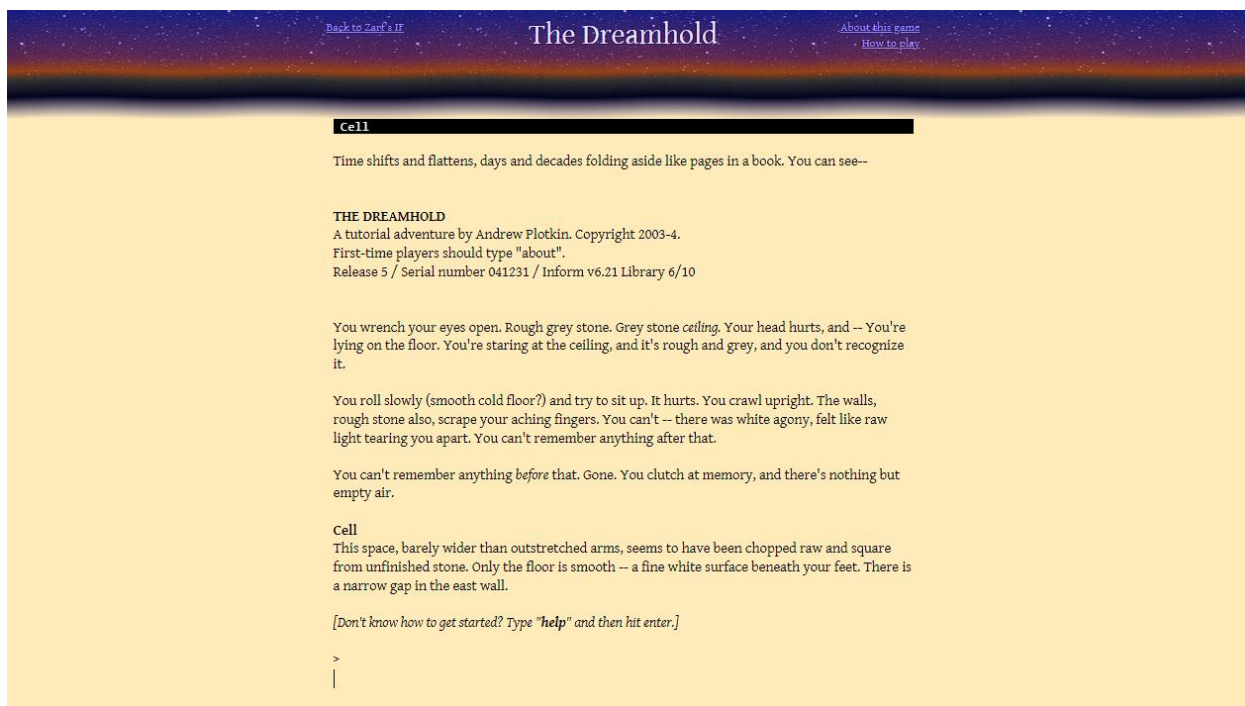


Рисунок 1.1 – Пример текстовой игры (The Dreamhold)

На протяжении всей истории компьютерных ролевых игр возникло достаточно много различных новых поджанров, которые обладают своими особенностями:

1 Настоящая РПГ (CRPG – Computer Role-Playing Game) – это игра с достаточно глубокими диалогами, с тщательно проработанной системой ролей и игровым миром, свободой действий, а также захватывающим сюжетом.

2 Action RPG (Экшен РПГ) – это игра, где основной игрового процесса выступает не ролевая составляющая, а боевая. Она сложнее, чем в простых RPG, так как чаще всего требует от игрока хорошей реакции или владение определенной боевой системой.

3 Тактические РПГ – RPG, разбавленное стратегией, где игрок контролирует группу (отряд). Развитие всех членов отряда осуществляется по классической для данного жанра схеме.

4 ММОРПГ (Массовая многопользовательская ролевая онлайн-игра) – как и в большинстве ролевых игр, игрок выбирает персонажа, который развивается, выполняя задания и продвигаясь по сюжету. В отличие от одиночной, в ММОРПГ играет очень много игроков одновременно.

5 Roguelike – это популярных поджанр ролевых игр, основой которого является случайная генерация игрового мира, а также необратимая смерть персонажа.

Жанр компьютерных ролевых игр сильно укрепился в современном

обществе: каждый год выходят десятки различных игр (Baldur's Gate 3, Elden Ring, Starfield и др.), которые вносят новые уникальные механики, графику и атмосферу.

## 1.2 Анализ игры “Dungeons & Dragons”

"Подземелья и Драконы" (Dungeons & Dragons, D&D) – первая коммерческая настольная ролевая игра, созданная Гэри Гигаксом и Дэйвом Арнесоном в США в 1974 году, и до сих пор являющаяся самой лучшей в своей категории. Почти все современные электронные игры чем-либо обязаны ей, в особенности игры жанра RPG. К примеру, "Baldur's Gate" – это компьютерная RPG, мотивом которой стала одноименная книга, написанная для проведения кампании в "Подземелья и Драконы" (рисунок 1.2).

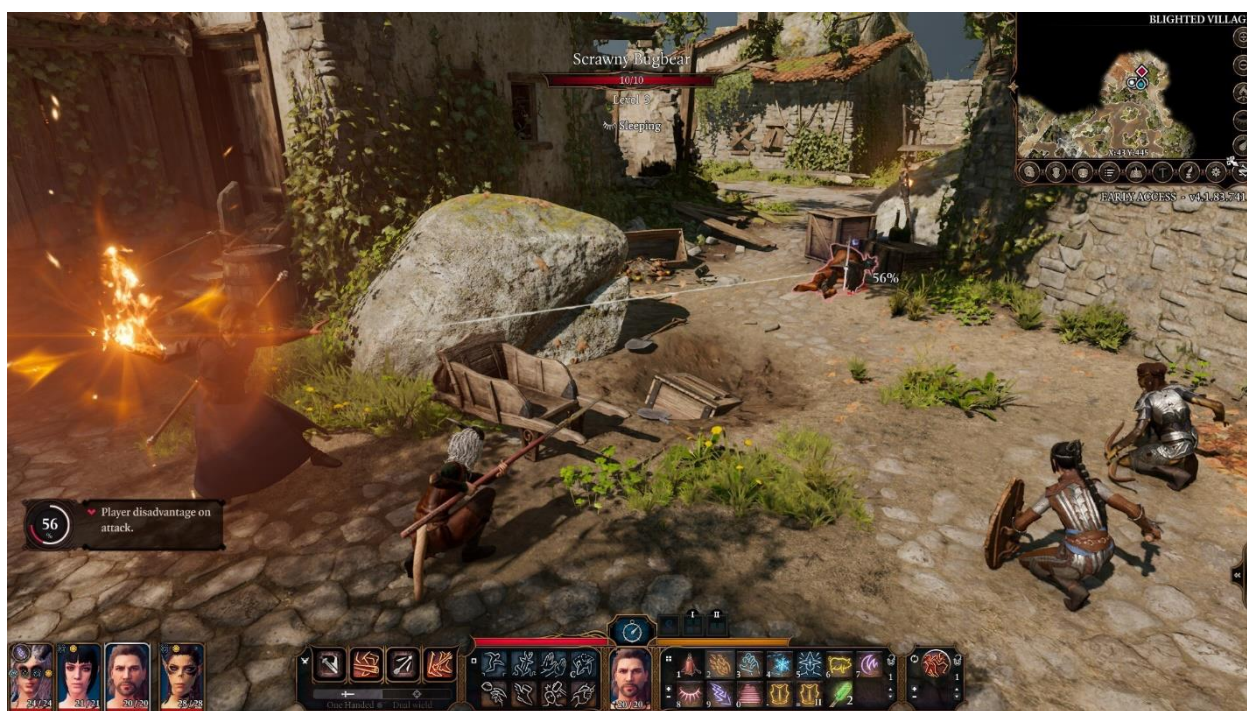


Рисунок 1.2 – Скриншот из компьютерной игры “Baldur's Gate 3”

D&D основана на правилах, которые обеспечивают рамки для взаимодействия игроков с окружающим миром и друг с другом, но это не значит, что нужно максимально им следовать. Система "Подземелья и драконы" отличается от других игр подобного жанра своей гибкостью: каждый игрок вправе дополнить или урезать официальные правила (рисунок 1.3) под свои конкретные нужды.





Рисунок 1.3 – Официальный трехтомник правил игры “Dungeons & Dragons”

Общая концепция представляет собой открытость действий игроков и влияние их последствий на вымышленный фэнтези мир, которым управляет мастер игры. "Подземелья" несут в себе свободу действий, отыгрыш роли и полёт фантазии от детских игр в воображаемых героев и импровизационного театра. Но в то же время D&D подобно компьютерным ролевым играм структурирует происходящее правилами для определения результатов действий.

