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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объектно-ориентированное программирование»

Студентка гр. 4384 _____ Калинина А.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

Цель работы.

Изучить принципы объектно-ориентированного программирования. Написать программу на языке C++, которая будет прототипом пошаговой игры с перемещением персонажа и сражением с врагами.

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).

Выполнение работы.

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

- Класс игрока;
- Класс врага;
- Класс игрового поля;

Архитектура программы.

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- Entity - базовый класс для всех игровых сущностей
- Player - класс игрока, наследуемый от Entity
- Enemy - класс врага, наследуемый от Entity
- Cell - класс клетки игрового поля
- GameField - класс игрового поля
  - класс игровой логики
- GameManager - главный класс управления игрой

Описание классов.

- Класс Entity

Базовый класс, содержащий общие характеристики и методы всех игровых сущностей. Создан для избегания дублирования кода в классах Player и Enemy.

Поля класса:

- ❖ health - текущее здоровье
- ❖ damage - наносимый урон
- ❖ x, y - координаты на поле

Методы класса:

- ➢ getHealth() - получение текущего здоровья
- ➢ getDamage() - получение значения урона
- ➢ getX(), getY() - получение координат
- ➢ takeDamage() - получение урона
- ➢ move() - перемещение сущности
- ➢ isAlive() - проверка жива ли сущность

- Класс Player

Класс содержит характеристики и методы игрока.

Поля класса:

- ❖ score - счет игрока

Методы класса:

- ➢ getScore() - получение счета
- ➢ addScore() - добавление очков к счету

- Класс Enemy

Класс содержит характеристики и методы вражеских персонажей. Наследует всю функциональность от класса Entity.

- Класс Cell

Класс представляет одну ячейку игрового поля.

Поля класса:

- ❖ content - символ содержимого ячейки

Методы класса:

- ➢ getContent() - получение содержимого
- ➢ setContent() - установка содержимого
- ➢ isEmpty() - проверка пуста ли ячейка
- ➢ clear() - очистка ячейки

- Класс GameField

Класс представляет игровое поле и управляет всеми клетками.

Поля класса:

- ❖ width, height - размеры поля
- ❖ grid - двумерный вектор клеток

Методы класса:

- ➢ getWidth(), getHeight() - получение размеров поля
- ➢ getCell() - получение клетки по координатам
- ➢ setCell() - установка содержимого клетки

- очистка клеток
  - isValidPosition() - проверка валидности позиции

- Класс GameLogic

Класс содержит всю игровую логику и правила игры.

Поля класса:

- field - ссылка на игровое поле
- player - ссылка на игрока
- enemies - ссылка на вектор врагов

Методы класса:

- processPlayerMove() - обработка хода игрока
- processEnemyMoves() - обработка ходов врагов
  y() - проверка победы игрока
- handleEnemyAttack() - обработка атаки врага

- Класс GameManager

Главный класс, управляющий всем игровым процессом.

Поля класса:

- field - игровое поле
- player - игрок
- enemies - вектор врагов
- gameLogic - объект игровой логики
  nning - флаг работы игры

Методы класса:

- run() - главный игровой цикл
- initializeEnemies() - инициализация врагов
- updateField() - обновление отображения поля
- processInput() - обработка пользовательского ввода
- displayGameInfo(), displayField() - отображение информации

