**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Расчетно-графическое задание

Дисциплина: Объектно-ориентированное программирование

по теме “Консольная РПГ ”

Выполнил: ст. группы ПВ-21  
Ковалев Павел Александрович

Проверил: Буханов Д.Г.

**Белгород 2020**

Оглавление

[**Цель работы** 3](#_Toc54617570)

[**Постановка задачи** 3](#_Toc54617571)

[**Ролевая система** 3](#_Toc54617572)

[**Необходимые элементы** 3](#_Toc54617573)

[**Важные элементы** 3](#_Toc54617574)

[**Исследование** 3](#_Toc54617575)

[**Необходимые элементы** 3](#_Toc54617576)

[**Важные элементы** 4](#_Toc54617577)

[**Боевая система** 4](#_Toc54617578)

[**Важные элементы** 4](#_Toc54617579)

[**Диаграмма классов** 4](#_Toc54617580)

[**Листинг программы** 5](#_Toc54617581)

[**Stlinclude.h** 5](#_Toc54617582)

[**gui\_functions.h** 5](#_Toc54617583)

[**Gui\_functions.cpp** 5](#_Toc54617584)

[**darr.h** 6](#_Toc54617585)

[**Main.cpp** 9](#_Toc54617586)

[**Game.h** 10](#_Toc54617587)

[**Game.cpp** 10](#_Toc54617588)

[**Character.h** 10](#_Toc54617589)

[**Character.cpp** 10](#_Toc54617590)

[**Item.h** 10](#_Toc54617591)

[**Item.cpp** 10](#_Toc54617592)

[**Inventory.h** 10](#_Toc54617593)

[**Inventory.cpp** 10](#_Toc54617594)

[**Weapon.h** 10](#_Toc54617595)

[**Weapon.cpp** 10](#_Toc54617596)

[**Armor.h** 10](#_Toc54617597)

[**Armor.cpp** 10](#_Toc54617598)

[**Event.h** 10](#_Toc54617599)

[**Event.cpp** 10](#_Toc54617600)

[**Enemy.h** 10](#_Toc54617601)

[**Enemy.cpp** 10](#_Toc54617602)

[**Результат работы программы** 10](#_Toc54617603)

**Цель работы:** Сделать консольную РПГ.

**Постановка задачи:** RPG - Role-Playing Game в приложении должны присутствовать role-play-system, exploration и battle system, а также должны реализованы некоторые их элементы:

## **Ролевая система**

Описывает способы создания, изменения и развития персонажей, повышающие их эффективность в игре.

### **Необходимые элементы**

* Игрок управляет одним (воплощение, аватар) или несколькими (группа, партия) уникальными персонажами. Это необходимое условие для всех ролевых игр. В отличие от чистых стратегий, его персонажи уникальны и имеют имя.
* Игрок постепенно улучшает характеристики и/или навыки (посредством внутриигровых значений, чаще всего очков опыта, получаемых за выполнение заданий, исследование мира, диалоги, сражения и так далее).
* В процессе прохождения происходят проверки характеристик и/или способностей и навыков.
* Персонажи могут улучшать свои характеристики и/или способности и навыки при помощи элементов снаряжения. Теоретически возможно создать компьютерную ролевую игру без снаряжения. Отсутствие элементов снаряжения может служить опознавательным знаком приключенческих игр, зачастую использующих инвентарь исключительно для предметов, которые применяются при решении головоломок.

### **Важные элементы**

* Игрок может создавать своих персонажей.
* Игрок должен планировать развитие персонажа (-жей). Этот элемент отражает стратегический аспект создания и развития персонажа, а также сочетания навыков в группе.
* Основным способом решения проблем, взаимодействия с игровым миром и преодоления препятствий является тактическое применение навыков/способностей персонажа/группы персонажей (навыки самого игрока вторичны). Если это условие не выполняется, то скорее всего, перед нами один из видов боевика, в котором решающее значение имеют навыки игрока, а не персонажа.

## **Исследование**

Описывает способы перемещения персонажа по игровому миру, всё, что он может найти, увидеть, с чем он может взаимодействовать. Например, игровые области, предметы и иные объекты.

### **Необходимые элементы**

* Персонаж игрока может взаимодействовать с игровым миром и находить новые игровые области.
* Персонаж игрока может находить предметы и хранить их в инвентаре. Теоретически возможно создать компьютерную ролевую игру без инвентаря.
* Персонаж игрока может находить источники информации. Компьютерная ролевая игра без поиска информации невозможна в принципе.

### **Важные элементы**

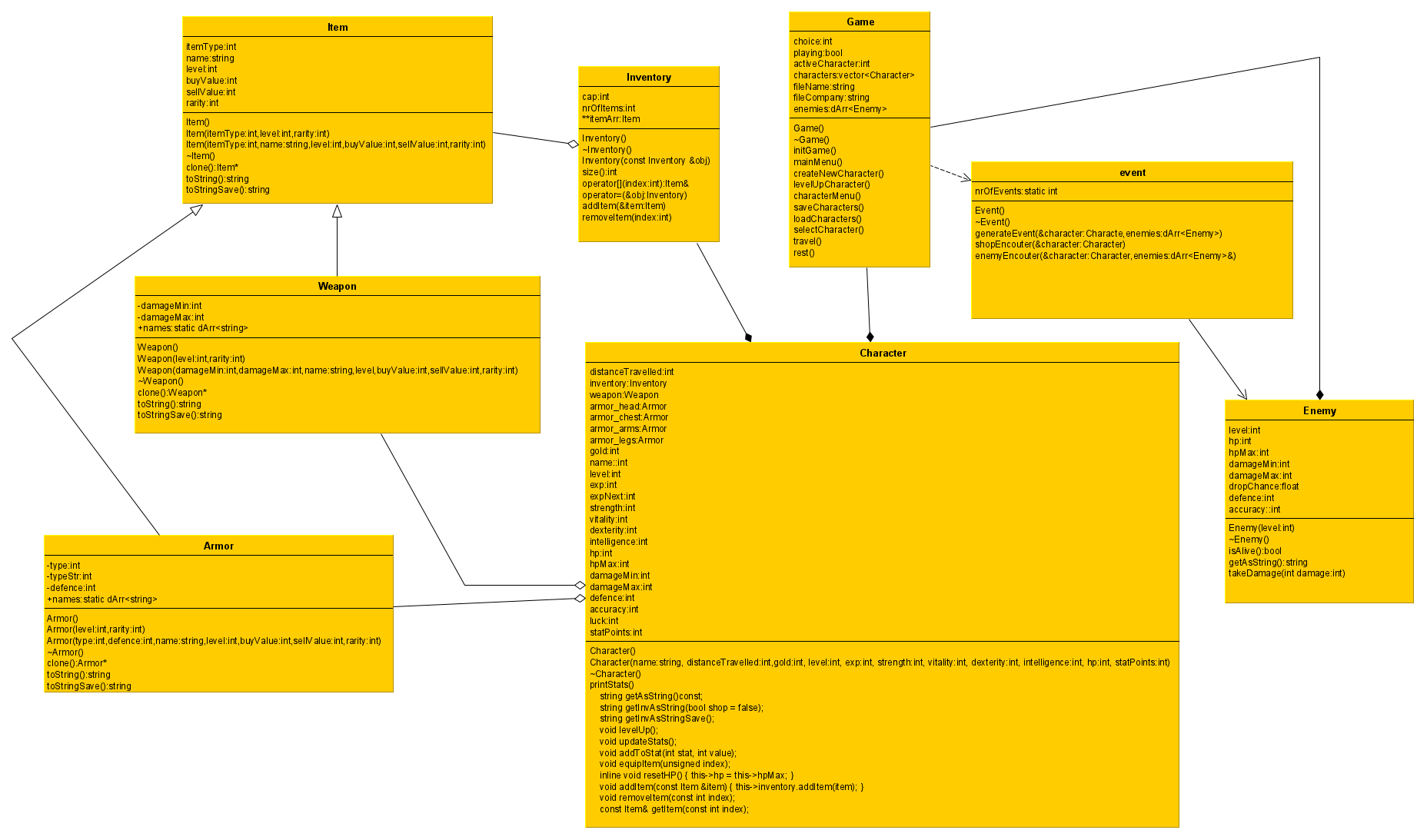
* В игре есть персонажи. Этот элемент отнесён к важным потому, что первые бродилки зачастую не имели других персонажей, кроме того, которым управлял игрок.
* Вы можете выбрать свой путь (хотя бы из нескольких вариантов).
* Персонаж может воздействовать на игровой мир (опускать рычаги, нажимать кнопки, открывать сундуки).
* Игровой мир воздействует на персонажа (-жей) (погода, ловушки, отравленные места).
* Существуют изначально недоступные области игрового мира, в которые можно попасть, лишь улучшив навыки персонажа, выполнив задание или решив головоломку (открыть замок, преодолеть препятствие, починить мост, развеять магию и так далее). Исследование игрового мира также должно зависеть от навыков.

## **Боевая система**

### **Важные элементы**

* Эффективность в сражении зависит от характеристик и навыков персонажа (количество повреждений, шанс попадания, возможность использования определённых видов оружия и так далее). Боевая система, не связанная с характеристиками и навыками персонажа, является верным признаком боевика, полагающегося исключительно на навыки игрока.
* В сражениях присутствует элемент случайности (броски внутриигровых кубиков). Практически во всех компьютерных ролевых играх присутствуют броски виртуальных кубиков и вероятностные функции.