Человек, играющий партию в D&D может быть гейм-мастером или обычным игроком. Гейм-мастер ведёт приключение: подготавливает игровой мир, определяет его законы и что может произойти в сценарии. Остальные члены группы выбирают себе персонажей и по ходу приключения исследуют мир, борются с монстрами, решают загадки и общаются с другими персонажами. Мастер решает, что происходит в игровом мире в ответ на их действия, а также отыгрывает всех остальных обитателей вселенной. Он действует как арбитр для правил и направляет историю, но сюжет развивается совместными усилиями гейм-мастера и игроков.

Первоначально игроку нужно создать персонажа, которого он будет отыгрывать. Создание заключается в заполнении специального листа, где будут описаны: класс, раса, характеристики, очки здоровья, способности, умения, инвентарь, мотивация, цель, слабости и др. В процессе игры лист будет дополняться новыми сведениями.

Вся механика D&D строится на случайных исходах бросков игровых костей. В рамках игры существует 6 их разновидностей: к4, к6, к8, к10, к12 и

к20. Каждая кость выполняет свою конкретную роль:

1 К20 – основная игральная кость, которая определяет исход заявок на действия от игроков, а также исход попадания атакой во врага. Он имеет 20 граней, на которых изображены числа от 1 до 20.

2 К12 – кость этого типа используется в основном для определения урона от оружия или навыков соответствующих классов персонажей. На её гранях находятся числа от 1 до 12.

3 К10 – эта кость позволяет определить результаты, подразумевающие десятичную систему, такие как определение критических попаданий при атаке или урон при накладывании заклинания. Он содержит числа от 0 до 9, которые могут быть умножены на 10 в случае броска двух таких костей, чтобы имитировать бросок к100 (100-гранной кости).

4 К8 – используется для определения урона от определенных навыков и заклинаний, а также для урона от сильного оружия. Кость имеет числа в диапазоне от 1 до 8.

5 К6 – шестигранные кубики используются в D&D наиболее часто. Они участвуют в определении урона от оружия, навыков и заклинаний, а также для определения некоторых результатов и случайных событий. На гранях кубика находятся числа от 1 до 6.

6 К4 – Четырехгранная кость используется в основном для определения незначительных результатов или мелких действий, таких как урон от импровизированного оружия. Кость содержит числа от 1 до 4.

Сражения происходят по правилу "step by step" (шаг за шагом). Это означает, что персонажи ходят по очереди, начиная с того, у кого выпало большее значение на к20. За ход игрок может сделать какое-либо основное действие, бонусное действие, перемещение и малое взаимодействие. Ходы персонажей объединяются в раунды, которые длятся 6 секунд игрового времени. Мастер вправе самостоятельно определить, какая заявка от игрока относится к определенному действию.

Как правило, сражения происходят на специальных боевых полях, но какие-то маленькие стычки гейм-мастер может отыграть и как обычные сцены, где нет четкой последовательности ходов.

После заявки на действие от игрока мастер устанавливает сложность броска двадцатигранника (значение от 1 до 20), и если игрок выбросил значение больше заявленного или равное ему, то действие считается успешным [5].



## **2 АРХИТЕКТУРА ИГРЫ "DUNGEONS & DATA"**

### **2.1 Описание общей структуры игры**

"Dungeons & Data" – это текстовая компьютерная ролевая игра, разработанная в качестве данного курсового проекта. Она представляет собой упрощенный аналог описанной выше настольной ролевой игры "Dungeons & Dragons" с сохранением основных её механик, интерпретированных под одного игрока.

Структура приложения имеет вид, типичный для данного жанра компьютерных игр, а именно MVC. MVC (Model-View-Controller) – это архитектурный шаблон, который структурирует приложение вокруг трех основных компонентов: модель, представление и контроллер. Каждый из компонентов предоставляет свою функциональность:

1 Модель отвечает за управление данными и логикой приложения. Она не зависит от других компонентов и обеспечивает независимый доступ к данным.

2 Представление отображает данные модели пользователю и обрабатывает пользовательский ввод. Этот компонент получает данные от модели и передает их пользователю в удобном виде, а также передает пользовательский ввод контроллеру.

3 Контроллер принимает пользовательский ввод из представления, интерпретирует его и обновляет модель или представление в соответствии с этим вводом. Контроллер является посредником между представлением и моделью, обеспечивая их взаимодействие.

MVC обеспечивает модульность и независимость компонентов, позволяя изменять каждый из них отдельно без влияния на остальные, что упрощает разработку, тестирование и поддержку приложения.

### **2.2 Компоненты, обеспечивающие работоспособность программы**

#### **2.2.1 Модель**

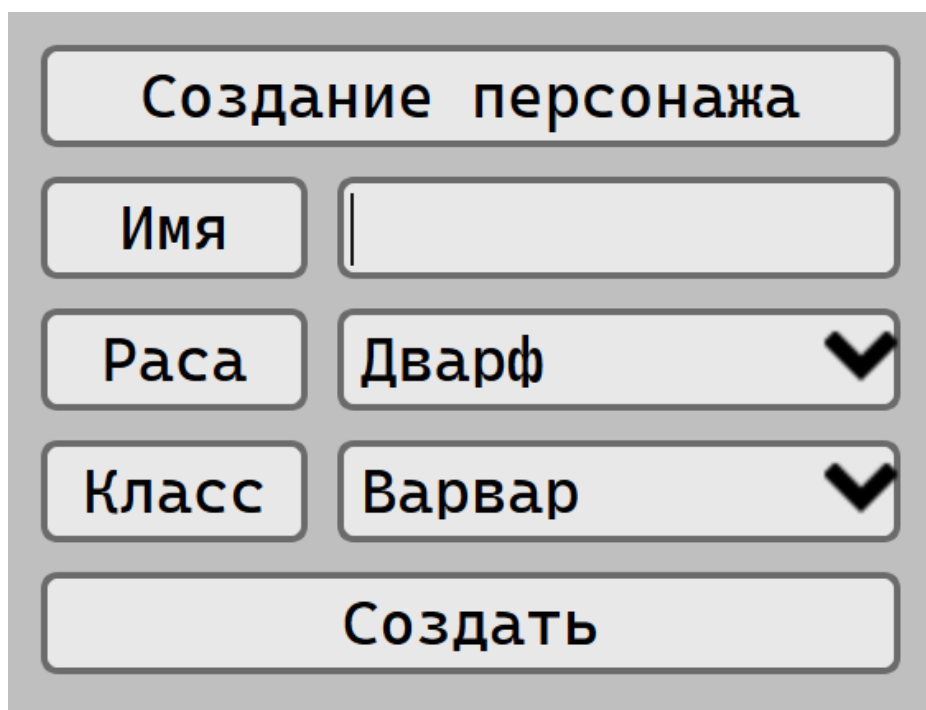
В качестве модели в игре "Dungeons & Data" предусмотрены несколько компонент, отвечающие за свой уникальный функционал:

- персонаж;
- неигровые персонажи;
- диалоги;
- предметы;

– игральные кости.

Персонаж представляет из себя набор данных, являющиеся его характеристиками, а также методами для их управления и изменения. Он является главной моделью, так как взаимодействие со всеми остальными компонентами приложения осуществляется именно через него.

Таким образом, игроки начинают свое знакомство с "Dungeons & Data" именно через форму создания своего персонажа (рисунок 2.1). Пользователь выбирает имя, расу и класс своего героя, тем самым неявно определяя его характеристики. Класс можно выбрать один из следующих: варвар, бард, жрец, друид, воин, монах, паладин, следопыт, плут, чародей, колдун, волшебник. Что касается расы, игроку дается на выбор: дварф, эльф, полурослик, человек, драконорожденный, гном, полуэльф, полуорк, тифлинг.



The image shows a user interface for creating a character. It has a title bar 'Создание персонажа'. Below it are three rows of controls: 'Имя' with a text input field, 'Раса' with a dropdown menu showing 'Дварф', and 'Класс' with a dropdown menu showing 'Варвар'. At the bottom is a large button labeled 'Создать'.

Рисунок 2.1 – Форма создания персонажа

Модель "Персонаж" обеспечивает весь необходимый функционал для взаимодействия с ним, предлагая все необходимые методы.

Неигровые персонажи (НПС) представляют из себя модели, имеющие практически такой же набор характеристик, как и персонаж, созданный пользователем. Взаимодействие с ними не является прямым, игрок общается с текстовой моделью, представляющей из себя дерево диалога, которое имитирует разговор с вариантами выбора ответа. При решении напасть на НПС в бой вступает уже сама модель неигрового персонажа со своими

характеристиками.

Предметы – это модели вещей, с которыми может взаимодействовать персонаж. Они делятся на 4 вида: безделушки, броня, оружие и зелья. Каждый обеспечивает свой функционал, необходимый для игры.