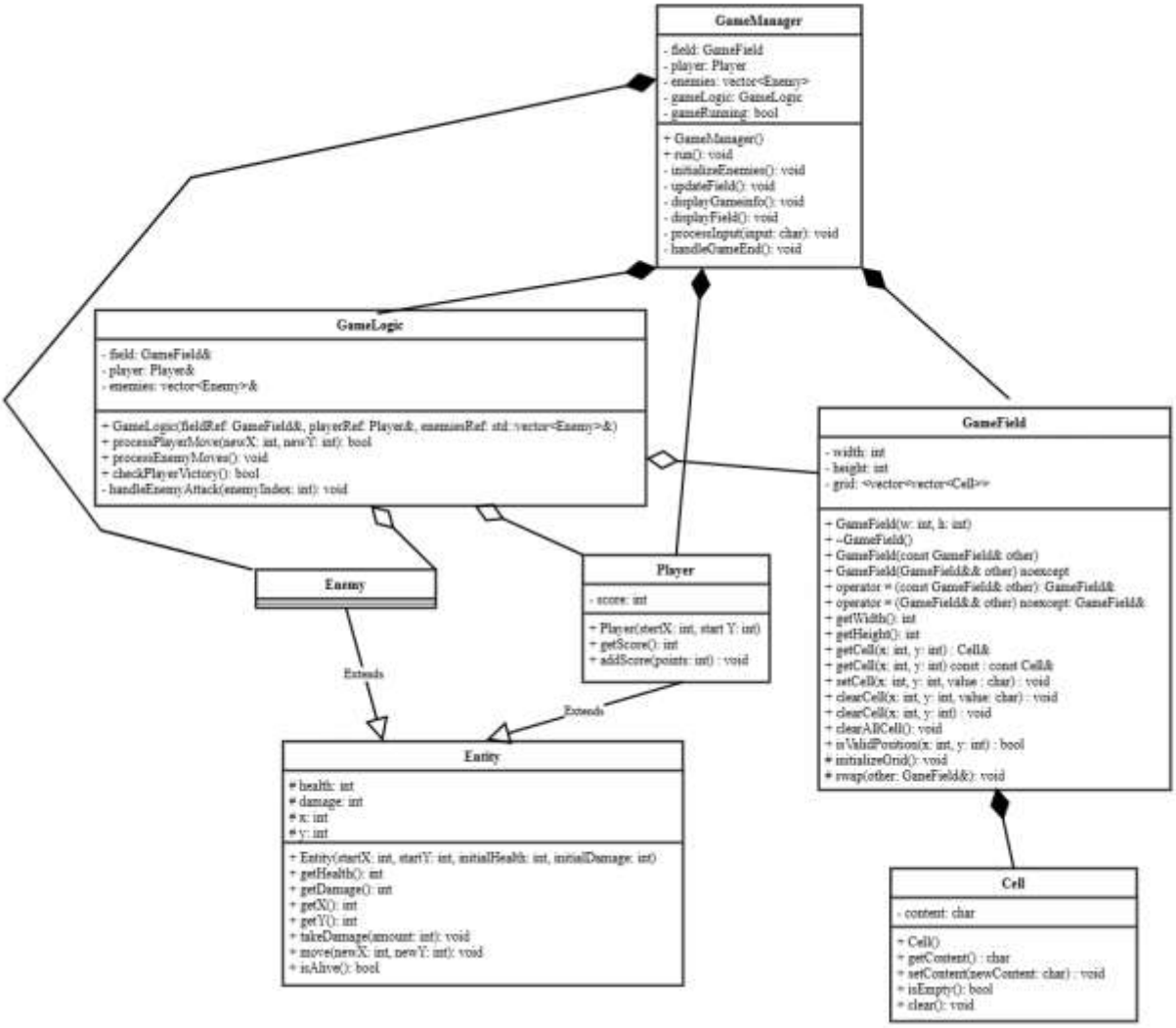


Диаграмма UML-классов

Разработанный программный код см. в приложении А.

Выводы.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Cell.h

```
#ifndef CELL_H
#define CELL_H

class Cell {
private:
    char content;

public:
    Cell();

    char getContent() const;
    void setContent(char newContent);
    bool isEmpty() const;
    void clear();
};

#endif
```

Cell.cpp

```
#include "Cell.h"

Cell::Cell() : content('.') {
}

char Cell::getContent() const {
    return content;
}

void Cell::setContent(char newContent) {
    content = newContent;
}

bool Cell::isEmpty() const {
    return content == '.';
}

void Cell::clear() {
    content = '.';
}
```

Enemy.cpp

```
#include "Enemy.h"

Enemy::Enemy(int startX, int startY)
    : Entity(startX, startY, 30, 5) {
}
```

Enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H

#include "Entity.h"

class Enemy : public Entity {
public:
    Enemy(int startX, int startY);
};

#endif
```

Entity.cpp

```
#include "Entity.h"

Entity::Entity(int startX, int startY, int initialHealth, int
initialDamage)
    : health(initialHealth), damage(initialDamage), x(startX),
y(startY) {
}

int Entity::getHealth() const {
    return health;
}

int Entity::getDamage() const {
    return damage;
}

int Entity::getX() const {
    return x;
}

int Entity::getY() const {
    return y;
}

void Entity::takeDamage(int amount) {
    health -= amount;
    if (health < 0) {
        health = 0;
    }
}

void Entity::move(int newX, int newY) {
    x = newX;
    y = newY;
```

```
}

bool Entity::isAlive() const {
    return health > 0;
}
```

Entity.h

```
#ifndef ENTITY_H
#define ENTITY_H

class Entity {
protected:
    int health;
    int damage;
    int x;
    int y;

public:
    Entity(int startX, int startY, int initialHealth, int
initialDamage);