# **Диаграмма классов**

****

# **Листинг программы**

##### **Stlinclude.h**

#ifndef STLINCLUDE\_H

#define STLINCLUDE\_H

#include <stdlib.h>

#include <string>

#include <ctime>

#include <vector>

#include <sstream>

#include <fstream>

#include <iostream>

#include <iomanip>

#include <math.h>

#include "dArr.h"

#include "gui\_functions.h"

#endif *//* *STLINCLUDE\_H*

##### **gui\_functions.h**

#ifndef GUI\_FUNCTIONS\_H

#define GUI\_FUNCTIONS\_H

#include <sstream>

#include <string>

#include <QTextCodec>

#include <QTextStream>

*namespace* **gui**

{

*const* std::string **menu\_title**(*const* std::string title);

*const* std::string **menu\_divider**(*const* unsigned amount = 30, *const* char symbol = '-');

*const* std::string **menu\_item**(*const* unsigned index, *const* std::string text);

}

#endif *//* *GUI\_FUNCTIONS\_H*

##### **Gui\_functions.cpp**

#include "gui\_functions.h"

*const* std::string gui::**menu\_title**(*const* std::string title)

{

std::stringstream ss;

ss << "= " << title << " ="

<< "\n"

<< "\n";

*return* ss.str();}

*const* std::string gui::**menu\_divider**(*const* unsigned amount, *const* char symbol)

{

std::stringstream ss;

ss << std::string(amount, symbol)

<< "\n"

<< "\n";

*return* ss.str();}

*const* std::string gui::**menu\_item**(*const* unsigned index, *const* std::string text)

{

std::stringstream ss;

ss << "- " << "(" << index << ") " << text

<< "\n";

*return* ss.str();}

##### **darr.h**

#ifndef DARR\_H

#define DARR\_H

*template*<*typename* T>

*class* **dArr**

{

*private*:

unsigned cap;

unsigned initialCap;

unsigned nrOfEl;

T\* \*arr;

void **expand**();

void **initialize**(unsigned from);

*public*:

**dArr**(unsigned size = 5);

**dArr**(*const* dArr& obj);

~**dArr**();

T& *operator*[] (*const* unsigned index);

void *operator*= (*const* dArr& obj);

unsigned **maxCap**();

unsigned **size**();

void **push**(*const* T element);

void **remove**(*const* unsigned index, bool ordered = *false*);

};

*template*<*typename* T>

dArr<T>::**dArr**(unsigned size)

{

*this*->initialCap = size;

*this*->cap = size;

*this*->nrOfEl = 0;

*this*->arr = *new* T\*[*this*->cap];

*this*->initialize(0);

}

*template*<*typename* T>

dArr<T>::**dArr**(*const* dArr& obj)

{

*this*->initialCap = obj.initialCap;

*this*->cap = obj.cap;

*this*->nrOfEl = obj.nrOfEl;

*this*->arr = *new* T\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfEl; i++)

{

*this*->arr[i] = *new* T(\*obj.arr[i]);

}

*this*->initialize(0);

}

*template*<*typename* T>

dArr<T>::~**dArr**()

{

*for* (size\_t i = 0; i < *this*->nrOfEl; i++)

{

*delete* *this*->arr[i];

}

*delete*[] arr;

}

*template*<*typename* T>

T& dArr<T>::*operator*[] (*const* unsigned index)

{

*if* (index < 0 || index >= *this*->nrOfEl)

*throw* "OUT OF BOUNDS INDEXING OPERATOR.";

*return* \**this*->arr[index];

}

*template*<*typename* T>

void dArr<T>::*operator*= (*const* dArr& obj)

{

*for* (size\_t i = 0; i < *this*->nrOfEl; i++)

{

*delete* *this*->arr[i];

}

*delete*[] arr;

*this*->initialCap = obj.initialCap;

*this*->cap = obj.cap;

*this*->nrOfEl = obj.nrOfEl;

*this*->arr = *new* T\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfEl; i++)

{

*this*->arr[i] = *new* T(\*obj.arr[i]);

}

*this*->initialize(0);

}

*template*<*typename* T>

void dArr<T>::**expand**()

{

*this*->cap \*= 2;

T\* \*tempArr = *new* T\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfEl; i++)

{

tempArr[i] = *this*->arr[i];

}

*delete*[]arr;

*this*->arr = tempArr;

*this*->initialize(*this*->nrOfEl);

}

*template*<*typename* T>

void dArr<T>::**initialize**(unsigned from)

{

*for* (size\_t i = from; i < *this*->cap; i++)

{

*this*->arr[i] = *nullptr*;

}

}

*template*<*typename* T>

unsigned dArr<T>::**maxCap**()

{

*return* *this*->cap;

}

*template*<*typename* T>

unsigned dArr<T>::**size**()

{

*return* *this*->nrOfEl;

}

*template*<*typename* T>

void dArr<T>::**push**(*const* T element)

{

*if* (*this*->nrOfEl >= *this*->cap)

*this*->expand();

*this*->arr[*this*->nrOfEl++] = *new* T(element);

}

*template*<*typename* T>

void dArr<T>::**remove**(*const* unsigned index, bool ordered)

{

*if* (index < 0 || index >= *this*->nrOfEl)

*throw* "OUT OF BOUNDS REMOVE.";

*if* (ordered)

{

*delete* *this*->arr[index];

*for* (size\_t i = 0; i < *this*->nrOfEl-1; i++)

{

*this*->arr[i] = *this*->arr[i + 1];

}

*this*->arr[--*this*->nrOfEl] = *nullptr*;

}

*else*

{

*delete* *this*->arr[index];

*this*->arr[index] = *this*->arr[*this*->nrOfEl - 1];

*this*->arr[--*this*->nrOfEl] = *nullptr*;

}

}

#endif *//* *DARR\_H*

##### **Main.cpp**

#include <QCoreApplication>

#include <game.h>

*//QString::fromUtf8().toLocal8Bit().data();*

int **main**(int argc, char \*argv[])

{

#ifdef Q\_OS\_WIN32

QTextCodec::setCodecForLocale(QTextCodec::codecForName("IBM 866"));

#endif

#ifdef Q\_OS\_LINUX

QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));

#endif

QCoreApplication app(*argc*, argv);

srand(time(NULL));

Game game;

game.initGame();

*while* (game.getPlaying())

{

game.mainMenu();

}

*return* app.exec();

}

##### **Game.h**

#ifndef GAME\_H

#define GAME\_H

#include "STLINCLUDE.h"

#include "Event.h"

*using* *namespace* std;

*class* **Game**

{

*private*:

int choice;

bool playing;

int activeCharacter;

std::vector<Character> characters;

std::string fileName;

dArr<Enemy> enemies;

*public*:

**Game**();

*virtual* ~***Game***();

void **initGame**();

void **mainMenu**();

void **createNewCharacter**();

void **levelUpCharacter**();

void **characterMenu**();

void **saveCharacters**();

void **loadCharacters**();

void **selectCharacter**();

void **travel**();

void **rest**();

*inline* bool **getPlaying**() *const* { *return* *this*->playing; }

};

#endif *//* *GAME\_H*

##### **Game.cpp**

#include "game.h"

Game::**Game**()

{

choice = 0;

playing = *true*;

activeCharacter = 0;

fileName = "characters.txt";

}

Game::~***Game***()

{

}

*//Functions*

void Game::**initGame**()

{

ifstream in;

in.open("characters.txt");

Weapon::initNames();

Armor::initNames();

*if* (in.is\_open())

*this*->loadCharacters();

*else*

{

createNewCharacter();

*this*->saveCharacters();

}

in.close();

}

void Game::**mainMenu**()

{

cout << QString::fromUtf8("Нажмите Enter, чтобы продолжить...").toLocal8Bit().data() << "\n";

cin.get();

system("CLS");

*if* (*this*->characters[activeCharacter].isAlive())

{

*if* (*this*->characters[activeCharacter].getExp() >=

*this*->characters[activeCharacter].getExpNext())

{

cout << QString::fromUtf8("Новый уровень доступен! \n\n").toLocal8Bit().data();

}

cout << QString::fromUtf8("= Главное меню =").toLocal8Bit().data() << "\n" << "\n";

cout << QString::fromUtf8("= Активный персонаж: ").toLocal8Bit().data() <<

*this*->characters[activeCharacter].getName() << " Nr: " <<

*this*->activeCharacter + 1 << "/" << *this*->characters.size() <<

" =" << "\n" << "\n";

cout << QString::fromUtf8("0: Выйти").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("1: Приключения").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("2: ВЫРАВНИВАНИЯ").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("3: Лагерь").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("4: Статистика персонажей").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("5: Создать нового персонажа").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("6: Выбрать персонажа").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("7: Сохранить персонажа").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("8: Загрузить персонажа").toLocal8Bit().data() << "\n";

cout << "\n";

cout << "\n" << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice > 9)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << "\n" << QString::fromUtf8("Ввод (0 - 8): ").toLocal8Bit().data();

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*switch* (*this*->choice)

{

*case* 0: *//QUIT*

playing = *false*;

*this*->saveCharacters();

*break*;

*case* 1: *//TRAVEL*

travel();

*break*;

*case* 2: *//LEVEL* *UP*

*this*->levelUpCharacter();

*break*;

*case* 3: *//REST*

rest();

*break*;

*case* 4: *//CHAR* *SHEET*

*this*->characterMenu();

*break*;

*case* 5: *//CREATE* *NEW* *CHAR*

createNewCharacter();

saveCharacters();

*break*;