Игральные кости в качестве модели имитируют бросок настоящих игральных костей, тем самым дают возможность случайного определения исхода действия игрока. Используется в сражениях и взаимодействии с НПС.

### **2.2.2 Контроллер**

В роли контроллера в данном проекте выступают 2 компонента:

- игровой мир;
- боевая система.

Игровой мир – это главный контроллер в игре, который обеспечивает взаимодействие остальных компонент программы с пользовательским интерфейсом. Именно в нем обрабатываются все команды игрока, обновляются модели и вызываются необходимые методы. Весь сюжет игры, взаимодействия, генерация случайных событий, а также боевая система, хранятся в это контроллере.

Боевая система – компонент программы, предоставляющий игрокам возможность вступать в битву с НПС. Битва представляет из себя последовательность раундов, состоящих из ходов игрока и соперника. Ход – это бросок кости на попадание и на урон, который будет нанесен оппоненту. В случае, когда у кого-либо из сражающихся существ не останется здоровья, он проигрывает.

### **2.2.3 Представление**

Представление в проекте – это пользовательский интерфейс. Он неотъемлемая часть любой компьютерной игры. Он позволяет пользователю взаимодействовать с компонентами приложения, а также получать эстетическое удовольствие от его реализации. Интерфейс игры "Dungeons & Data" включает в себя различные элементы взаимодействия с программой, оформленные в едином стиле (рисунок 2.2). Среди них можно выделить: меню, кнопки, ярлыки, выпадающие списки, текстовые поля и таблицы. Данные элементы сгруппированы между собой и разбиты на своеобразные блоки, выполняющие определенные функции:

1 Меню. Имеет три кнопки, нажав на которые пользователь может выйти из игры, начать новую игру или же продолжить старую.

2 Характеристики персонажа. Включает в себя всю необходимую для

игры информацию о персонаже, обновляющуюся автоматически по мере необходимости.

3 Инвентарь. Имеет вид таблицы, где предоставлена информация об вещах, которыми персонаж обладает на данный момент. Также имеется кнопка, нажав на которую можно использовать выбранный предмет.

4 Диалог. Представляет из себя окно, в которое выводится сюжетный текст, а также описываются действия, совершаемые персонажем или НПС. Имеется строка ввода, предназначенная для ввода команды или текста пользователя, и кнопка отправки текста. Является главным источником информации для игрока.

5 Игровая кость. Данный блок состоит из окна вывода графики (имитация вращения кубика), кнопки "Вращать" и выпадающего списка, в котором можно выбрать необходимую кость.

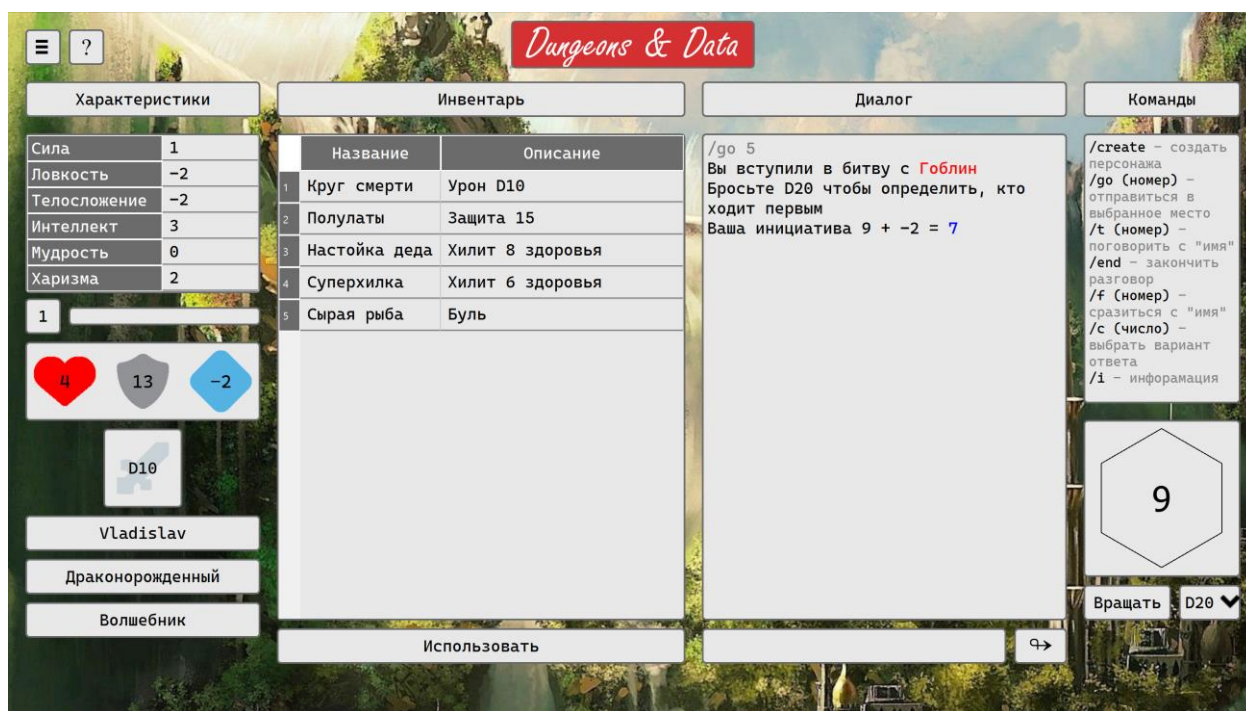


Рисунок 2.2 – Пользовательский интерфейс игры

Интерфейс разработан с учетом пользовательского удобства использования, придавая особое внимание созданию интуитивно понятной игровой среды. Предусмотрена кнопка "?", которая выполняет роль подсказки: выводит список всех команд, тем самым помогая пользователю, если тот что-либо забыл.

## **3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**

### **3.1 Обоснование необходимости разработки**

В наше время индустрия компьютерных игр развивается быстро и постоянно. Новые технологии и платформы приносят удивительные возможности, которые привлекают внимание как взрослых, так и детей, и обеспечивают захватывающий игровой опыт. Это стимулирует разработчиков следовать трендам и постоянно выпускать новые продукты, которые заслужат одобрение общества.

В свою очередь, создание текстовой RPG это явный способ подчеркнуть достоинство жанров компьютерных игр прошлого, которые отличались своей простотой визуализации и своеобразной механикой. А адаптация системы НРИ "Dungeons & Dragons" для одного игрока даст возможность погрузиться в игровой прогресс без вовлечения других людей.

Разработка игры – это креативный и трудоемкий процесс, который требует значительных усилий и ресурсов. В рамках курсового проекта, написание такой программы имеет множество практических преимуществ.

Были выделены следующие выгоды разработки программного продукта:

1 Проектирование игры позволяет применить знания, полученные во время учебы, в реальном проекте. Наряду с проектированием логики программы разработчик создает дизайн игрового процесса, что развивает его креативные способности.

2 Разработка игры является отличным способом улучшить навыки программирования. Изучая новые стези, можно попрактиковаться в разработке сложных программных проектов, требующими более расширенной базы знаний.

3 Разработчики могут ознакомиться с основами управления проектом, такими как планирование, управление ресурсами и сроками, а также контроль качества. Данные навыки очень востребованы в современном мире.

4 Текстовая RPG позволит пользователям получить незабываемый игровой опыт, расслабиться после важных дел, погрузившись в мир компьютерных ролевых игр.

В целом, разработка игры в качестве курсового проекта способствует повышению мотивации и интереса к учебе, поскольку она представляет собой захватывающий проект, который может вдохновить человека к творчеству и самореализации.

## 3.2 Технологии и инструменты

### 3.2.1 Язык программирования

Язык программирования, на котором будет реализована система, заслуживает большого внимания, так как погружение в него будет происходить с начала конструирования программы и до самого конца. Поэтому нужно максимально ответственно подойти к его выбору.

Если язык хорошо знаком программистам, то работа выполняется более производительно. Данные, полученные при помощи модели оценки Сосото II, показывают, что программисты, которые долго работают с определенным языком, примерно на 30% более продуктивны, чем программисты, обладающие аналогичным опытом, но на другом языке [6].

Отталкиваясь от этой информации и личных предпочтений, был выбран язык C++ (рисунок 3.1). C++ – компилируемый, статически и нестрого типизированный, высокоуровневый язык программирования. Рассмотрим подробнее каждую его характеристику:

1 Компилируемый язык программирования – язык программирования, исходный код которого преобразуется компилятором в машинный код. В результате компиляции получается исполняемый файл, который выполняется операционной системой.

2 Статическая типизация указывает на то, что тип данных привязывается к объекту при его создании и не может быть изменен после. Это снижает риск ошибки программиста и улучшает читаемость кода.

3 Нестрогая типизация допускает приведение одних типов данных к другим.

