



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

**ОТЧЕТ  
О ХОДЕ РЕАЛИЗАЦИИ ТЕКСТОВОЙ ПОШАГОВОЙ RPG  
ИГРЫ НА ЯЗЫКЕ C++**

**по дисциплине  
«ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ»**

Выполнил студент группы ИКБО-04-21

Мерзляков К. Р.

Преподаватель

Евстигнеева О. А.

Москва 2021

## СОДЕРЖАНИЕ

1 Постановка задачи и ее анализ.....	4
1.1 Постановка задачи.....	4
1.2 Анализ задачи .....	4
1.3 Формализация требований .....	6
2 Выбор названия игры.....	7
3 Файловая структура проекта .....	10
4 Проектирование и реализация системы сохранений.....	11
4.1 Проектировка системы сохранений .....	11
4.2 Реализация системы сохранений на языке с++ .....	16
5 Проектирование и реализация системы случайных встреч .....	30
5.1 Проектирование системы случайных встреч .....	30
5.2 Реализация системы случайных встреч на языке C++ .....	30
6 Реализация вызываемых функций в файле «Functions.cpp».....	33
6.1 Реализация функции isNum() .....	33
6.2 Реализация функции correctName() .....	34
6.3 Реализация функции print() .....	34
7 Реализация игровых классов «knight» и «wizard».....	36
7.1 Реализация игрового класса «knight» .....	36
7.2 Реализация игрового класса «wizard» .....	41
8 Реализация классов игровых противников.....	42
9 Реализация системы сражений с противником в файле «Battle.cpp» .	44
9.1 Сражение с противником .....	44
9.2 Сохранение игры при достижении нового уровня .....	49

10 Выводы .....	52
11 Дополнительные материалы .....	53

# **1 ПОСТАНОВКА ЗАДАЧИ И ЕЕ АНАЛИЗ**

## **1.1 Постановка задачи**

Необходимо разработать текстовую пошаговую игру в жанрке RPG на языке программирования C++. Так как важным критерием является использование в качестве единственного средства разработки языка программирования C++ использование игровых движков (Unity, gamemaker studio 2 и др.) становится невозможным (следовательно, программа будет работать в консоли).

## **1.2 Анализ задачи**

Исходя из условий, игра является текстовой, следовательно, использование графики не предполагается, что позволяет снизить затраты на создания текстур, которые в проекте по программированию не играют никакой роли, до нуля. Так же это упрощает создание игры и делает возможным ее реализацию в консоли.

Любая современная игра позволяет игроку сохранять его игровой процесс. В противном случае игрок довольно быстро теряет интерес к продукту. Это связано с тем, что люди любят видеть результат своих действий. Многие, если не все, инкрементальные игры базируются на сохранении числовых результатов игрока, видимости его прогресса. Это важная психологическая почва практически любой игры. Если же сохранения достижений игрока не будет, как уже было сказано ранее, пользователь быстрее потеряет интерес к продукту, что является большой проблемой: чем больше времени игрок проводит за одной игрой, тем больше вероятность того, что он поделится информацией о ее существовании с другими потенциальными игроками.

Есть несколько способов выбора основных направлений игры: упор на сюжет, упор на геймплей. Рассмотрим оба варианта:

К плюсам сюжетно-ориентированной игры (при хорошо проработанном сюжете) можно отнести:

1. Запоминающаяся история.

2. Персонажи, к которым игрок привыкает и начинает чувствовать по отношению к ним эмоции.
3. Появляющееся исходя из пунктов 1 и 2 желание игрока рассказать об игре своим знакомым. Принцип сарафанного радио создает дополнительную рекламу.

#### Минусы сюжетно-ориентированной игры:

1. Требуется больше работы для создания качественного сценария, грамотного прописывания сценария, нежели работы программиста, что для проекта по дисциплине «Процедурное программирование» неприемлемо.
2. Необходимость присутствия музыкального сопровождения для создания необходимой атмосферы.
3. Не имея соответствующего образования (сценариста) вероятность написать качественный, запоминающийся, вызывающий эмоции сюжет практически невозможно.
4. Ограниченное время прохождения игры.

Теперь для сравнения выделим плюсы и минусы геймплейно-ориентированной игры:

1. Интересный процесс игры, заставляющий возвращаться в нее и тратить там неограниченное количество времени.
2. Возникающее в связи с интересным геймплеем желание привлечь в игру своих знакомых. Принцип сарафанного радио, основанный на геймплее.
3. Отсутствует необходимость в музыкальном сопровождении.
4. Все усилия тратятся на написание кода, что соответствует дисциплине «Процедурное программирование».

5. Не требуются специальные навыки (например, сценарное мастерство). Навыки программиста в учет не берутся, так как это основа направления «Программная инженерия» (09.03.04).

Минусы геймплейно-ориентированной игры:

1. Экспоненциально возросшая сложность кода.

Исходя из плюсов и минусов каждой из возможных направленностей игры выберем оптимальную: В виду того, что сюжетная ориентация имеет больше минусов, нежели у геймплейная, некоторые из которых фатальные (пункт 1 в «Минусах сюжетно-ориентированной игры»), за основу игры берется геймплейная ориентация.

### **1.3 Формализация требований**

Формализуем требования к проекту:

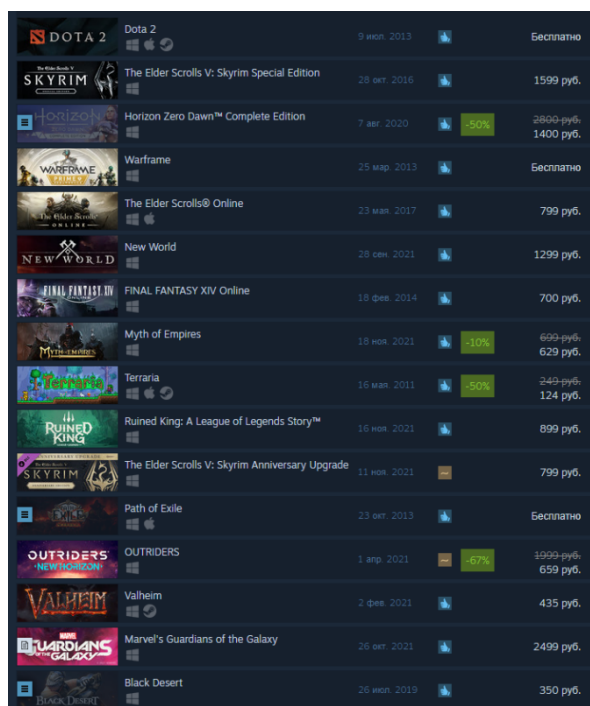
- Текстовая игра (отсутствие графики)
- Проект выполняется в консоли
- Отсутствие сюжета
- Геймплей как основа игры
- Наличие сохранений
- C++ как единственное средство разработки

















## 2 ВЫБОР НАЗВАНИЯ ИГРЫ

Название является крайне важной составляющей игры. В обществе существует поговорка «Встречают по одежке, а провожают по уму». Конечно, это лишь устное народное творчество, которое не имеет доказательной базы. Однако если это суждение существует уже сотни лет, значит люди с ним согласны. Если перекладывать эту поговорку на плоскость игростроя, то получится примерно следующее: «Встречают по названию, провожают по наполнению». Исходя из этого, можно понять, что назвать игру «GaMe1423» или похожим образом просто недопустимо.

Если провести небольшой анализ, то можно понять, что RPG на с названием на русском языке практически нет. Так, если зайти в самый популярный на данный момент времени цифровой магазин видеоигр «Steam» и выставить жанр «Ролевая игра» (равносильно RPG), то можно увидеть, что игр с русскоязычным названием, как говорилось выше, практически нет. Приведем в пример первые 15 игр из поискового запроса (DLC не считаем за отдельную игру, так как это дополнение к уже указанному в списке продукту, по сути, являющееся его частью) (см рис. 1).

Рисунок 1 – результат запроса «Ролевая игра» в цифровом магазине Steam



	Dota 2	9 июл. 2013	Бесплатно
	The Elder Scrolls V: Skyrim Special Edition	28 окт. 2016	1599 руб.
	Horizon Zero Dawn™ Complete Edition	7 апр. 2020	2800 руб. 1400 руб.
	Warframe	25 мар. 2013	Бесплатно
	The Elder Scrolls® Online	23 мар. 2017	799 руб.
	New World	28 дек. 2021	1299 руб.
	FINAL FANTASY XIV Online	18 фев. 2014	700 руб.
	Myth of Empires	18 июл. 2021	699 руб. 629 руб.
	Terraria	16 мар. 2011	249 руб. 124 руб.
	Ruined King: A League of Legends Story™	16 июл. 2021	899 руб.
	The Elder Scrolls V: Skyrim Anniversary Upgrade	11 июл. 2021	799 руб.
	Path of Exile	23 окт. 2013	Бесплатно
	OUTRIDERS	1 апр. 2021	1099 руб. 659 руб.
	Valheim	2 фев. 2021	435 руб.
	Marvel's Guardians of the Galaxy	26 окт. 2021	2499 руб.
	Black Desert	26 мар. 2019	350 руб.

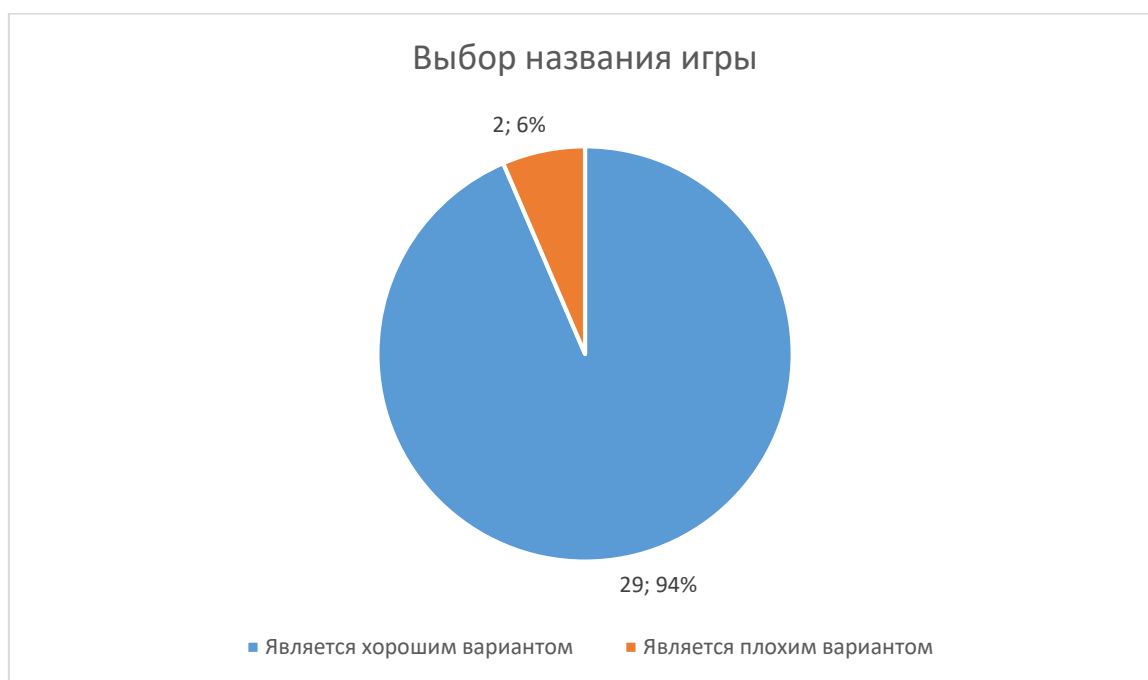
Так, среди приведенных 15 игр нет ни одной, название которой было бы переведено на русский язык. Исходя из проведенного выше небольшого анализа можно сделать вывод, что название игры должно быть на английском языке.