*case* 6: *//SELECT* *CHAR*

selectCharacter();

*break*;

*case* 7: *//SAVE* *CHARS*

saveCharacters();

*break*;

*case* 8: *//LOAD* *CHARS*

loadCharacters();

*break*;

*default*:

*break*;

}

}

*else*

{

cout << QString::fromUtf8("= Вы умерли загрузить сохранения? =").toLocal8Bit().data() << "\n" << "\n";

cout << QString::fromUtf8("(0) Да, (1) Нет ").toLocal8Bit().data() << "\n";

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice < 0 || *this*->choice > 1)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("(0) Да, (1) Нет ").toLocal8Bit().data() << "\n";

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*if* (*this*->choice == 0)

*this*->loadCharacters();

*else*

playing = *false*;

}

}

void Game::**createNewCharacter**()

{

string name = "";

cout << QString::fromUtf8("Ввод имени персонажа: ").toLocal8Bit().data();

getline(*cin*, *name*);

*for* (size\_t i = 0; i < *this*->characters.size(); i++)

{

*while* (name == *this*->characters[i].getName())

{

cout << "Error! Character already exists!" << "\n";

cout << "Character name: ";

getline(*cin*, *name*);

}

}

characters.push\_back(Character());

activeCharacter = characters.size() - 1;

characters[activeCharacter].initialize(name);

}

void Game::**levelUpCharacter**()

{

*this*->characters[activeCharacter].levelUp();

*if* (*this*->characters[activeCharacter].getStatPoints() > 0)

{

cout << QString::fromUtf8("У вас есть статпоинты для распределения!").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8("Стат для улучшения: ").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("0: Сила ").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("1: Живучесть ").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("2: Ловкость ").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("3: Интеллект ").toLocal8Bit().data() << "\n";

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice > 3)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("Стат для улучшения: ").toLocal8Bit().data() << "\n";

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*switch* (*this*->choice)

{

*case* 0: *//STRENGTH*

*this*->characters[activeCharacter].addToStat(0, 1);

*break*;

*case* 1:*//VITALITY*

*this*->characters[activeCharacter].addToStat(1, 1);

*break*;

*case* 2:*//DEXTERITY*

*this*->characters[activeCharacter].addToStat(2, 1);

*break*;

*case* 3:*//INTELLIGENCE*

*this*->characters[activeCharacter].addToStat(3, 1);

*break*;

*default*:

*break*;

}

}

}

void Game::**characterMenu**()

{

*do*

{

system("CLS");

cout << gui::menu\_title("CHARACTER MENU");

cout << gui::menu\_divider();

characters[activeCharacter].printStats();

cout << gui::menu\_divider();

cout << QString::fromUtf8("= Меню =").toLocal8Bit().data() << "\n";

cout << gui::menu\_item(0, "Back");

cout << gui::menu\_item(1, "Print Inventory");

cout << gui::menu\_item(2, "Equip Item");

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice < 0 || *this*->choice > 2)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("= Меню =").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("0: Назад").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("1: Показать инвентарь").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("2: Одеть предмет").toLocal8Bit().data() << "\n";

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*switch* (*this*->choice)

{

*case* 1:

cout << *this*->characters[*this*->activeCharacter].getInvAsString();

*break*;

*case* 2:

cout << *this*->characters[*this*->activeCharacter].getInvAsString();

cout << QString::fromUtf8("Ввод id предмета: ").toLocal8Bit().data();

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice < 0 || *this*->choice >= *this*->characters[*this*->activeCharacter].getInventorySize())

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("Ввод id предмета: ").toLocal8Bit().data();

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*this*->characters[*this*->activeCharacter].equipItem(*this*->choice);

*break*;

*default*:

*break*;

}

*if* (*this*->choice > 0)

{

cout << QString::fromUtf8("Нажмите Enter, чтобы продолжить...").toLocal8Bit().data() << "\n";

cin.get();

}

} *while* (*this*->choice > 0);

}

void Game::**saveCharacters**()

{

ofstream outFile(fileName);

*if* (outFile.is\_open())

{

*for* (size\_t i = 0; i < *this*->characters.size(); i++)

{

outFile << *this*->characters[i].getAsString() << "\n";

outFile << *this*->characters[i].getInvAsStringSave() << "\n";

}

}

outFile.close();

}

void Game::**loadCharacters**()

{

ifstream inFile(fileName);

*this*->characters.clear();

string name = "";

int distanceTravelled = 0;

int gold = 0;

int level = 0;

int exp = 0;

int strength = 0;

int vitality = 0;

int dexterity = 0;

int intelligence = 0;

int hp = 0;

int statPoints = 0;

*//Item*

int itemType = 0;

int defence = 0;

int type = 0;

int damageMin = 0;

int damageMax = 0;

*//name*

*//level*

int buyValue = 0;

int sellValue = 0;

int rarity = 0;

Inventory tempItems;

string line = "";

stringstream str;

*if* (inFile.is\_open())

{

*while* (getline(inFile, *line*))

{

str.str(line);

str >> name;

str >> distanceTravelled;

str >> gold;

str >> level;

str >> exp;

str >> strength;

str >> vitality;

str >> dexterity;

str >> intelligence;

str >> hp;

str >> statPoints;

*//Create* *characyer*

Character temp(name, distanceTravelled, gold, level,

exp, strength, vitality, dexterity, intelligence,

hp, statPoints);

*//Weapon*

str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

damageMin >> damageMax;

Weapon weapon(damageMin, damageMax, name, level, buyValue, sellValue, rarity);

*//Armor* *head*

str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

defence >> type;

Armor armor\_head(type, defence, name, level, buyValue, sellValue, rarity);

*//Armor* *chest*

str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

defence >> type;

Armor armor\_chest(type, defence, name, level, buyValue, sellValue, rarity);

*//Armor* *arms*

str >>

itemType >> name >> level >>

rarity >> buyValue >>sellValue >>

defence >> type;

Armor armor\_arms(type, defence, name, level, buyValue, sellValue, rarity);

*//Armor* *legs*

str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

defence >> type;

Armor armor\_legs(type, defence, name, level, buyValue, sellValue, rarity);

temp.setWeapon(weapon);

temp.setArmorHead(armor\_head);

temp.setArmorChest(armor\_chest);

temp.setArmorArms(armor\_arms);

temp.setArmorLegs(armor\_legs);

*//Add* *Inventory* *Items*

str.clear();

line.clear();

getline(inFile, *line*);

str.str(line);

*while* (str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

damageMin >> damageMax)

{

temp.addItem(

Weapon

(

damageMin,

damageMax,

name,

level,

buyValue,

sellValue,

rarity

)

);

}

str.clear();

line.clear();

getline(inFile, *line*);

str.str(line);

*while* (str >>

itemType >> name >> level >>

rarity >> buyValue >> sellValue >>

defence >> type)

{

temp.addItem(

Armor

(

type,

defence,

name,

level,

buyValue,

sellValue,

rarity

)

);

}

*this*->characters.push\_back(Character(temp));

cout << QString::fromUtf8("Персонаж ").toLocal8Bit().data() << temp.getName() << QString::fromUtf8(" загружен!\n").toLocal8Bit().data();

str.clear();

}

}

inFile.close();

*if* (*this*->characters.size() <= 0)

{

*throw* "ERROR! NO CHARACTERS LOADED OR EMPTY FILE!";

}

}

void Game::**selectCharacter**()

{

cout << QString::fromUtf8("Выбор персонажа: ").toLocal8Bit().data() << "\n\n";

*for* (size\_t i = 0; i < *this*->characters.size(); i++)

{

cout << QString::fromUtf8("Ввод index персонажа:").toLocal8Bit().data() << i << " = " << *this*->characters[i].getName() << " Level: " << *this*->characters[i].getLevel() << "\n";

}

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice >= *this*->characters.size() || *this*->choice < 0)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("Выбрать персонажа: ").toLocal8Bit().data() << "\n";

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*this*->activeCharacter = *this*->choice;

cout << *this*->characters[*this*->activeCharacter].getName() << QString::fromUtf8(" ВЫБРАН!").toLocal8Bit().data() << "\n\n";

}

void Game::**travel**()

{

*this*->characters[activeCharacter].travel();

Event ev;

ev.generateEvent(*this->characters[activeCharacter]*, *this->enemies*);

}

void Game::**rest**()

{

int restCost = *this*->characters[*this*->activeCharacter].getHPMax() - *this*->characters[*this*->activeCharacter].getHP();

cout << QString::fromUtf8("= Отдых =").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8("Отдых стоит вам: ").toLocal8Bit().data() << restCost << "\n";

cout << QString::fromUtf8("Ваше золото: ").toLocal8Bit().data() << *this*->characters[*this*->activeCharacter].getGold() << "\n";

cout << "HP: " << *this*->characters[*this*->activeCharacter].getHP() << " / " << *this*->characters[*this*->activeCharacter].getHPMax() << "\n\n";

*if* (*this*->characters[*this*->activeCharacter].getGold() < restCost)

{

cout << QString::fromUtf8("Нет денег!").toLocal8Bit().data() << "\n\n";

}

*else* *if* (*this*->characters[*this*->activeCharacter].getHP() >= *this*->characters[*this*->activeCharacter].getHPMax())

{

cout << QString::fromUtf8("Вы полность вылечелись!").toLocal8Bit().data() << "\n\n";

}

*else*

{

cout << QString::fromUtf8("\n\n Отдых? (0) Да, (1) Нет...").toLocal8Bit().data() << "\n\n";

cin >> *this*->choice;

