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

ОТЧЕТ

по лабораторной работе №3

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:

1. Создать класс(ы) игры, который реализует основной цикл игры, и которому передаются команды от пользователя. Игровой цикл состоит из следующих шагов:
    a. Начало игры
    b. Запуск уровня
    c. Ход игрока. Ход, атака или применение заклинания.
    d. Ход союзников - если имеются
    e. Ход врагов
    f. Ход вражеской базы и башни - если имеются

    Условие прохождения уровня студент определяет самостоятельно. Если игрок проигрывает, то игроку должно предлагаться начать заново игру, либо выйти из программы.

    Все взаимодействие должно происходить через классы игры.
2. Реализовать систему сохранения и загрузки игры. Пользователь должен иметь возможность сохранить игру в любой момент. Пользователь должен иметь возможность загружаться при запуске программы (или выбрать новую), либо во время игры. Сохранения должны оставаться в консистентном состоянии между запусками игры.
3. Добавить обработку исключительных ситуаций для загрузки/сохранения, например, невозможность записать в файл, нельзя загрузиться так как файл не существует или в нем некорректные данные.

На 8/4/1.5 баллов:

4. Реализовать переход на следующий уровень, после прохождения уровня. При переходе на следующий уровень создается новое поле другого размера с более сильными врагами. При переходе на следующий уровень, значение жизни игрока восстанавливается, и половина его карточек заклинаний случайным образом удаляется.

На 10/5/2 баллов:

5. Реализовать прокачку игрока при переходе между уровнями. Пользователь может улучшить характеристики игрока или улучшить заклинание (что и как улучшать определяет студент). Для этого нужно расширить игровой цикл.

Примечания:

- Класс игры может знать о игровых сущностях, но не наоборот;
- При работе с файлом используйте идиому RAII;
- Исключения должны обязательно обрабатываться, и программа не должна завершаться;
- Исключения должны быть информативными (содержать информацию о том, что и где произошло), на разные виды исключительных ситуаций должны быть свои исключения.

Выполнение работы.

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

- Основной игровой цикл, включающий начало игры, запуск уровня, ходы игрока, союзников, врагов и вражеской базы/башен;
- Возможность завершения уровня по условию (например, поражение всех врагов) и перезапуска игры при поражении игрока;
- Система сохранения и загрузки игры, позволяющая сохранить прогресс в любой момент и восстановить его при последующем запуске;
- Обработка исключительных ситуаций при работе с файлами: проверка существования файла, корректности данных, а также ошибок ввода-вывода при сохранении и загрузке.

Архитектура программы.

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- Game – используется для запуска игры, создания уровня, выполнения ходов игрока и врагов. Также здесь реализовано сохранение и загрузка состояния игры.
- GameSaveException – используется для ошибок, возникающих во время сохранения игры.
- GameLoadException – используется для ошибок, возникающих при загрузке игры.

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 – возвращает текущую строку позиции игрока
- start_level – создание нового уровня, размещение игрока и врагов.
- player_turn – 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.
- ask_restart_or_exit – запрос у игрока, хочет ли он начать заново или выйти из игры.
- 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 GameSaveException

Класс служит для передачи сообщения о проблеме при сохранении.

Методы класса:

- GameSaveException – конструктор, принимающий сообщение об ошибке.

• Класс GameLoadException

Класс служит для передачи сообщения о проблеме при загрузке.

Методы класса:

- GameLoadException – конструктор, принимающий описание ошибки.

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 <iostream>
#include <locale>
#include "game.h"

int main() {
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    setlocale(LC_ALL, "");

    Game game;
    game.run();

    return 0;
}
```

player.cpp

```
#include "player.h"

Player::Player(int health, int damage, int score)
    : health_(health), damage_(damage), score_(score) {
}

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);
    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_;
    const int damage_;
    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 <stdexcept>
#include <algorithm>
#include <memory>
#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::saveToFile(const std::string& filename) const {
    std::ofstream out(filename);
    if (!out.is_open()) {
        throw GameSaveException("Не удалось открыть файл '" + filename +
"' для записи.");
    }
    out << player_.row() << " " << player_.col() << " " <<
player_.health() << " " << player_.score() << "\n";
    if (!out) {
        throw GameSaveException("Ошибка записи данных игрока в файл '" +
filename + "'.");
    }
    out << enemies_.size() << "\n";
```