Как было сказано в «Анализе задачи», игра не будет сюжетно-ориентированной, поэтому название не получится привязать к истории, о которой повествует игра. Однако мы можем связать название игры с ее геймплеем. Так как геймплей будет заключаться в сражении с несколькими разнообразными противниками, в качестве названия можно взять «The Dungeon Conqueror». Стоит уточнить, что в игровой культуре и культуре ЛитRPG (откуда, собственно, и появились RPG), слово «Dungeon» (дословно «Подземелье») означает место с большим количеством противников, с которыми можно сражаться, и (иногда) сокровищ. Из обычной этимологии этого слова так же следует, что это место находится под землей. Так как игрок будет сражаться в этом подземелье с противниками, можно сказать, что он будет его завоевывать. Его можно назвать завоевателем (Conqueror).

Проверим нашу гипотезу с помощью опроса. Сформулируем краткий вопрос, который будет задан респонденту: «Является ли название «The Dungeon Conqueror» хорошим выбором для текстовой RPG игры без сюжета, в которой противник будет сражаться с бесконечным количеством противников и получать после победы деньги». Так же сформулируем ответы: «Является хорошим вариантом» и «Является плохим вариантом». Из 31 одного человека, что прошли опрос, 29 считаю данное название хорошим выбором, 2 – плохим (см. диаграмма 1)



Диаграмма 1 – результат опроса «Выбор названия игры»



Из результатов опроса становится понятно, что данное название подходит игре.

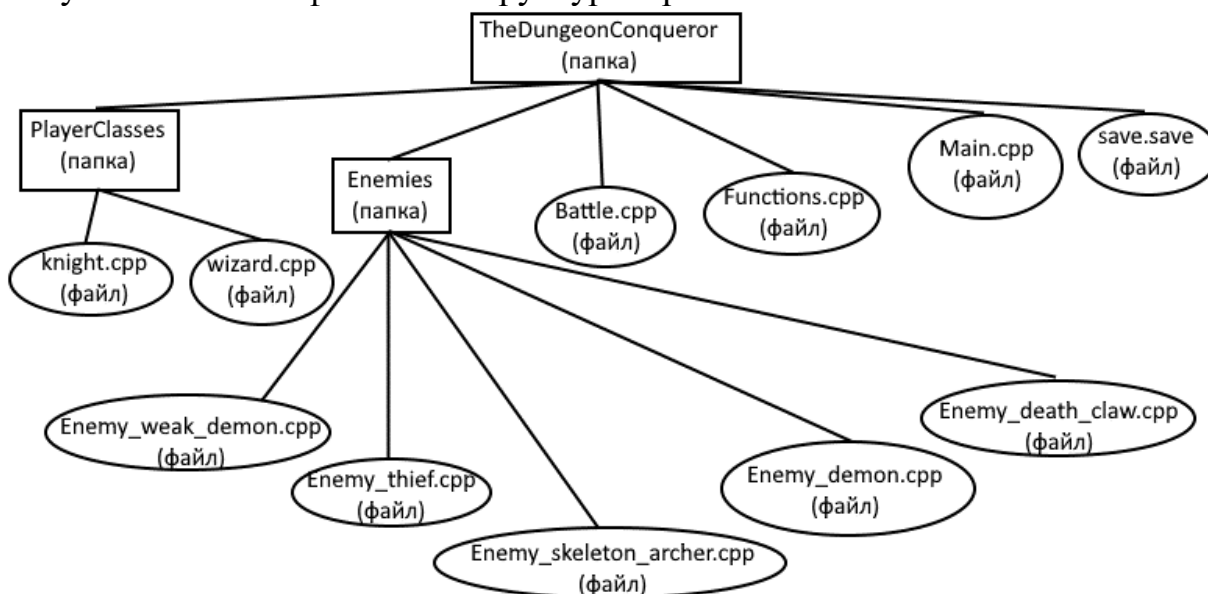
### 3 ФАЙЛОВАЯ СТРУКТУРА ПРОЕКТА

Корректная файловая структура является крайне основополагающей вещью при реализации любого проекта. Файлы должны располагаться в соответствующих папках, должны быть объединены в них по схожести каких-либо критериев.

Определим грамотную файловую структуру для данного проекта.

Исходной (главной) папкой, в которой будут лежать все остальные файлы игры будет папка с названием «TheDungeonConqueror» (по названию игры). Внутри папки будут располагаться файлы «Main.cpp», «Functions.cpp», «Battle.cpp», а также папки «Enemies» и «PlayerClasses». Реализация всех этих файлов будет описана позже. Внутри папки «Enemies» будут располагаться противники. В данной игре будет пять типов противников, назовем файлы соответственно названиям противников: «Enemy\_weak\_demon.cpp», «Enemy\_thief.cpp», «Enemy\_skeleton\_archer.cpp», «Enemy\_demon.cpp», «Enemy\_death\_claw.cpp». Внутри папки «PlayerClasses» будут располагаться классы игроков. В данной игре их будет два. Назовем файлы соответственно названиям классов: «knight.cpp», «wizard.cpp». На рисунке 2 можно увидеть схему расположения файлов.

Рисунок 2 – схема файловой структуры проекта



## **4 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ СОХРАНЕНИЙ**

### **4.1 Проектировка системы сохранений**

Как говорилось ранее, система сохранений – обязательный критерий при реализации данной игры. Для того, чтобы понять, как ее реализовывать, необходимо продумать логику работы системы сохранения.

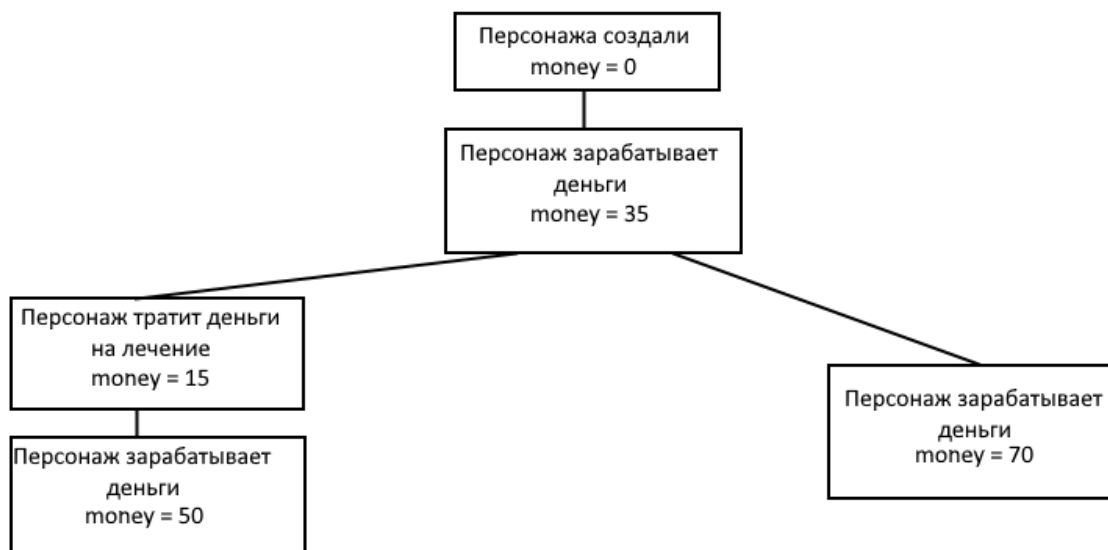
Итак, сначала нужно понять, какие данные мы будем сохранять. Для того, чтобы при загрузке игры пользователь мог более удобно выбирать игрового персонажа необходимо позволить каждому персонажу присваивать имя. Таким образом, одной из переменных, которые будет сохраняться, будет имя.

Одним из основных критериев RPG игры является возможность выбора игрового класса. Так как данный проект так же принадлежит к этому жанру, необходимо реализовать несколько игровых классов. Таким образом, следующей переменной, которую необходимо сохранять, будет класс.

Как говорилось в «Анализе задачи», людям нравится видеть (даже мнимое) развитие, чувствовать, что он чего-то достиг. Таким образом, игра должна предоставлять игроку это ощущение. В видеоиграх жанра RPG это обеспечивается системой уровней. В зависимости от уровня игрового персонажа сила его атаки и максимальное количество здоровья так же будут изменяться. Таким образом, уровень так же необходимо сохранять.

Количество денег, которые есть у конкретного персонажа, постоянно меняется. Так, например, когда персонаж только создается, количество его денег  $money = 0$ . Когда персонаж убивает противника, количество денег увеличивается на, например, 35. Количество денег  $money = 35$ . Игрок может подлечиться и потратить на это некоторую сумму средств, например, 20. Тогда его количество денег  $money = 35 - 20 = 15$ . После этого игрок может победить еще одного противника, увеличив количество денег до  $money = 50$ . Однако он может не лечиться и сразу победить противника. Тогда количество денег  $money = 70$ . Как видно из данного примера, количество денег  $money$  необходимо сохранять.

Рисунок 3 – иллюстрация примера принципа подсчета денег в игре



Силу атаки можно не сохранять. Это связано с тем, что на каждом уровне атака всегда имеет свое собственное, постоянное значение, в отличие от количества здоровья. Например, на первом уровне сила атаки  $attack = 10$ , на втором  $attack = 15$ , на третьем  $attack = 20$  и так далее. Таким образом, силу атаки можно рассчитывать непосредственно в коде в зависимости от уровня игрового персонажа.

Количество здоровья во всех играх жанра RPG реализуется следующим образом: задается некий максимум здоровья для персонажа, выше которого оно просто не может подниматься. Так же задается текущее здоровье персонажа. Например, максимальное здоровье игрока  $maxHealth = 500$ . Однако это не означает, что на данный момент здоровье игрока  $currentHealth = maxHealth = 500$ . Текущее состояние здоровья может быть меньше максимального. Например,  $currentHealth = 440 < maxHealth$ . Такое может случаться в случае, если противник атаковал игрока и нанес ему урон. Максимальное значение здоровья  $maxHealth$  не изменилось, однако текущее значение здоровья  $currentHealth$  уменьшилось на силу атаки противника. Таким образом, значение  $maxHealth$  на каждом уровне постоянно и меняется только при увеличении уровня. Следовательно, максимальное значение здоровья можно не сохранять, а рассчитывать непосредственно в коде в

зависимости от игрового персонажа. Однако значение текущего здоровья `currentHealth` может меняться в течении уровня, следовательно, рассчитывать значение текущего здоровья в зависимости от уровня нельзя. Его необходимо сохранять.

Количество маны, необходимое для совершения атаки, работает по схожему с количеством здоровья принципу. Конечно, причины изменения количества маны отличаются от причин изменения количества здоровья. Однако основной принцип работы аналогичен. Следовательно, значение `maxManaPoint` не сохраняется, но значение `currentManaPoint` сохраняется.

Исходя из сказанного выше, сохраняются следующие переменные (их названия могут отличаться в итоговой реализации):

1. Переменная, хранящая в себе имя конкретного персонажа
2. Переменная, хранящая в себе выбранный класс персонажа
3. Переменная, хранящая в себе уровень персонажа
4. Переменная, хранящая в себе количество денег персонажа
5. Переменная, хранящая в себе размер текущего здоровья персонажа
6. Переменная, хранящая в себе размер текущей маны персонажа

Таким образом, мы выделили шесть переменных, которые будут сохраняться для каждого из игровых персонажей. Теперь необходимо понять, куда сохранять данные переменные и в какой форме это делать.

Обозначим возможные варианты записи данных в файл. Мы можем записывать данные в файл формата `.txt`, в файл своего собственного формата или же в базу данных `SQL`. Рассмотрим все три варианта.