*while* (cin.fail() || *this*->choice < 0 || *this*->choice > 1)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("\n\n Отдых? (0) Да, (1) Нет...").toLocal8Bit().data() << "\n\n";

cin >> *this*->choice;

}

cin.ignore(100, '\n');

cout << "\n";

*if* (*this*->choice == 0)

{

*this*->characters[*this*->activeCharacter].resetHP();

*this*->characters[*this*->activeCharacter].payGold(restCost);

cout << QString::fromUtf8("Отдых!").toLocal8Bit().data() << "\n\n";

}

*else*

{

cout << QString::fromUtf8("Может в следующий раз...").toLocal8Bit().data() << "\n\n";

}

}

}

##### **Character.h**

#ifndef CHARACTER\_H

#define CHARACTER\_H

#include "STLINCLUDE.h"

#include "Inventory.h"

*using* *namespace* std;

*class* **Character**

{

*private*:

int distanceTravelled;

Inventory inventory;

Weapon weapon;

Armor armor\_head;

Armor armor\_chest;

Armor armor\_arms;

Armor armor\_legs;

int gold;

std::string name;

int level;

int exp;

int expNext;

int strength;

int vitality;

int dexterity;

int intelligence;

int hp;

int hpMax;

int damageMin;

int damageMax;

int defence;

int accuracy;

int luck;

int statPoints;

*public*:

**Character**();

**Character**(string name, int distanceTravelled,

int gold, int level,

int exp, int strength, int vitality,

int dexterity, int intelligence,

int hp, int statPoints);

*virtual* ~***Character***();

void **initialize**(*const* std::string name);

void **printStats**() *const*;

string **getAsString**()*const*;

string **getInvAsString**(bool shop = *false*);

string **getInvAsStringSave**();

void **levelUp**();

void **updateStats**();

void **addToStat**(int stat, int value);

void **equipItem**(unsigned index);

*inline* void **resetHP**() { *this*->hp = *this*->hpMax; }

void **addItem**(*const* Item &item) { *this*->inventory.addItem(item); }

void **removeItem**(*const* int index);

*const* Item& **getItem**(*const* int index);

*inline* *const* int& **getDistTravel**() *const* { *return* *this*->distanceTravelled; }

*inline* *const* std::string& **getName**() *const* { *return* *this*->name; }

*inline* *const* int& **getLevel**() *const* { *return* *this*->level; }

*inline* *const* int& **getExp**() *const* { *return* *this*->exp; }

*inline* *const* int& **getExpNext**() *const* { *return* *this*->expNext; }

*inline* *const* int& **getStatPoints**() *const* { *return* *this*->statPoints; }

*inline* *const* int& **getHP**() *const* { *return* *this*->hp; }

*inline* *const* int& **getHPMax**() *const* { *return* *this*->hpMax; }

*inline* *const* bool **isAlive**() { *return* *this*->hp > 0; }

*inline* *const* int& **getDamageMin**() *const* { *return* *this*->damageMin; }

*inline* *const* int& **getDamageMax**() *const* { *return* *this*->damageMax; }

*inline* *const* int **getDamage**()*const* { *return* rand() % (*this*->damageMax + *this*->weapon.getDamageMax()) + (*this*->damageMin + *this*->weapon.getDamageMin()); }

*inline* *const* int& **getDefence**() *const* { *return* *this*->defence; }

*inline* *const* int **getAddedDefence**()*const* { *return* *this*->armor\_arms.getDefence() + *this*->armor\_chest.getDefence() + *this*->armor\_legs.getDefence() + *this*->armor\_head.getDefence(); }

*inline* *const* int& **getAccuracy**() *const* { *return* *this*->accuracy; }

*inline* *const* int **getGold**() *const* { *return* *this*->gold; }

*inline* *const* int **getInventorySize**()*const* { *return* *this*->inventory.size(); }

*inline* void **setDistTravelled**(*const* int& distance) { *this*->distanceTravelled = distance; }

*inline* void **travel**() { *this*->distanceTravelled++; }

*inline* void **gainExp**(*const* int exp) { *this*->exp += exp; }

*inline* void **gainGold**(*const* int gold) { *this*->gold += gold; }

*inline* void **payGold**(*const* int gold) { *this*->gold -= gold; }

void **takeDamage**(*const* int damage);

*inline* void **setWeapon**(Weapon weapon) { *this*->weapon = weapon; }

*inline* void **setArmorHead**(Armor armor\_head) { *this*->armor\_head = armor\_head; }

*inline* void **setArmorChest**(Armor armor\_chest) { *this*->armor\_chest = armor\_chest; }

*inline* void **setArmorArms**(Armor armor\_arms) { *this*->armor\_arms = armor\_arms; }

*inline* void **setArmorLegs**(Armor armor\_legs) { *this*->armor\_legs = armor\_legs; }

};

#endif *//* *CHARACTER\_H*

##### **Character.cpp**

#include "character.h"

Character::**Character**()

{

*this*->distanceTravelled = 0;

*this*->gold = 0;

*this*->name = "";

*this*->level = 0;

*this*->exp = 0;

*this*->expNext = 0;

*this*->strength = 0;

*this*->vitality = 0;

*this*->dexterity = 0;

*this*->intelligence = 0;

*this*->hp = 0;

*this*->hpMax = 0;

*this*->damageMin = 0;

*this*->damageMax = 0;

*this*->defence = 0;

*this*->accuracy = 0;

*this*->luck = 0;

*this*->statPoints = 0;

}

Character::**Character**(string name, int distanceTravelled,

int gold, int level,

int exp, int strength, int vitality,

int dexterity, int intelligence,

int hp, int statPoints)

{

*this*->distanceTravelled = distanceTravelled;

*this*->gold = gold;

*this*->name = name;

*this*->level = level;

*this*->exp = exp;

*this*->expNext = 0;

*this*->strength = strength;

*this*->vitality = vitality;

*this*->dexterity = dexterity;

*this*->intelligence = intelligence;

*this*->hp = hp;

*this*->hpMax = 0;

*this*->damageMin = 0;

*this*->damageMax = 0;

*this*->defence = 0;

*this*->accuracy = 0;

*this*->luck = 0;

*this*->statPoints = statPoints;

*this*->updateStats();

}

Character::~***Character***()

{

}

*//Functions*

void Character::**initialize**(*const* string name)

{

*this*->distanceTravelled = 0;

*this*->gold = 100;

*this*->name = name;

*this*->level = 1;

*this*->exp = 0;

*this*->strength = 5;

*this*->vitality = 5;

*this*->dexterity = 5;

*this*->intelligence = 5;

*this*->statPoints = 0;

*this*->updateStats();

}

void Character::**printStats**() *const*

{

cout << "= Character Sheet =" << "\n";

cout << "= Name: " << *this*->name << "\n";

cout << "= Level: " << *this*->level << "\n";

cout << "= Exp: " << *this*->exp << "\n";

cout << "= Exp to next level: " << *this*->expNext << "\n";

cout << "= Statpoints: " << *this*->statPoints << "\n";

cout << "\n";

cout << "= Strenght: " << *this*->strength << "\n";

cout << "= Vitality: " << *this*->vitality << "\n";

cout << "= Dexterity: " << *this*->dexterity << "\n";

cout << "= Intelligence: " << *this*->intelligence << "\n";

cout << "\n";

cout << "= HP: " << *this*->hp << " / " << *this*->hpMax << "\n";

cout << "= Damage: " << *this*->damageMin << "( +" << *this*->weapon.getDamageMin() << ")" << " - "<< *this*->damageMax << "( +" << *this*->weapon.getDamageMax() << ")" << "\n";

cout << "= Defence: " << *this*->defence << "( +" << std::to\_string(*this*->getAddedDefence()) << ")" << "\n";

cout << "= Accuracy: " << *this*->accuracy << "\n";

cout << "= Luck: " << *this*->luck << "\n";

cout << "\n";

cout << "= Distance Travelled: " << *this*->distanceTravelled << "\n";

cout << "= Gold: " << *this*->gold << "\n";

cout << "\n";

cout << "= Weapon: " << *this*->weapon.getName()

<< " Lvl: " << *this*->weapon.getLevel()

<< " Dam: " << *this*->weapon.getDamageMin() << " - " << *this*->weapon.getDamageMax() << "\n";

cout << "= Armor Head: " << *this*->armor\_head.getName()

<< " Lvl: " << *this*->armor\_head.getLevel()

<< " Def: " << *this*->armor\_head.getDefence() << "\n";

cout << "= Armor Chest: " << *this*->armor\_chest.getName()

<< " Lvl: " << *this*->armor\_chest.getLevel()

<< " Def: " << *this*->armor\_chest.getDefence() << "\n";

cout << "= Armor Arms: " << *this*->armor\_arms.getName()

<< " Lvl: " << *this*->armor\_arms.getLevel()

<< " Def: " << *this*->armor\_arms.getDefence() << "\n";

cout << "= Armor Legs: " << *this*->armor\_legs.getName()

<< " Lvl: " << *this*->armor\_legs.getLevel()

<< " Def: " << *this*->armor\_legs.getDefence() << "\n" << "\n";

}

string Character::**getAsString**() *const*

{

*return* name + " "

+ to\_string(distanceTravelled) + " "

+ to\_string(gold) + " "

+ to\_string(level) + " "

+ to\_string(exp) + " "

+ to\_string(strength) + " "

+ to\_string(vitality) + " "

+ to\_string(dexterity) + " "

+ to\_string(intelligence) + " "