```
    if (!out) {
        throw GameSaveException("Ошибка записи количества врагов в файл
'" + filename + "'.");
    }
    for (const auto& enemy : enemies_) {
        if (enemy.is_alive()) {
            out << enemy.row() << " " << enemy.col() << " " <<
enemy.health() << " " << enemy.damage() << "\n";
            if (!out) {
                throw GameSaveException("Ошибка записи данных врага в
файл '" + filename + "'.");
            }
        }
    }
    out << hand_.size() << "\n";
    if (!out) {
        throw GameSaveException("Ошибка записи количества заклинаний в
файл '" + filename + "'.");
    }
    for (const auto& spell_ptr : hand_.spells_) {
        if (auto direct_spell =
dynamic_cast<DirectDamageSpell*>(spell_ptr.get())) {
            out << "0 " << direct_spell->getDamage() << " " <<
direct_spell->getRadius() << "\n";
        }
        else if (auto area_spell =
dynamic_cast<AreaDamageSpell*>(spell_ptr.get())) {
            out << "1 " << area_spell->getDamage() << " " << area_spell-
>getRadius() << "\n";
        }
        else {
            throw GameSaveException("Неизвестный тип заклинания при
сохранении.");
        }
        if (!out) {
            throw GameSaveException("Ошибка записи заклинания в файл '" +
filename + "'.");
        }
    }
    out.close();
    if (out.fail()) {
        throw GameSaveException("Ошибка закрытия файла '" + filename +
"'.");
    }
}
void Game::loadFromFile(const std::string& filename) {
    std::ifstream in(filename);
    if (!in.is_open()) {
        throw GameLoadException("Файл '" + filename + "' не найден или не
доступен для чтения.");
    }
    int player_row, player_col, player_health, player_score;
    in >> player_row >> player_col >> player_health >> player_score;
    if (!in) {
        throw GameLoadException("Файл '" + filename + "' поврежден:
невозможно прочитать данные игрока.");
    }
    size_t num_enemies;
```

```
    in >> num_enemies;
    if (!in) {
        throw GameLoadException("Файл '" + filename + "' поврежден:
невозможно прочитать количество врагов.");
    }
    std::vector<Enemy> loaded_enemies;
    for (size_t i = 0; i < num_enemies; ++i) {
        int enemy_row, enemy_col, enemy_health, enemy_damage;
        in >> enemy_row >> enemy_col >> enemy_health >> enemy_damage;
        if (!in) {
            throw GameLoadException("Файл '" + filename + "' поврежден:
невозможно прочитать данные врага " + std::to_string(i) + ".");
        }
        Enemy new_enemy(enemy_health, enemy_damage);
        new_enemy.set_position(enemy_row, enemy_col);
        loaded_enemies.push_back(new_enemy);
    }
    size_t num_spells;
    in >> num_spells;
    if (!in) {
        throw GameLoadException("Файл '" + filename + "' поврежден:
невозможно прочитать количество заклинаний.");
    }
    Hand temp_hand(3);
    temp_hand.clear_spells();
    for (size_t i = 0; i < num_spells; ++i) {
        int spell_type;
        int damage, radius;
        in >> spell_type >> damage >> radius;
        if (!in) {
            throw GameLoadException("Файл '" + filename + "' поврежден:
невозможно прочитать данные заклинания " + std::to_string(i) + ".");
        }
        if (spell_type == 0) {

temp_hand.add_spell(std::make_unique<DirectDamageSpell>(damage, radius));
        }
        else if (spell_type == 1) {
            temp_hand.add_spell(std::make_unique<AreaDamageSpell>(damage,
radius));
        }
        else {
            throw GameLoadException("Файл '" + filename + "' поврежден:
неизвестный тип заклинания " + std::to_string(spell_type) + ".");
        }
    }
    enemies_.clear();
    hand_.clear_spells();
    player_.set_position(player_row, player_col);
    player_.take_damage(player_.health() - player_health);
    player_.add_score(player_score - player_.score());
    enemies_ = std::move(loaded_enemies);
    hand_.spells_ = std::move(temp_hand.spells_);
    field_ = GameField(10, 10);
    field_.place_player(player_);
    for (const auto& enemy : enemies_) {
        if (enemy.is_alive()) {
```

```
            field_.update_cell(enemy.row(), enemy.col(),
Cell::EntityType::kEnemy);
        }
    }
    game_over_ = false;
    level_complete_ = false;
    in.close();
}
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;
        if (!load(load_filename)) {
            std::cout << "Не удалось загрузить игру. Начинаем новую.\n";
            start();
        }
        else {
            bool running = true;
            while (running) {
                bool level_over = false;
                while (!level_over) {
                    field_.print_field(player_, enemies_);
                    player_turn();
                    if (check_defeat_condition()) {
                        std::cout << "Вы погибли!\n";
                        level_over = true;
                        break;
                    }
                    enemies_turn();
                    if (check_defeat_condition()) {
                        std::cout << "Вы погибли!\n";
                        level_over = true;
                        break;
                    }
                    if (check_victory_condition()) {
                        std::cout << "Все враги уничтожены! Уровень
пройден!\n";
                        level_over = true;
                        break;
                    }
                }
                ask_restart_or_exit(running);
                if (!running) break;
            }
            return;
        }
    }
    start();
}
void Game::start() {
    bool running = true;
    while (running) {
        start_level();
```