    // Общие методы для всех сущностей
    int getHealth() const;
    int getDamage() const;
    int getX() const;
    int getY() const;

    void takeDamage(int amount);
    void move(int newX, int newY);
    bool isAlive() const;
};

#endif
```

GameField.cpp

```
#include "GameField.h"
#include <stdexcept>
#include <utility>

// Метод swap для обмена состояниями
void GameField::swap(GameField& other) noexcept {
    std::swap(width, other.width);
    std::swap(height, other.height);
    std::swap(grid, other.grid);
}

void GameField::initializeGrid() {
    grid.resize(height);
    for (int i = 0; i < height; i++) {
        grid[i].resize(width);
    }
}

GameField::GameField(int w, int h) : width(w), height(h) {
    initializeGrid();
}
```

```
// Конструктор копирования через swap
GameField::GameField(const GameField& other)
    : width(0), height(0) {
    // Создаем временный объект-копию
    GameField temp(other.width, other.height);

    // Копируем содержимое всех ячеек
    for (int i = 0; i < other.height; i++) {
        for (int j = 0; j < other.width; j++) {
            temp.grid[i][j] = other.grid[i][j];
        }
    }

    // Обмениваем состояния с временным объектом
    swap(temp);
}

// Конструктор перемещения через swap
GameField::GameField(GameField&& other) noexcept
    : width(0), height(0) {
    swap(other);
}

// Оператор присваивания копированием через swap (copy-and-swap)
GameField& GameField::operator=(const GameField& other) {
    GameField temp(other);  // Создаем копию через конструктор
копирования
    swap(temp);             // Обмениваем состояния
    return *this;           // temp уничтожается со старыми данными
}

// Оператор присваивания перемещением через swap
GameField& GameField::operator=(GameField&& other) noexcept {
    swap(other);
    return *this;
}

int GameField::getWidth() const {
    return width;
}

int GameField::getHeight() const {
    return height;
}

Cell& GameField::getCell(int x, int y) {
    if (!isValidPosition(x, y)) {
        throw std::out_of_range("Position is out of field bounds");
    }
    return grid[y][x];
}

const Cell& GameField::getCell(int x, int y) const {
    if (!isValidPosition(x, y)) {
        throw std::out_of_range("Position is out of field bounds");
    }
    return grid[y][x];
}
```

```
void GameField::setCell(int x, int y, char value) {
    if (isValidPosition(x, y)) {
        grid[y][x].setContent(value);
    }
}

void GameField::clearCell(int x, int y) {
    if (isValidPosition(x, y)) {
        grid[y][x].clear();
    }
}

void GameField::clearAllCells() {
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            clearCell(x, y);
        }
    }
}

bool GameField::isValidPosition(int x, int y) const {
    return x >= 0 && x < width && y >= 0 && y < height;
}
```

GameField.h

```
#ifndef GAMEFIELD_H
#define GAMEFIELD_H

#include "Cell.h"
#include <vector>

class GameField {
private:
    int width;
    int height;
    std::vector<std::vector<Cell>> grid;

    void initializeGrid();
    void swap(GameField& other) noexcept;

public:
    GameField(int w, int h);
    ~GameField() = default;

    // Правило пяти с использованием swap
    GameField(const GameField& other);
    GameField(GameField&& other) noexcept;
    GameField& operator=(const GameField& other);
    GameField& operator=(GameField&& other) noexcept;

    int getWidth() const;
    int getHeight() const;
    Cell& getCell(int x, int y);
    const Cell& getCell(int x, int y) const;
    void setCell(int x, int y, char value);
    void clearCell(int x, int y);
```

```
    void clearAllCells();
    bool isValidPosition(int x, int y) const;
};

#endif
```

GameManager.cpp

```
#include "GameManager.h"
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

GameManager::GameManager()
    : field(15, 15), player(0, 0), gameLogic(field, player, enemies),
gameRunning(true) {
    srand(time(nullptr));
    initializeEnemies();
    updateField();
}

void GameManager::initializeEnemies() {  // Инициализация врагов
    for (int enemyCount = 0; enemyCount < 2; enemyCount++) {
        int x, y;
        bool validPosition = false;

        while (!validPosition) {
            x = rand() % field.getWidth();
            y = rand() % field.getHeight();
            validPosition = true;

