МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Студентка гр. 4384 _____ Зайченко Е.Э.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Ознакомиться с основами объектно-ориентированного программирования и применить их на практике, разработав на языке C++ прототип пошаговой игры, включающей перемещение игрового персонажа по карте и сражение с врагом.

Задание.

На 6/3/1 баллов:

Создать класс считывающий ввод пользователя и преобразующий ввод пользователь в объект команды.

Создать класс отрисовки игры. Данный класс определяет то, как должно отображаться игра.

Создать шаблонный класс управления игрой. В качестве параметра шаблона должен передаваться класс, отвечающий за считывание и преобразование ввода. У себя он создает объект класса из параметра шаблона и получает от него команды, а далее вызывает нужное действие у классов игры. Данный класс не должен создавать объект класса игры. Реализация должна быть такой, что можно масштабировать программу, например, реализовать получение команд через интернет без использования реализации интерфейса, и просто подставить новый класс в качестве параметра шаблона.

Создать шаблонный класс визуализации игры. . В качестве параметра шаблона должен передаваться класс, отвечающий за способ отрисовки игры. Данный класс создает объект класса отрисовки игры, и реагирует на изменения в игре, и вызывает команду отрисовку. Реализация должна быть такой, что можно масштабировать программу, например, реализовать отрисовку в виде веб-страницы без использования реализации интерфейса, и просто подставить новый класс в качестве параметра шаблона.

На 8/4/1.5 баллов:

Добавить возможность настраивать управление игрой через файл (то, на какие клавиши должна выполняться та или иная команда). Если команды некорректные: отсутствует информация для какой-то команды, на одну клавишу

две разные команды назначены, для одной команды назначены две разные клавиши, то в таком случае управление должно устанавливаться по умолчанию.

На 10/5/2 баллов:

Добавить систему логирования событий в игре. Система должна реагировать на игровые события, и записывать об этом событии (то и кому сколько урона было нанесено, на какие координаты перешел игрок, получение заклинания, и.т.д.). Игровые сущности не должны напрямую вызывать систему логировать, а только лишь информировать о событии. Запись может идти как в файл, так и в терминал, способ логирования определяется пользователем через параметры запуска программы.

Примечания:

После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду

Для представления команды можно разработать системы классов

Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”

Выполнение работы.

Была реализована программа, содержащая все указанные в условии лабораторной работы классы и их поля и методы, а именно:

- Класс, считывающий ввод пользователя и преобразующий ввод пользователь в объект команды.
- Класс отрисовки игры. Данный класс определяет то, как должно отображаться игра.
- Шаблонный класс управления игрой.

Архитектура программы.

В программе реализована иерархия классов, соответствующая принципам ООП.

Описание классов.

Основные классы:

- Класс Game

  Класс содержит игровые объекты (поле, игрока, врагов и руку), а также методы для управления игровым процессом.

  Поля класса:

  - field_ – игровое поле.
  - player_ – объект игрока.
  - enemies – список врагов, находящихся на игровом поле.
  - hand_ – рука игрока, содержащая его заклинания.
  - game_over_ – флаг, показывающий, завершена ли игра поражением.
  - level_complete_ – флаг, показывающий успешное прохождение уровня.

  Методы класса:

  - game – конструктор, инициализирующий поле, игрока, врагов и руку.
  - run – основной метод запуска, который предлагает игроку начать новую игру или загрузить сохранённую.
  - start_level – создание нового уровня, размещение игрока и врагов.
  - execute – ход игрока (движение, атаки, заклинания, команды).
  - enemies_turn – ход врагов (движение к игроку, атаки).
  - check_victory_condition – проверка поражены ли все враги.
  - check_defeat_condition – проверка, жив ли игрок.
  - get_enemy_at – поиск врага по координатам клетки.
  - remove_dead_enemies – удаление всех пораженных врагов.
  - spawn_enemy – создание нового врага.
  - save – сохранение состояния игры.
  - saveToFile – запись данных в файл (игрок, враги, заклинания).

- load – загрузка состояния игры.
- loadFromFile – чтение данных из файла и восстановление игрового состояния.

- Класс ConsoleInputReader

  Класс, отвечающий за считывание действий пользователя из консоли и преобразование их в объект Command.

  Методы класса:

  - read – считывает выбор пользователя через меню, создаёт и возвращает объект Command, обрабатывает ошибки пользовательского ввода (некорректные символы, строки).
- Класс ConsoleRenderer

  Класс, отвечающий за отрисовку состояния игры в консоль.

  Методы класса:

  - render – Вызывает print_field у поля, передаёт туда позицию игрока и врагов, реализует консольный способ отображения.
- Класс GameView

  Шаблонный класс визуализации игры. Позволяет подключать разные подсистемы отображения.

  Методы класса:

  - GameView – конструктор, связывающий отображаемую игру и вызуализатор.
  - unpdate – вызывает render, отображает актуальное состояние игры.
- Класс ActionType

  Enum класс, содержащий типы игровых действий, которые может выполнять игрок после команды ввода.

