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

ОТЧЕТ

по лабораторной работе №1

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 игрока, который должен хранить информацию об игроке (его жизни, урон, очки, и т.д. - студент сам определяет необходимые для работы характеристики). Объект класса игрока должен перемещаться по карте. Если у игрока кончаются жизни, то происходит конец игры.

Создать класс врага, который хранит параметры жизней и урона. Объектами класса врага управляет компьютер. При перемещении, если враг пытается перейти на клетку с игроком, то перемещение не происходит, и игроку наносится урон.

Создать класс квадратного/прямоугольного игрового поля, по которому перемещаются игрок и враги. Игровое поле не должно быть меньше 10 на 10 клеток, и не больше 25 на 25 клеток. Размеры поля задаются через конструктор. Рекомендуется для хранения информации об отдельных клетках поля создать отдельный класс.

Реализовать конструкторы перемещения и копирования для поля, а также соответствующие операторы присваивания с копированием и перемещением (должна происходить глубокая копия).

На 8/4/1.5 баллов:

Реализовать непроходимые клетки на поле. При попытке врагов или игрока перейти на такую клетку, перемещение не происходит. Заполнения поля непроходимыми клетками происходит в момент создания поля.

Добавить возможность для игрока переключаться на ближний или дальний бой с изменением значения наносимого урона. Такое переключение требует один ход.

На 10/5/2 баллов:

Добавить класс вражеского здания. Такое здание размещается на карте, и раз в несколько ходов создает нового врага возле себя. Количество ходов до создания нового врага задается в конструкторе.

Реализовать замедляющие клетки на поле. Если игрок переходит на такую клетку, то он он не может двигаться на следующий ход.

Выполнение работы.

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

- Класс игрока с характеристиками здоровья, урона, очков и возможностью перемещения по игровому полю;
- Класс врага с характеристиками здоровья и урона;
- Класс игрового поля, на котором размещаются игрок и враги;
- Проверка завершения игры при потере здоровья игроком.

Архитектура программы.

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- Player – класс игрока с характеристиками здоровья, урона, очков и возможностью перемещения по полю;
- Enemy – класс врага с характеристиками здоровья и урона, управляемый компьютером;
- GameField – класс игрового поля, на котором размещаются игрок и враги;
- Cell – класс отдельной клетки игрового поля.

Дополнительные enum классы:

- Cell::EntityType – класс типов содержимого клетки (kEmpty, kPlayer, kEnemy).

Описание классов.

Основные классы:

- Класс Player

  Класс содержит характеристики и методы игрового персонажа.

  Поля класса:

  - health – текущее здоровье
  - damage – базовый урон
  - score – очки игрока
  - row_, col_ - координаты игрока на игровом поле

  Методы класса:

  - health() – возвращает текущее здоровье
  - damage() – возвращает значение урона
  - score() – возвращает очки игрока
  - take_damage(int damage) – уменьшает здоровье на указанное количество урона
  - add_score(int points) – увеличивает очки игрока на указанное количество
  - is_alive() – проверяет, жив ли игрок (true, если здоровье > 0)
  - set_position(int row, int col) – устанавливает координаты игрока
  - row() – возвращает текущую строку позиции игрока
  - col() – возвращает текущий столбец позиции игрока

- Класс Enemy

  Класс содержит характеристики и методы врага.

  Поля класса:

  - health – текущее здоровье
  - damage – базовый урон
  - row_, col_ - координаты игрока на игровом поле

  Методы класса:

  - health() – возвращает текущее здоровье
  - damage() – возвращает значение урона

- set_position(int row, int col) – устанавливает координаты врага
- row() – возвращает текущую строку позиции врага
- col() – возвращает текущий столбец позиции врага

- Класс Cell

  Класс содержит характеристики и методы клеток игрового поля.

  Поля класса:

  - type_ – тип содержимого клетки (EntityType)

  Методы класса:

  - type() – возвращает текущий тип клетки
  - set_type(EntityType type) – устанавливает тип клетки

- Класс GameField

  Класс содержит характеристики и методы игрового поля, а также характеристики и методы взаимодействия персонажей с игровым полем.

  Поля класса:

  - rows_ – количество строк игрового поля
  - cols_ – количество столбцов игрового поля
  - grid_ – двумерный вектор клеток (Cell)

  Методы класса:

  - is_valid_position(int row, int col) – проверяет, что координаты находятся внутри поля
  - is_empty(int row, int col) – проверяет, пуста ли клетка
  - place_player(const Player& player) – размещает игрока на поле
  - place_enemy(const Enemy& enemy) – размещает врага на поле
  - clear_cell(int row, int col) – очищает клетку
  - move_player(int new_row, int new_col, Player& player) – перемещает игрока на указанную клетку, если она пуста
  - move_enemy(int new_row, int new_col, Enemy& enemy, Player& player) – перемещает врага; если враг попадает на игрока, наносит ему урон