            // Проверяем, что позиция не совпадает с игроком
            if (x == 0 && y == 0) {
                validPosition = false;
                continue;
            }

            // Клетка занята
            if (!field.getCell(x, y).isEmpty()) {
                validPosition = false;
                continue;
            }

            // Проверяем, что позиция не совпадает с уже созданными
врагами
            for (int i = 0; i < enemies.size(); i++) {
                if (enemies[i].getX() == x && enemies[i].getY() == y)
{
                    validPosition = false;
                    break;
                }
            }
        }
```

```
        enemies.push_back(Enemy(x, y));
    }
}

void GameManager::run() {
    cout << "Game Started! Use WASD to move, Q to quit" << endl;

    while (gameRunning && player.isAlive()) {
        displayGameInfo();
        displayField();

        if (gameLogic.checkPlayerVictory()) {
            cout << "Player victory! All enemies are destroyed!" <<
endl;
            cout << "Final score: " << player.getScore() << endl;
            gameRunning = false;
            break;
        }

        cout << "Move (WASD): ";
        char input;
        cin >> input;

        if (input == 'q' || input == 'Q') {
            break;
        }

        processInput(input);
        gameLogic.processEnemyMoves();
        updateField();

        if (!player.isAlive()) {
            cout << "GAME OVER! Your score: " << player.getScore() <<
endl;
            gameRunning = false;
            break;
        }
    }

    handleGameEnd();
}

void GameManager::processInput(char input) {
    int newX = player.getX();
    int newY = player.getY();

    switch (input) {
    case 'w': case 'W': newY--; break;
    case 's': case 'S': newY++; break;
    case 'a': case 'A': newX--; break;
    case 'd': case 'D': newX++; break;
    default:
        cout << "Invalid input!" << endl;
        return;
    }

    gameLogic.processPlayerMove(newX, newY);
}
```

```
void GameManager::updateField() {
    // Очищаем поле
    for (int y = 0; y < field.getHeight(); y++) {
        for (int x = 0; x < field.getWidth(); x++) {
            field.clearCell(x, y);
        }
    }

    // Ставим врагов
    for (int i = 0; i < enemies.size(); i++) {
        if (enemies[i].isAlive()) {
            field.setCell(enemies[i].getX(), enemies[i].getY(), 'E');
        }
    }

    // Ставим игрока
    field.setCell(player.getX(), player.getY(), 'P');
}

void GameManager::displayGameInfo() const {
    cout << "Health: " << player.getHealth();
    cout << " | Score: " << player.getScore();
    cout << " | Position: (" << player.getX() << "," << player.getY()
<< ")" << endl;
}

void GameManager::displayField() const {
    for (int y = 0; y < field.getHeight(); y++) {
        for (int x = 0; x < field.getWidth(); x++) {
            cout << field.getCell(x, y).getContent() << " ";
        }
        cout << endl;
    }
}

void GameManager::handleGameEnd() const {
    if (player.isAlive() && !gameLogic.checkPlayerVictory()) {
        cout << "Game finished. Final score: " << player.getScore() <<
endl;
    }
}
```

GameManager.h

```
#ifndef GAMEMANAGER_H
#define GAMEMANAGER_H

#include "GameField.h"
#include "Player.h"
#include "Enemy.h"
#include "GameLogic.h"
#include <vector>

class GameManager {
private:
    GameField field;
    Player player;
    std::vector<Enemy> enemies;
```

```
    GameLogic gameLogic;
    bool gameRunning;

    void initializeEnemies();
    void updateField();
    void displayGameInfo() const;
    void displayField() const;
    void processInput(char input);
    void handleGameEnd() const;

public:
    GameManager();
    void run();
};

#endif
```

Main.cpp

```
#include "GameManager.h"

int main() {
    GameManager game;
    game.run();
    return 0;
}
```

Player.cpp

```
#include "Player.h"

Player::Player(int startX, int startY)
    : Entity(startX, startY, 100, 10), score(0) {
}

int Player::getScore() const {
    return score;
}

void Player::addScore(int points) {
    score += points;
}
```

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include "Entity.h"

class Player : public Entity {
private:
    int score;

public:
    Player(int startX, int startY);
```

```
    int getScore() const;
    void addScore(int points);
};

#endif
```

GameLogic.cpp

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

using namespace std;