4 Высокоуровневый язык программирования – это язык программирования, который обладает высоким уровнем абстракции, что позволяет создавать компьютерные программы более быстро и просто. В таких языках используются конструкции, которые ближе к естественному языку человека, для описания структур данных и операций над ними.

Современный C++ может многое предложить. Многие современные функциональные возможности языка имеют практическую ценность. Они обеспечивают повышенную производительность и надежность, их легко понять и применить, трудно случайно использовать не по назначению. Кроме того, современный C++ предлагает богатый набор инструментов для разработки высокоэффективных приложений. Новые возможности языка, такие как семантика перемещения, умные указатели, асинхронное программирование, позволяют создавать более эффективный и управляемый код [7].



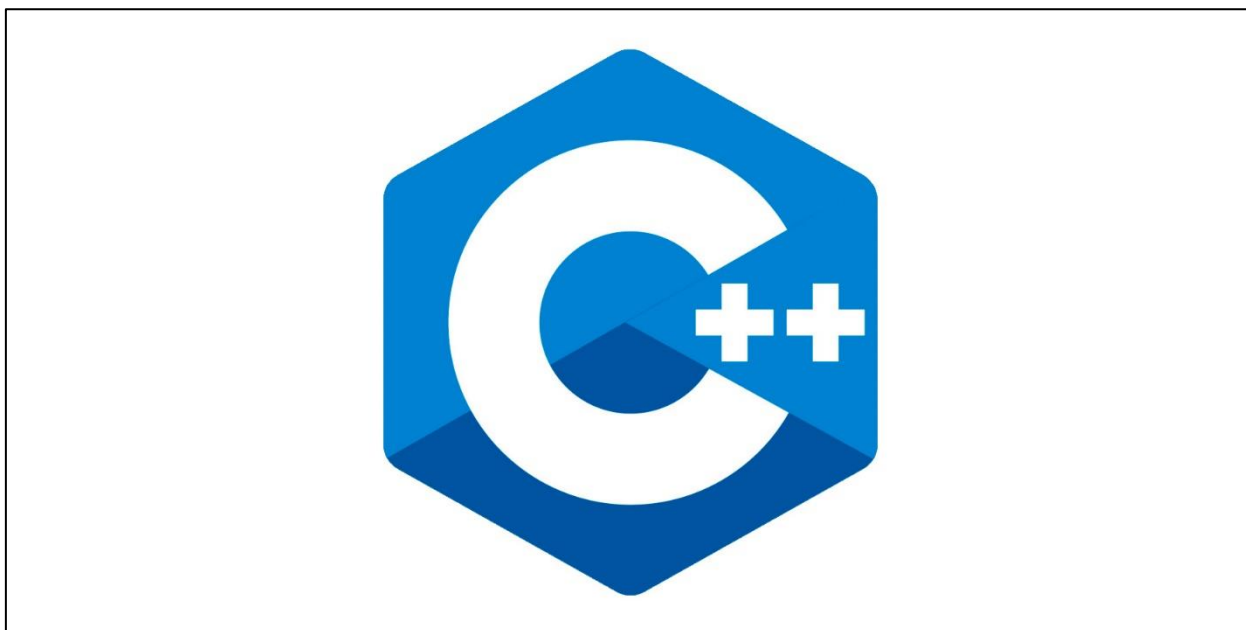


Рисунок 3.1 – Язык программирования C++

Важной особенностью C++ в сравнении с другими языками программирования является возможность напрямую работать с памятью, что позволяет оптимизировать использование ресурсов и уменьшить накладные расходы, связанные с автоматическим управлением памятью, таким как сборка мусора. Это приводит к улучшению производительности приложения и увеличению скорости выполнения программы.

C++ вкупе со всеми его функциональными возможностями является наилучшим выбором языка программирования в рамках данного курсового проекта. Рассмотрим основные причины, почему это так:

1 C++ является высокопроизводительным, следовательно, он позволяет создавать быстрые и эффективные приложения. В игровой разработке производительность играет ключевую роль, поскольку игры должны быть отзывчивыми.

2 C++ один из наиболее популярных и широко используемых языков программирования в игровой индустрии. Это означает, что существует обширное сообщество разработчиков, фреймворков и ресурсов, которые могут помочь в написании проекта.

3 C++ подходит для разработки крупных проектов благодаря своей модульности, возможности организации кода в библиотеки и поддержке пространств имен.

### 3.2.2 Библиотеки и фреймворки

Фреймворк – заготовка, готовая модель в программировании для быстрой разработки, на основе которой можно написать собственный код. Он задает структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта [8].

Для реализации игры "Dungeons & Data" был выбран фреймворк Qt версии 6.6.1. Он позволит облегчить работу с графикой и пользовательским интерфейсом, а также привнесет в проект свои особенности.

Qt – это кроссплатформенный фреймворк для разработки программного обеспечения с открытым исходным кодом, который широко используется для создания графических интерфейсов (GUI), приложений сетевого взаимодействия, игр и других программных продуктов (рисунок 3.2).



Рисунок 3.2 – Фреймворк Qt

Однако не следует принимать Qt как фреймворк только для создания пользовательского интерфейса, так как в отличие от популярной библиотеки SFML он предоставляет большое количество новых возможностей для разработчика, не касающихся интерфейса. Можно выделить следующие особенности Qt:

1 Одним из главных преимуществ Qt является его кроссплатформенность. С помощью этого фреймворка можно создавать приложения,

которые могут работать на различных операционных системах, таких как Windows, macOS, Linux, а также на мобильных устройствах под управлением Android и iOS.

2 Qt предоставляет мощные инструменты для создания графических пользовательских интерфейсов. Это включает в себя широкий набор виджетов, стилей и всевозможных средств для их проектирования.

3 Данный фреймворк построен на модульной архитектуре, что позволяет разработчику использовать только те компоненты, которые необходимы для конкретного проекта. Такой подход упрощает разработку и уменьшает размер итогового приложения [9].

Важно упомянуть, что в Qt существует своя иерархия классов, которая обеспечивает удобство и гибкость в разработке приложений. Иерархия базируется на концепции объектно-ориентированного программирования и включает в себя множество базовых классов, от которых наследуются более специфические классы для реализации конкретного функционала.

Одним из ключевых элементов этой иерархии является класс QObject, от которого наследуются практически все классы в Qt (рисунок 3.3). QObject предоставляет базовую функциональность для управления объектами в Qt, такую как система событий, сигналы и слоты, управление памятью и т.д.

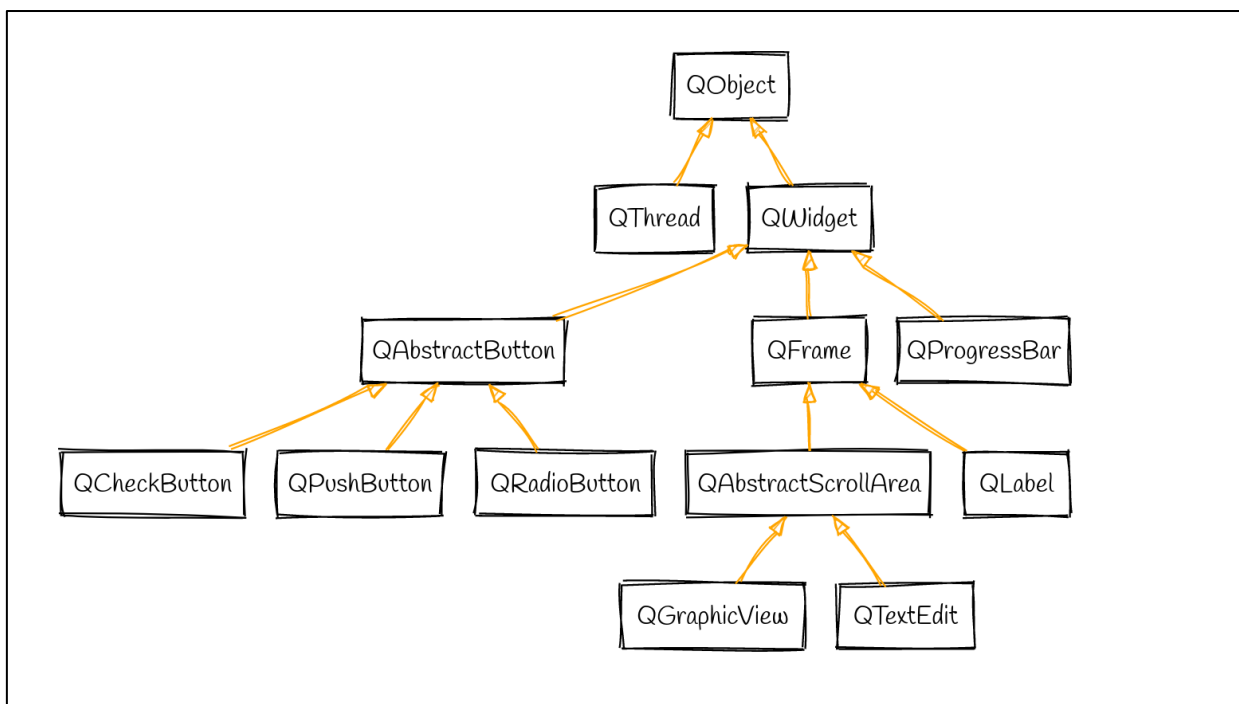


Рисунок 3.3 – Диаграмма наследования некоторых классов в Qt

## **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ**

### **4.1 Общий анализ алгоритмов**

В современных реалиях пользователям требуется достаточно качественные приложения с максимально продуманной структурой и функциональными возможностями, поэтому разработчику нужно тщательно подойти к выбору алгоритмов, которые будут обеспечивать данный функционал.