Способ с использование записи данных в файл формата `.txt` отличается простотой своей реализации, однако имеет серьезный минус: файлы формата `.txt` распознаются как текстовые по умолчанию на подавляющем количестве компьютеров. Таким образом, пользователь может открыть файл двойным щелчком без особых усилий, ведь операционная система сама подобрала необходимое приложение для открытие файла (например, блокнот).

Второй способ (использование собственного формата файла), как и первый, отличается простотой своей реализации. Однако бесспорным его плюсом над первым способом является то, что наше расширение файла, которое мы придумали, не будет определяться системой по умолчанию как файл текстового формата. То есть, пользователь не сможет открыть его просто дважды кликнув по файлу. Ему придется выбирать приложение, с помощью которого открывать подобные файлы. Однако не каждый пользователь решит открывать подобный файл. Этому есть множество причин. Так, например, некоторые пользователи могут быть уверены в том, что у них нет необходимого программного обеспечения, чтобы открыть данный файл. Или же просто могут бояться взаимодействовать с файлами, имеющими незнакомое расширение, ведь они могут случайно что-то повредить. К минусам можно отнести, что такая защита является больше «проверкой на дурака». Человек, который хотя бы немного разбирается в программном обеспечении, сможет открыть данный файл.

Третий способ (использование базы данных SQL) является самым надежным в плане защиты информации. Человек, не имеющий прикладных знаний о устройстве баз данных не сможет открыть файл и что-то в нем изменить. К минусам данного способа можно отнести сложность реализации и необходимость использования стороннего программного обеспечения (например, систему управления базами данных «Microsoft SQL Server»), что запрещено требованиями к проекту.

Таким образом, ввиду того, что третий способ не подходит под изначальные критерии, а первый способ проигрывает по защищенности второму, для реализации сохранений рациональнее всего использовать второй способ (реализация системы сохранений с помощью текстового файла собственного формата).

Исходя из того, что был выбран второй способ реализации сохранений, необходимо выбрать расширение файла сохранения. Самым очевидным вариантом для файла, в который сохраняется игра, является .save. Данное

название расширения соответствует содержимому, поэтому с точки зрения грамотного названия переменных (и файлов) проблем нет.

Файл является текстовым. Нужно определить, как будет выглядеть сохраненная информация внутри файла. Сохранять переменные вместе с их названиями не рационально: это занимает много ненужного места, дает пользователю (если тот все же вскрыл файл) понимание, какая переменная за что отвечает, что упрощает подделку сохранений. Из этого можно сделать вывод, что название переменных, которые сохраняются, не должно присутствовать в файле. В таком случае, нужно как-то разделять между собой записанные данные, чтобы программа понимала, где заканчивается, например, имя игрока и начинается его уровень. Можно писать каждую переменную в новой строке. Однако если при большом количестве игровых персонажей файл станет неоправданно длинным, ведь, например, при 7 игровых персонажах будет уже  $7 * 6 = 42$  строчки. В противовес предыдущему варианту, можно писать все данные об одном конкретном персонаже в одну строчку и отделять переменные друг от друга специальным символом, например, символом «:». В таком случае при 7 созданных персонажах будет всего семь строчек. Для разработчика это будет удобно при дебаге, ведь все данные об одном конкретном персонаже в одной строчке, в строго определенном порядке. Однако игрок, который (если тот все же вскрыл файл) не знает, в каком порядке конкретно идут данные, поэтому он не сможет что-то поменять не рискуя сломать сохранение, что с большой вероятностью остановит его. Ввиду описанных выше причин, последний вариант является оптимальным.

По сути, порядок записи переменных не играет большого значения. Главное, чтобы во всей программе он был одинаков. Так как мы уже описали выше все записываемые переменные и пронумеровали их, возьмем порядок оттуда.

Переменные друг от друга будем отделять знаком «:», т.е после каждой переменной будет стоять «:». Под каждого персонажа будет отдельная строка.

Таким образом, запись сохранения в файле будет выглядеть примерно так, как изображено на рисунке 3.

Рисунок 4 – пример сохранения двух персонажей в файле

```
Ник1:Класс1:Уровень1:Деньги1:Здоровье1:Мана1:  
Ник2:Класс2:Уровень2:Деньги2:Здоровье2:Мана2:
```

## 4.2 Реализация системы сохранений на языке с++

В файле «main.cpp» будет две логические части:

1. Создание новой игры и загрузка старой игры
2. Встречи со случайными противниками

В данный момент времени остановимся на первом пункте.

Так как для корректной работы программы нам так или иначе придется подключать библиотеки, подключим все необходимые сразу: `iostream`, `string`, `fstream`, `Windows.h`, `conio.h`, `cstdlib`, `time.h`.

Так же необходимо к главному файлу подключить все побочные файлы, которые указаны на рис. 2: `Functions.cpp`, `Battle.cpp`, `Enemy_thief.cpp`, `Enemy_skeleton_archer.cpp`, `Enemy_weak_demon.cpp`, `Enemy_demon.cpp`, `Enemy_death_claw.cpp`, `knight.cpp`, `wizard.cpp`. Помимо этого, установим пространство имен `std`. Подключение всех библиотек и файлов, а также установка пространства имен `std` проиллюстрировано рис. 5.



Рисунок 5 – подключение необходимых внешних ресурсов и установка пространства имен

```
#include "Functions.cpp"
#include "Battle.cpp"

#include "Enemies/Enemy_thief.cpp"
#include "Enemies/Enemy_skeleton_archer.cpp"
#include "Enemies/Enemy_weak_demon.cpp"
#include "Enemies/Enemy_demon.cpp"
#include "Enemies/Enemy_death_claw.cpp"

#include "PlayerClasses/knight.cpp"
#include "PlayerClasses/wizard.cpp"
using namespace std;
```

Рисунок 6 – сообщение приветствия игрока

Пока игрок не нажмет любую из кнопок на клавиатуре приветствие не исчезнет, что обеспечивается циклом с предусловием «while», условием выхода из которого является нажатие любой из кнопок («\_getch()»). Так как программа не предусматривает асинхронной работы разных частей ее кода, пока условие «if (\_getch()) break» не выполнится, программа не пойдет дальше.

работа, третья – за выбранный режим работы: создание нового персонажа (1) или загрузка уже имеющейся игры (2).

Также создадим два объекта, один класса knight, другой класса wizard (реализация классов будет показана и объяснена далее). В зависимости от того, какой будет класс у персонажа игрока будет использоваться один из двух объектов.

Создадим переменную типа bool, в зависимости от которой первая логическая часть будет меняться на вторую (иными словами, когда программа будет переходить от этапа создания персонажа / загрузки сохранения к непосредственно геймплею). Назовем ее flag и присвоим значение false (т.е, это означает, что ячейка сохранения, с которой дальше будет происходить работа, еще не выбрана). Реализация этого блока кода на рисунке 7.

Рисунок 7 – объявление и инициализация отвечающих за сохранения переменных, двух объектов игрока

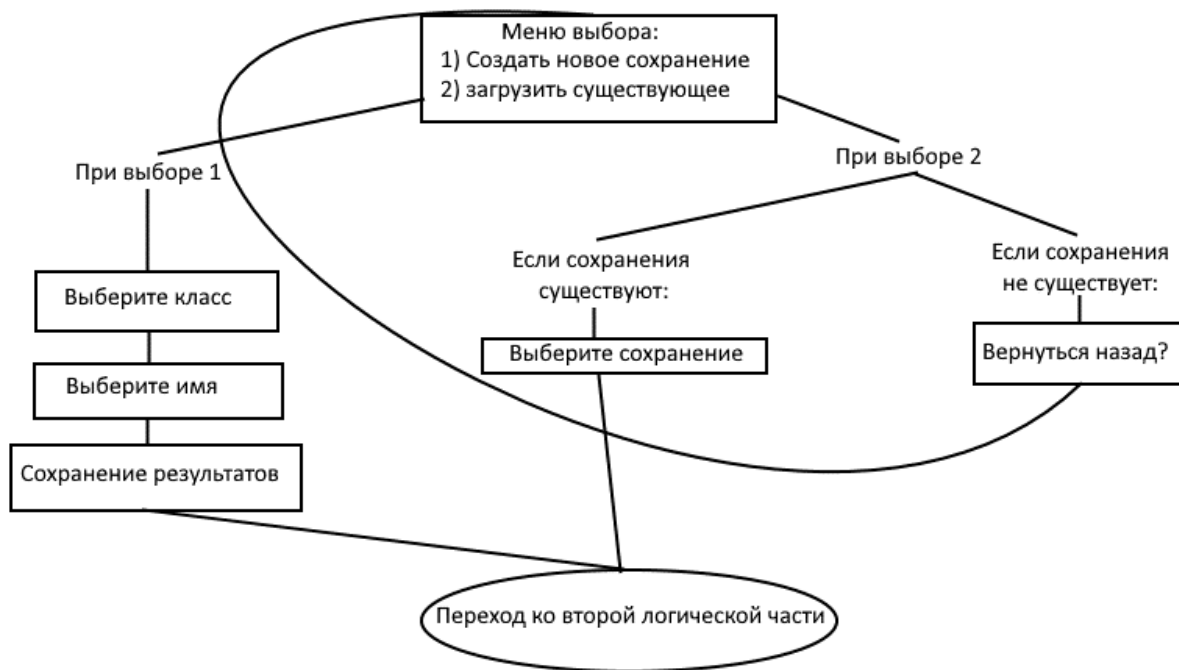
```
int ClassChoice = 0;
int ChoiceSave = 0;
int GameStartchoice = 0;

knight playerKnight;
wizard playerWizard;

bool flag = false;
```

Теперь перейдем непосредственно к написанию логики. Сначала создадим цикл while(!flag). Данный цикл будет выполняться, пока переменная flag != true. Это необходимо для той ситуации, когда игрок, например, захочет загрузить сохранение, выберет эту опцию в изначальном меню, однако потом обнаружится, что сохраненных игр нет и ему нужно создавать новую. В таком случае благодаря циклу while этот блок кода начнется еще раз, что позволит игроку заново выбрать: загрузить сохранение или же создать новое. Более наглядно эта система описана на рисунке 8.

Рисунок 8 – схема работы загрузки и сохранений



Исходя из описанной выше и продемонстрированной на рис. 8 логики, дальнейший код системы сохранений будет реализовываться в этом цикле `while(!flag)`. Ввиду того, что он довольно большой, его придется рассматривать по кускам.

В первом блоке кода внутри цикла `while(!flag)` мы открывает на чтение и на запись с условием добавления к уже существующему тексту, но не перезаписи «`ios::app`». Так же очистим консоль от того, что находится на ней сейчас, ведь дальше пользователю надо будет выбрать режим сохранения. С помощью созданной нами функции `print` (ее создание будет описано позже, данная функция просто выводит текст с задержкой после каждой буквы и характерным звуком печати) выводим на экран текст, в котором предлагаем игроку выбрать: начать новую игру или загрузить сохранение. Также на всякий случай еще раз обнулим переменную «`GameStartchoice`», ведь она могла сохранить свое значение с предыдущего прогона цикла `while(!flag)` (если он был). Далее используем цикл `while` с постусловием. Цикл будет повторяться до тех пор, пока игрок не введет одно из двух допустимых значений (1 или 2 соответственно). Внутри цикла мы создаем строковую

переменную choiceStr и читаем ее. Затем с помощью созданное нами функции isNum (ее реализация будет описана позже) проверяем, является ли данная строка числом. Если это так, то мы присваиваем переменной GameStartchoice переменную choiceStr, приведенную к типу int с помощью функции atoi. Далее происходит проверка выхода из цикла. Если переведенное из строки в число значение переменной GameStartchoice не является 1 или 2, то цикл повторяется. Так как изначально GameStartchoice = 0, если проверка isNum выдала false и переменной GameStartchoice не было присвоено новое значение, она продолжает быть равной нулю. И ввиду того, что 0 != 1 и 0 != 2 цикл продолжает свою работу.