  - None – действие отсутствует
  - Move – движение
  - Attack – атака

- CastSpell – использование заклинания
- ShowSpells – вывод списка заклинаний
- Save – сохранение игры
- Load – загрузка игры

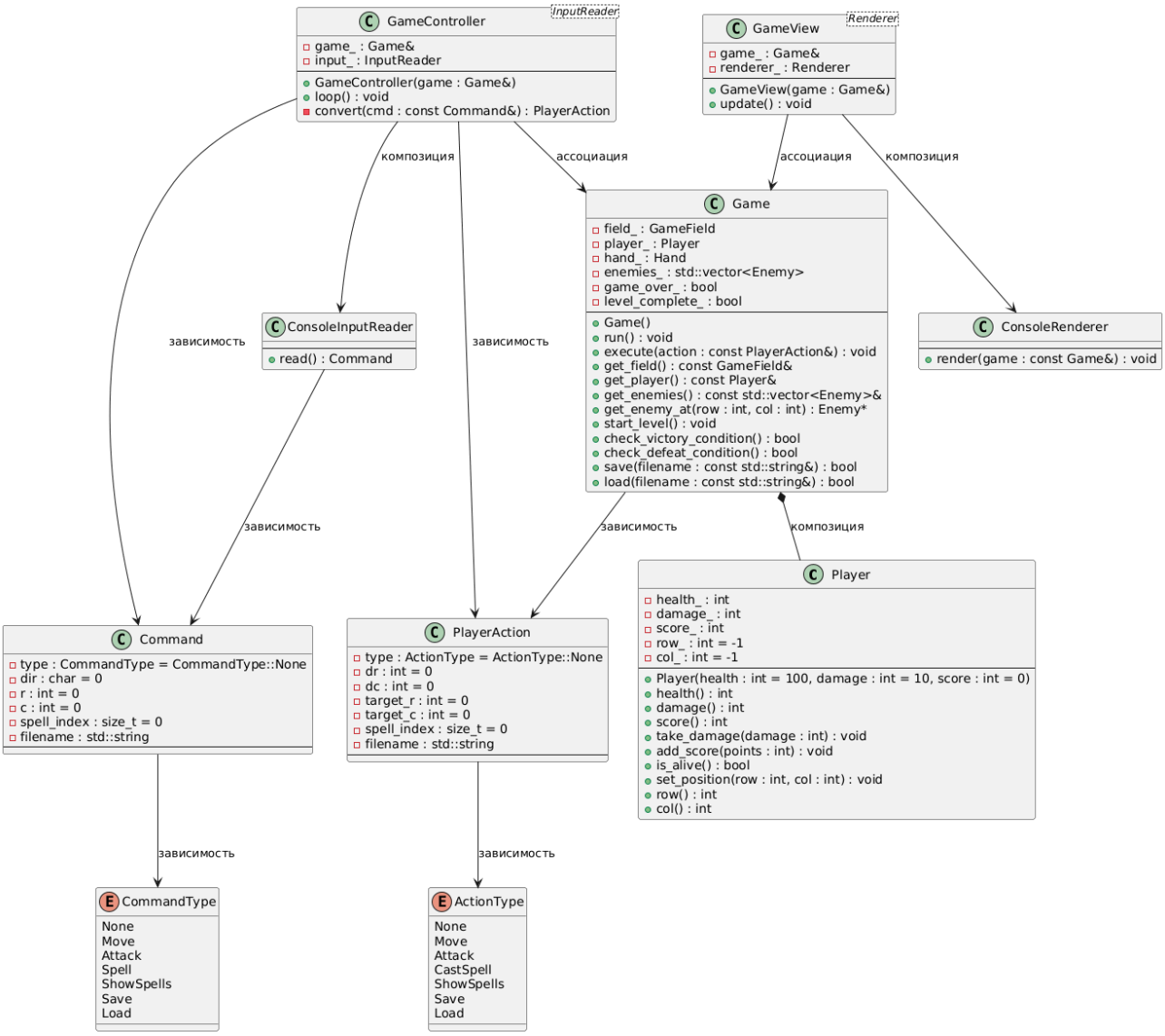
- Класс CommandType

  Enum класс, содержащий виды команд, приходящих от устройства ввода (движение, атака, применение заклинания, вывод имеющихся заклинаний, сохранение игры, загрузка сохраненной игры).

  - None – действие отсутствует
  - Move – движение
  - Attack – атака
  - CastSpell – использование заклинания
  - ShowSpells – вывод списка заклинаний
  - Save – сохранение игры
  - Load – загрузка игры

Разработанный программный код см. в приложении А.

UML-диаграмма



Вывод.