Игровой движок "Dungeons & Data" должен обеспечивать выполнение основных потребностей текстовой RPG игры (приложение Б). В данном проекте выделяются следующие из них:

- обработка пользовательского ввода;
- боевая механика;
- механика диалогов;
- бросок кубика;
- управление инвентарём;
- управление персонажем.

Для реализации этих потребностей были спроектированы 5 ключевых алгоритмов, описывающих основной функционал приложения:

- алгоритм управления системой боя;
- алгоритм обработки событий;
- алгоритм выборов и последствий;
- алгоритм прогрессии персонажей;
- алгоритм взаимодействия с неигровыми персонажами.

Для более читаемого кода был разработан файл, хранящий пользовательские константы, используемые далее в коде (листинг А.1). В добавок к ним спроектированы несколько алгоритмов менее сложной структуры, однако так же необходимых, как и основные: алгоритм случайного броска кубика (листинг А.2), парсер JSON объектов (листинг А.3), алгоритм использования предметов и др.

Далее будут более подробно рассмотрены основные алгоритмы, перечисленные выше [10].

### **4.2 Алгоритм управления системой боя**

Боевая система – неотъемлемая часть любой RPG игры, поэтому ей было

уделено достаточно много внимания (приложение В). Она должна обеспечивать сражение между персонажем и какой-либо игровой сущностью.

Алгоритм управления системой боя находится внутри класса, который описывает все необходимые компоненты для его осуществления. Среди этих компонент можно выделить:

- указатели на персонажа игрока и на NPC;
- флаг состояния хода и флаг состояния первого хода;
- переменная, хранящая фазу боя ("бросок инициативы", "бросок попадания", "бросок урона" и "нет броска");
- переменные, хранящие последнее значение броска игрока и броска врага, а также тип кости, которую нужно бросить;
- функция начала сражения, которая запускает основной алгоритм;

Сам алгоритм находится внутри функции, которая является Qt слотом, принимающим от интерфейса значение броска кубика и его тип для последующей обработки. Обработка заключается в проверке состояния боя, описываемого переменными, приведенными выше.

Первое условие проверяет, какая фаза боя в данный момент. Если фаза – "бросок инициативы" (самая первая фаза), то алгоритм дает право сделать первый бросок игроку, а затем врагу. После обоих бросков и их сравнения алгоритм вычисляет, кто первым бросает кубик на попадание и ставит соответствующий флаг, а также ставит следующую фазу – бросок на попадание.

Затем тот, у кого больше инициатива, бросает кость на попадание. В случае попадания (сумма броска кости и модификаторов больше, чем защита соперника) ставится фаза "бросок урона", и тот же самый участник боя бросает кубик на урон, который наносится оппоненту. Если урон, нанесенный оппоненту, больше его оставшегося здоровья, то данный участник проигрывает сражение. В противном случае, ход переходит следующему сражающемуся, и фаза меняется на "бросок на попадание".

Если игрок выигрывает сражение, то ему в инвентарь дается случайный предмет и начисляется опыт, зависящий от сложности оппонента.

В данном алгоритме предусмотрена проверка на ошибку игрока: бросок неверного кубика. В таком случае на экран выводится сообщение о том, какую кость бросил пользователь и какую следует. После этого игрок может спокойно ещё раз перебросить кубик. Весь описанный выше функционал сопровождается выводом информации пользователю для наглядного отображения процесса боя (листинг А.4).

### 4.3 Алгоритм выборов и последствий

Алгоритм выборов и последствий предлагает удобный инструмент для обработки пользовательского ввода и обновления статуса игры (приложение Г). Он представляется из себя логику своеобразной командной строки, анализирующей ввод игрока и запускающий какие-либо другие алгоритмы или функции.

Данный алгоритм находится внутри основного класса игры в функции, являющейся Qt слотом, принимающим строку от пользователя. В случае неверного ввода, выводится соответствующее сообщение о том, что команда не найдена. Всего существует 7 команд:

1 `"/create"` – команда для создания персонажа. Запускает специальное меню, в котором можно указать нужные характеристики.

2 `"/i"` – команда, выводящая текущее состояние игры. Если персонаж находится в диалоге, то выводятся текст и возможные варианты, если же нет, то выводится текущее место пребывания, его описание, списки NPC и мест, в которые можно пойти.

3 `"/go номер места"` – команда для смены локации. Вместо "номер места" следует написать номер из списка мест, предоставленных локацией, в которой находится персонаж.

4 `"/f номер NPC"` – команда для запуска сражения с выбранным неигровым персонажем. Если игрок находится в диалоге с кем-то, то данную команду следует прописывать без "номер NPC", чтобы начать сражение с этим NPC.

5 `"/t номер NPC"` – команда для начала диалога с выбранным NPC.

6 `"/с номер варианта"` – команда для выбора определенного варианта ответа неигровому персонажу.

7 `"/end"` – команда, позволяющая закончить разговор с неигровым персонажем.

Также, в зависимости от состояния игры, некоторые команды могут быть недействительными. Например, в состоянии "создание" нельзя прописать никакую команду кроме `"/create"`, а в состоянии "диалог" нельзя выполнить команду `"/go"` (листинг А.5).

Введен запрет на написание команд в момент сражения и обработки событий.



## 4.4 Алгоритм взаимодействия с неигровыми персонажами

Алгоритм взаимодействия с неигровыми персонажами представляет собой поиск возможных путей в графе (приложение Д), где узлы графа являются узлами диалога, а рёбра – возможными переходами.

Он состоит из двух функций:

1 Рекурсивная функция, которая исследует всевозможные пути, начиная с заданного узла (листинг А.6).

2 Функция запуска предыдущей рекурсивной функции. Она очищает список возможных переходов и вызывает рекурсию от корневого узла дерева диалогов (листинг А.7).

Список возможных переходов выводится сразу после выбора следующего узла. Каждый узел диалога – это структура, имеющая несколько полей:

- id узла;
- текст;
- опция перехода;
- список из id следующих узлов;
- флаг, отвечающий за то, был ли игрок в данном узле или нет;
- номер события, которое выполняется после перехода на данный узел.

Давайте подробнее рассмотрим поиск в глубину для данного графа (рисунок 4.1).

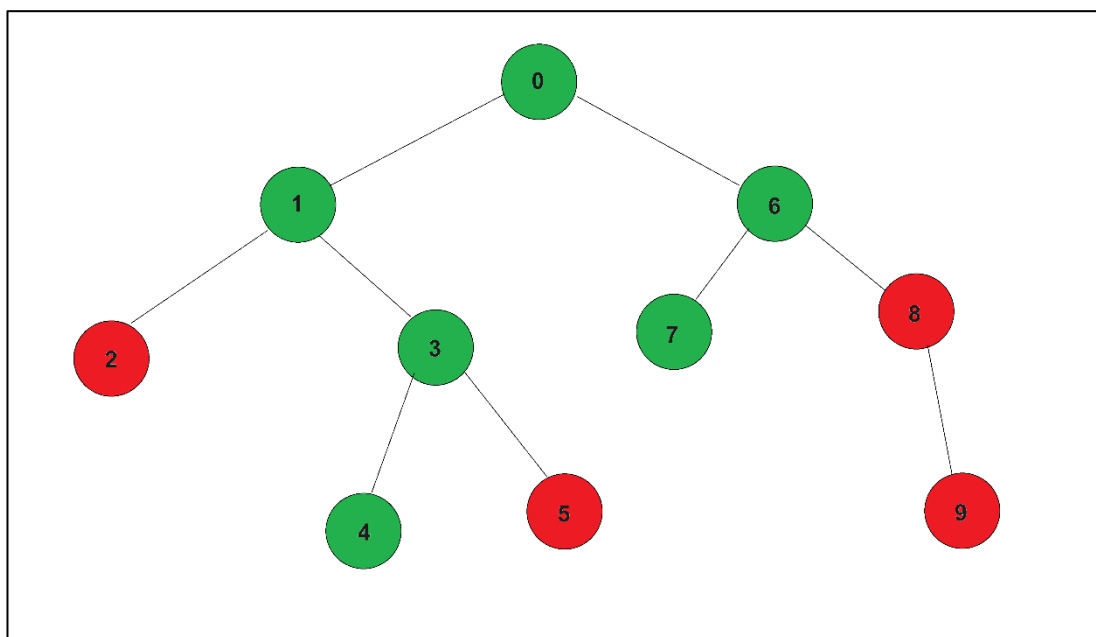


Рисунок 4.1 – Исходный граф

Рекурсивная функция проходит по всем опциям в текущем узле диалога. Если опция уже была пройдена или является пустой (зеленый цвет), то она игнорируется (рисунок 4.2).