Рисунок 9 – выбор между созданием нового сохранения и загрузкой старого

```
ifstream SaveRead("save.save");
ofstream SaveWrite("save.save", ios::app);

system("cls");

print("1 - New Game/n2 - Upload a save/n");

GameStartchoice = 0;
do {
    string choiceStr;
    cin >> choiceStr;
    if (isNum(choiceStr)) GameStartchoice = atoi(choiceStr.c_str());
    if (GameStartchoice < 1 or GameStartchoice > 2) print("Incorrect choice: ");
} while (GameStartchoice < 1 or GameStartchoice > 2);
```

После завершения работы цикла значение переменной «GameStartchoice» обязательно будет равняться либо 1, либо 2. Теперь реализуем оба режима работы: создание нового сохранения и загрузку старого. Начнем с первого. Так как в тексте, который мы выводили на экран, 1 – это новая игра, а 2 – загрузка существующей, при создании нового сохранения будет происходить при «GameStartchoice» = 1.

Таким образом, создадим условие, условием входа в которое является «GameStartchoice» = 1. Весь данный блок будет находиться внутри этого условия.

Так как возвращение к меню выбора после выбора создания нового сохранения не предполагается (см. рис 8), присваиваем переменной `flag` значение `true`. Очищаем экран от предыдущего текста, предлагаем выбрать один из двух классов: 1 – рыцарь, 2 – маг.

Дальше аналогично предыдущему разу создаем цикл с постусловием, в результате выполнения которого переменной «ClassChoice» присваивается одно из двух значений: 1 или 2.

Далее просим игрока (с помощью функции `print`) ввести ник игрового персонажа. Создаем переменную «name» и читаем ее в цикле с постусловием. Если результат проверки переменной «name» функцией `correctName` (реализация которой будет описана далее) будет равняться 1 (ник корректен), то цикл завершается. Если нет, цикл повторяется, пока ник не станет корректен. Далее, в зависимости от того, какой игровой класс выбрал персонаж, применяем метод `SetName` (реализация будет показана позже) к объекту `playerKnight` (если выбран рыцарь) или к объекту `playerWizard` (если выбран маг). Этот метод, как следует из названия, устанавливает персонажу имя.

Далее, на 94 строчке, записываем в файл ник персонажа (в дополнение к той информации, которая уже была в файле, если он не был пуст), знак разделитель, выбранный класс, уровень персонажа (так как мы только его создали, его уровень равняется 1), знак разделить, количество денег (которое равняется 0 по той же самой причине), знак разделитель. Далее, в зависимости от выбранного класса, сохраняем максимальное значение здоровья и маны. Если выбран рыцарь, максимальное здоровье = 600, а максимальная мана 100. В противном случае, если выбран маг, максимальное здоровье равняется 150, а максимальная мана 600. В файл записывает максимальный уровень здоровья, знак разделитель, максимальный уровень маны, знак разделитель. Так как это все необходимые переменные были записаны, выполняем в файле переход на новую строку (создаем ее для следующей ячейки сохранения).


На 98 строчке мы создаем переменную «gameSaves». Ее значение в данном случае нам будет не важно, ведь она нужна лишь для корректной работы getline.

Так как мы при создании нового сохранения мы дописывали строчку в конец файла можно сделать вывод, что на данный момент это сохранение является последним из всех. Создадим цикл while с предусловием, который будет исполняться, пока весь файл не будет прочтен. В цикле будем построчно (с помощью getline) читать строки-сохранения и прибавлять переменной «ChoiceSave» единицу с каждым прочтением. Таким образом, эта переменная будет сохранять номер прочтенной только что строки. Когда файл будет прочтен полностью, сохраненный номер строки (значение переменной «ChoiceSave») будет равен номеру последней строки, чего мы и добивались.

Закрываем открытые ранее на запись и на чтение потоки, так как они больше не будут использоваться внутри функции main.

Код данного блока можно увидеть на рисунке 10.

Рисунок 10 – код создания нового сохранения



```
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

if (GameStartChoice == 1) {
    flag = 1;
    system("cls");

    print("Select a class:/n1 - Knight/n2 - Wizard/n");

    do {
        string choiceStr;
        cin >> choiceStr;
        if (isNum(choiceStr)) ClassChoice = atoi(choiceStr.c_str());
        if (ClassChoice < 1 or ClassChoice > 2) print("Incorrect choice: ");
    } while (ClassChoice < 1 or ClassChoice > 2);

    print("Enter the name of the character:/n");
    string name;

    do {
        cin >> name;
        if (!correctName(name)) print("Incorrect name/n");
    } while (!correctName(name));

    if (ClassChoice == 1) playerKnight.SetName(name);
    else playerWizard.SetName(name);

    SaveWrite << name << ":" << ClassChoice << ":1:0:";
    if (ClassChoice == 1) SaveWrite << "600:100:\n";
    else SaveWrite << "150:600:\n";

    string gameSaves;
    while (!SaveRead.eof()) {
        getline(SaveRead, gameSaves);
        ChoiceSave++;
    }

    SaveRead.close();
    SaveWrite.close();

    system("cls");
}
```

Только что мы реализовали создание нового сохранения, которое отрабатывает при условии «GameStartchoice» = 1. Однако может быть ситуация «GameStartchoice» = 2, когда происходит чтение уже существующего сохранения.

Так как это два взаимоисключающих режима, реализуем их через if else. If уже был написан в предыдущем блоке кода, осталось дописать к нему else. После else запишем условие «GameStartchoice» = 2. По факту, оно не обязательно, ведь из-за цикла while, который мы использовали для чтения данной переменной, она может принимать только одно из этих двух значений. Однако написание этого условия сильно повышает читаемость кода.

Таким образом, весь следующий блок будет внутри этого условия if.

Очистим консоль от текста, который был в ней до этого. Создадим переменную «gameSaves» типа string и запишем в нее строку из файла сохранений с помощью getline (поток чтения и записи еще не закрыт, ведь это делалось в другом условии if else, которое, если выполняется данное, не выполнялось). Если строка пуста, значит сохраненных игр в файле нет. Говорим об этом пользователю с помощью функции print, создаем переменную «crutch» типа string и читаем ее. Благодаря этому, мы, можно сказать, ставим игру на паузу, пока игрок не введет какое-нибудь (абсолютно любое) значение. После этого цикл while (!flag) повторяется еще раз (см. рис. 8)

Если строка не пуста, то у пользователя есть сохраненные игры. Создадим переменную типа int «score» = 1, которая будет отвечать за номер выводимого на экран сохранения. Предлагаем игроку (с помощью функции print) выбрать одно из существующих сохранений. Далее мы создаем цикл while с постусловием, который выполняется до тех пор, пока файл не будет прочтен полностью. Далее мы будем работать внутри этого цикла, пока не будет сказано обратное. Выводим значение переменной «score» (то есть номер ячейки сохранения), следом за ним выводим тире, которое нужно для лучшей читаемости текста.

Реализуем проверку сохранения на верное количество знаков-разделителей (количество знаков-разделителей всегда строго определено и одинаково для всех корректных сохранений, равно шести). Создадим переменную «SeparatorCountCounter» типа `int`, в которую будет записываться количество знаков-разделителей «:» в строке. Используем цикл `for`, чтобы проверить все символы строки с начала до конца. Внутри цикла делаем проверку `if` с условием, что значение текущего символа равняется «:». Если это так, то увеличиваем значение переменной «SeparatorCountCounter» на единицу. Так же в цикле `for` делаем еще одну проверку `if`, условием которой будет, что значение «SeparatorCountCounter» > 6 (это связано с тем, что количество разделителей в строке-сохранении не может быть больше шести). Если условие выполняется, то мы выходим из цикла `for`.

Вне цикла `for` (но все еще в цикле `while` с постусловием) создаем проверку `if` с условием, что «SeparatorCountCounter» != 6. Если это условие выполняется, выводим сообщение об ошибке в сохранении и завершаем программу.

Так как первым в сохранении идет ник, сначала будем выводить его. Создадим переменную-счетчик типа `int` и назовем ее «`i`». Эта переменная будет отвечать за индекс символа в конкретной строке. Создаем цикл `while` с предусловием, который будет повторяться, пока не будет встречен символ «:» в строке «`gameSaves`». Внутри цикла выводим значение символа «`gameSaves[i]`» и инкрементируем значение переменной «`i`» на единицу для того, чтобы двигаться по строке. С помощью этого цикла мы выводим ник персонажа, который находится в данном сохранении.

После того, как мы закончили выводить ник персонажа отделим его пробелом от того, что будет дальше (строка 142). Увеличим значение «`i`» на единицу, так как сейчас «`gameSaves[i]`» = «:», а нам нужно идущее следом значение.

Сейчас символ строки «`gameSaves[i]`» = номеру класса (номер класса – всегда однозначное число). Однако если игрок как-то испортил сохранения,



это может быть не так. Создадим новую переменную «ClassChoiceStr» типа string. Следом создадим цикл с предусловием while, который будет выполняться до тех пор, пока мы не встретим знак разделитель «:». Внутри цикла прибавляем к только созданной нами строке символ строки gameSaves[i], затем увеличиваем значение переменной «i» на единицу. После выполнения цикла мы получаем строку, в которой находятся все символы, которые в файле стоят на месте номера класса. Если файл сохранения не корректировался самим игроком, там будет значение «1» или «2», но в противном случае там может быть другой символ. Увеличим значение «i» на единицу. Реализуем проверку корректности переменной «ClassChoiceStr» (строка 151). Ее значение должно быть числом (проверяется функцией «isNum»), равным 1 или 2. Если это не так, выводим сообщение об ошибке и завершаем программу.

Используем конструкцию switch-case. Значение кейсов будет выбираться в зависимости от значения переменной «ClassChoiceStr», приведенной к типу int. Только что мы сделали проверку. Если программа продолжила свою работу и дошла до конструкции switch-case, значит значение строки «ClassChoiceStr» = «1» или «2». Реализуем кейс для первого случая. Внутри него выводим название класса «Knight». Реализуем кейс для второго случая. Внутри него выводим название класса «Wizard». Больше в кейсе мы ничего не пишем.

Так как мы уже прибавили к «i» единицу, сейчас символ строки «gameSaves[i]» = уровню персонажа. Однако, как и в случае с номером класса, это может быть не так в случае, когда игрок изменял файл сохранений. Создадим переменную «LvlStr» типа string и сделаем аналогичную пошлому разу проверку (строки 164 – 173) (критерии завершения работы программы будут другими). После выводим на экран «LvlStr» (числовое значение уровня), следом выводим слово «Lvl», что переводится как «уровень».

Далее аналогично прошлым двум разам поступаем с количеством денег (строки 178 – 190), количеством здоровья (строки 192 – 204), количеством

маны (строки 206 – 217). В последнем случае не увеличиваем значение переменной «i» после нахождения знака-разделителя «:», так как мы уже достигли конца строки.

Записываем новую строку в переменную «gameSaves» (для того, чтобы при следующем прогоне просматривалась другая строка) и прибавляем к «score» единицу (так как мы считаем количество сохранений). Больше в цикле с постусловием while мы ничего не пишем.

После выполнения цикла while с постусловием в переменной «score» находится количество строк сохранения + 1 (так как мы увеличиваем «score» на 1 в самом конце цикла).

Создаем цикл с постусловием while, условием работы которого является принадлежность переменной «ChoiceSave» от 1 до score – 1 включительно. Внутри цикла создаем строку «choiceStr» и сразу же читаем ее. Если в строку ввели только цифры, то присваиваем переменной «ChoiceSave» значение переменной «choiceStr», приведенное к типу int. Если строка не соответствует описанным выше критериям, выводим сообщение о том, что ответ некорректен. Больше в цикле с постусловием while ничего не пишем.