+ to\_string(hp) + " "

+ to\_string(statPoints) + " "

+ *this*->weapon.*toStringSave*()

+ *this*->armor\_head.*toStringSave*()

+ *this*->armor\_chest.*toStringSave*()

+ *this*->armor\_arms.*toStringSave*()

+ *this*->armor\_legs.*toStringSave*();

}

string Character::**getInvAsString**(bool shop)

{

string inv;

*for* (size\_t i = 0; i < *this*->inventory.size(); i++)

{

*if* (shop)

{

inv += to\_string(i) + ": " + *this*->inventory[i].*toString*() + "\n" + " - Sell value: "

+ std::to\_string(*this*->inventory[i].getSellValue()) + "\n";

}

*else*

{

inv += to\_string(i) + ": " + *this*->inventory[i].*toString*() + "\n";

}

}

*return* inv;

}

string Character::**getInvAsStringSave**()

{

string inv;

*for* (size\_t i = 0; i < *this*->inventory.size(); i++)

{

*if*(*this*->inventory[i].getItemType() == itemTypes::*WEAPON*)

inv += *this*->inventory[i].*toStringSave*();

}

inv += "\n";

*for* (size\_t i = 0; i < *this*->inventory.size(); i++)

{

*if* (*this*->inventory[i].getItemType() == itemTypes::*ARMOR*)

inv += *this*->inventory[i].*toStringSave*();

}

*return* inv;

}

void Character::**levelUp**()

{

*if* (*this*->exp >= *this*->expNext)

{

*this*->exp -= *this*->expNext;

*this*->level++;

*this*->expNext = *static\_cast*<int>((50 / 3)\*((pow(level, 3)

- 6 \* pow(level, 2))

+ 17 \* level - 12)) + 100;

*this*->statPoints++;

*this*->updateStats();

cout << QString::fromUtf8("Ваш уровень теперь ").toLocal8Bit().data() << *this*->level << "!" << "\n\n";

}

*else*

{

cout << QString::fromUtf8("Нужно большо опыта!").toLocal8Bit().data() << "\n\n";

}

}

void Character::**updateStats**()

{

*this*->expNext = *static\_cast*<int>(

(50 / 3)\*((pow(level, 3)

- 6 \* pow(level, 2))

+ 17 \* level - 12)) + 100;

*this*->hpMax = (*this*->vitality \* 5) + (*this*->strength) + *this*->level\*5;

*this*->damageMin = *this*->strength;

*this*->damageMax = *this*->strength + 2;

*this*->defence = *this*->dexterity + (*this*->intelligence / 2);

*this*->accuracy = (*this*->dexterity / 2) + intelligence;

*this*->luck = *this*->intelligence;

*this*->hp = *this*->hpMax;

}

void Character::**addToStat**(int stat, int value)

{

*if* (*this*->statPoints < value)

cout << QString::fromUtf8("Ошибка! Нужны статпоинты!").toLocal8Bit().data() << "\n";

*else*

{

*switch* (stat)

{

*case* 0:

*this*->strength += value;

cout << QString::fromUtf8("Сила выросла!").toLocal8Bit().data() << "\n";

*break*;

*case* 1:

*this*->vitality += value;

cout << QString::fromUtf8("Живучесть выросла!").toLocal8Bit().data() << "\n";

*break*;

*case* 2:

*this*->dexterity += value;

cout << QString::fromUtf8("Ловкость выросла!").toLocal8Bit().data() << "\n";

*break*;

*case* 3:

*this*->intelligence += value;

cout << QString::fromUtf8("Интеллект вырос!").toLocal8Bit().data() << "\n";

*break*;

*default*:

cout << QString::fromUtf8("Нет такой статистики!").toLocal8Bit().data() << "\n";

*break*;

}

*this*->statPoints -= value;

*this*->updateStats();

}

}

void Character::**equipItem**(unsigned index)

{

*if* (index < 0 || index >= *this*->inventory.size())

{

cout << QString::fromUtf8("Выбран неправельный предмет!").toLocal8Bit().data() << "\n\n";

}

*else*

{

Weapon \*w = *nullptr*;

w = *dynamic\_cast*<Weapon\*>(&*this*->inventory[index]);

Armor \*a = *nullptr*;

a = *dynamic\_cast*<Armor\*>(&*this*->inventory[index]);

*//Is* *weapon*

*if* (w != *nullptr*)

{

*if* (*this*->weapon.getRarity() >= 0)

*this*->inventory.addItem(*this*->weapon);

*this*->weapon = \*w;

*this*->inventory.removeItem(index);

}

*else* *if* (a != *nullptr*)

{

*switch* (a->getType())

{

*case* armorType::*HEAD*:

*if* (*this*->armor\_head.getRarity() >= 0)

*this*->inventory.addItem(*this*->armor\_head);

*this*->armor\_head = \*a;

*this*->inventory.removeItem(index);

*break*;

*case* armorType::*CHEST*:

*if* (*this*->armor\_chest.getRarity() >= 0)

*this*->inventory.addItem(*this*->armor\_chest);

*this*->armor\_chest = \*a;

*this*->inventory.removeItem(index);

*break*;

*case* armorType::*ARMS*:

*if* (*this*->armor\_arms.getRarity() >= 0)

*this*->inventory.addItem(*this*->armor\_arms);

*this*->armor\_arms = \*a;

*this*->inventory.removeItem(index);

*break*;

*case* armorType::*LEGS*:

*if*(*this*->armor\_legs.getRarity() >= 0)

*this*->inventory.addItem(*this*->armor\_legs);

*this*->armor\_legs = \*a;

*this*->inventory.removeItem(index);

*break*;

*default*:

cout << QString::fromUtf8("Ошибка неправельный тип брони!").toLocal8Bit().data() << "\n\n";

*break*;

}

}

*else*

{

cout << QString::fromUtf8("Ошибка предмет, не является броней или оружием!").toLocal8Bit().data();

}

}

}

void Character::**removeItem**(*const* int index)

{

*if* (index < 0 || index >= *this*->inventory.size())

cout << QString::fromUtf8("Ошибка, невозможно удалить предмет, <removeItem Character>").toLocal8Bit().data() << "\n\n";

*else*

{

*this*->inventory.removeItem(index);

}

}

*const* Item& Character::**getItem**(*const* int index)

{

*if* (index < 0 || index >= *this*->inventory.size())

{

cout << QString::fromUtf8("Ошибка, невозможно удалить предмет, <getItem Character>").toLocal8Bit().data() << "\n\n";

*throw*("ERROR OUT OF BOUNDS, GETITEM CHARACTER");

}

*return* *this*->inventory[index];

}

void Character::**takeDamage**(*const* int damage)

{

*this*->hp -= damage;

*if* (*this*->hp <= 0)

{

*this*->hp = 0;

}

}

##### **Item.h**

#ifndef ITEM\_H

#define ITEM\_H

#include "STLINCLUDE.h"

*enum* **itemTypes** {*WEAPON* = 0, *ARMOR*, *ACCESSORIES*};

*class* **Item**

{

*private*:

int itemType;

std::string name;

int level;

int buyValue;

int sellValue;

int rarity;

*public*:

**Item**();

**Item**(int itemType,int level,int rarity);

**Item**(int itemType,std::string name,int level,int buyValue,int sellValue,int rarity);

*virtual* ~***Item***();

*inline* std::string **debugPrint**() *const* { *return* *this*->name; }

*virtual* Item\* ***clone***()*const* = 0;

*virtual* std::string ***toString***()*const* = 0;

*virtual* std::string ***toStringSave***()*const* = 0;

*inline* *const* std::string& **getName**() *const* { *return* *this*->name; }

*inline* *const* int& **getLevel**() *const* { *return* *this*->level; }

*inline* *const* int& **getBuyValue**() *const* { *return* *this*->buyValue; }

*inline* *const* int& **getSellValue**() *const* { *return* *this*->sellValue; }

*inline* *const* int& **getRarity**() *const* { *return* *this*->rarity; }

*inline* *const* int& **getItemType**() *const* { *return* *this*->itemType; }

*inline* void **setName**(std::string name) { *this*->name = name; }

};

*enum* **rarity** {

*COMMON* = 0,

*UNCOMMON*,

*RARE*,

*LEGENDARY*,

*MYTHIC*

};

#endif *//* *ITEM\_H*

##### **Item.cpp**

#include "item.h"

Item::**Item**()

{

*this*->itemType = -1;

*this*->name = "EMPTY";

*this*->level = 0;

*this*->buyValue = 0;

*this*->sellValue = 0;

*this*->rarity = -1;

}

Item::**Item**(int itemType, int level, int rarity)

{

*this*->name = "RANDOM";

*this*->level = rand() % (level+2) + 1;

*this*->rarity = rarity;

*this*->buyValue = (*this*->level + *this*->rarity) + *this*->level\**this*->rarity\*10;

*this*->sellValue = *this*->buyValue / 2;

*this*->itemType = itemType;

}

Item::**Item**(int itemType, std::string name, int level, int buyValue, int sellValue, int rarity)

{

*this*->itemType = itemType;

*this*->name = name;

*this*->level = level;

*this*->buyValue = buyValue;

*this*->sellValue = sellValue;

*this*->rarity = rarity;

}

Item::~***Item***()

{

}

##### **Inventory.h**

#ifndef INVENTORY\_H

#define INVENTORY\_H

#include "STLINCLUDE.h"

#include "Weapon.h"

#include "Armor.h"

#include "Accessories.h"

*class* **Inventory**

{

*private*:

int cap;

int nrOfItems;