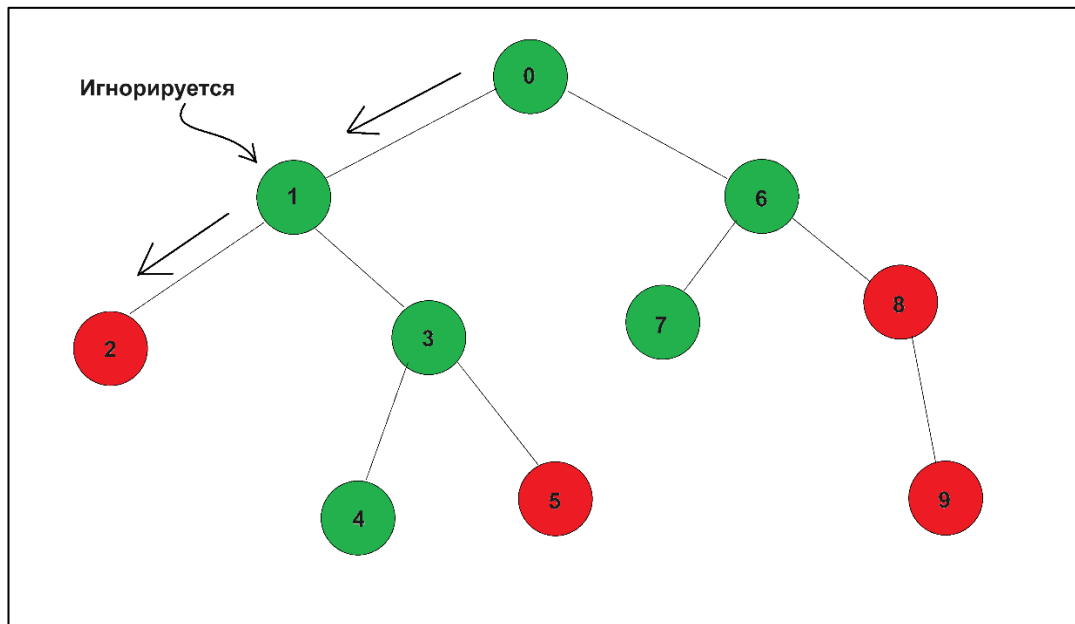


Рисунок 4.2 – Игнорирование вершины

В противном случае (красный цвет) проверяется, содержится ли данный узел в списке возможных переходов, если нет, то он добавляется, и данная рекурсивная ветка заканчивается (рисунок 4.3).

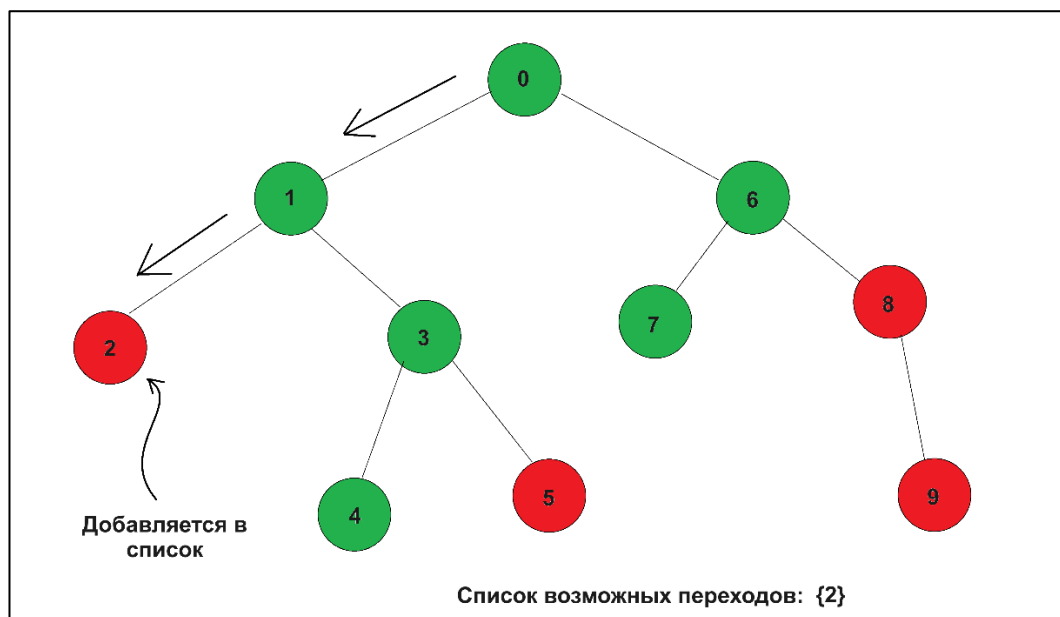


Рисунок 4.3 – Добавление узла в список

Затем функция рекурсивно вызывается для каждой опции, которая ещё не была пройдена, чтобы исследовать пути дальше (рисунок 4.4).

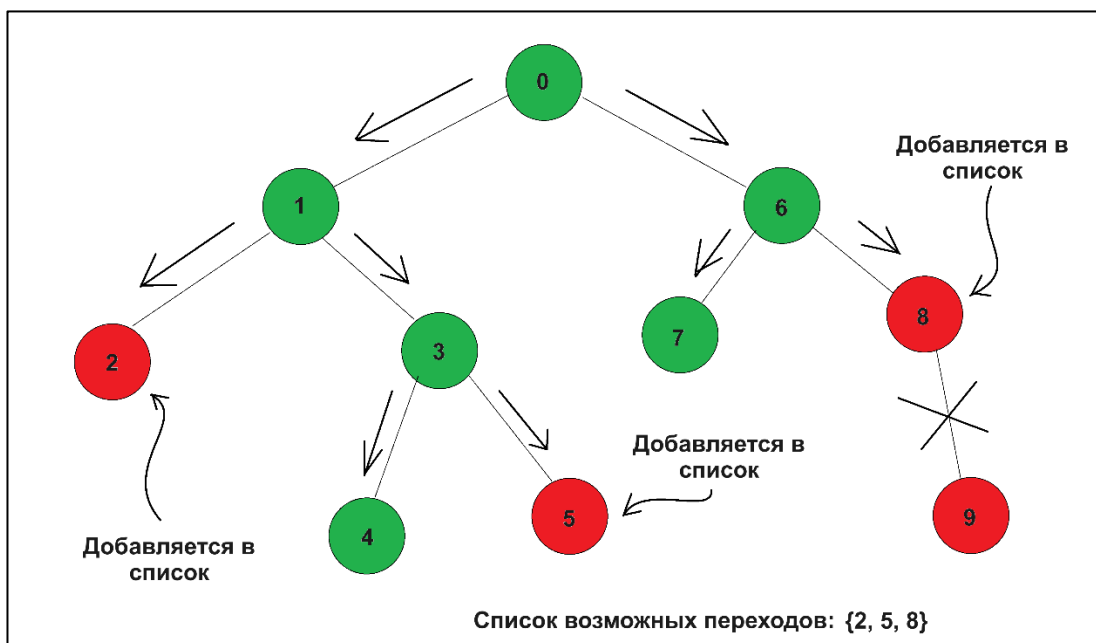


Рисунок 4.4 – Конечный вариант обхода

В конечном итоге мы имеем полный список вершин, в которых не был персонаж. Этот список затем выводится и используется для проверки ввода.

#### 4.5 Алгоритм обработки событий

Алгоритм обработки событий представляет из себя функцию, в которую передают параметр, отвечающий за событие (приложение E). В зависимости от типа события, выполняются различные действия:

1 Если событие – сражение (FIGHT\_EVENT), игра переходит в режим битвы и запускается функция боя с НПС.

2 Если событие – получение награды (PRIZE\_EVENT), выбирается случайный предмет из списка предметов. Если игрок находится в режиме диалога, предмет добавляется в инвентарь игрока, и выводится информация о полученном предмете и опыте. Затем ищутся возможные пути диалога, и, если их нет, заканчивается диалог. В противном случае выводятся доступные варианты ответа. Вне режима диалога персонажу просто добавляется предмет (листинг A.8).

3 Если событие – завершение диалога (END\_EVENT), игра переходит в свободный режим, и диалог завершается.