Переоткроем файл «save.save» в потоке SaveRead для чтения (строки 232 – 233). Создадим цикл for, который будет считать количество строк до тех пор, пока прочтенная строка не будет тем сохранением, которое выбрал игрок (строки 235 – 237).

Далее создадим переменную-счетчик «i» и строковую переменную «name». Запишем в переменную «name» ник выбранного игрового персонажа (строки 241 – 244), следом записываем в переменную «ClassChoice» класс выбранного персонажа, приведенный к типу int. Аналогично в только что созданную переменную типа int «lvl» записываем из файла уровень выбранного персонажа. Так же поступаем с количеством денег (строки 250 – 255), количеством здоровья (строки 256 – 261), количеством маны (строки 262 – 266).

Далее используем конструкцию switch-case, в которой кейсы будут выбираться в зависимости от значения переменной «ClassChoice». Первый кейс будет выполняться при условии, что «ClassChoice» = 1. Внутри этого кейса мы, используя методы класса «knight» (их реализация будет показана позже) устанавливаем объекту «playerKnight» ник (строка 269), уровень (строка 270), здоровье и ману (строка 271), количество денег. Вторым кейсом выполняется в том случае, когда переменная «ClassChoice» = 2. Реализация второго кейса схожа с первым, за исключением того, что во втором кейсе мы используем методы класса «wizard» у объекта «playerWizard».

Далее очищаем консоль от присутствующего на ней текста, присваиваем переменной «flag» значение true (так как нам больше не нужно, чтобы цикл сохранения while(!flag) повторялся). Закрываем открытые потоки на чтение и на запись.

На этом заканчивается код загрузки существующего сохранения (см. рис. 11). Вместе с ним завершается и первая логическая часть файла «main.cpp» (создание новой игры и загрузка старой игры). Больше в цикле while (!flag) ничего не дописывается. Весь следующий код пишется вне while (!flag).

Рисунок 11 – код загрузки существующего сохранения

```

110 else if (gameStartChoice == 2) {
111     system("cls");
112     string gameSaves;
113     getline(SaveRead, gameSaves);
114     if (gameSaves == "") {
115         print("No saved games/nback?");
116         string crutch;
117         cin >> crutch;
118     }
119 }
120 else {
121     int score = 1;
122     print("Select a save cell: /n");
123     do {
124         int SeparatorCountCounter = 0;
125         for (int i = 0; i < gameSaves.length(); i++) {
126             if (gameSaves[i] == ':') SeparatorCountCounter++;
127             if (SeparatorCountCounter > 6) break;
128         }
129         if (SeparatorCountCounter != 6) {
130             cout << "SAVE ERROR";
131             exit(0);
132         }
133         print(score);
134         print(" - ");
135         int i = 0;
136         while (gameSaves[i] != ':') {
137             print(gameSaves[i]);
138             i++;
139         }
140         cout << " ";
141         i++;
142         string ClassChoiceStr = "";
143         while (gameSaves[i] != ':') {
144             ClassChoiceStr += gameSaves[i];
145             i++;
146         }
147         if (!isNum(ClassChoiceStr) or (ClassChoiceStr != "1" and ClassChoiceStr != "2")) {
148             print("SAVE ERROR");
149             exit(0);
150         }
151         switch (atoi(ClassChoiceStr.c_str())) {
152             case 1:
153                 print("Knight");
154                 break;
155             case 2:
156                 print("Wizard");
157                 break;
158         }
159         string LvlStr = "";
160         while (gameSaves[i] != ':') {
161             LvlStr += gameSaves[i];
162             i++;
163         }
164         if (!isNum(LvlStr) or atoi(LvlStr.c_str()) < 1) {
165             print("SAVE ERROR");
166             exit(0);
167         }
168         print(LvlStr);
169         print(" Lvl");
170         string moneyStr = "";
171         while (gameSaves[i] != ':') {
172             moneyStr += gameSaves[i];
173             i++;
174         }
175         if (!isNum(moneyStr) or atoi(moneyStr.c_str()) < 0) {
176             print("SAVE ERROR");
177             exit(0);
178         }
179         i++;
180         print("Money - ");
181         print(moneyStr);
182         print(" ");
183         string hpStr = "";
184         while (gameSaves[i] != ':') {
185             hpStr += gameSaves[i];
186             i++;
187         }
188         if (!isNum(hpStr) or atoi(hpStr.c_str()) < 0) {
189             print("SAVE ERROR");
190             exit(0);
191         }
192     } while (ChoiceSave < 1 or ChoiceSave > score - 1);
193     print("Incorrect choice: ");
194     SaveRead.close();
195     ifstream SaveRead("save.save");
196     for (int i = 0; i < ChoiceSave; i++) {
197         getline(SaveRead, gameSaves);
198     }
199     int i = 0;
200     string name = "";
201     while (gameSaves[i] != ':') {
202         name += gameSaves[i];
203         i++;
204     }
205     i++;
206     ClassChoice = gameSaves[i] - '0';
207     i += 2;
208     int lvl = gameSaves[i] - '0';
209     i += 2;
210     string moneyStr = "";
211     while (gameSaves[i] != ':') {
212         moneyStr += gameSaves[i];
213         i++;
214     }
215     i++;
216     string hpStr = "";
217     while (gameSaves[i] != ':') {
218         hpStr += gameSaves[i];
219         i++;
220     }
221     i++;
222     string mpStr = "";
223     while (gameSaves[i] != ':') {
224         mpStr += gameSaves[i];
225         i++;
226     }
227     switch (ClassChoice) {
228         case 1:
229             playerKnight.SetName(name);
230             playerKnight.SetLvl(lvl);
231             playerKnight.SetHpAndMp(atoi(hpStr.c_str()), atoi(mpStr.c_str()));
232             playerKnight.setMoney(atoi(moneyStr.c_str()));
233             break;
234         case 2:
235             playerWizard.SetName(name);
236             playerWizard.SetLvl(lvl);
237             playerWizard.setHpAndMp(atoi(hpStr.c_str()), atoi(mpStr.c_str()));
238             playerWizard.setMoney(atoi(moneyStr.c_str()));
239             break;
240     }
241     system("cls");
242     flag = 1;
243     SaveRead.close();
244     SaveWrite.close();
245 }

```

Выведем полные характеристики выбранного персонажа с помощью метода «GetSpecifications». Предложим игроку продолжить игру или же выйти из нее. В цикле while с постусловием будем считывать ответ до тех пор, пока пользователь не введет один из допустимых ответов («1» или «2») (см. рис. 12).

## Рисунок 12 – спецификации персонажа

```
290     if (ClassChoice == 1) playerKnight.GetSpecifications();
291     else playerWizard.GetSpecifications();
292
293     if (GameStartchoice == 1) print("Do you want to start your adventure ?/n1 - Yes / 2 - No ");
294     else print("Do you want to continue your adventure ?/n1 - Yes / 2 - No ");
295
296     do {
297         string choiceStr;
298         cin >> choiceStr;
299         if (isNum(choiceStr)) {
300             if (atoi(choiceStr.c_str()) == 2) exit(0);
301             if (atoi(choiceStr.c_str()) == 1) break;
302         }
303     } while (true);
```

## 5 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ СЛУЧАЙНЫХ ВСТРЕЧ

### 5.1 Проектирование системы случайных встреч

Спроектируем систему случайных встреч. Всего, как было показано ранее (рис. 2), в игре будет 5 видов противников. Противники должны отличаться по своим характеристикам (количество здоровья, наносимого урона), ведь если они будут одинаковыми, игрок не будет видеть между ними разницы. Однако стоит понимать, что нельзя сразу же давать игроку сражаться с самым сильным противником, наращивание характеристик должно происходить постепенно, в зависимости от силы персонажа, за которого играет персонаж. Таким образом, самым логичным вариантом будет разделить пять противников по уровням. На первом уровне будет встречаться самый слабый, на втором уровне самый слабый + тот, кто чуть сильнее и так далее. Выбор на каждом уровне между конкретными противниками можно реализовать с помощью рандома.

Так же стоит учитывать, что силы противников должны соответствовать ожиданиям игрока. Таким образом, противник «weak demon» просто не может быть сильнее противника «demon», ведь это противоречит логике.

Составим схему доступных противников по каждому уровню, где сверху вниз противники будут становиться сильнее (рис. 12).

Рисунок 12 – схема противников по уровням

1 уровень	2 уровень	3 уровень	4 уровень	5+ уровни
Enemy_thief	Enemy_thief	Enemy_thief	Enemy_thief	Enemy_thief
	Enemy_skeleton_archer	Enemy_skeleton_archer	Enemy_skeleton_archer	Enemy_skeleton_archer
		Enemy_weak_demon	Enemy_weak_demon	Enemy_weak_demon
			Enemy_demon	Enemy_demon
				Enemy_death_claw

### 5.2 Реализация системы случайных встреч на языке C++

Создадим по одному объекту каждого класса врагов (строчки 305 – 309), а также переменную «DefeatedEnemies» типа int, в которую будет записываться количество побежденных противников.

В зависимости от значения (true или false) проверки условия if будет выполняться одна из двух частей кода. Вторая часть кода полностью аналогична первой за исключением того, что там функции вызываются с передаваемым объектом «playerWizard», а не «playerKnight». Условием проверки if будет «ClassChoice» = 1 (то есть, класс персонажа – «knight»). Дальнейший код будет записываться внутри этого if а.

Создадим бесконечный цикл while (true) (игра не должна заканчиваться до тех пор, пока игрок сам того не захочет), а так же предложим выйти из игры, если игрок того хочет (строки 315 – 316). Делаем это внутри цикла while (true) для того, чтобы игрок мог выйти из игры после каждого сражения с противником.

Создадим переменную «rand» типа time\_t. Следом за ней, создадим switch-case. Значение кейсов будет выбираться в зависимости от значения «playerKnight.GetLvl()» (то есть, в зависимости от уровня персонажа). Всего будет пять кейсов: первый работает при первом уровне, второй при втором, третий при третьем, четвертый при четвертом. Последних кейс будет «default», иными словами, он будет отрабатывать при всех остальных значениях. Внутри первого кейса вызываем функцию «battle» (будет реализована позже), в которую передаем объект нашего персонажа «playerKnight», объект единственного для первого уровня противника «thief», ссылку на переменную «DefeatedEnemies» и номер выбранного сохранения «ChoiceSave». Во втором кейсе нам нужно будет вызывать функцию battle с одним из двух противников, выбираемым случайно. Присвоим переменной «rand» остаток от деления времени компьютера в секундах и цифры 2. Создадим еще один switch-case с двумя кейсами: первый вызывается, когда «rand» = 1, второй же является кейсом по умолчанию. Внутри первого кейса вызываем функцию «battle» с противником «skeleton\_archer». Внутри кейса «default» - с противником «thief». В остальных трех кейсах поступаем аналогично этому, с тем лишь различием, что «rand» будет вычисляться по формуле «rand» = time(NULL) % количество противников на данном уровне. Соответственно, внутри

вложенных switch-сас`ов будет количество кейсов, равное количеству противников на уровне. Противники будут вызываться согласно схеме на рисунке 12.