GameLogic::GameLogic(GameField& fieldRef, Player& playerRef,
std::vector<Enemy>& enemiesRef)
    : field(fieldRef), player(playerRef), enemies(enemiesRef) {
}

bool GameLogic::processPlayerMove(int newX, int newY) {
    if (!field.isValidPosition(newX, newY)) {
        cout << "Cannot move there - out of bounds!" << endl;
        return false;
    }

    // Проверяем, есть ли враг на целевой клетке
    for (int i = 0; i < enemies.size(); i++) {
        if (enemies[i].isAlive() && enemies[i].getX() == newX &&
enemies[i].getY() == newY) {
            // Атакуем врага
            enemies[i].takeDamage(player.getDamage());
            cout << "You attacked enemy! Enemy health: " <<
enemies[i].getHealth() << endl;

            if (!enemies[i].isAlive()) {
                player.addScore(10);
                cout << "Enemy defeated! +10 points" << endl;
            }
            return false; // Игрок не перемещается
        }
    }

    // Свободная клетка - перемещаемся
    player.move(newX, newY);
    player.addScore(1);
    return true;
}

void GameLogic::processEnemyMoves() {
    for (int i = 0; i < enemies.size(); i++) {
        if (!enemies[i].isAlive()) continue;

        int direction = rand() % 4;
        int oldX = enemies[i].getX();
        int oldY = enemies[i].getY();
        int newX = oldX;
        int newY = oldY;
```

```
        switch (direction) {
        case 0: newX++; break; // вправо
        case 1: newX--; break; // влево
        case 2: newY++; break; // вниз
        case 3: newY--; break; // вверх
        }

        if (field.isValidPosition(newX, newY)) {
            bool canMove = true;

            // Если враг пытается перейти на клетку с игроком
            if (newX == player.getX() && newY == player.getY()) {
                canMove = false;
                handleEnemyAttack(i);
            }

            // Проверяем, есть ли другие враги на целевой клетке
            for (int j = 0; j < enemies.size(); j++) {
                if (i != j && enemies[j].isAlive() &&
enemies[j].getX() == newX && enemies[j].getY() == newY) {
                    canMove = false;
                    break;
                }
            }

            if (canMove) {
                enemies[i].move(newX, newY);
            }
        }
    }
}

bool GameLogic::checkPlayerVictory() const {
    for (int i = 0; i < enemies.size(); i++) {
        if (enemies[i].isAlive()) {
            return false;
        }
    }
    return true;
}

void GameLogic::handleEnemyAttack(int enemyIndex) {
    player.takeDamage(enemies[enemyIndex].getDamage());
    cout << "Enemy " << enemyIndex + 1 << " attacked you! Lost "
        << enemies[enemyIndex].getDamage() << " health!" << endl;
}
```

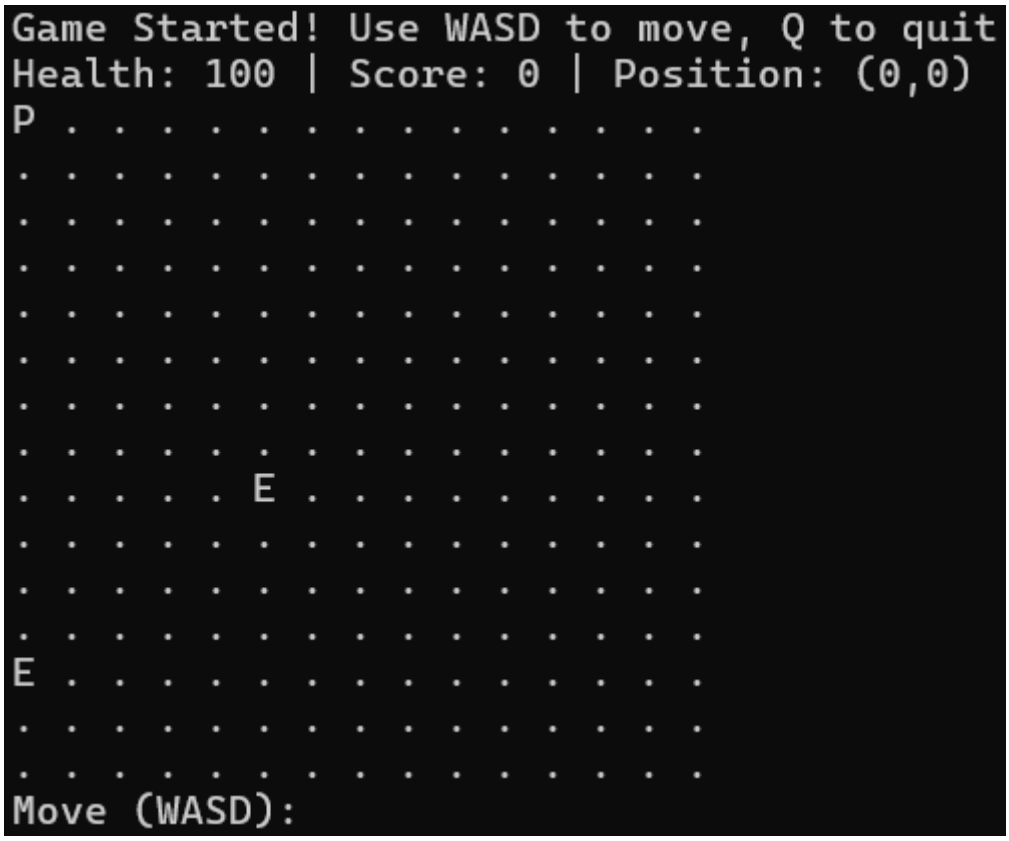
GameLogic.h