Item \*\*itemArr;

void **expand**();

void **initialize**(*const* int from = 0);

*public*:

**Inventory**();

~**Inventory**();

**Inventory**(*const* Inventory &obj);

*inline* int **size**()*const* { *return* *this*->nrOfItems; };

Item& *operator*[](*const* int index);

void *operator*=(*const* Inventory &obj);

void **addItem**(*const* Item &item);

void **removeItem**(int index);

*inline* void **debugPrint**() *const*

{

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

std::cout << *this*->itemArr[i]->debugPrint() << std::endl;

}

}

};

#endif *//* *INVENTORY\_H*

##### **Inventory.cpp**

#include "inventory.h"

Inventory::**Inventory**()

{

*this*->cap = 5;

*this*->nrOfItems = 0;

*this*->itemArr = *new* Item\*[cap];

*this*->initialize();

}

Inventory::~**Inventory**()

{

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

*delete* *this*->itemArr[i];

}

*delete*[] *this*->itemArr;

}

Inventory::**Inventory**(*const* Inventory &obj)

{

*this*->cap = obj.cap;

*this*->nrOfItems = obj.nrOfItems;

*this*->itemArr = *new* Item\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

*this*->itemArr[i] = obj.itemArr[i]->*clone*();

}

initialize(*this*->nrOfItems);

}

Item& Inventory::*operator*[](*const* int index)

{

*if* (index < 0 || index >= *this*->nrOfItems)

*throw*("BAD INDEX!");

*return* \**this*->itemArr[index];

}

void Inventory::*operator*=(*const* Inventory &obj)

{

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

*delete* *this*->itemArr[i];

}

*delete*[] *this*->itemArr;

*this*->cap = obj.cap;

*this*->nrOfItems = obj.nrOfItems;

*this*->itemArr = *new* Item\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

*this*->itemArr[i] = obj.itemArr[i]->*clone*();

}

initialize(*this*->nrOfItems);

}

void Inventory::**expand**()

{

*this*->cap \*= 2;

Item \*\*tempArr = *new* Item\*[*this*->cap];

*for* (size\_t i = 0; i < *this*->nrOfItems; i++)

{

tempArr[i] = *this*->itemArr[i];

}

*delete*[] *this*->itemArr;

*this*->itemArr = tempArr;

*this*->initialize(*this*->nrOfItems);

}

void Inventory::**initialize**(*const* int from)

{

*for* (size\_t i = from; i < cap; i++)

{

*this*->itemArr[i] = *nullptr*;

}

}

void Inventory::**addItem**(*const* Item &item)

{

*if* (*this*->nrOfItems >= *this*->cap)

{

expand();

}

*this*->itemArr[*this*->nrOfItems++] = item.*clone*();

}

void Inventory::**removeItem**(int index)

{

*if* (index < 0 || index >= *this*->nrOfItems)

*throw*("OUT OF BOUNDS REMOVE ITEM INVENTORY");

*delete* *this*->itemArr[index];

*this*->itemArr[index] = *this*->itemArr[*this*->nrOfItems - 1];

*this*->itemArr[--*this*->nrOfItems] = *nullptr*;

}

##### **Weapon.h**

#ifndef WEAPON\_H

#define WEAPON\_H

#include "STLINCLUDE.h"

#include "Item.h"

*class* **Weapon** : *public* Item

{

*private*:

int damageMin;

int damageMax;

*public*:

**Weapon**();

**Weapon**(int level,int rarity);

**Weapon**(int damageMin,int damageMax,std::string name,int level,int buyValue,int sellValue,int rarity);

*virtual* ~***Weapon***();

*virtual* Weapon\* ***clone***()*const*;

std::string ***toString***()*const*;

std::string ***toStringSave***()*const*;

*inline* int **getDamageMin**()*const* { *return* *this*->damageMin; }

*inline* int **getDamageMax**()*const* { *return* *this*->damageMax; }

*static* dArr<std::string> names;

*static* void **initNames**();

};

#endif *//* *WEAPON\_H*

##### **Weapon.cpp**

#include "weapon.h"

dArr<std::string> Weapon::names;

void Weapon::**initNames**()

{

Weapon::names.push("Bitter Steel");

Weapon::names.push("Leaf Cutter");

Weapon::names.push("Face Shredder");

}

Weapon::**Weapon**()

:Item()

{

*this*->damageMax = 0;

*this*->damageMin = 0;

}

Weapon::**Weapon**(int level, int rarity)

:Item(itemTypes::*WEAPON*, level, rarity)

{

*this*->damageMax = rand() % level \* (rarity+1);

*this*->damageMax += (rarity+1) \* 5;

*this*->setName(Weapon::names[rand() % Weapon::names.size()]);

*if* (rarity == 3)

*this*->damageMax += level \* 5;

*else* *if* (rarity == 4)

*this*->damageMax += level \* 10;

*this*->damageMin = *this*->damageMax / 2;

}

Weapon::**Weapon**(int damageMin, int damageMax, std::string name, int level, int buyValue, int sellValue, int rarity)

: Item(itemTypes::*WEAPON*, name, level, buyValue, sellValue, rarity)

{

*this*->damageMin = damageMin;

*this*->damageMax = damageMax;

}

Weapon::~***Weapon***()

{

}

Weapon\* Weapon::***clone***()*const*

{

*return* *new* Weapon(\**this*);

}

std::string Weapon::***toString***()*const*

{

std::string str =

*this*->getName()

+ " | Lvl: "

+ std::to\_string(*this*->getLevel())

+ " | Rarity: "

+ std::to\_string(*this*->getRarity())

+ " | Damage: "

+ std::to\_string(*this*->damageMin)

+ " - "

+ std::to\_string(*this*->damageMax);

*return* str;

}

std::string Weapon::***toStringSave***()*const*

{

std::string str =

std::to\_string(*this*->getItemType())

+ " "

+ *this*->getName()

+ " "

+ std::to\_string(*this*->getLevel())

+ " "

+ std::to\_string(*this*->getRarity())

+ " "

+ std::to\_string(*this*->getBuyValue())

+ " "

+ std::to\_string(*this*->getSellValue())

+ " "

+ std::to\_string(*this*->damageMin)

+ " "

+ std::to\_string(*this*->damageMax)

+ " ";

*return* str;

}

##### **Armor.h**

#ifndef ARMOR\_H

#define ARMOR\_H

#include "STLINCLUDE.h"

#include "Item.h"

*enum* **armorType** { *HEAD* = 0, *CHEST*, *ARMS*, *LEGS* };

*class* **Armor** : *public* Item

{

*private*:

int type;

std::string typeStr;

int defence;

*public*:

**Armor**();

**Armor**(int level,int rarity);

**Armor**(int type,int defence,std::string name,int level,int buyValue,int sellValue,int rarity);

*virtual* ~***Armor***();

*virtual* Armor\* ***clone***()*const*;

std::string ***toString***()*const*;

std::string ***toStringSave***()*const*;

*inline* int **getDefence**()*const* { *return* *this*->defence; }

*inline* int **getType**()*const* { *return* *this*->type; }

*static* dArr<std::string> names;

*static* void **initNames**();

};

#endif *//* *ARMOR\_H*

##### **Armor.cpp**

#include "armor.h"

dArr<std::string> Armor::names;

void Armor::**initNames**()

{

Armor::names.push("Devilish Defender");

Armor::names.push("Angelish Defender");

Armor::names.push("Iron Sheets");

}

Armor::**Armor**()

:Item()

{

*this*->type = -1;

*this*->defence = 0;

}

Armor::**Armor**(int level, int rarity)

:Item(itemTypes::*ARMOR*, level, rarity)

{

*this*->defence = rand() % (level \* (rarity+1));

*this*->defence += (rarity + 1) \* 5;

*this*->type = rand() % 4;

*switch* (*this*->type)

{

*case* armorType::*HEAD*:

*this*->typeStr = "Head";

*break*;

*case* armorType::*CHEST*:

*this*->typeStr = "Chest";

*break*;

*case* armorType::*ARMS*:

*this*->typeStr = "Arms";

*break*;

*case* armorType::*LEGS*:

*this*->typeStr = "Legs";

*break*;

*default*:

*this*->typeStr = "ERROR INVALID!";

*break*;

}

*this*->setName(Armor::names[rand() % Armor::names.size()]);

*if*(rarity == 3)

*this*->defence += level \* 5;

*else* *if* (rarity == 4)

*this*->defence += level \* 10;

}

Armor::**Armor**(int type, int defence, std::string name, int level, int buyValue, int sellValue, int rarity)

: Item(itemTypes::*ARMOR*, name, level, buyValue, sellValue, rarity)

{

*this*->type = type;

*this*->defence = defence;

}

Armor::~***Armor***()

{

}

Armor\* Armor::***clone***()*const*

{

*return* *new* Armor(\**this*);

}

std::string Armor::***toString***()*const*

{

std::string str =

*this*->getName()

+ " | Type: "

+ *this*->typeStr

+ " | Lvl: "

+ std::to\_string(*this*->getLevel())

+ " | Rarity: "

+ std::to\_string(*this*->getRarity())

+ " | Def: "

+ std::to\_string(*this*->defence);

*return* str;

}

std::string Armor::***toStringSave***()*const*

{

std::string str =

std::to\_string(*this*->getItemType())

+ " "

+ *this*->getName()

+ " "

+ std::to\_string(*this*->getLevel())

+ " "

+ std::to\_string(*this*->getRarity())

+ " "