Рисунок 13 – реализация случайных встреч для объекта «playerKnight»

```

313 while (true) {
314
315     print("/nTo exit the game, press 'ESC'. Press to continue/n/n");
316     if (_getch() == 27) exit(0);
317
318     time_t rand;
319     switch (playerKnight.GetLvl()) {
320     case 1:
321         Battle(playerKnight, thief, &DefeatedEnemies, ChoiceSave);
322         break;
323     case 2:
324         rand = time(NULL) % 2;
325         switch (rand) {
326         case 1:
327             Battle(playerKnight, skeleton_archer, &DefeatedEnemies, ChoiceSave);
328             break;
329             default:
330                 Battle(playerKnight, thief, &DefeatedEnemies, ChoiceSave);
331                 break;
332         }
333         break;
334     case 3:
335         rand = time(NULL) % 3;
336         switch (rand) {
337         case 1:
338             Battle(playerKnight, skeleton_archer, &DefeatedEnemies, ChoiceSave);
339             break;
340             case 2:
341                 Battle(playerKnight, weak_demon, &DefeatedEnemies, ChoiceSave);
342                 break;
343             default:
344                 Battle(playerKnight, thief, &DefeatedEnemies, ChoiceSave);
345                 break;
346         }
347         break;
348     case 4:
349         rand = time(NULL) % 4;
350         switch (rand) {
351         case 1:
352             Battle(playerKnight, skeleton_archer, &DefeatedEnemies, ChoiceSave);
353             break;
354             case 2:
355                 Battle(playerKnight, weak_demon, &DefeatedEnemies, ChoiceSave);
356                 break;
357             case 3:
358                 Battle(playerKnight, demon, &DefeatedEnemies, ChoiceSave);
359                 break;
360             default:
361                 Battle(playerKnight, thief, &DefeatedEnemies, ChoiceSave);
362                 break;
363         }
364         break;
365     default:
366         rand = time(NULL) % 5;
367         switch (rand) {
368         case 1:
369             Battle(playerKnight, skeleton_archer, &DefeatedEnemies, ChoiceSave);
370             break;
371             case 2:
372                 Battle(playerKnight, weak_demon, &DefeatedEnemies, ChoiceSave);
373                 break;
374             case 3:
375                 Battle(playerKnight, demon, &DefeatedEnemies, ChoiceSave);
376                 break;
377             case 4:
378                 Battle(playerKnight, death_claw, &DefeatedEnemies, ChoiceSave);
379                 break;
380             default:
381                 Battle(playerKnight, thief, &DefeatedEnemies, ChoiceSave);
382                 break;
383         }
384         break;
385     }
386 }
387

```

Как говорилось ранее, для объекта «playerWizard» код будет аналогичным изображенному на рисунке 13.



## 6 РЕАЛИЗАЦИЯ ВЫЗЫВАЕМЫХ ФУНКЦИЙ В ФАЙЛЕ «FUNCTIONS.CPP»

### 6.1 Реализация функции isNum()

Функция «isNum()» осуществляет проверку на то, является ли переданная в функцию строка числом.

Сначала подключим все необходимые библиотеки и установим пространство имен std.

Далее объявим функцию «isNum()» с возвращаемым значением типа bool и одним получаемым параметром «choice» строкового типа. Реализуем функцию.

Объявим внутри функции переменные целочисленного типа «i» и «score», обнулим их. Создадим цикл for, который будет проходить по значениям переменной «i» от нуля до размера строки. Внутри цикла с помощью функции «isdigit()» проверяем символ строки «choice[i]» на то, является ли он цифрой. Если это так, увеличиваем значение переменной «score» на единицу. После выполнения цикла происходит еще одна проверка: «i» = «score». Эта проверка выдает истину в том случае, если строка состоит только из цифр. Если проверка выдала истину, функция возвращает логическую единицу, иначе – логический ноль (см. рис. 14).

Рисунок 14 – реализация функции «isNum()»

```
1  #include <iostream>
2  #include <string>
3  #include <Windows.h>
4  #include <conio.h>
5  #include <cstdlib>
6  using namespace std;
7
8  bool isNum(string choice) {
9      int i = 0;
10     int score = 0;
11     for (i; i < choice.size(); i++) {
12         if (isdigit(choice[i])) score++;
13     }
14     if (i == score) return 1;
15     else {
16         return 0;
17     }
18 }
```

## 6.2 Реализация функции correctName()

Функция «correctName()» нужна для отбраковки недопустимых ников. В нашем случае, такими являются ники, содержащие символ-разделитель «:» или же просто слишком длинные ники (например, больше 10 символов).

Объявим функцию «correctName()» с возвращаемым значением типа bool и одним получаемым параметром «name» строкового типа. Реализуем функцию.

Создадим цикл, который будет проходить по значениям переменной «i» от нуля и до размера строки. Внутри цикла напишем условие (условие описано чуть выше), при котором функция вернет логический ноль (false). После for`а запишем «return 1». Если условие в цикле не выполнится, цикл закончится и функция возвратит единицу, в противном случае возвратиться ноль. Код функции можно наблюдать на рисунке 15.

Рисунок 15 – реализация функции «correctName()»

```
21 bool correctName(string name){  
22     for (int i = 0; i < name.length(); i++) {  
23         if (name[i] == ':' or name.length() > 10) return 0;  
24     }  
25     return 1;  
26 }
```

## 6.3 Реализация функции print()

Функция print() нужна для реализации побуквенного вывода текста с характерным звуком печати.

Объявим функцию «print()» без возвращаемого значения и с одним получаемым параметром «strOne» строкового типа. Реализуем функцию.

Создадим цикл, который будет проходить по значениям переменной «i» от нуля и до размера строки. Внутри for`а разместим проверку if с условием текущий символ «strOne[i]» != «\n» или следующий символ «strOne[i + 1]» != «\n» и количество символов строго больше индекса следующего символа. Иными словами, данная конструкция проверяет, нет ли «\n» в строке. Если нет (условие if выдает истину) выводим текущий символ, ставим паузу на 35мс.,

включаем звук частотой 14000 Гц. и длиной 1мс. Если же конструкция «\n» все-таки обнаружена в строке, выполняется переход на следующую строку и переменная «i» дополнительно увеличивается (так как «\n» занимает два символа, и ни один из них не должен выводиться).

Однако в данный момент функция «print()» может принимать только строки. Перегрузим функцию, чтобы она могла обрабатывать и другие типы данных.

В первой перегрузке функции у нас так же будет один входной параметр и она так же не будет иметь возвращаемого значения. Однако принимаемое значение будет называться «charMessage» и будет типа «char».

Внутри функции выводим символ «charMessage», ставим паузу на 35мс., включаем звук частотой 14000 Гц. и длиной 1мс.

Во второй перегрузке опять не будет возвращаемого значения, принимаемое значение будет типа int и будет называться «intMessage».

Внутри функции выводим переменную «intMessage», ставим паузу на 35мс., включаем звук частотой 14000 Гц. и длиной 1мс.

Код функции и ее перегрузок можно наблюдать на рисунке 16.

Рисунок 16 – реализация функции «print()» и ее перегрузок

```
28 void print(string strOne) {
29     for (int i = 0; i < strOne.length(); i++) {
30         if (strOne[i] != '/' or (strOne[i + 1] != 'n' and i + 1 < strOne.length())) {
31             cout << strOne[i];
32             Sleep(35);
33             Beep(14000, 1);
34         }
35         else {
36             cout << endl;
37             i++;
38         }
39     }
40 }
41
42 void print(char charMessage) {
43     cout << charMessage;
44     Sleep(35);
45     Beep(14000, 1);
46 }
47
48 void print(int intMessage) {
49     cout << intMessage;
50     Sleep(35);
51     Beep(14000, 1);
52 }
53 }
```

## 7 РЕАЛИЗАЦИЯ ИГРОВЫХ КЛАССОВ «KNIGHT» И «WIZARD»

### 7.1 Реализация игрового класса «knight»

Согласно рисунку 2, игровой класс «knight» должен быть реализован в файле «knight.cpp».

Сначала подключим библиотеки и установим пространство имен std. После этого создадим класс «knight» и инициализируем приватные целочисленные поля класса «hp» = 600, «maxHp» = 600, «mp» = 100, «maxMp» = 100, «damage» = 40, «lvl» = 1, «money» = 0 и строковую переменную «name» (см. рис. 17). Переменные являются приватными для реализации принципа инкапсуляции.

Рисунок 17 – объявление и инициализация приватных полей

```
9      private:
10          int hp = 600 ;
11          int maxHp = 600;
12          int mp = 100;
13          int maxMp = 100;
14          int damage = 40;
15          int lvl = 1;
16          int money = 0;
17          string name;
```

Далее будут объявляться публичные методы класса.

Метод «healing()» не имеет возвращаемого значения и не принимает никакие параметры. Он увеличивает количество здоровья, которое у персонажа сейчас на 300. Если после увеличения текущее количество здоровья становится больше, чем максимальное количество здоровья, мы присваиваем текущему количеству здоровья максимальное количество здоровья. Так же следует уменьшить количество денег персонажа (строка 23). Код метода на рисунке 18.

Рисунок 18 – реализация публичного метода «healing()»

```
19 public:
20 void healing() {
21     hp += 300;
22     if (hp > maxHp) hp = maxHp;
23     money -= 20;
24 }
```

Метод «SetName()» не имеет возвращаемого значения, принимает строковую переменную «name». В данный момент в зоне видимости метода «SetName()» находится переменная «name», которую мы передали в метод при его вызове и поле «name», принадлежащее объекту. Нам необходимо присвоить последнему первое. Для этого используем ключевое слово «this->», которое позволяет обращаться к полю текущего класса (см. рис. 19).

Рисунок 19 – реализация публичного метода «SetName()»

```
27 void SetName(string name) {
28     this->name = name;
29 }
```

Метод «MpRecovery()» не имеет возвращаемого значения и не принимает никакие параметры. Его тело будет пустым. По сути, данный метод нужен лишь для того, чтобы класс «knight» имел все те же поля и переменные, что и класс «wizard», реализуя тем самым принцип полиморфизма.

Метод «newLevel()» не имеет возвращаемого значения и не принимает никакие параметры. Он будет увеличивать уровень персонажа, повышая соответствующие характеристики. Увеличим значение поля lvl на единицу. Новый урон, который наносит персонаж, будет рассчитываться по формуле: «damage += 0.1 \* lvl \* damage». Максимальное количество здоровья также изменится, как и максимальное количество маны (строчки 35 – 37). Так же можно реализовать восстановления части здоровья и маны (строчки 38 – 39). Если после восстановления текущее значение маны или здоровья стало больше максимального, нужно приравнять его к максимальному. Код метода «newLevel()» на рисунке 20.

Рисунок 20 – реализация публичного метода «newLevel()»

```

33 void newLevel() {
34     lvl++;
35     damage += 0.1 * lvl * damage;
36     maxHp += 0.1 * lvl * hp;
37     maxMp += 0.1 * lvl * mp;
38     hp += 0.3 * maxHp;
39     mp += 0.3 * maxMp;
40     if (hp > maxHp) hp = maxHp;
41     if (mp > maxMp) mp = maxMp;
42 }

```

Метод «setHpAndMp()» не имеет возвращаемого значения, принимает целочисленные переменные «hp» и «mp». Внутри метода мы присваиваем полям класса «mp» и «hp» переданные переменные «mp» и «hp» (см. рис. 21).

Рисунок 21 – реализация публичного метода «setHpAndMp()»

```

44 void setHpAndMp(int hp, int mp) {
45     this->hp = hp;
46     this->mp = mp;
47 }

```

Метод «SetLvl()» не имеет возвращаемого значения, принимает целочисленную переменную «lvl». Внутри метода мы присваиваем полю класса «lvl» значение переданной в метод переменной «lvl» (строка 50). Далее в цикле for мы повышаем значение полей «damage», «maxHp» и «maxMp» до необходимых для уровня значений (строки 53 – 55). Код метода представлен на рисунке 22.