```
#ifndef GAMELOGIC_H
#define GAMELOGIC_H

#include "GameField.h"
#include "Player.h"
#include "Enemy.h"
#include <vector>

class GameLogic {
```

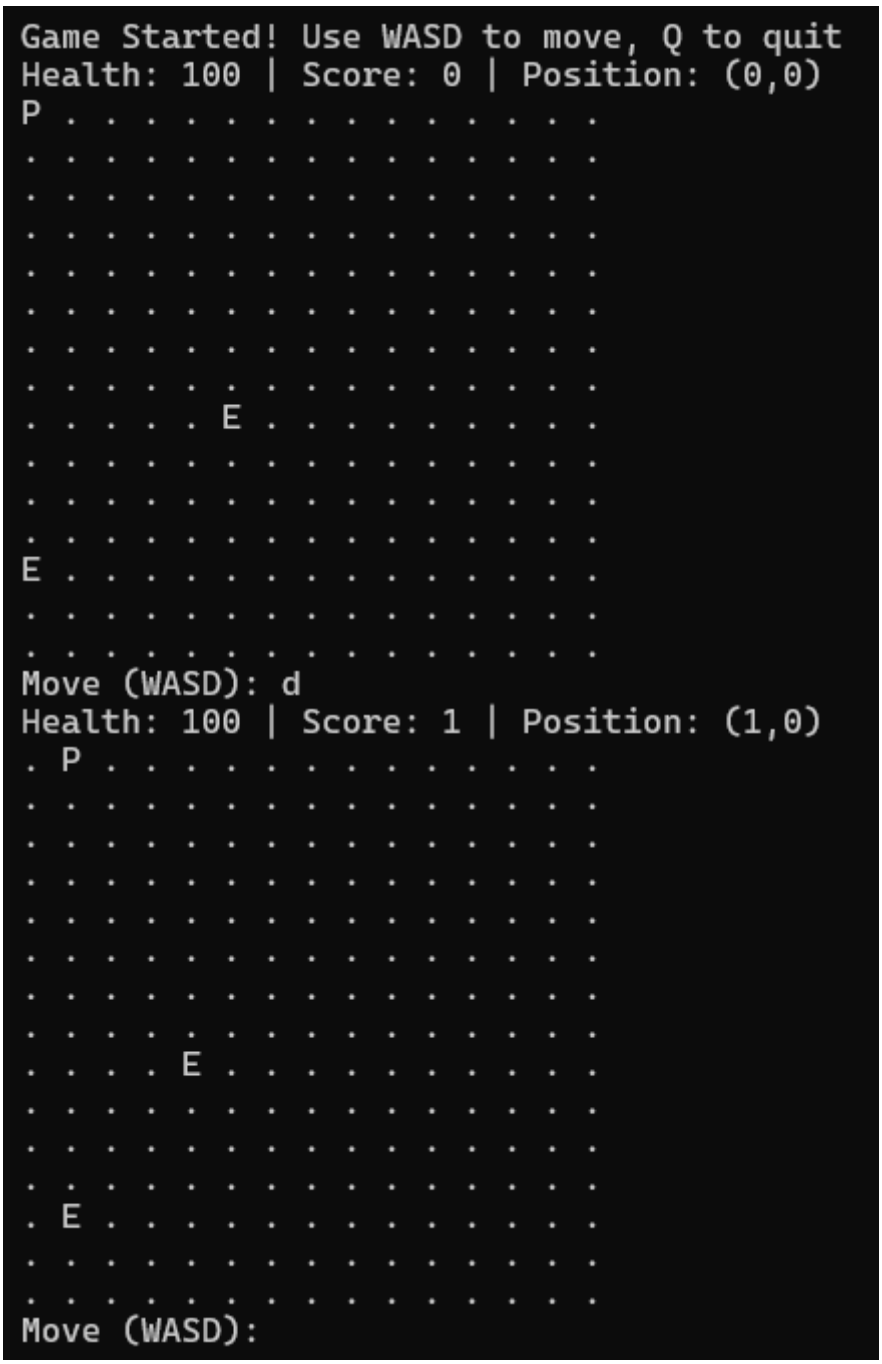
```
private:
    GameField& field;
    Player& player;
    std::vector<Enemy>& enemies;