- is_game_over(const Player& player) – проверяет завершение игры (если здоровье игрока ≤ 0)
- Конструкторы копирования и перемещения, операторы присваивания с копированием и перемещением

Дополнительные enum классы:

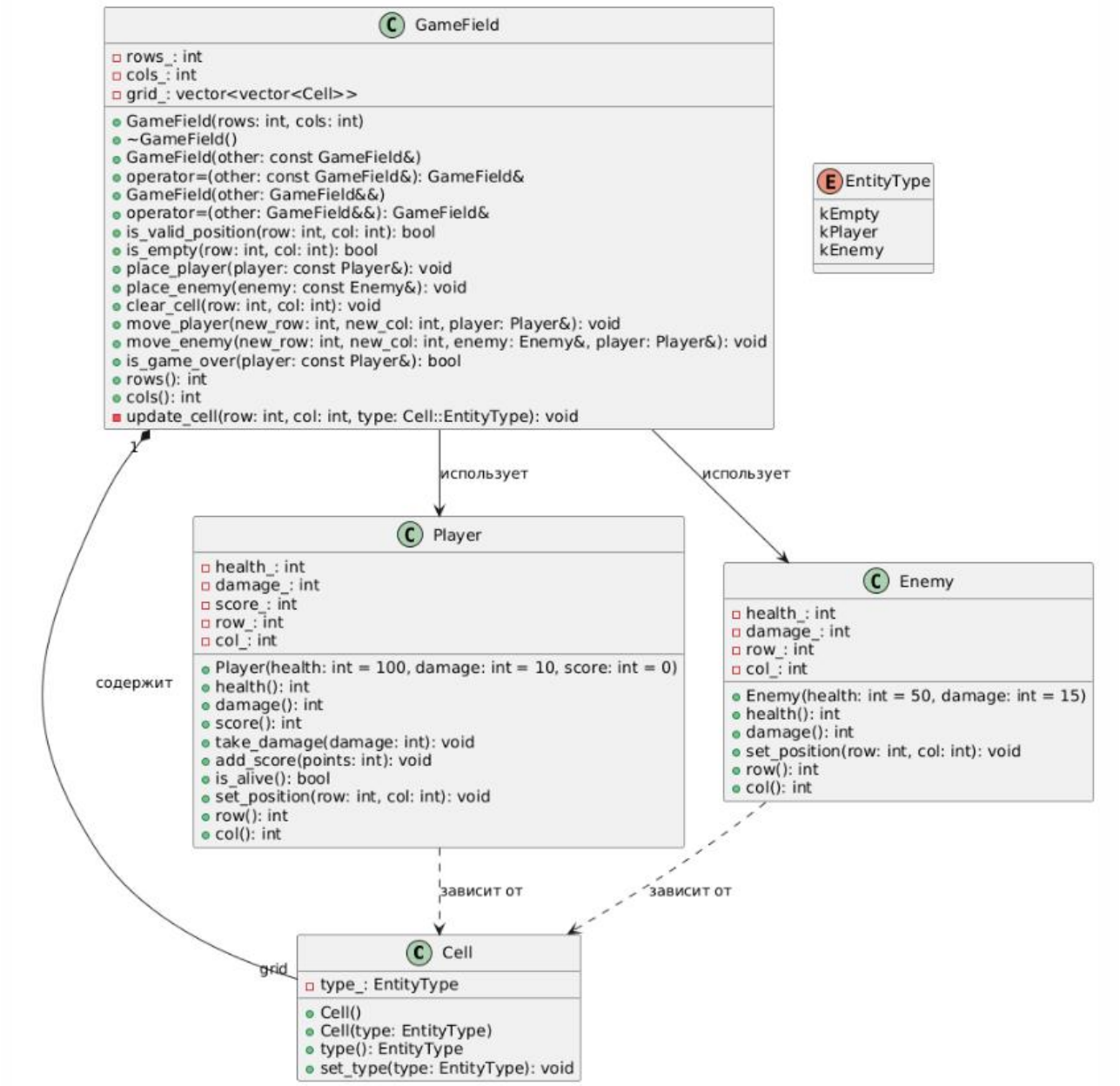
- Класс Cell::EntityType

  Значения:

  - kEmpty – пустая клетка
  - kPlayer – клетка с игроком
  - kEnemy – клетка с врагом

Разработанный программный код см. в приложении А.