Рисунок 22 – реализация публичного метода «SetLvl()»

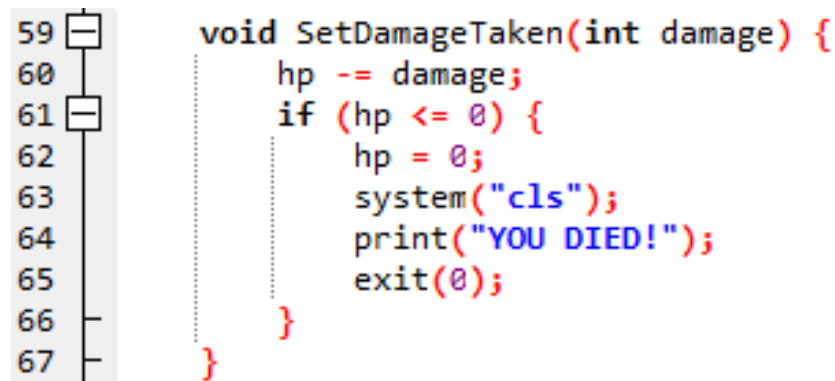
```

49 void SetLvl(int lvl) {
50     this->lvl = lvl;
51
52     for (int i = 2; i <= this->lvl; i++) {
53         damage += 0.1 * i * damage;
54         maxHp += 0.1 * i * hp;
55         maxMp += 0.1 * i * mp;
56     }
57 }

```

Метод «SetDamageTaken()» не имеет возвращаемого значения, принимает целочисленную переменную «damage». Внутри метода мы вычитаем из текущего количества здоровья «hp» значение переданной переменной «damage». Если в результате этого значение текущего здоровья «hp»  $\leq 0$ , приравниваем «hp» = 0, очищаем консоль, выводим сообщение о проигрыше и выходим из программы. Код метода представлен на рисунке 23.

Рисунок 23 – реализация публичного метода «SetDamageTaken ()»



The image shows a UML diagram on the left and C++ code on the right. The UML diagram has a vertical line with boxes at lines 59, 61, and 66, and a horizontal line at line 67. The C++ code is as follows:

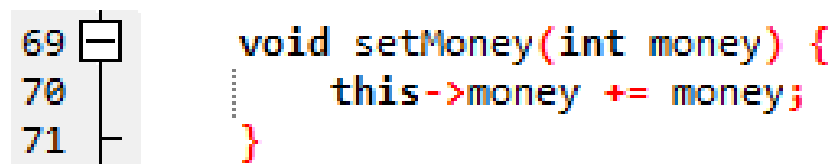
```

void SetDamageTaken(int damage) {
    hp -= damage;
    if (hp <= 0) {
        hp = 0;
        system("cls");
        print("YOU DIED!");
        exit(0);
    }
}

```

Метод «setMoney()» не имеет возвращаемого значения, принимает целочисленную переменную «money». Внутри метода мы увеличиваем значение поля класса «money» на переданное в метод целочисленное значение «money» (см. рис. 24).

Рисунок 24 – реализация публичного метода «SetDamageTaken ()»



The image shows a UML diagram on the left and C++ code on the right. The UML diagram has a vertical line with boxes at lines 69, 70, and 71, and a horizontal line at line 71. The C++ code is as follows:

```

void setMoney(int money) {
    this->money += money;
}

```

Метод «GetMp()» возвращает значение поля «mp», метод «GetHp()» – поля hp, «GetMaxHp» – поля «maxHp», «GetDamage» – поля «damage», «GetMoney» – поля «money», «GetName» – поля «name», «GetLvl» – поля «lvl», «GetClass» – номер класса (см. рис. 25).

Рисунок 25 – реализация геттеров класса «knight»

```

73  int GetMp() {
74      return mp;
75  }
76
77  int GetMaxMp() {
78      return maxMp;
79  }
80
81  int GetHp() {
82      return hp;
83  }
84
85  int GetMaxHp() {
86      return maxHp;
87  }
88
89  int GetDamage() {
90      return damage;
91  }
92
93  int GetMoney() {
94      return money;
95  }
96
97  string GetName() {
98      return name;
99  }
100
101  int GetLvl() {
102      return lvl;
103  }
104
105  int GetClass() {
106      return 1;
107  }

```

Метод «GetSpecifications()» не имеет возвращаемого значения, не принимает параметры. Внутри метода выводим через функцию «print» ник персонажа «name», уровень персонажа «lvl», текущее здоровье персонажа «hp», максимальное здоровье персонажа «maxHp», текущую ману персонажа «mp», максимальную ману персонажа «maxMp», урон персонажа «damage», количество денег персонажа «money» (см. рис. 26).

Рисунок 26 – реализация публичного метода «GetSpecifications()»

```

113 void GetSpecifications() {
114     print("Name - ");
115     print(name);
116     print("/nLvl - ");
117     print(lvl);
118     print("/nHP - ");
119     print(hp);
120     cout << "/";
121     print(maxHp);
122     print("/nMP - ");
123     print(mp);
124     cout << "/";
125     print(maxMp);
126     print("/nDamage - ");
127     print(damage);
128     print("/nMoney - ");
129     print(money);
130     print("/n");
131 }

```



## 7.2 Реализация игрового класса «wizard»

Большая часть кода класса «wizard» аналогична классу «knight», однако есть несколько исключений.

Во-первых, отличаются значения частных полей (см. рис. 27)

Рисунок 26 – значения частных полей класса «wizard»

```
9      private:
10         int hp = 150;
11         int maxHp = 150;
12         int mp = 600;
13         int maxMp = 600;
14         int damage = 200;
15         int lvl = 1;
16         int money = 0;
17         string name;
```

Во-вторых, метод «MpRecovery()» класса «wizard», в отличие от аналогичного метода класса «knight» не имеет пустого тела. Внутри метода мы увеличиваем значение переменной «mp» на четверть от максимального значения маны «maxMp» (см. рис. 27)

Рисунок 27 – реализация публичного метода «MpRecovery()»

```
31      void MpRecovery() {
32          mp += (maxMp / 4);
33          if (mp > maxMp) mp = maxMp;
34      }
```

## 8 РЕАЛИЗАЦИЯ КЛАССОВ ИГРОВЫХ ПРОТИВНИКОВ

Реализуем класс «Enemy\_thief». Подключим библиотеки и установим пространство имен std. После этого создадим класс «Enemy\_thief». Внутри класса объявим и инициализируем приватные поля «hp» = 130, «damage» = 15, «money» = 17. Далее реализуем публичные методы класса.

Метод «Reborn()» устанавливает полю «hp» его изначальное значение, не имеет возвращаемого значения, не принимает входные данные.

Метод «SetDamageTaken()» не имеет возвращаемого значения, принимает переменную «damage» типа «int». Уменьшает текущее здоровье на переданное в метод значение «damage». Если текущее здоровье становится меньше нуля, присваивает ему ноль.

Метод «GetHp()» возвращает значение поля «hp», «GetDamage» – поля «gamage», «GetMoney» – поля «money», «GetClass» возвращает номер класса (номер класса равен номеру уровня игрового персонажа, на котором этот противник впервые может появиться), «GetClassName» – имя противника. На рисунке 28 можно наблюдать реализацию класса «Enemy\_thief».

Рисунок 28 – реализация класса «Enemy\_thief»

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <Windows.h>
5  #include <conio.h>
6  #include <cstdlib>
7  #include <time.h>
8  using namespace std;
9
10 class Enemy_thief {
11 private:
12     int hp = 130;
13     int damage = 15;
14     int money = 17;
15 public:
16     void Reborn() {
17         hp = 130;
18     }
19     void SetDamageTaken(int damage) {
20         hp -= damage;
21         if (hp < 0) hp = 0;
22     }
23
24     int GetHp() {
25         return hp;
26     }
27
28     int GetDamage() {
29         return damage;
30     }
31
32     int GetMoney() {
33         return money;
34     }
35     int GetClass() {
36         return 1;
37     }
38     string GetClassName() {
39         return "Thief";
40     }
41 };
```

Остальные классы противников реализуются аналогично за тем лишь исключением, что значения приватных полей отличаются у каждого класса. Так же отличается возвращаемое имя в методе «GetClassName()» (оно зависит от названия класса) и присваиваемое полю «hp» значение (оно, как была сказано выше, зависит от изначального значения поля «hp»).

## 9 РЕАЛИЗАЦИЯ СИСТЕМЫ СРАЖЕНИЙ С ПРОТИВНИКОМ В ФАЙЛЕ «BATTLE.CPP»

### 9.1 Сражение с противником

Подключим библиотеки и установим пространство имен std.

Данная функция должна принимать в качестве параметров объекты всех тех классов, что были описаны ранее. Для этого используем template. По сути, мы создаем шаблон функции, который будет принимать помимо прочих данных ссылку на объект «playerObject» класса «player» и объект «enemyObject» класса «enemy». Сами классы «player» и «enemy» определяются во время исполнения кода, после того, как мы подадим в функцию необходимые параметры.

Так же помимо вышеописанного функция принимает указатель на переменную «DefeatedEnemies» и переменную «ChoiceSave» (типа int).

Внутри функции выводим с помощью функции «print» сообщение о том, что игрок встретил противника, выводим имя этого противника и предлагаем игроку попытаться сбежать. Далее мы создаем логическую переменную «EscapeFlag», которая будет отвечать за то, пытался ли игрок совершить попытку побега. Далее мы создаем цикл бесконечный while(true) с постусловием, внутри которого считываем только что созданную внутри цикла строковую переменную «EscapeFlag». Если проверка введенного значения функцией «isNum()» выдает истину (то есть ввели число), то мы дополнительно проверяем, какое число ввели. Если была введена единица (то есть игрок попытается сбежать), то мы присваиваем «EscapeFlag» логическую единицу и выходим из цикла, если же был введен ноль, то мы оставляем переменную «EscapeFlag» равной нулю и тоже выходим из цикла. Если же переменная не равна ни одному из этих значений, мы выводим сообщение о том, что выбор некорректен и цикл продолжается.

Далее, как и в прошлый раз, создаем переменную «rand», в которую записываем время компьютера в секундах. Если остаток от деления «rand» на

2 равен единице и игрок совершал попытку побега, что выводим сообщение о том, что игроку удалось сбежать и возвращаем значение ноль (выходим из функции). В противном же случае выводим сообщение о том, что попытка побега провалилась (см. рис. 29).

Рисунок 29 – реализация кода попытки побега

```
13      print("You met a " + enemyObject.GetClassName() + ". Try to escape ? 1 - Yes / 2 - NO /n");
14
15      bool EscapeFlag = 0;
16
17      do {
18
19          string choiceStr;
20          cin >> choiceStr;
21          if (isNum(choiceStr)) {
22              if (atoi(choiceStr.c_str()) == 1) {
23                  EscapeFlag = 1;
24                  break;
25              }
26              if (atoi(choiceStr.c_str()) == 2) {
27                  break;
28              }
29          }
30          print("Incorrect choice: ");
31
32      } while (true);
33
34      time_t rand = time(NULL);
35      if (rand % 2 == 1 and EscapeFlag) {
36          print("You have successfully escaped !/n");
37          return 0;
38      }
39      else if (EscapeFlag) print("A failed escape attempt!/n/n/n");
```

Далее создаем бесконечный цикл с предусловием while (true). Внутри цикла помещаем условие, пытался ли игрок бежать. Если это так, то выводим, сообщение, что враг нас атакует. Ставим паузу на 90 мс. Далее выводим сообщение о том, что получен урон. Создаем целочисленную переменную «PlayerTakeGamage» и присваиваем ей силу атаки противника (строка 48), затем устанавливаем нанесенный урон игроку (строка 49). После выводим на экран нанесенный урон (переменную «PlayerTakeGamage») и снова ставим паузу на 90 мс. Далее выводим информацию о текущем состоянии здоровья (см. рис. 30)

Рисунок 30 – атака противника