Была изучена парадигма объектно-ориентированного программирования. Была реализована программа на языке C++ включающая основные классы игры с необходимыми полями и методами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#define NOMINMAX
#include <Windows.h>
#include <locale>

#include "game.h"
#include "input_reader.h"
#include "renderer.h"
#include "game_controller.h"
#include "game_view.h"

int main() {

    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    setlocale(LC_ALL, "");

    Game game;

    GameController<ConsoleInputReader> controller(game);
    GameView<ConsoleRenderer> view(game);


    game.run();


    while (true) {


        view.update();

        controller.loop();

        if (game.check_victory_condition()) {
            std::cout << "\n=== Уровень пройден! ===\n";
            std::cout << "Начать новый уровень? (y/n): ";
            char c;
            std::cin >> c;

            if (c == 'y' || c == 'Y') {
                game.start_level();
                continue;
            }
            else {
                std::cout << "Игра завершена." << std::endl;
                break;
            }
        }
```

```
        if (game.check_defeat_condition()) {
            std::cout << "\n=== Вы погибли! ===\n";
            std::cout << "Начать заново? (y/n): ";
            char c;
            std::cin >> c;

            if (c == 'y' || c == 'Y') {
                game.start_level();
                continue;
            }
            else {
                std::cout << "Игра завершена." << std::endl;
                break;
            }
        }
    }

    return 0;
}
```

player.cpp

```
#include "player.h"
Player::Player(int health, int damage, int score)
    : health_(health), damage_(damage), score_(score) { // Теперь
можно копировать damage_
}
void Player::take_damage(int damage) {
    if (damage > 0) {
        health_ -= damage;
        if (health_ < 0) health_ = 0;
    }
}
void Player::add_score(int points) {
    if (points > 0) {
        score_ += points;
    }
}
bool Player::is_alive() const {
    return health_ > 0;
}
void Player::set_position(int row, int col) {
    row_ = row;
    col_ = col;
}
```

player.h

```
#ifndef GAME_PLAYER_H_
#define GAME_PLAYER_H_
class Player {
public:
```

```
    explicit Player(int health = 100, int damage = 10, int score =
0);
    // Убрана строка: Player& operator=(const Player&) = delete;
    int health() const { return health_; }
    int damage() const { return damage_; }
    int score() const { return score_; }
    void take_damage(int damage);
    void add_score(int points);
    bool is_alive() const;
    void set_position(int row, int col);
    int row() const { return row_; }
    int col() const { return col_; }
private:
    int health_;
    int damage_; // Убран const
    int score_;
    int row_ = -1;
    int col_ = -1;
};
#endif
```

enemy.cpp

```
#include "enemy.h"

Enemy::Enemy(int health, int damage)
    : health_(health), damage_(damage) {
}

void Enemy::take_damage(int damage) {
    if (damage > 0) {
        health_ -= damage;
        if (health_ < 0) health_ = 0;
    }
}

void Enemy::set_position(int row, int col) {
    row_ = row;
    col_ = col;
}
```

enemy.h

```
#ifndef GAME_ENEMY_H_
#define GAME_ENEMY_H_

class Enemy {
public:
    explicit Enemy(int health = 50, int damage = 15);

    int health() const { return health_; }
    int damage() const { return damage_; }
```

```
    void take_damage(int damage);

    void set_position(int row, int col);
    int row() const { return row_; }
    int col() const { return col_; }

    bool is_alive() const { return health_ > 0; }

private:
    int health_;
    int damage_;
    int row_ = -1;
    int col_ = -1;
};

#endif
```

game_field.cpp

```
#include "game_field.h"
#include <iostream>

GameField::GameField(int rows, int cols)
    : rows_(rows), cols_(cols), grid_(rows,
std::vector<Cell>(cols)) {
    if (rows < 1 || cols < 1) {
        throw std::invalid_argument("Field size must be
positive.");
    }
}

GameField::GameField(const GameField& other)
    : rows_(other.rows_), cols_(other.cols_), grid_(other.grid_) {
}

GameField& GameField::operator=(const GameField& other) {
    if (this != &other) {
        rows_ = other.rows_;
        cols_ = other.cols_;
        grid_ = other.grid_;
    }
    return *this;
}

GameField::GameField(GameField&& other) noexcept
    : rows_(other.rows_), cols_(other.cols_),
grid_(std::move(other.grid_)) {
}

GameField& GameField::operator=(GameField&& other) noexcept {
    if (this != &other) {
        rows_ = other.rows_;
```