```
        bool level_over = false;
        while (!level_over) {
            field_.print_field(player_, enemies_);
            player_turn();
            if (check_defeat_condition()) {
                std::cout << "Вы погибли!\n";
                level_over = true;
                break;
            }
            enemies_turn();
            if (check_defeat_condition()) {
                std::cout << "Вы погибли!\n";
                level_over = true;
                break;
            }
            if (check_victory_condition()) {
                std::cout << "Все враги уничтожены! Уровень пройден!\n";
                level_over = true;
                break;
            }
        }
        ask_restart_or_exit(running);
        enemies_.clear();
    }
}
void Game::player_turn() {
    std::cout << "\n=== Ход игрока ===\n";
    std::cout << "1 - ход\n2 - атака\n3 - заклинание\n4 - показать
заклинания\n5 - сохранить игру\n6 - загрузить игру\nВаш выбор: ";
    int choice;
    if (!(std::cin >> choice)) {
        std::cin.clear();
        std::string dum; std::getline(std::cin, dum);
        std::cout << "Некорректный ввод — пропускаем ход.\n";
        return;
    }
    if (choice == 1) {
        char move;
        std::cout << "Ход (W/A/S/D): ";
        std::cin >> move;
        int dr = 0, dc = 0;
        switch (std::tolower(move)) {
        case 'w': dr = -1; dc = 0; break;
        case 's': dr = 1; dc = 0; break;
        case 'a': dr = 0; dc = -1; break;
        case 'd': dr = 0; dc = 1; break;
        default:
            std::cout << "Некорректная клавиша!\n";
            return;
        }
        int new_r = player_.row() + dr;
        int new_c = player_.col() + dc;
        field_.move_player(new_r, new_c, player_);
    }
    else if (choice == 2) {
        std::cout << "Введите координаты врага для атаки (r c): ";
        int r, c;
        if (!(std::cin >> r >> c)) {
```

```
            std::cin.clear();
            std::string dum; std::getline(std::cin, dum);
            std::cout << "Некорректные координаты.\n";
            return;
        }
        for (auto& enemy : enemies_) {
            if (enemy.is_alive() && enemy.row() == r && enemy.col() == c)
{
                enemy.take_damage(player_.damage());
                std::cout << "Вы нанесли " << player_.damage() << " урона
врагу!\n";
                if (!enemy.is_alive()) {
                    field_.clear_cell(r, c);
                }
                break;
            }
        }
        remove_dead_enemies();
    }
    else if (choice == 3) {
        hand_.show_spells();
        std::cout << "Введите номер заклинания (0-based): ";
        size_t index;
        if (!(std::cin >> index)) {
            std::cin.clear();
            std::string dum; std::getline(std::cin, dum);
            std::cout << "Некорректный ввод индекса.\n";
            return;
        }
        std::cout << "Введите координаты цели (r c): ";
        int r, c;
        if (!(std::cin >> r >> c)) {
            std::cin.clear();
            std::string dum; std::getline(std::cin, dum);
            std::cout << "Некорректные координаты.\n";
            return;
        }
        hand_.use_spell(index, *this, field_, player_, r, c);
    }
    else if (choice == 4) {
        hand_.show_spells();
    }
    else if (choice == 5) {
        std::cout << "Введите имя файла для сохранения: ";
        std::string filename;
        std::cin >> filename;
        save(filename);
    }
    else if (choice == 6) {
        std::cout << "Введите имя файла для загрузки: ";
        std::string filename;
        std::cin >> filename;
        std::cout << "Это перезапишет текущую игру. Продолжить? (y/n): ";
        char confirm;
        std::cin >> confirm;
        if (std::tolower(confirm) == 'y') {
            if (load(filename)) {
                return;
```

```
            }
            else {
                std::cout << "Загрузка не удалась. Продолжаем текущую
игру.\n";
            }
        }
        else {
            std::cout << "Загрузка отменена.\n";
        }
    }
    else {
        std::cout << "Неизвестная команда.\n";
    }
}
void Game::start_level() {
    std::cout << "\nЗапуск 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);
    }
}
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();
}
bool Game::check_victory_condition() const {
    return std::all_of(enemies_.begin(), enemies_.end(), [](const Enemy&
e) { return !e.is_alive(); }) || enemies_.empty();
```

```
}
bool Game::check_defeat_condition() const {
    return !player_.is_alive();
}
void Game::ask_restart_or_exit(bool& running) {
    std::cout << "\nНачать заново? (y/n): ";
    char c;
    std::cin >> c;
    if (c == 'y' || c == 'Y') {
        player_.set_position(0, 0);
        player_.take_damage(player_.health() - 100);
        player_.add_score(-player_.score());

        enemies_.clear();
        hand_ = Hand(3);
        field_ = GameField(10, 10);

        start_level();
    }
    else {
        running = false;
    }
}

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);
}
```

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"
class Game {
public:
    Game();
    void run();
    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;
private:
    void start();
    void start_level();
    void player_turn();
    void enemies_turn();
    bool check_victory_condition() const;
    bool check_defeat_condition() const;
    void ask_restart_or_exit(bool& running);
    void saveToFile(const std::string& filename) const;
    void loadFromFile(const std::string& filename);
private:
    GameField field_;
    Player player_;
    Hand hand_;
    std::vector<Enemy> enemies_;
    bool game_over_;
    bool level_complete_;
};
#endif
```

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_
```