4 Если событие – проверка характеристики (STRENGTH\_EVENT, DEXTERITY\_EVENT, и т.д.), выводится сообщение с запросом броска кубика соответствующего типа и ставятся нужные флаги, для последующего принятия этого значения.

5 Если событие – попадание в ловушку (TRAP\_EVENT), выводится сообщение о попадании в ловушку и запрашивается бросок кубика на спасение, в зависимости от результата выводится соответствующее сообщение.

Все события кроме ловушки могут быть вызваны внутри диалога. Попадание в ловушку или начало сражения возникают после перехода в некоторые локации.

#### **4.6 Алгоритм прогрессии персонажей**

Алгоритм прогрессии персонажей состоит из двух частей: создание персонажа и развитие персонажа.

Создание происходит с помощью специальной формы, где игрок может выбрать имя, класс и расу для своего персонажа. После отправки формы данные, введенные пользователем, обрабатываются. В зависимости от того, какой класс выбрал игрок, персонажу дается определенное снаряжение и выставляются некоторые характеристики, раса же дает бонус к определенным характеристикам. Все остальные характеристики выбираются случайным образом (листинг А.9).

Алгоритм развития персонажа заключается в выставлении линии прокачки навыков и здоровья. При создании персонажа ему автоматически выставляется количество опыта необходимое для достижения определенного уровня, растущее в алгебраической прогрессии от уровня к уровню. Когда персонаж набирает достаточное количество опыта для повышения уровня, его здоровье повышается на значение, выпавшее на специальной кости (листинг А.10), а также всплывает окно, в котором пользователь может выбрать характеристику, которую хочет повысить на единицу (приложение Ж).

Опыт начисляется за победу над вражескими сущностями (сущность признается вражеской, если уровень опасности больше 0) и за общение с НПС.

## ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была разработана текстовая RPG, основанная на популярной настольной ролевой игре Dungeons & Dragons. В процессе создания были основаны навыки написания крупных приложений с графическим интерфейсом на языке C++ и фреймворком Qt. Все цели, поставленные вначале, были выполнены.

Также был изучен архитектурный шаблон MVC, благодаря которому приложение получилось связным и одновременно открытым для расширения.

Проект представляет собой успешное сочетание технических навыков, творческого подхода и внимания к деталям. Разработанная игра предоставляет увлекательный и захватывающий игровой опыт для пользователей, а также отличную практическую основу для изучения и применения концепций программирования и разработки игр.

Были успешно реализованы:

1 Алгоритм обхода графа в глубину, учитывая состояние узлов, с последующим использованием для реализации взаимодействия с неигровыми персонажами.

2 Алгоритм выборов и последствий для обработки пользовательского ввода и отклика игры.

3 Алгоритм боевой системы с интуитивно понятным интерфейсом и проработанной механикой.

Однако, несмотря на достигнутые результаты, проект остается открытым для дальнейшего развития и улучшения. Возможным направлением доработки является расширение функциональности игры, добавление новых элементов геймплея, улучшение графики, а также оптимизация производительности.

В целом, данный курсовой проект предоставляет ценный опыт как разработчику, так и пользователям. Создание игр – это увлекательное и в то же время довольно сложное занятие, требующее усидчивости и внимания к деталям. Однако дальнейший пользовательский опыт оправдывает все затраты.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Доманов, А. Т. Стандарт предприятия / А. Т. Доманов, Н. И. Сорока. – Минск : БГУИР, 2017. – 167 с.
- [2] Что такое РПГ [Электронный ресурс]. – Режим доступа: <https://igrasan.ru/chto-takoe-rpg/>. – Дата доступа: 18.04.2024.
- [3] Настольные ролевые игры [Электронный ресурс]. – Режим доступа: [https://gamesisart.ru/theory\\_nri\\_kk.html](https://gamesisart.ru/theory_nri_kk.html). – Дата доступа: 18.04.2024.
- [4] Текстовые игры [Электронный ресурс]. – Режим доступа: <https://cubiq.ru/luchshie-tekstovye-igry/>. – Дата доступа: 18.04.2024.
- [5] Книга игрока / Дж. Кроуфорд [и др.]. – США : Hasbro SA, 2014. –
- [6] Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. – СПб. : БХВ, 2024. – 59 с.
- [7] Современный C++: безопасное использование / Дж. Лакос [и др.]. – М. : ДМК ПРЕСС, 2023. – 32 с.
- [8] Фреймворк: что это такое и для чего нужен [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/framework/>. – Дата доступа: 20.04.2024.
- [9] About Qt - Qt Wiki [Электронный ресурс]. – Режим доступа: [https://wiki.qt.io/About\\_Qt](https://wiki.qt.io/About_Qt). – Дата доступа: 20.04.2024.
- [10] Vladk-jpg/Dnd (github.com) [Электронный ресурс]. – Режим доступа: <https://github.com/Vladk-jpg/Dnd>. – Дата доступа: 05.05.2024.



# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг программного кода

Листинг А.1 – Используемые константы и подключаемые библиотеки

```
#include <QObject>
#include <QRandomGenerator>
#include <QTimer>
#include <QColor>
#include <QWidget>
#include <QString>
#include <QVector>
#include <QDebug>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QMainWindow>
#include <QMessageBox>
#include <QTextEdit>
#include <QFile>
#include <QJsonArray>
#include <QJsonDocument>
#include <QJsonObject>
enum modifiers { STRENGTH, DEXTERITY, CONSTITUTION, INTELLEGENCE,
WISDOM, CHARISMA };
enum gameStates { CREATION, FIGHT, DIALOG, FREE };
enum itemTypes { HEAL, DAMAGE, DEFENCE, TRINKET, SPELL };
enum battleRolls { INITIATIVE_ROLL, HIT_ROLL, DAMAGE_ROLL, NO_ROLL };
enum placeTypes { CITY, BUILDING, ENVIRONMENT };
enum events {
    NO_EVENT,
    STRENGTH_EVENT,
    DEXTERITY_EVENT,
    CONSTITUTION_EVENT,
    INTELLEGENCE_EVENT,
    WISDOM_EVENT,
    CHARISMA_EVENT,
    FIGHT_EVENT,
    TRAP_EVENT,
    PRIZE_EVENT,
    END_EVENT
};
const int D4 = 4;
```

```

const int D6 = 6;
const int D8 = 8;
const int D10 = 10;
const int D12 = 12;
const int D20 = 20;
const int MAX_LVL = 20;
const int DEF_EXP = 300;
const int ARMOR_ID_START = 1;
const int ARMOR_ID_FINISH = 10;
const int WEAPON_ID_START = 11;
const int WEAPON_ID_FINISH = 35;
const int SPELL_ID_START = 26;
const int SPELL_ID_FINISH = 35;
const int POTION_ID_START = 36;
const int POTION_ID_FINISH = 40;
const int TRINKET_ID_START = 41;
const int TRINKET_ID_FINISH = 45;

```

## Листинг А.2 – Алгоритм броска кубика

```

void startRolling(int type)
{
    m_rollsCount = 0;
    m_interval = STARTINTERVAL;
    m_timer.start(m_interval);
    this->type = type;
}
void roll()
{
    int result = QRandomGenerator::global()->bounded(1, type + 1);
    emit rolled(type, result, m_rollsCount >= MAXROLLS - 1);
    m_rollsCount++;
    if (m_rollsCount >= MAXROLLS) {
        stopRolling();
    }
    m_interval *= INCREMENT;
    m_timer.setInterval(m_interval);
}

```

## Листинг А.3 – Структура диалога и её парсер из JSON

```

struct Dialog
{
    QString name;

```

```

struct DialogNode
{
    int id;
    QString option;
    QString dialog;
    QVector<int> options;
    int event;
    bool was = false;
};
QVector<DialogNode> nodes;
};

Dialog World::parseDialog(const QString &filePath)
{
    Dialog dialog;

    QByteArray jsonData = file.readAll();
    QJsonDocument doc = QJsonDocument::fromJson(jsonData);
    QJsonObject rootObject = doc.object();
    dialog.name = rootObject.value("name").toString();
    QJsonArray nodesArray = rootObject.value("dialog_nodes").toArray();

    for (const auto &nodeValue : nodesArray) {
        QJsonObject nodeObject = nodeValue.toObject();
        Dialog::DialogNode dialogNode;
        dialogNode.id = nodeObject.value("id").toInt();
        dialogNode.option = nodeObject.value("option").toString();
        dialogNode.dialog = nodeObject.value("dialog").toString();
        dialogNode.event = nodeObject.value("event").toInt();
        QJsonArray optionsArray = nodeObject.value("options").toArray();
        for (const auto &optionValue : optionsArray) {
            if (optionValue.isDouble()) {
                dialogNode.options.append(optionValue.toInt());
            }
        }
        dialog.nodes.append(dialogNode);
    }
    file.close();
    return dialog;
}