```
        cols_ = other.cols_;
        grid_ = std::move(other.grid_);
    }
    return *this;
}

bool GameField::is_valid_position(int row, int col) const {
    return row >= 0 && row < rows_ && col >= 0 && col < cols_;
}

bool GameField::is_empty(int row, int col) const {
    return is_valid_position(row, col) &&
        grid_[row][col].type() == Cell::EntityType::kEmpty;
}

void GameField::update_cell(int row, int col, Cell::EntityType
type) {
    if (is_valid_position(row, col)) {
        grid_[row][col].set_type(type);
    }
}

void GameField::place_player(const Player& player) {
    update_cell(player.row(), player.col(),
Cell::EntityType::kPlayer);
}

void GameField::clear_cell(int row, int col) {
    update_cell(row, col, Cell::EntityType::kEmpty);
}

void GameField::move_player(int new_row, int new_col, Player&
player) {
    if (!is_valid_position(new_row, new_col)) return;
    if (!is_empty(new_row, new_col)) return;

    clear_cell(player.row(), player.col());
    player.set_position(new_row, new_col);
    place_player(player);
}

void GameField::move_enemy(int new_row, int new_col, Enemy& enemy,
Player& player) {
    if (!is_valid_position(new_row, new_col)) return;

    if (new_row == player.row() && new_col == player.col()) {
        player.take_damage(enemy.damage());
        return;
    }

    if (!is_empty(new_row, new_col)) return;

    clear_cell(enemy.row(), enemy.col());
```

```
    enemy.set_position(new_row, new_col);
    update_cell(new_row, new_col, Cell::EntityType::kEnemy);
}

void GameField::print_field(const Player& player, const
std::vector<Enemy>& enemies) const {
    std::vector<std::vector<char>> temp(rows_,
std::vector<char>(cols_, '.'));

    temp[player.row()][player.col()] = 'P';

    for (const auto& e : enemies) {
        if (e.is_alive()) {
            if (is_valid_position(e.row(), e.col()))
                temp[e.row()][e.col()] = 'E';
        }
    }

    std::cout << "   ";
    for (int c = 0; c < cols_; ++c) {
        std::cout << c << ' ';
    }
    std::cout << '\n';

    for (int r = 0; r < rows_; ++r) {
        std::cout << r << ": ";
        for (int c = 0; c < cols_; ++c) {
            std::cout << temp[r][c] << ' ';
        }
        std::cout << '\n';
    }
}
```

game_field.h

```
#ifndef GAME_GAME_FIELD_H_
#define GAME_GAME_FIELD_H_
#include <vector>
#include <stdexcept>
#include "cell.h"
#include "player.h"
#include "enemy.h"
class GameField {
public:
    GameField(int rows, int cols);
    ~GameField() = default;
    GameField(const GameField& other);
    GameField& operator=(const GameField& other);
    GameField(GameField&& other) noexcept;
    GameField& operator=(GameField&& other) noexcept;
    bool is_valid_position(int row, int col) const;
    bool is_empty(int row, int col) const;
```

```
    void place_player(const Player& player);
    void clear_cell(int row, int col);
    void move_player(int new_row, int new_col, Player& player);
    void move_enemy(int new_row, int new_col, Enemy& enemy,
Player& player);
    void print_field(const Player& player, const
std::vector<Enemy>& enemies) const;
    int rows() const { return rows_; }
    int cols() const { return cols_; }
    void update_cell(int row, int col, Cell::EntityType type);
private:
    int rows_;
    int cols_;
    std::vector<std::vector<Cell>> grid_;
};
#endif
```

cell.h

```
#ifndef GAME_CELL_H_
#define GAME_CELL_H_

class Cell {
public:
    enum class EntityType {
        kEmpty,
        kPlayer,
        kEnemy
    };

    Cell() = default;
    explicit Cell(EntityType type) : type_(type) {}

    EntityType type() const { return type_; }
    void set_type(EntityType type) { type_ = type; }

private:
    EntityType type_ = EntityType::kEmpty;
};

#endif // GAME_CELL_H_
```

hand.cpp

```
#include "hand.h"

Hand::Hand(size_t max_size) : max_size_(max_size) {
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    spells_.push_back(std::make_unique<DirectDamageSpell>(20, 1));
    if (spells_.size() < max_size_)
        spells_.push_back(std::make_unique<AreaDamageSpell>(20,
2));
```

```
}

void Hand::add_random_spell() {
    if (spells_.size() >= max_size_) {
        std::cout << "Рука заполнена, нельзя добавить новое
заклинание.\n";
        return;
    }

    int choice = std::rand() % 2;
    if (choice == 0)
        spells_.push_back(std::make_unique<DirectDamageSpell>(20,
1));
    else
        spells_.push_back(std::make_unique<AreaDamageSpell>(20,
2));

    std::cout << "Игрок получил новое заклинание: " <<
spells_.back()->name() << "\n";
}

void Hand::use_spell(size_t index, Game& game, GameField& field,
Player& player, int row, int col) {
    if (index >= spells_.size()) {
        std::cout << "Некорректный выбор заклинания.\n";
        return;
    }
    spells_[index]->use(game, field, player, row, col);
}

void Hand::show_spells() const {
    std::cout << "Заклинания в руке:\n";
    for (size_t i = 0; i < spells_.size(); ++i)
        std::cout << "  [" << i << "] " << spells_[i]->name() <<
"\n";
}
```

hand.h

```
#ifndef GAME_HAND_H_
#define GAME_HAND_H_
#include <vector>
#include <memory>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "spell.h"
#include "direct_damage_spell.h"
#include "area_damage_spell.h"
class Game;
class Hand {
public:
    explicit Hand(size_t max_size = 3);
```