UML



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

Название файла: 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_
```

Название файла: cell.cpp

```
#include "cell.h"
```

Название файла: 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 set_position(int row, int col);
    int row() const { return row_; }
    int col() const { return col_; }

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

#endif // GAME_ENEMY_H_
```

Название файла: enemy.cpp

```
#include "enemy.h"

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

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

Название файла: 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;

    // Copy
    GameField(const GameField& other);
    GameField& operator=(const GameField& other);

    // Move
    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 place_enemy(const Enemy& enemy);
    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);

    bool is_game_over(const Player& player) const;

    int rows() const { return rows_; }
    int cols() const { return cols_; }

private:
    void update_cell(int row, int col, Cell::EntityType type);

    int rows_;
    int cols_;
    std::vector<std::vector<Cell>> grid_;
```

```
};

#endif // GAME_GAME_FIELD_H_
```

Название файла: game_field.cpp

```
#include "game_field.h"

GameField::GameField(int rows, int cols)
    : rows_(rows), cols_(cols), grid_(rows, std::vector<Cell>(cols))
{
    if (rows < 10 || rows > 25 || cols < 10 || cols > 25) {
        throw std::invalid_argument("Field size must be between
10x10 and 25x25.");
    }
}

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

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

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

```
    // Move assignment
    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::place_enemy(const Enemy& enemy) {
        update_cell(enemy.row(), enemy.col(), Cell::EntityType::kEnemy);
    }

    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);
    place_enemy(enemy);
}

bool GameField::is_game_over(const Player& player) const {
    return !player.is_alive();
}
```

Название файла: player.h

```
#ifndef GAME_PLAYER_H_
#define GAME_PLAYER_H_

class Player {
public:
```

```
        explicit Player(int health = 100, int damage = 10, int score =
0);

        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 // GAME_PLAYER_H_
```

Название файла: 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;
}
```

Название файла: main.cpp

```
#include <iostream>
#include "cell.h"
#include "player.h"
#include "enemy.h"
#include "game_field.h"
#define NOMINMAX
#include <Windows.h>

void test_player() {
    std::cout << "=== Тестирование класса Player (Игрок) ===\n";
    Player player(100, 10, 0);
    player.set_position(2, 3);

    std::cout << "Начальные данные игрока:\n";
    std::cout << "  Здоровье: " << player.health()
        << ", Урон: " << player.damage()
        << ", Очки: " << player.score()
        << ", Позиция: (" << player.row() << ", " << player.col()
<< ")\n";
```

```
        std::cout << "Игрок получает 30 урона...\n";
        player.take_damage(30);
        std::cout << "  Текущее здоровье: " << player.health() << "\n";

        std::cout << "Игрок получает очки (+15)...\n";
        player.add_score(15);
        std::cout << "  Очки игрока: " << player.score() << "\n";

        std::cout << "Статус: " << (player.is_alive() ? "жив" : "мертв")
<< "\n\n";
    }

    void test_enemy() {
        std::cout << "=== Тестирование класса Enemy (Враг) ===\n";
        Enemy enemy(50, 15);
        enemy.set_position(1, 1);

        std::cout << "Параметры врага:\n";
        std::cout << "  Здоровье: " << enemy.health()
            << ", Урон: " << enemy.damage()
            << ", Позиция: (" << enemy.row() << ", " << enemy.col() <<
")\n\n";
    }

    void test_cell() {
        std::cout << "=== Тестирование класса Cell (Клетка) ===\n";
        Cell cell;
        std::cout << "  Состояние по умолчанию: kEmpty (пусто)\n";

        cell.set_type(Cell::EntityType::kPlayer);
        std::cout << "  После установки: kPlayer (игрок)\n";

        cell.set_type(Cell::EntityType::kEnemy);
        std::cout << "  После установки: kEnemy (враг)\n\n";
    }

    void test_game_field() {
```

```
        std::cout << "=== Тестирование класса GameField (Игровое поле)
===\n";
        GameField field(10, 10);
        Player player(100, 10, 0);
        Enemy enemy(50, 15);

        player.set_position(0, 0);
        enemy.set_position(0, 1);

        field.place_player(player);
        field.place_enemy(enemy);

        std::cout << "Игрок и враг размещены на поле.\n";
        std::cout << "Игрок: (" << player.row() << ", " << player.col()
<< ")\n";
        std::cout << "Враг:  (" << enemy.row() << ", " << enemy.col()
<< ")\n";

        std::cout << "Враг атакует игрока (переходит на его
клетку)...\n";
        field.move_enemy(0, 0, enemy, player);

        std::cout << "Здоровье игрока после атаки: " << player.health()
<< "\n";

        if (field.is_game_over(player))
            std::cout << "Игра окончена — игрок погиб.\n";
        else
            std::cout << "Игрок выжил!\n\n";

        std::cout << "Игрок перемещается в (1, 0)...\n";
        field.move_player(1, 0, player);
        std::cout << "Новая позиция игрока: (" << player.row() << ", "
<< player.col() << ")\n\n";
    }

    int main() {
        SetConsoleOutputCP(1251);
        SetConsoleCP(1251);
```

```
    setlocale(LC_ALL, "");
    try {
        std::cout << "=== Тестирование программы ===\n\n";
        test_player();
        test_enemy();
        test_cell();
        test_game_field();
        std::cout << "=== Все тесты успешно завершены ===\n";
    }
    catch (const std::exception& e) {
        std::cerr << "Ошибка: " << e.what() << "\n";
        return 1;
    }
    return 0;
}
```