```

Листинг А.4 – Функция обеспечивающая алгоритм боя без вывода сообщений

```

if (type == needRoll) {

```

```

if (phase == INITIATIVE_ROLL) {
    needRoll = D20;
    if (yourTurn) {
        playerLastRoll = roll + player->getMod(DEXTERITY);
        yourTurn = false;
        emit enemyRoll(needRoll);
    } else {
        enemyLastRoll = roll + enemy->getMod(DEXTERITY);
        if (playerLastRoll >= enemyLastRoll) {
            yourTurn = true;
            firstStep = true;
        } else {
            yourTurn = false;
            firstStep = false;
            emit enemyRoll(needRoll);
        }
        phase = HIT_ROLL;
    }
} else if (phase == HIT_ROLL) {
    if (yourTurn) {
        playerLastRoll = roll + bonus();
        if (playerLastRoll >= enemy->getDefence()) {
            needRoll = player->getDamage();
            phase = DAMAGE_ROLL;
        } else {
            yourTurn = false;
            needRoll = D20;
            emit enemyRoll(needRoll);
            if (!firstStep) {
                round++;
            }
        }
    } else {
        enemyLastRoll = roll + enemy->getMod(STRENGTH);
        if (enemyLastRoll >= player->getDefence()) {
            phase = DAMAGE_ROLL;
            needRoll = enemy->getDamage();
            emit enemyRoll(needRoll);
        } else {
            yourTurn = true;
            needRoll = D20;
            if (firstStep) {
                round++;
            }
        }
    }
}

```

```

    }
}
} else if (phase == DAMAGE_ROLL) {
    if (yourTurn) {
        enemy->getHeart(roll + bonus());
        if (!enemy->isAlive()) {
            phase = NO_ROLL;
            player->addExp(enemy->getDanger() * 200);
            emit fightEnd(enemy->getName());

        } else {
            phase = HIT_ROLL;
            yourTurn = false;
            needRoll = D20;
            emit enemyRoll(needRoll);
        }
        if (!firstStep) {
            round++;
        }
    } else {
        player->getHeart(roll + enemy->getMod(STRENGTH));
        if (!player->isAlive()) {
            phase = NO_ROLL;
            emit gameOver();
        } else {
            phase = HIT_ROLL;
            yourTurn = true;
            needRoll = D20;
        }
        if (firstStep) {
            round++;
        }
    }
}
}

```

Листинг А.5 – выдержка из алгоритма выборов и последствий

```

else if (state == DIALOG) {
    if (command.mid(0, 2) == "/c") {
        QString option = command.mid(3);
        chooseOption(option, false);

    } else if (command.mid(0, 2) == "/f") {
        state = FIGHT;
    }
}

```

```

fightToNPC(QString::number(currentEntity.toInt() + 1));

} else if (command.mid(0, 4) == "/end") {
    state = FREE;
    defeated[currentPlace->npcs[currentEntity.toInt()]] = true;
    emit sendText("Диалог закончился\n", Qt::black);

} else if (command.mid(0, 2) == "/i") {
    emit sendText("\nВарианты:\n", Qt::darkBlue);
    for (const int var : possibleWays) {
        emit sendText(QString::number(var) + " " + dialog.nodes[var].option + "\n",
            Qt::black);
    }

} else {
    emit sendText("Команда не найдена\n", Qt::red);
}
}

```

#### Листинг А.6 – рекурсивная функция для обхода графа

```

void World::checkWay(int id)
{
    for (const int &node : dialog.nodes[id].options) {
        if (dialog.nodes[node].was || dialog.nodes[node].option == "") {
            dialog.nodes[node].was = true;
            checkWay(node);
        } else if (!possibleWays.contains(node)) {
            possibleWays.append(node);
        }
    }
}

```

#### Листинг А.7 – Функция для запуска рекурсии обхода графа

```

void World::findPossibleWays()
{
    possibleWays.clear();
    checkWay(0);
}

```

#### Листинг А.8 – Выдержка из алгоритма обработки событий



```

else if (event == PRIZE_EVENT) {
    int randomNum = QRandomGenerator::global()->bounded(1, 5);
    if (randomNum == 1) {
        randomNum = QRandomGenerator::global()->bounded(ARMOR_ID_START,
ARMOR_ID_FINISH + 1);
    } else if (randomNum == 2) {
        if (player->getGameClass() == "Варвар" || player->getGameClass() == "Воин"
            || player->getGameClass() == "Паладин" || player->getGameClass() == "Следопыт"
            || player->getGameClass() == "Плут") {
            randomNum = QRandomGenerator::global()->bounded(WEAPON_ID_START,
SPELL_ID_START);
        } else {
            randomNum = QRandomGenerator::global()->bounded(WEAPON_ID_START,
WEAPON_ID_FINISH + 1);
        }
    } else if (randomNum == 3) {
        randomNum = QRandomGenerator::global()->bounded(POTION_ID_START,
POTION_ID_FINISH + 1);
    } else {
        randomNum = QRandomGenerator::global()->bounded(TRINKET_ID_START,
TRINKET_ID_FINISH + 1);
    }
    if (state == DIALOG) {
        currentNode->was = true;
        player->inventory.push_back(items[randomNum]);
        player->addExp(200);
        findPossibleWays();
        if (possibleWays.empty()) {
            state = FREE;
            defeated[currentPlace->npcs[currentEntity.toInt()]] = true;
        } else {
            emit sendText("\nВарианты:\n", Qt::darkBlue);
            for (const int var : possibleWays) {
                emit sendText(QString::number(var) + " " + dialog.nodes[var].option + "\n",
Qt::black);
            }
        }
    } else {
        player->inventory.push_back(items[randomNum]);
    }
    emit blockInput(false);
}

```

Листинг А.9 – Часть алгоритма создания персонажа

```

void CreationForm::on_createButton_clicked()
{
    QString name = ui->name->text();
    QString race = ui->race->currentText();
    QString gameClass = ui->gameClass->currentText();
    int str = 0, dex = 0, con = 0, intel = 0, wis = 0, cha = 0, health = 0;
    player->inventory.clear();
    str = QRandomGenerator::global()->bounded(-2, 3);
    dex = QRandomGenerator::global()->bounded(-2, 3);
    con = QRandomGenerator::global()->bounded(-2, 3);
    intel = QRandomGenerator::global()->bounded(-2, 3);
    wis = QRandomGenerator::global()->bounded(-2, 3);
    cha = QRandomGenerator::global()->bounded(-2, 3);

    if (gameClass == "Бавар") {
        health = D12;
        str = 2;
        dex = 2;
        intel = -2;
        player->inventory.push_back(world->getItem(15));
        player->inventory.push_back(world->getItem(4));
        player->inventory.push_back(world->getItem(37));
    }

    ...

    if (race == "Дварф") {
        str += 2;
        con += 2;
    }

    ...

    health += con;
    player->inventory.push_back(world->getItem(QRandomGenerator::global()->bounded(41,
46)));
    player->setHealth(health);
    player->setGameClass(gameClass);
    player->setRace(race);
    player->setName(name);
    player->setMod(STRENGTH, str);
    player->setMod(DEXTERITY, dex);
    player->setMod(CONSTITUTION, con);
}

```

```

player->setMod(INTELLEGENCE, intel);
player->setMod(WISDOM, wis);
player->setMod(CHARISMA, cha);
player->useItem(0);
player->useItem(1);
emit Completed(player);
}

```

#### Листинг А.10 – функции прокачки персонажа

```

void Player::addExp(int incr)
{
    exp += incr;
    while (exp >= maxExp[lvl - 1]) {
        exp -= maxExp[lvl - 1];
        lvl++;
        lvlUpForm->show();
    }
}

void Player::handleMod(int mod)
{
    setMod(mod, getMod(mod) + 1);
    maxHealth += QRandomGenerator::global()->bounded(1, healDice + 1);
    health = maxHealth;
    emit sendText("Ваша характеристика №" + QString::number(mod + 1) + " повысилась на 1",
        Qt::darkGreen);
}

```

**ПРИЛОЖЕНИЕ Б**  
**(обязательное)**  
**Функциональная схема программы**

**ПРИЛОЖЕНИЕ В**  
**(обязательное)**  
**Блок-схема алгоритма,**  
**реализующего управление системой боя**

**ПРИЛОЖЕНИЕ Г**  
**(обязательное)**  
**Блок-схема алгоритма,**  
**реализующего механизм выборов и последствий**

**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Блок-схема алгоритма,**  
**реализующего поиск в глубину**

**ПРИЛОЖЕНИЕ Е**  
**(обязательное)**  
**Блок-схема алгоритма,**  
**реализующего обработку событий**



**ПРИЛОЖЕНИЕ Ж**  
**(обязательное)**  
**Блок-схема алгоритма,**  
**реализующего прогрессию персонажа**