```
    void add_random_spell();
    void use_spell(size_t index, Game& game, GameField& field,
Player& player, int row, int col);
    void show_spells() const;
    size_t size() const { return spells_.size(); }
    void clear_spells() { spells_.clear(); }
    void add_spell(std::unique_ptr<Spell> spell) {
        if (spells_.size() < max_size_) {
            spells_.push_back(std::move(spell));
        }
    }
    friend class Game;
private:
    std::vector<std::unique_ptr<Spell>> spells_;
    size_t max_size_;
};
#endif
```

spell.h

```
#ifndef GAME_SPELL_H_
#define GAME_SPELL_H_
class Game;
#include "game_field.h"
#include "player.h"
#include "enemy.h"
class Spell {
public:
    virtual ~Spell() = default;
    virtual void use(Game& game, GameField& field, Player& player,
int target_row, int target_col) = 0;
    virtual const char* name() const = 0;
    virtual int getType() const = 0;
    virtual int getDamage() const = 0;
    virtual int getRadius() const = 0;
};
#endif
```

direct_damage_spell.cpp

```
#include "direct_damage_spell.h"
#include <cmath>
#include "game.h"

void DirectDamageSpell::use(Game& game, GameField& field, Player&
player, int target_row, int target_col) {
    if (!field.is_valid_position(target_row, target_col)) {
        std::cout << "Цель вне поля!\n";
        return;
    }
```

```
    int dist = std::abs(player.row() - target_row) +
std::abs(player.col() - target_col);
    if (dist > radius_) {
        std::cout << "Цель вне радиуса заклинания!\n";
        return;
    }

    Enemy* enemy = game.get_enemy_at(target_row, target_col);
    if (enemy) {
        enemy->take_damage(damage_);
        std::cout << "Заклинание '" << name() << "' нанесло " <<
damage_
            << " урона врагу в клетке (" << target_row << ", " <<
target_col
            << "). Здоровье врага теперь: " << enemy->health() <<
"\n";

        if (!enemy->is_alive()) {
            std::cout << "Враг уничтожен!\n";
            field.clear_cell(target_row, target_col);
            game.remove_dead_enemies();
        }
    }
    else {
        std::cout << "В клетке (" << target_row << ", " <<
target_col << ") врагов нет.\n";
    }
}
```

direct_damage_spell.h

```
#ifndef GAME_DIRECT_DAMAGE_SPELL_H_
#define GAME_DIRECT_DAMAGE_SPELL_H_
#include "spell.h"
#include <cmath>
#include <iostream>
class DirectDamageSpell : public Spell {
public:
    explicit DirectDamageSpell(int damage, int radius = 1)
        : damage_(damage), radius_(radius) {
    }
    void use(Game& game, GameField& field, Player& player, int
target_row, int target_col) override;
    const char* name() const override { return "Прямой урон"; }
    int getType() const override { return 0; }
    int getDamage() const override { return damage_; }
    int getRadius() const override { return radius_; }
private:
    int damage_;
    int radius_;
};
#endif
```

area_damage_spell.cpp

```
#include <cmath>
#include "area_damage_spell.h"
#include "game.h"

void AreaDamageSpell::use(Game& game, GameField& field, Player&
player, int target_row, int target_col) {
    if (!field.is_valid_position(target_row, target_col)) {
        std::cout << "Цель вне поля!\n";
        return;
    }

    std::cout << "Заклинание '" << name() << "' нанесло " <<
damage_
        << " урона по области с центром в (" << target_row << ", "
<< target_col << ")\n";

     for (int r = target_row - radius_; r <= target_row + radius_;
++r) {
        for (int c = target_col - radius_; c <= target_col +
radius_; ++c) {
            if (!field.is_valid_position(r, c)) continue;
            Enemy* enemy = game.get_enemy_at(r, c);
            if (enemy) {
                enemy->take_damage(damage_);
                std::cout << " -> Враг в клетке (" << r << ", " <<
c
                    << ") получил " << damage_ << " урона.
Осталось: " << enemy->health() << "\n";
                if (!enemy->is_alive()) {
                    std::cout << "   Враг в клетке (" << r << ", "
<< c << ") уничтожен!\n";
                    field.clear_cell(r, c);
                }
            }
        }
    }

    game.remove_dead_enemies();
}
```

area_damage_spell.h

```
#ifndef GAME_AREA_DAMAGE_SPELL_H_
#define GAME_AREA_DAMAGE_SPELL_H_
#include "spell.h"
#include <iostream>
class AreaDamageSpell : public Spell {
public:
```

```
    explicit AreaDamageSpell(int damage, int radius = 2)
        : damage_(damage), radius_(radius) {
    }
    void use(Game& game, GameField& field, Player& player, int
target_row, int target_col) override;
    const char* name() const override { return "Урон по области"; }
    int getType() const override { return 1; }
    int getDamage() const override { return damage_; }
    int getRadius() const override { return radius_; }
private:
    int damage_;
    int radius_;
};
#endif
```

game.cpp