public:
    GameLogic(GameField& fieldRef, Player& playerRef,
std::vector<Enemy>& enemiesRef);

    bool processPlayerMove(int newX, int newY);
    void processEnemyMoves();
    bool checkPlayerVictory() const;
    void handleEnemyAttack(int enemyIndex);
};

#endif
```

ПРИЛОЖЕНИЕ Б

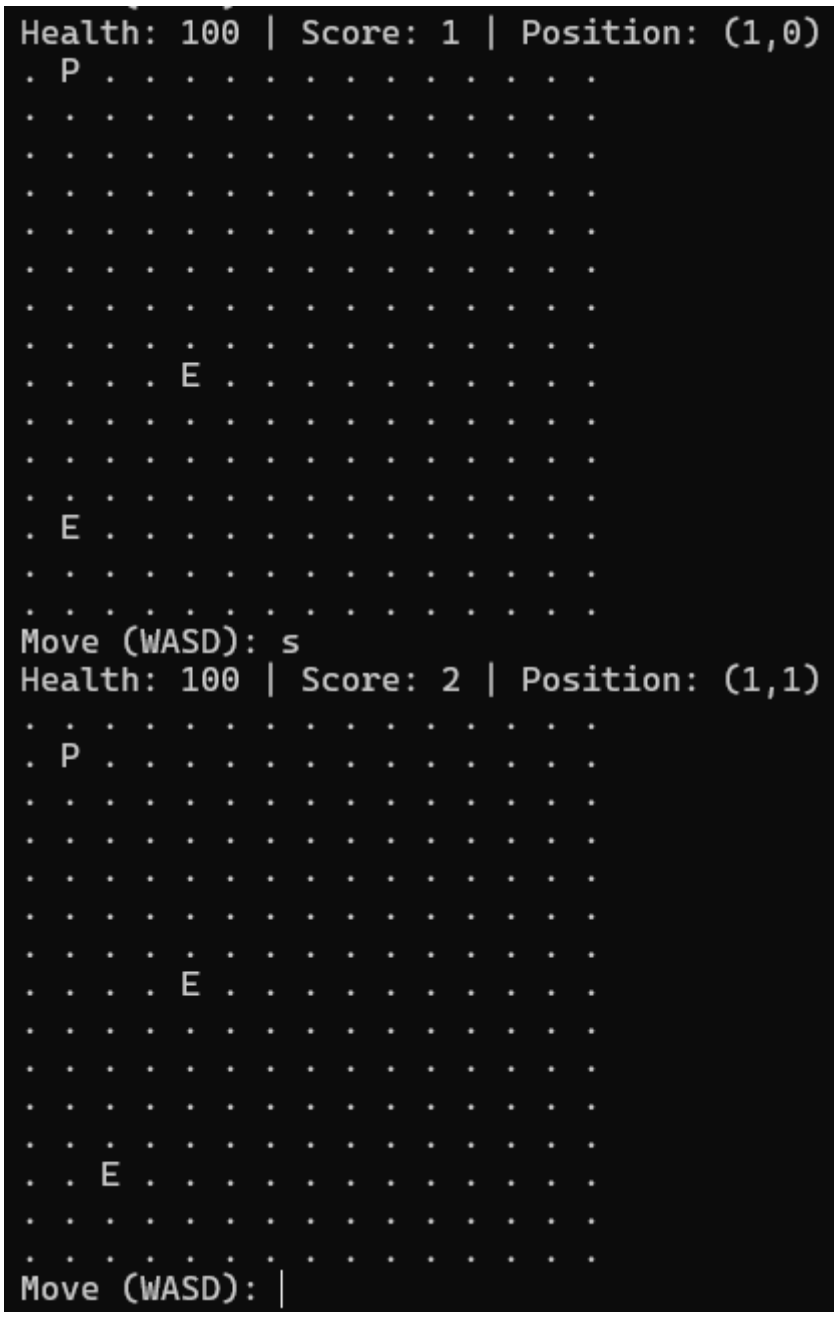
ТЕСТИРОВАНИЕ



1. Запуск игры

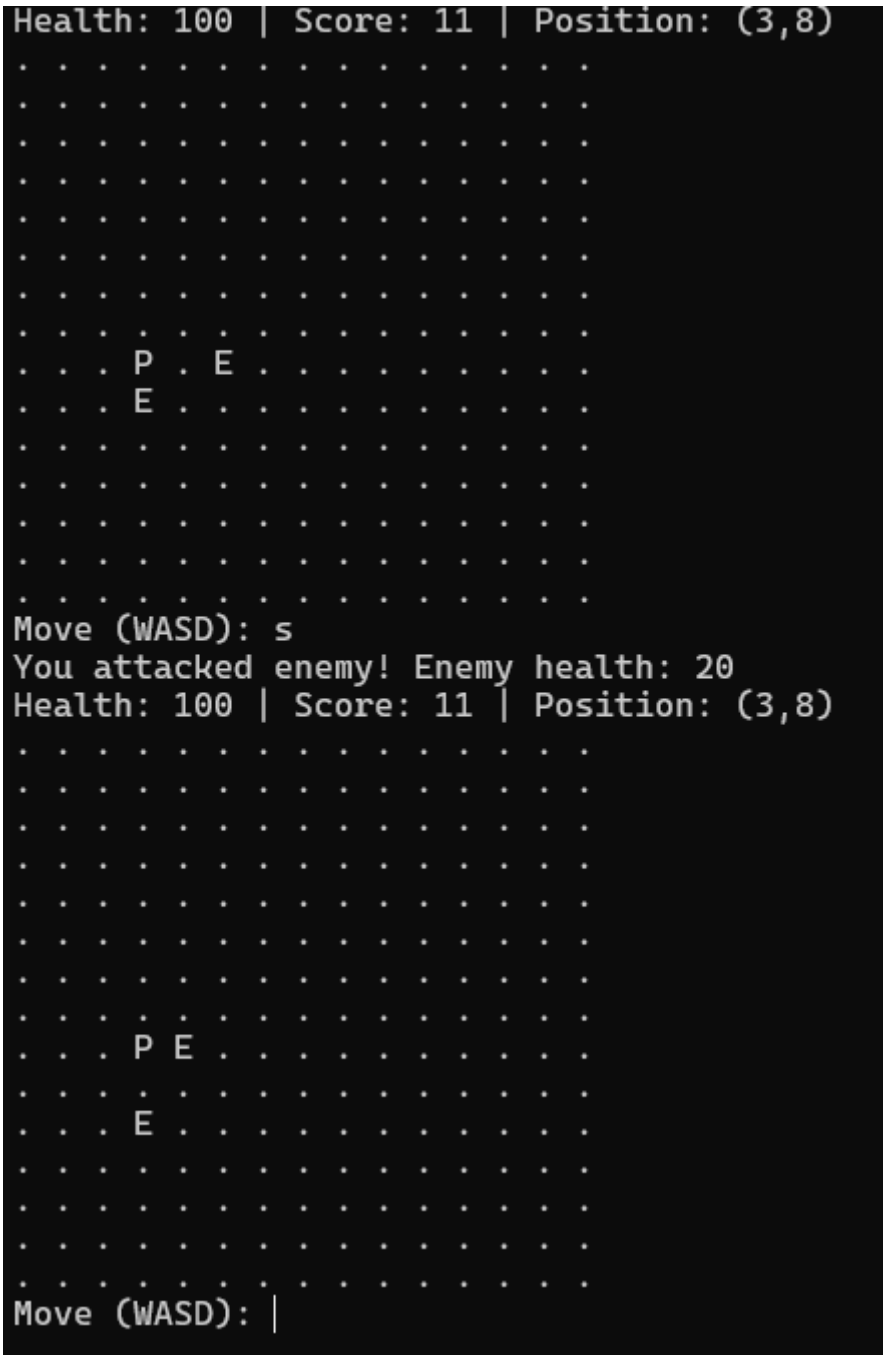


2. Перемещение вправо



3. Перемещение вниз

4. Враг наносит нам урон



5. Мы наносим урон врагу