```
43 ☐ if (EscapeFlag == 1) {  
44     print("The " + enemyObject.GetClassName() + " is attacking you!\n");  
45  
46     Sleep(90);  
47     print("Damage received: ");  
48     int PlayerTakeGamage = enemyObject.GetDamage();  
49     playerObject.SetDamageTaken(PlayerTakeGamage);  
50     print(PlayerTakeGamage);  
51     Sleep(90);  
52     print("\nYour health: ");  
53     print(playerObject.GetHp());  
54     cout << "/";  
55     print(playerObject.GetMaxHp());  
56     print("\n");  
57     EscapeFlag = 0;  
58     Sleep(90);  
59 }
```

Далее в зависимости от класса мы предлагаем игроку сделать свой ход. Если персонаж игрока принадлежит классу «knight», игрок может восстановить здоровье за 20 монеток или атаковать. Если же персонаж принадлежит классу «wizard», к возможностям рыцаря прибавляется так же вариант восстановления здоровья. Так как код для класса рыцаря является урезанным кодом для класса мага, рассмотрим код для класса мага.

Выводим на экран сообщение, в котором игроку предлагается атаковать, восстановить здоровье или восстановить ману. Создаем бесконечный цикл с постусловием `while(true)`. Внутри цикла считываем только что созданную локальную строковую переменную «choiceStr». В зависимости от значения переменной «choiceStr» мы либо атакуем (1), либо лечимся (2), либо восстанавливаем ману (3).

Когда переменная «choiceStr» = 2 мы делаем дополнительную проверку, что количество денег игрока  $\geq 20$ . Если это так, то вызываем у объекта игрока метод «healing()», ставим паузу на 90мс., выводим количество здоровья, опять ставим паузу на 90мс., выводим количество денег. В противном же случае выводим сообщение о том, что денег недостаточно и показываем игроку его количество денег (см. рис. 31).

Рисунок 31 – код лечения

```

111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
    if (atoi(choiceStr.c_str()) == 2) {
        if (playerObject.GetMoney() >= 20) {
            playerObject.healing();
            Sleep(90);
            print("Your health: ");
            print(playerObject.GetHp());
            cout << "/";
            print(playerObject.GetMaxHp());
            Sleep(90);
            print("/nYour money: ");
            print(playerObject.GetMoney());
            print("/n/n");
            break;
        }
        else {
            print("You don't have enough coins (");
            print(playerObject.GetMoney());
            print("/20");
        }
    }
    else

```

Если значение переменной «choiceStr» = 1, выводим сообщение о том, что урон нанесен. Создаем целочисленную переменную «EnemyTakeGamage» и присваиваем ей силу атаки игрового персонажа. Выводим сообщение с количеством нанесенного урона, ставим паузу на 90мс. Затем выводим значение маны (у рыцаря мана не выводится). После этого устанавливаем противнику нанесенный урон (строка 143) и выводим на экран здоровье противника, ставим паузу на 90мс (см. рис. 32).

Рисунок 32 – код атаки игрового персонажа


```

130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
    } else
    if (atoi(choiceStr.c_str()) == 1) {
        print("/n");
        print("You're doing damage: ");
        int EnemyTakeGamage = playerObject.GetDamage();
        print(EnemyTakeGamage);
        Sleep(90);
        print("/nYour mana: ");
        print(playerObject.GetMp());
        cout << "/";
        print(playerObject.GetMaxMp());
        print("/n");
        Sleep(90);
        enemyObject.SetDamageTaken(EnemyTakeGamage);
        print("/n" + enemyObject.GetClassName() + " Health: ");
        print(enemyObject.GetHp());
        print("/n/n");
        Sleep(90);
        break;
    }
    else

```

Если же значение переменной «choiceStr» = 3 (такое условие есть только у класса «wizard»). Применяем к объекту игрока метод «MpRecovery()», выводим на экран количество маны (см. рис. 33).


Рисунок 33 – код восстановления маны

```
150       if (atoi(choiceStr.c_str()) == 3) {  
151     playerObject.MpRecovery();  
152     print("Your Mp: ");  
153     print(playerObject.GetMp());  
154     cout << "/";  
155     print(playerObject.GetMaxMp());  
156     print("/n/n");  
157     Sleep(90);  
158     break;  
159 }
```

Далее код пишется вне цикла while (true) выбора хода.

Если здоровье противника опустилось ниже нуля, выводим информацию о том, что противник убит. Увеличиваем количество побежденных противников и выходим из цикла. В противном же случае пишем код атаки противника (см. рис. 30, кроме 43 и 59 строк).

Рисунок 33 – предполагаемая смерть противника и его новая атака

```
165       if (enemyObject.GetHp() <= 0) {  
166     print("/n/nYou killed a " + enemyObject.GetClassName() + "!/n/n");  
167     enemyObject.Reborn();  
168     (*DefeatedEnemies)++;  
169     break;  
170 }  
171  
172 if (enemyObject.GetClass() == 1) print("The " + enemyObject.GetClassName() + " is attacking you!/n");  
173 Sleep(90);  
174 print("Damage received: ");  
175 int PlayerTakeGamage = enemyObject.GetDamage();  
176 playerObject.SetDamageTaken(PlayerTakeGamage);  
177 print(PlayerTakeGamage);  
178 Sleep(90);  
179 print("/nYour health: ");  
180 print(playerObject.GetHp());  
181 cout << "/";  
182 print(playerObject.GetMaxHp());  
183 print("/n");  
184 EscapeFlag = 0;  
185 Sleep(90);
```

Далее код пишется вне цикла while (true) сражения с противником, так как мы реализовали взаимные атаки персонажа и врага и больше нам не нужно ничего писать в данном цикле.

Выведем на экран информацию о полученных деньгах и о текущем балансе.



## 9.2 Сохранение игры при достижении нового уровня

Игра будет сохраняться при достижении игровым персонажем нового уровня. Достижение нового уровня будет осуществляться с помощью проверки `if` с условием, что количество побежденных врагов «`DefeatedEnemies`» больше, чем текущий уровень игрока умножить на два.

Если это условие выполняется, то ставим паузу на 90мс., затем выводим сообщение о том, что получен новый уровень. Применяем метод «`newLevel()`» к объекту игрока, выводим полученный уровень.

Далее открываем на запись файл «`SaveGameHelper.save`» (это временный файл, который создается на время сохранения прогресса, сразу же после завершения сохранения он удаляется, поэтому он не был изображен на рисунке 2) и на чтение файл «`save.save`» в потоках «`SaveGameWriteHelper`» и «`SaveRead` соответственно», создаем пустую строковую переменную «`Buffer`».

Если выбранное для записи сохранение не является первым в списке, то мы записываем первую строчку из файла «`save.save`» в файл «`SaveGameHelper.save`» с помощью переменной «`Buffer`».

Далее создаем цикл `for`, который проходит от «`i`» = 0 до «`ChoiceSave`» – 2 (исходя из условий цикла, программа заходит в него только в том случае, когда «`ChoiceSave`» > 2). В цикле записываем переписываем переменные из файла «`save.save`» в файл «`SaveGameHelper.save`».

После цикла `for` считываем строку из файла «`save.save`» в переменную «`Buffer`». Если выбранное сохранение не является первым, записываем в файл «`SaveGameHelper.save`» символ переноса строки, имя персонажа, знак разделитель, класс персонажа, знак разделитель, уровень персонажа, знак разделитель, количество денег, знак разделитель, количество маны, знак разделитель. В противном случае записываем все то же самое, но без знака переноса строки.

Получается, что вместо сохранения под номером «`ChoiceSave`» в файл «`SaveGameHelper.save`» записывается новое сохранение с текущими данными персонажа.

Далее в цикле с предусловием while, который продолжается до тех пор, пока поток чтения «SaveRead» не пуст, переписываем оставшиеся сохранения из файла «save.save» в файл «SaveGameHelper.save». Закрываем потоки «SaveGameWriteHelper», и «SaveRead», удаляем файл «save.save».

Такая сложная система с построчной записью сохранений в файл «SaveGameHelper.save» в зависимости от номера сохранения нужна для того, чтобы в файле не возникало пустых строк, что критически важно для корректной работы программы.

Далее создаем поток записи файла «save.save» под названием «SaveWrite», тем самым создавая его, и поток чтения файла «SaveGameHelper.save» с названием «SaveGameReadHelper».

Считываем строчку из файла «SaveGameHelper.save» и записываем в файл «save.save».

Далее в цикле while, который продолжается до тех пор, пока поток чтения «SaveRead» не пуст, переписываем оставшиеся сохранения из файла «SaveGameHelper.save» в файл «save.save», не забывая ставить символы переноса строки.

Разделение записи на первую строку и оставшиеся крайне важно, потому что в противном случае появится одна пустая строка.

Закрываем потоки «SaveWrite» и «SaveGameReadHelper», удаляем временный файл «SaveGameHelper.save». Обнуляем количество побежденных врагов.

Реализация сохранения игры при достижении нового уровня показана на рисунке 34.

Рисунок 34 – реализация сохранения игры при достижении нового

```

196 if (*DefeatedEnemies >= playerObject.GetLv1() * 2) {
197     Sleep(90);
198     print("/nYou've got a new level: ");
199     playerObject.newLevel();
200     print(playerObject.GetLv1());
201     print("/n/n");
202
203
204     ofstream SaveGameWriteHelper("SaveGameHelper.save");
205     ifstream SaveRead("save.save");
206     string Buffer = "";
207
208     if (ChoiceSave != 1) {
209         getline(SaveRead, Buffer);
210         SaveGameWriteHelper << Buffer;
211     }
212
213     for (int i = 0; i < ChoiceSave - 2; i++) {
214         getline(SaveRead, Buffer);
215         SaveGameWriteHelper << "\n" << Buffer;
216     }
217     getline(SaveRead, Buffer);
218
219     if (ChoiceSave != 1) {
220         SaveGameWriteHelper << endl << playerObject.GetName() <<
221             ":" + to_string(playerObject.GetClass()) + ":" <<
222             playerObject.GetLv1() << ":" << playerObject.GetMoney() <<
223             ":" << playerObject.GetHp() << ":" << playerObject.GetMp() << ":";
224     }
225     else {
226         SaveGameWriteHelper << playerObject.GetName() <<
227             ":" + to_string(playerObject.GetClass()) + ":" <<
228             playerObject.GetLv1() << ":" << playerObject.GetMoney() << ":" <<
229             playerObject.GetHp() << ":" << playerObject.GetMp() << ":";
230     }
231
232 while (!SaveRead.eof()) {
233     getline(SaveRead, Buffer);
234     SaveGameWriteHelper << "\n" << Buffer;
235 }
236
237 SaveGameWriteHelper.close();
238 SaveRead.close();
239
240 remove("save.save");
241
242 ofstream SaveWrite("save.save");
243 ifstream SaveGameReadHelper("SaveGameHelper.save");
244
245 getline(SaveGameReadHelper, Buffer);
246 SaveWrite << Buffer;
247
248 while (!SaveGameReadHelper.eof()) {
249     getline(SaveGameReadHelper, Buffer);
250     SaveWrite << endl << Buffer;
251 }
252 SaveWrite.close();
253 SaveGameReadHelper.close();
254
255 remove("SaveGameHelper.save");
256
257 *DefeatedEnemies = 0;
258 }

```

## **10 ВЫВОДЫ**

Удалось реализовать текстовую консольную пошаговую игру жанра RPG. Работоспособность игры демонстрируется в приложенном в «дополнительных материалах» видеоролике.

## **11 ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ**

1. Демонстрация работоспособности игры : [сайт].

– URL: <https://youtu.be/3II7NMzHOjI> (дата обращения: 25.11.21). –

Режим доступа: по ссылке. – Вид: электронный.