```
#include "game.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <string>
#include <cstdlib>
#include <ctime>
#include <cctype>
Game::Game()
    : field_(10, 10),
    player_(100, 10),
    hand_(3),
    game_over_(false),
    level_complete_(false)
{
    std::srand(static_cast<unsigned>(std::time(nullptr)));
}
bool Game::save(const std::string& filename) const noexcept {
    try {
        saveToFile(filename);
        std::cout << "Игра   успешно   сохранена   в  '" <<
filename << "'.\n";
        return true;
    }
    catch (const GameSaveException& e) {
        std::cerr << "Ошибка сохранения: " << e.what() << "\n";
        return false;
    }
}
bool Game::load(const std::string& filename) noexcept {
    try {
        loadFromFile(filename);
        std::cout << "Игра   успешно   загружена   из  '" <<
filename << "'.\n";
```

```
        return true;
    }
    catch (const GameLoadException& e) {
        std::cerr << " Ошибка   загрузки : " << e.what() << "\n";
        return false;
    }
}
void Game::run() {
    std::cout << "Добро пожаловать в игру!\n";
    std::cout << "Загрузить сохранённую игру?  (y/n): ";
    char load_choice;
    std::cin >> load_choice;
    if (std::tolower(load_choice) == 'y') {
        std::cout << "Введите имя файла сохранения: ";
        std::string load_filename;
        std::cin >> load_filename;
        load(load_filename);
    }
    start_level();
}
void Game::start_level() {
    std::cout << "\n Запуск   уровня ...\n";
    enemies_.clear();
    player_.set_position(0, 0);
    field_.clear_cell(player_.row(), player_.col());
    field_.place_player(player_);
    for (int i = 0; i < 4; ++i) {
        int r = std::rand() % field_.rows();
        int c = std::rand() % field_.cols();
        if (r == player_.row() && c == player_.col()) { --i;
continue; }
        enemies_.emplace_back(20, 10);
        enemies_.back().set_position(r, c);
        field_.update_cell(r, c, Cell::EntityType::kEnemy);
    }
}
void Game::execute(const PlayerAction& action) {
    bool player_acting = false;
    switch (action.type) {
    case ActionType::Move: {
        int nr = player_.row() + action.dr;
        int nc = player_.col() + action.dc;
        field_.move_player(nr, nc, player_);
        player_acting = true;
        break;
    }
    case ActionType::Attack: {
        Enemy* e = get_enemy_at(action.target_r, action.target_c);
        if (e) {
            e->take_damage(player_.damage());
            std::cout << " Вы   нанесли  " << player_.damage() <<
"  урона   врагу !\n";
```

```
            if (!e->is_alive()) field_.clear_cell(action.target_r,
action.target_c);
            remove_dead_enemies();
        }
        else {
            std::cout << " В   этой   клетке   никого   нет .\n";
        }
        player_acting = true;
        break;
    }
    case ActionType::CastSpell: {
        if (action.spell_index >= hand_.size()) {
            std::cout << "Некорректный индекс заклинания.\n";
        }
        else {
            hand_.use_spell(action.spell_index, *this, field_,
player_, action.target_r, action.target_c);
            player_acting = true;
        }
        break;
    }
    case ActionType::ShowSpells:
        hand_.show_spells();
        break;
    case ActionType::Save:
        if (!action.filename.empty()) save(action.filename);
        break;
    case ActionType::Load:
        if (!action.filename.empty()) load(action.filename);
        break;
    default:
        break;
    }
    if (player_acting) {
        enemies_turn();
    }
}
void Game::enemies_turn() {
    std::cout << "\n===  Ход   врагов  ===\n";
    for (auto& enemy : enemies_) {
        if (!enemy.is_alive()) continue;
        int er = enemy.row();
        int ec = enemy.col();
        int pr = player_.row();
        int pc = player_.col();
        int dr = (pr > er) ? 1 : (pr < er ? -1 : 0);
        int dc = (pc > ec) ? 1 : (pc < ec ? -1 : 0);
        int new_r = er + dr;
        int new_c = ec + dc;
        if (new_r == player_.row() && new_c == player_.col()) {
            player_.take_damage(enemy.damage());
            std::cout << "Враг в (" << er << "," << ec << ")
атаковал игрока на "
```