+ std::to\_string(*this*->getBuyValue())

+ " "

+ std::to\_string(*this*->getSellValue())

+ " "

+ std::to\_string(*this*->defence)

+ " "

+ std::to\_string(*this*->type)

+ " ";

*return* str;

}

##### **Event.h**

#ifndef EVENT\_H

#define EVENT\_H

#include "STLINCLUDE.h"

#include "Character.h"

#include "Enemy.h"

#include "Boss.h"

*class* **Event**

{

*private*:

*public*:

**Event**();

*virtual* ~***Event***();

void **generateEvent**(Character &character, dArr<Enemy>& enemies);

void **shopEncouter**(Character &character);

void **enemyEncouter**(Character &character, dArr<Enemy>& enemies);

*static* int nrOfEvents;

};

#endif *//* *EVENT\_H*

##### **Event.cpp**

#include "event.h"

int Event::nrOfEvents = 4;

*using* *namespace* std;;

Event::**Event**()

{

}

Event::~***Event***()

{

}

void Event::**generateEvent**(Character &character, dArr<Enemy>& enemies)

{

int i = rand() % Event::nrOfEvents;

*switch* (i)

{

*case* 0:

*//Enemy* *encounter*

*this*->enemyEncouter(*character*, *enemies*);

*break*;

*case* 1:

*//Shop*

*this*->shopEncouter(*character*);

*break*;

*default*:

*break*;

}

}

*//Different* *events*

void Event::**shopEncouter**(Character &character)

{

int choice = 0;

bool shopping = *true*;

Inventory merchantInv;

string inv;

*//Init* *merchant* *inv*

int nrOfItems = rand() % 20 + 10;

bool coinToss = *false*;

*for* (size\_t i = 0; i < nrOfItems; i++)

{

coinToss = rand() % 2;

*if*(coinToss > 0)

merchantInv.addItem(Weapon(character.getLevel() + rand() % 5, rand() % 4));

*else*

merchantInv.addItem(Armor(character.getLevel() + rand() % 5, rand() % 4));

}

*while* (shopping)

{

system("CLS");

cout << QString::fromUtf8("= Меню магазина =").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8("0: Покинуть").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("1: Купить").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("2: Продать").toLocal8Bit().data() << "\n";

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

*while* (cin.fail() || choice > 3 || choice < 0)

{

system("CLS");

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("= Меню магазина =").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8("0: Покинуть").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("1: Купить").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("2: Продать").toLocal8Bit().data() << "\n";

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

}

cin.ignore(100, '\n');

cout << "\n";

*//Shop*

*switch* (choice)

{

*case* 0: *//Leave*

shopping = *false*;

*break*;

*case* 1: *//Buy*

cout << QString::fromUtf8("= Меню покупки =").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8(" - Золото: ").toLocal8Bit().data() <<character.getGold() << "\n\n";

inv.clear();

*for* (size\_t i = 0; i < merchantInv.size(); i++)

{

inv += to\_string(i) + ": " + merchantInv[i].*toString*() + QString::fromUtf8("\n - цена: ").toLocal8Bit().data() + to\_string(merchantInv[i].getBuyValue()) + "\n";

}

cout << inv << "\n";

cout << QString::fromUtf8("Золото: ").toLocal8Bit().data() << character.getGold() << "\n";

cout << QString::fromUtf8("Выбор предмета (-1 to cancel): ").toLocal8Bit().data();

cin >> choice;

*while* (cin.fail() || choice > merchantInv.size() || choice < -1)

{

system("CLS");

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("Золото: ").toLocal8Bit().data() << character.getGold() << "\n";

cout << QString::fromUtf8("Выбор предмета (-1 to cancel): ").toLocal8Bit().data();

cin >> choice;

}

cin.ignore(100, '\n');

cout << "\n";

*if* (choice == -1)

{

cout << QString::fromUtf8("Назад...").toLocal8Bit().data() << "\n";

}

*else* *if* (character.getGold() >= merchantInv[choice].getBuyValue())

{

character.payGold(merchantInv[choice].getBuyValue());

character.addItem(merchantInv[choice]);

cout << QString::fromUtf8("Купил предмет ").toLocal8Bit().data() << merchantInv[choice].getName() << " -" << merchantInv[choice].getBuyValue() << "\n";

merchantInv.removeItem(choice);

}

*else*

{

cout << QString::fromUtf8("Вы не можете себе позволить этот предмет!").toLocal8Bit().data() << "\n";

}

*break*;

*case* 2: *//Sell*

cout << character.getInvAsString(*true*) << "\n";

cout << QString::fromUtf8("= Меню продажи =").toLocal8Bit().data() << "\n\n";

cout << QString::fromUtf8(" - Золото: ").toLocal8Bit().data() << character.getGold() << "\n\n";

*if* (character.getInventorySize() > 0)

{

cout << QString::fromUtf8("Золото: ").toLocal8Bit().data() << character.getGold() << "\n";

cout << QString::fromUtf8("Выбор предмета (-1 to cancel): ").toLocal8Bit().data();

cin >> choice;

*while* (cin.fail() || choice > character.getInventorySize() || choice < -1)

{

system("CLS");

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << QString::fromUtf8("Золото: ").toLocal8Bit().data() << character.getGold() << "\n";

cout << QString::fromUtf8("Выбор предмета (-1 to cancel): ").toLocal8Bit().data();

cin >> choice;

}

cin.ignore(100, '\n');

cout << "\n";

*if* (choice == -1)

{

cout << QString::fromUtf8("Назад...").toLocal8Bit().data() << "\n";

}

*else*

{

character.gainGold(character.getItem(choice).getSellValue());

cout << QString::fromUtf8("Предмет продан!").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("Золота получено: ").toLocal8Bit().data() << character.getItem(choice).getSellValue() << "\n\n";

character.removeItem(choice);

}

}

*else*

{

cout << QString::fromUtf8("Инвентарь пуст...").toLocal8Bit().data() << "\n";

}

*break*;

*default*:

*break*;

}

cout << QString::fromUtf8("Нажмите Enter, чтобы продолжить...").toLocal8Bit().data() << "\n";

cin.get();

}

cout << QString::fromUtf8("Вы покинули магазин..").toLocal8Bit().data() << "\n\n";

}

void Event::**enemyEncouter**(Character &character, dArr<Enemy>& enemies)

{

bool playerTurn = *false*;

int choice = 0;

int coinToss = rand() % 2 + 1;

*if* (coinToss == 1)

playerTurn = *true*;

*else*

playerTurn = *false*;

bool escape = *false*;

bool playerDefeated = *false*;

bool enemiesDefeated = *false*;

int nrOfEnemies = rand() % 5 + 1;

*for* (size\_t i = 0; i < nrOfEnemies; i++)

{

enemies.push(Enemy(character.getLevel() + rand()%3));

}

*//Battle* *variables*

int damage = 0;

int gainExp = 0;

int gainGold = 0;

int playerTotal = 0;

int enemyTotal = 0;

int combatTotal = 0;

int combatRollPlayer = 0;

int combatRollEnemy = 0;

*while* (!escape && !playerDefeated && !enemiesDefeated)

{

*if* (playerTurn && !playerDefeated)

{

cout << QString::fromUtf8("= Ход игрока =").toLocal8Bit().data() << "\n\n";

cout << "Continue..." << "\n\n";

cin.get();

system("CLS");

cout << QString::fromUtf8("= Боевое меню =").toLocal8Bit().data() << "\n\n";

cout << "HP: " << character.getHP() << " / " << character.getHPMax() << "\n\n";

cout << "0: Escape" << "\n";

cout << "1: Attack" << "\n";

cout << "2: Defend" << "\n";

cout << "3: Use Item" << "\n";

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

*while* (cin.fail() || choice > 3 || choice < 0)

{

system("CLS");

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << "= BATTLE MENU =" << "\n\n";

cout << "0: Escape" << "\n";

cout << "1: Attack" << "\n";

cout << "2: Defend" << "\n";

cout << "3: Use Item" << "\n";

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

}

cin.ignore(100, '\n');

cout << "\n";

*//Moves*

*switch* (choice)

{

*case* 0: *//ESCAPE*

escape = *true*;

*break*;

*case* 1: *//ATTACK*

*//Select* *enemy*

cout << "Select enemy: " << "\n\n";

*for* (size\_t i = 0; i < enemies.size(); i++)

{

cout << i << ": "

<< "Level: " << enemies[i].getLevel() << " - " <<

"HP: " << enemies[i].getHp() << "/" << enemies[i].getHpMax() << " - " <<

"Defence: " << enemies[i].getDefence() << " - " <<

"Accuracy: " << enemies[i].getAccuracy() << " - " <<

"Damage: " << enemies[i].getDamageMin() << " - " << enemies[i].getDamageMax() <<

"\n";

}

cout << "\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

*while* (cin.fail() || choice >= enemies.size() || choice < 0)

{

cout << QString::fromUtf8("Неправильный ввод!").toLocal8Bit().data() << "\n";

cin.clear();

cin.ignore(100, '\n');

cout << "Select enemy: " << "\n\n";

cout << QString::fromUtf8("Ввод: ").toLocal8Bit().data();

cin >> choice;

}

cin.ignore(100, '\n');

cout << "\n";