```
                << enemy.damage() << "  урона ! "
                << " Здоровье   игрока : " << player_.health() <<
"\n";
            continue;
        }
        field_.move_enemy(new_r, new_c, enemy, player_);
    }
    remove_dead_enemies();
}
Enemy* Game::get_enemy_at(int row, int col) {
    for (auto& e : enemies_)
        if (e.row() == row && e.col() == col && e.is_alive())
return &e;
    return nullptr;
}
void Game::remove_dead_enemies() {
    enemies_.erase(std::remove_if(enemies_.begin(),  enemies_.end(),
        [](const Enemy& e) { return !e.is_alive(); }),
enemies_.end());
}
void Game::spawn_enemy(int health, int damage, int row, int col) {
    enemies_.emplace_back(health, damage);
    enemies_.back().set_position(row, col);
}
void Game::saveToFile(const std::string& filename) const {
    std::ofstream file(filename);
    if (!file.is_open()) {
        throw GameSaveException("Не удалось открыть файл для
сохранения: " + filename);
    }
    file << player_.health() << " " << player_.score() << " " <<
player_.row() << " " << player_.col() << "\n";
    file << enemies_.size() << "\n";
    for (const auto& enemy : enemies_) {
        file << enemy.health() << " " << enemy.damage() << " " <<
enemy.row() << " " << enemy.col() << "\n";
    }
    file << hand_.size() << "\n";
    for (size_t i = 0; i < hand_.size(); ++i) {
        file << hand_.spells_[i]->getType() << " " <<
hand_.spells_[i]->getDamage() << " " << hand_.spells_[i]-
>getRadius() << "\n";
    }
    file.close();
}
void Game::loadFromFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        throw GameLoadException("Не удалось открыть файл для
загрузки: " + filename);
    }
    int p_health, p_score, p_row, p_col;
    file >> p_health >> p_score >> p_row >> p_col;
```

```
    player_ = Player(p_health, player_.damage(), p_score); // Тут
возникает ошибка из-за operator= = delete и const damage_
    player_.set_position(p_row, p_col);
    int num_enemies;
    file >> num_enemies;
    enemies_.clear();
    for (int i = 0; i < num_enemies; ++i) {
        int e_health, e_damage, e_row, e_col;
        file >> e_health >> e_damage >> e_row >> e_col;
        enemies_.emplace_back(e_health, e_damage);
        enemies_.back().set_position(e_row, e_col);
    }
    int num_spells;
    file >> num_spells;
    hand_.clear_spells();
    for (int i = 0; i < num_spells; ++i) {
        int type, damage, radius;
        file >> type >> damage >> radius;
        if (type == 0) {

hand_.add_spell(std::make_unique<DirectDamageSpell>(damage,
radius));
        }
        else if (type == 1) {

hand_.add_spell(std::make_unique<AreaDamageSpell>(damage,  radius));
        }
    }
    file.close();
}
bool Game::check_victory_condition() const {
    for (const auto& enemy : enemies_) {
        if (enemy.is_alive()) {
            return false;
        }
    }
    return true;
}
bool Game::check_defeat_condition() const {
    return !player_.is_alive();
}
```

game.h

```
#ifndef GAME_GAME_H_
#define GAME_GAME_H_

#include <iostream>
#include <vector>
#include <algorithm>
#include <memory>
#include <string>
```

```
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <stdexcept>

#include "save_exception.h"
#include "load_exception.h"
#include "player.h"
#include "enemy.h"
#include "game_field.h"
#include "hand.h"
#include "action_type.h"
#include "action.h"

class Game {
public:
    Game();

    void run();
    void execute(const PlayerAction& action);

    const GameField& get_field() const { return field_; }
    const Player& get_player() const { return player_; }
    const std::vector<Enemy>& get_enemies() const { return
enemies_; }

    Enemy* get_enemy_at(int row, int col);
    void remove_dead_enemies();
    void spawn_enemy(int health, int damage, int row, int col);

    bool save(const std::string& filename) const noexcept;
    bool load(const std::string& filename) noexcept;

    void start_level();
    bool check_victory_condition() const;
    bool check_defeat_condition() const;

private:
    void start();
    void enemies_turn();
    void ask_restart_or_exit(bool& running);
    void saveToFile(const std::string& filename) const;
    void loadFromFile(const std::string& filename);

    GameField field_;
    Player player_;
    Hand hand_;
    std::vector<Enemy> enemies_;

    bool game_over_;
    bool level_complete_;
};
```

```
#endif // GAME_GAME_H_
```

save_exception.h

```
#ifndef GAME_SAVE_EXCEPTION_H_
#define GAME_SAVE_EXCEPTION_H_
#include <exception>
#include <string>

class GameSaveException : public std::exception {
public:
    explicit GameSaveException(const std::string& msg) : msg_(msg)
{}
    const char* what() const noexcept override { return
msg_.c_str(); }
private:
    std::string msg_;
};
#endif // GAME_SAVE_EXCEPTION_H_
```

load_exception.h

```
#ifndef GAME_LOAD_EXCEPTION_H_
#define GAME_LOAD_EXCEPTION_H_
#include <exception>
#include <string>

class GameLoadException : public std::exception {
public:
    explicit GameLoadException(const std::string& msg) : msg_(msg)
{}
    const char* what() const noexcept override { return
msg_.c_str(); }
private:
    std::string msg_;
};
#endif // GAME_LOAD_EXCEPTION_H_
```

action_type.h

```
#pragma once
enum class ActionType {
    None,
    Move,
    Attack,
    CastSpell,
    ShowSpells,
    Save,
    Load
};
```

action.h

```
#pragma once
#include <string>
#include "action_type.h"

struct PlayerAction {
    ActionType type = ActionType::None;
    int dr = 0;
    int dc = 0;
    int target_r = 0;
    int target_c = 0;
    size_t spell_index = 0;
    std::string filename;
};
```

command_type.h

```
#pragma once
#include <string>
#include "command_type.h"
```

```
struct Command {
    CommandType type = CommandType::None;
    char dir = 0;
    int r = 0;
    int c = 0;
    size_t spell_index = 0;
    std::string filename;
};
```

command.h

```
#pragma once
#include <string>
#include "command_type.h"

struct Command {
    CommandType type = CommandType::None;
    char dir = 0;
    int r = 0;
    int c = 0;
    size_t spell_index = 0;
    std::string filename;
};
```

game_controller.h

```
#pragma once
#include "command.h"
#include "action.h"
#include "action_type.h"
#include "game.h"

template <typename InputReader>
class GameController {
public:
    GameController(Game& game) : game_(game), input_() {}

    void loop() {
        Command cmd = input_.read();
        PlayerAction action = convert(cmd);
        game_.execute(action);
    }

private:
    Game& game_;
    InputReader input_;

    PlayerAction convert(const Command& cmd) {
        PlayerAction action;

        switch (cmd.type) {
```

```
        case CommandType::Move:
            action.type = ActionType::Move;
            switch (cmd.dir) {
            case 'w': case 'W': action.dr = -1; action.dc = 0;
break;
            case 's': case 'S': action.dr = 1;  action.dc = 0;
break;
            case 'a': case 'A': action.dr = 0;  action.dc = -1;
break;
            case 'd': case 'D': action.dr = 0;  action.dc = 1;
break;
            }
            break;

        case CommandType::Attack:
            action.type = ActionType::Attack;
            action.target_r = cmd.r;
            action.target_c = cmd.c;
            break;

        case CommandType::Spell:
            action.type = ActionType::CastSpell;
            action.spell_index = cmd.spell_index;
            action.target_r = cmd.r;
            action.target_c = cmd.c;
            break;

        case CommandType::ShowSpells:
            action.type = ActionType::ShowSpells;
            break;

        case CommandType::Save:
            action.type = ActionType::Save;
            action.filename = cmd.filename;
            break;

        case CommandType::Load:
            action.type = ActionType::Load;
            action.filename = cmd.filename;
            break;

        default:
            action.type = ActionType::None;
            break;
        }

        return action;
    }
};
```

game_view.h

```
#pragma once
#include "renderer.h"
#include "game.h"

template <typename Renderer>
class GameView {
public:
    GameView(Game& game) : game_(game), renderer_() {}

    void update() {
        renderer_.render(game_);
    }

private:
    Game& game_;
    Renderer renderer_;
};
```

input_reader.h

```
#pragma once
#include <iostream>
#include <string>
#include "command.h"

class ConsoleInputReader {
public:
    Command read() {
        Command cmd;
        std::cout << "\n=== Ход игрока ===\n";
        std::cout << "1 - ход\n2 - атака\n3 - заклинание\n4 -
показать заклинания\n5 - сохранить\n6 - загрузить\nВаш выбор: ";

        int choice;
        if (!(std::cin >> choice)) {
            std::cin.clear();
            std::string dump; std::getline(std::cin, dump);
            return cmd;
        }

        switch (choice) {
        case 1: {
            cmd.type = CommandType::Move;
            std::cout << "Ход (W/A/S/D): ";
            std::cin >> cmd.dir;
            break;
        }
        case 2: {
            cmd.type = CommandType::Attack;
            std::cout << "Введите координаты атаки (r c): ";
            std::cin >> cmd.r >> cmd.c;
```

```
            break;
        }
        case 3: {
            cmd.type = CommandType::Spell;
            std::cout << "Введите номер заклинания: ";
            std::cin >> cmd.spell_index;
            std::cout << "Введите координаты цели (r c): ";
            std::cin >> cmd.r >> cmd.c;
            break;
        }
        case 4:
            cmd.type = CommandType::ShowSpells;
            break;
        case 5:
            cmd.type = CommandType::Save;
            std::cout << "Введите имя файла: ";
            std::cin >> cmd.filename;
            break;
        case 6:
            cmd.type = CommandType::Load;
            std::cout << "Введите имя файла: ";
            std::cin >> cmd.filename;
            break;

        default:
            cmd.type = CommandType::None;
            break;
        }

        return cmd;
    }
};
```

renderer.h

```
#pragma once
#include "game.h"

class ConsoleRenderer {
public:
    void render(const Game& game) {
        game.get_field().print_field(
            game.get_player(),
            game.get_enemies()
        );
    }
};
```