*//Attack* *roll*

combatTotal = enemies[choice].getDefence() + character.getAccuracy();

enemyTotal = enemies[choice].getDefence() / (double)combatTotal \* 100;

playerTotal = character.getAccuracy() / (double)combatTotal \* 100;

combatRollPlayer = rand() % playerTotal + 1;

combatRollEnemy = rand() % enemyTotal + 1;

cout << "Combat total: " << combatTotal << "\n";

cout << "Enemy percent: " << enemyTotal << "\n";

cout << "Player percent: " << playerTotal << "\n\n";

cout << "Player roll: " << combatRollPlayer << "\n";

cout << "Enemy roll: " << combatRollEnemy << "\n\n";

*if* (combatRollPlayer > combatRollEnemy) *//Hit*

{

cout << "HIT! " << "\n\n";

damage = character.getDamage();

enemies[choice].takeDamage(damage);

cout << "Damage: " << damage << "!" << "\n\n";

*if* (!enemies[choice].isAlive())

{

cout << "ENEMY DEFEATED!" << "\n\n";

gainExp = enemies[choice].getExp();

character.gainExp(gainExp);

gainGold = rand() % enemies[choice].getLevel() \* 10 + 1;

character.gainGold(gainGold);

cout << "EXP GAINED: " << gainExp << "\n";

cout << "GOLD GAINED: " << gainGold << "\n\n";

*//Item* *roll*

int roll = rand() % 100 + 1;

int rarity = -1;

*if* (roll > 20)

{

rarity = 0; *//Common*

roll = rand() % 100 + 1;

*if* (roll > 30)

{

rarity = 1; *//Uncommon*

roll = rand() % 100 + 1;

*if* (roll > 50)

{

rarity = 2; *//Rare*

roll = rand() % 100 + 1;

*if* (roll > 70)

{

rarity = 3; *//Legendary*

roll = rand() % 100 + 1;

*if* (roll > 90)

{

rarity = 4; *//Mythic*

}

}

}

}

}

*if* (roll >= 0)

{

roll = rand() % 100 + 1;

*if* (roll > 50)

{

Weapon tempW(character.getLevel(), rarity);

character.addItem(tempW);

cout << "WEAPON DROP!" << "\n";

}

*else*

{

Armor tempA(character.getLevel(), rarity);

character.addItem(tempA);

cout << "ARMOR DROP!" << "\n";

}

}

enemies.remove(choice);

}

}

*else* *//Miss*

{

cout << QString::fromUtf8("Промах! \n\n").toLocal8Bit().data();

}

*break*;

*case* 2: *//DEFEND*

*break*;

*case* 3: *//ITEM*

*break*;

*default*:

*break*;

}

*//End* *turn*

playerTurn = *false*;

}

*else* *if*(!playerTurn && !playerDefeated && !escape && !enemiesDefeated)

{

cout << QString::fromUtf8("=Ход врага=").toLocal8Bit().data() << "\n";

cout << QString::fromUtf8("Продолжить...").toLocal8Bit().data() << "\n\n";

cin.get();

system("CLS");

*//Enemy* *attack*

*for* (size\_t i = 0; i < enemies.size(); i++)

{

cout << QString::fromUtf8("Продолжить...").toLocal8Bit().data() << "\n\n";

cin.get();

system("CLS");

cout << QString::fromUtf8("Враг: ").toLocal8Bit().data() << i << "\n\n";

*//Attack* *roll*

combatTotal = enemies[i].getAccuracy() + (character.getDefence() + character.getAddedDefence());

enemyTotal = enemies[i].getAccuracy() / (double)combatTotal \* 100;

playerTotal = (character.getDefence() + character.getAddedDefence()) / (double)combatTotal \* 100;

combatRollPlayer = rand() % playerTotal + 1;

combatRollEnemy = rand() % enemyTotal + 1;

cout << "Combat total: " << combatTotal << "\n";

cout << "Enemy percent: " << enemyTotal << "\n";

cout << "Player percent: " << playerTotal << "\n\n";

cout << "Player roll: " << combatRollPlayer << "\n";

cout << "Enemy roll: " << combatRollEnemy << "\n\n";

*if* (combatRollPlayer < combatRollEnemy) *//Hit*

{

cout << "HIT! " << "\n\n";

damage = enemies[i].getDamage();

character.takeDamage(damage);

cout << "Damage: " << damage << "!" << "\n";

cout << "HP: " << character.getHP() << " / " << character.getHPMax() << "\n\n";

*if* (!character.isAlive())

{

cout << QString::fromUtf8("Вы проиграли!").toLocal8Bit().data() << "\n\n";

playerDefeated = *true*;

}

}

*else* *//Miss*

{

cout << QString::fromUtf8("Промах! \n\n").toLocal8Bit().data();

}

}

*//End* *turn*

playerTurn = *true*;

}

*//Conditions*

*if* (!character.isAlive())

{

playerDefeated = *true*;

}

*else* *if* (enemies.size() <= 0)

{

enemiesDefeated = *true*;

}

}

}

##### **Enemy.h**

#ifndef ENEMY\_H

#define ENEMY\_H

#include "STLINCLUDE.h"

*class* **Enemy**

{

*private*:

int level;

int hp;

int hpMax;

int damageMin;

int damageMax;

float dropChance;

int defence;

int accuracy;

*public*:

**Enemy**(int level = 0);

*virtual* ~***Enemy***();

*inline* bool **isAlive**() { *return* *this*->hp > 0; }

std::string **getAsString**()*const*;

void **takeDamage**(int damage);

*inline* int **getLevel**()*const* { *return* *this*->level; }

*inline* int **getDamageMin**()*const* { *return* *this*->damageMin; }

*inline* int **getDamageMax**()*const* { *return* *this*->damageMax; }

*inline* int **getDamage**()*const* { *return* rand() % *this*->damageMax + *this*->damageMin; }

*inline* int **getExp**()*const* { *return* *this*->level \* 100; }

*inline* int **getHp**()*const* { *return* *this*->hp; }

*inline* int **getHpMax**()*const* { *return* *this*->hpMax; }

*inline* int **getDefence**()*const* { *return* *this*->defence; }

*inline* int **getAccuracy**()*const* { *return* *this*->accuracy; }

};

#endif *//* *ENEMY\_H*

##### **Enemy.cpp**

#include "enemy.h"

Enemy::**Enemy**(int level)

{

*this*->level = level;

*this*->hpMax = rand()% (level \* 10) + (level \* 2);

*this*->hp = *this*->hpMax;

*this*->damageMin = *this*->level \* 1;

*this*->damageMax = *this*->level \* 2;

*this*->dropChance = rand() % 100 + 1;

*this*->defence = rand() % level\*5 + 1;

*this*->accuracy = rand() % level\*5 + 1;

}

Enemy::~***Enemy***()

{

}

std::string Enemy::**getAsString**()*const*

{

*return* "Level: " + std::to\_string(*this*->level) + "\n" +

"Hp: " + std::to\_string(*this*->hp) + " / " + std::to\_string(*this*->hpMax) + "\n" +

"Damage: " + std::to\_string(*this*->damageMin) + " - " + std::to\_string(*this*->damageMax) + "\n" +

"Defence: " + std::to\_string(*this*->defence) + "\n" +

"Accuracy: " + std::to\_string(*this*->accuracy) + "\n" +

"Drop chance: " + std::to\_string(*this*->dropChance) + "\n";

}

void Enemy::**takeDamage**(int damage)

{

*this*->hp -= damage;

*if* (*this*->hp <= 0)

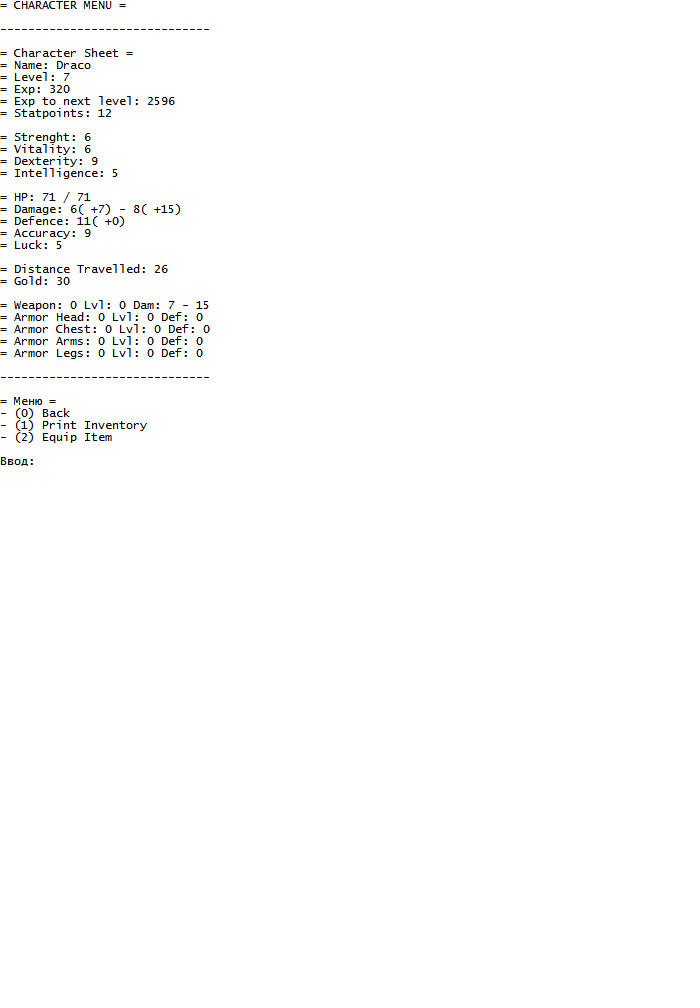
{

*this*->hp = 0;

}

}

# **Результат работы программы**

**