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

# ОТЧЕТ

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

Студентка гр. 4384 _____ Мулюкина Е.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

**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:

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.

**Выполнение работы.**

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

- Система уровней с увеличением сложности (количество врагов растёт)
- Ход врагов: каждый ход игрока — все живые враги делают шаг к игроку
- Полная сериализация игры (сохранение/загрузка): игрок, заклинания, враги, поле, прогресс
- Красивый текстовый интерфейс с подсказками и визуализацией целей

Также были изменены некоторые предыдущие классы для взаимодействия с новыми классами.

**Архитектура программы.**

В программе реализована иерархия классов, соответствующая принципам ООП без «божественных» классов.

Внесены изменения в класс с общей логикой игры – GameController, отвечающие за главное меню, запуск и переход на новые уровни и ход врагов.

Реализованные этапы игрового цикла:

- Начало игры — метод showMainMenu() предлагает выбор: новая игра / загрузка / выход
- Запуск уровня — при старте новой игры или победе вызывается resetGame() и initializeGame(), отображается сообщение «УРОВЕНЬ X»
- Ход игрока — реализован через getPlayerInput() и processPlayerMove(), поддерживает:
    - Перемещение (w/a/s/d)
    - Применение заклинаний (c → processSpellCast())

- Ход врагов — реализован метод moveEnemies(), вызывается каждый ход игрока. Все живые враги делают один шаг в сторону игрока, избегая столкновений друг с другом и игроком

Условие прохождения уровня: уничтожение всех врагов на текущем уровне. При победе: автоматический переход на следующий уровень с увеличением сложности (врагов становится больше), начисление бонусных очков. При поражении: игроку предлагается начать заново (r), загрузить сохранение (l) или выйти (q)

Всё взаимодействие с игроком, врагами, полем и заклинаниями происходит исключительно через класс GameController.

**Система сохранения и загрузки игры**

Реализована полная бинарная сериализация состояния игры через методы saveGame() / loadGame() и внутренние saveImpl() / loadImpl().

Все классы реализуют save() / load().

Игра может быть сохранена в любой момент и корректно восстановлена

Сохраняются и восстанавливаются:

- Игрок (здоровье, позиция, очки, рука с заклинаниями)
- Все враги (позиция, здоровье)
- Игровое поле
- Текущий уровень, количество убитых врагов, флаг активности игры
- Все заклинания в руке с сохранением типа (getType()) и состояния

Возможности сохранения:

- Автосохранение при выходе из игры
- Сохранение по команде (s)
- Загрузка при старте игры или в любой момент по команде (l)
- Сохранение остаётся консистентным между запусками

**Обработка исключительных ситуаций при сохранении/загрузке**

Реализована полноценная обработка ошибок через собственные классы исключений:

Все критические операции (открытие файла, чтение, запись) обернуты в try-catch. Программа никогда  аварийно не завершается — всегда выводит понятное сообщение

Обрабатываемые ситуации:

- SaveLoadException – любая ошибка сохранения/загрузки.
- FileOpenException - не удалось открыть файл (нет прав, не существует, диск полный и т.д.)
- FileWriteException - ошибка записи в файл
- FileReadException - ошибка чтения из файла
- CorruptedDataException - файл повреждён
- VersionMismatchException - версия файла не совпадает с текущей программой
- TooManyEnemiesException – слишком много врагов указано при запуске игры (более 5)

При любой ошибке пользователю выводится понятное сообщение, игра не падает, предлагается продолжить без сохранения или выйти.

**Класс BinaryFile:**

Автоматическое закрытие файла при выходе из области видимости (в том числе при исключениях),

Немедленное обнаружение ошибок открытия файла,

Отсутствие утечек ресурсов и повреждённых сохранений.

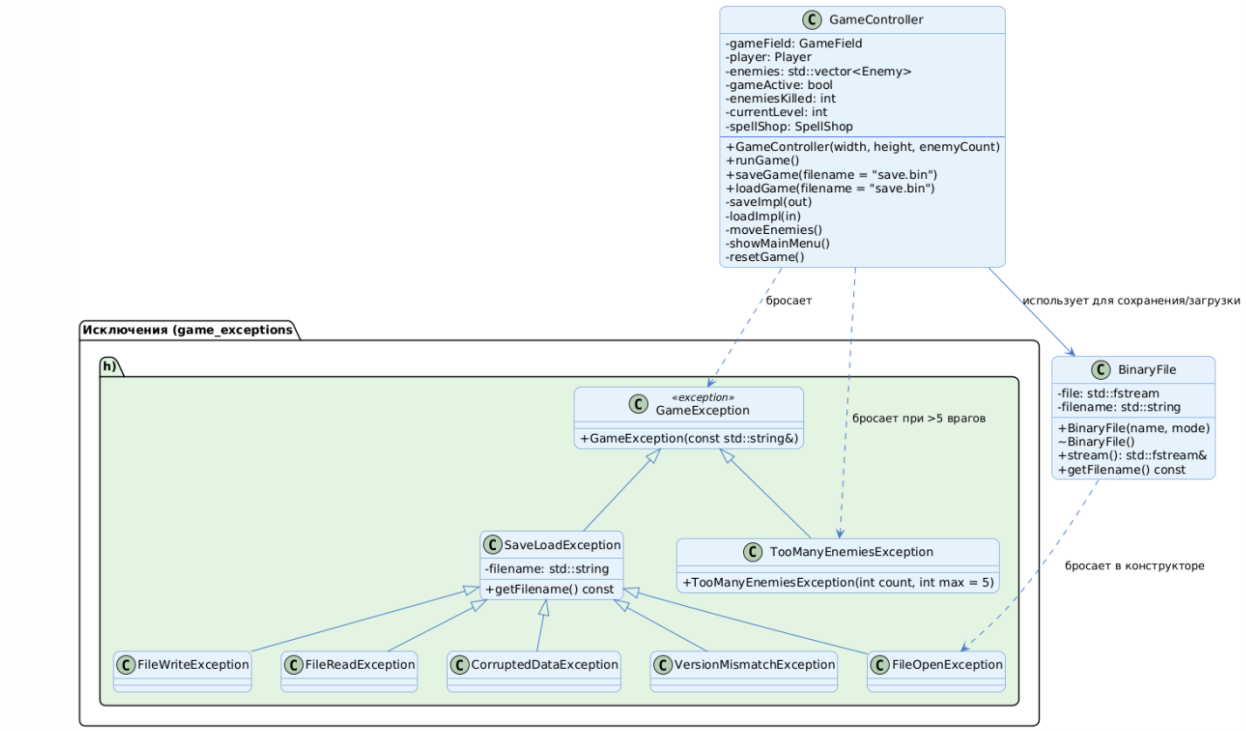
Поля:

- File - Внутренний поток для чтения/записи данных
- Filename - Имя файла, с которым работает объект

Методы:

- ~BinaryFile() - Деструктор. Автоматически закрывает файл при выходе из области видимости
- stream() - Возвращает ссылку на внутренний поток для записи/чтения
- getFilename() - Возвращает имя файла

**UML-диаграммы классов.**




GameController -> BinaryFile – ассоциация

GameController -> GameExceptions – зависимость (временно использует)

**Выводы.**

В процессе выполнения третьей лабораторной работы я взяла за основу код из первой и второй лабораторных работ и добавила туда выполнение нужных заданий. Были созданы новый класс бинарного файла, новые классы исключений, которые «ловят» ошибки, связанные не только с процессом игры, но и с сохранением и загрузкой файла. Благодаря им программа корректно завершается с ошибкой, а не просто «вылетает». Создан класс бинарного файла как RAII-обёртка для безопасного выполнения операций сохранения и загрузки игры. Также реализован переход на новый уровень, осуществляемый при убийстве всех врагов. На новом уровне врагов становится на два больше.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: binary_file.h

```
#ifndef BINARY_FILE_H
#define BINARY_FILE_H

#include <fstream>
#include "game_exceptions.h"

class BinaryFile {
    std::fstream file;
    std::string filename;
public:
    BinaryFile(const std::string& name, std::ios::openmode mode)
        : filename(name) {
        file.open(name, std::ios::binary | mode);
        if (!file.is_open()) {
            throw FileOpenException(name, "Check permissions or
disk space");
        }
    }
    ~BinaryFile() { if (file.is_open()) file.close(); }

    std::fstream& stream() { return file; }
    const std::string& getFilename() const { return filename; }
};

#endif
```

Название файла: game_exceptions.h

```
#ifndef GAME_EXCEPTIONS_H
#define GAME_EXCEPTIONS_H

#include <stdexcept>
#include <string>
```

```
class GameException : public std::runtime_error {
public:
    using std::runtime_error::runtime_error;
};

class SaveLoadException : public GameException {
protected:
    std::string filename;
public:
    SaveLoadException(const std::string& msg, const std::string&
file = "")
        : GameException(msg), filename(file) {
    }
    const std::string& getFilename() const { return filename; }
};

class FileOpenException : public SaveLoadException {
public:
    FileOpenException(const std::string& file, const std::string&
reason = "")
        : SaveLoadException("Cannot open file '" + file + "'" +
            (reason.empty() ? "" : ": " + reason), file) {
    }
};

class FileWriteException : public SaveLoadException {
public:
    FileWriteException(const std::string& file)
        : SaveLoadException("Write failed to '" + file + "'", file)
{
    }
};

class FileReadException : public SaveLoadException {
public:
    FileReadException(const std::string& file)
        : SaveLoadException("Read failed from '" + file + "'", file)
{
    }
```

```
};

class CorruptedDataException : public SaveLoadException {
public:
    CorruptedDataException(const std::string& file, const
std::string& detail)
        : SaveLoadException("Corrupted save '" + file + "': " +
detail, file) {
    }
};

class VersionMismatchException : public SaveLoadException {
public:
    VersionMismatchException(const std::string& file, int expected,
int got)
        : SaveLoadException("Version mismatch in '" + file + "':
expected " +
            std::to_string(expected) + ", got " +
std::to_string(got), file) {
    }
};

class TooManyEnemiesException : public GameException {
public:
    TooManyEnemiesException(int count, int max = 5)
        : GameException("Too many enemies: " + std::to_string(count)
+
            ". Maximum allowed: " + std::to_string(max)) {
    }
};

#endif
```

Название файла: player.h

```
#ifndef PLAYER_H
#define PLAYER_H
```

```
#include "game_object.h"
#include "game_constants.h"

class Player : public GameObject {
public:
    Player();
    void move(int deltaX, int deltaY);
};

#endif
```

Название файла: player.cpp

```
#include "player.h"

Player::Player()
    : GameObject(Position(0, 0),
        GameConstants::INITIAL_PLAYER_HEALTH,
        GameConstants::INITIAL_PLAYER_DAMAGE) {
}

void Player::move(int deltaX, int deltaY) {
    currentPosition.setX(currentPosition.getX() + deltaX);
    currentPosition.setY(currentPosition.getY() + deltaY);
}
```

Название файла: enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H

#include "game_object.h"
#include "game_constants.h"

class Enemy : public GameObject {
public:
    Enemy(const Position& position);
    bool canAttackPlayer(const Position& playerPosition) const;

private:
    static constexpr int ATTACK_RANGE = 1;
};
```

```
#endif
```

Название файла: enemy.cpp

```
#include "enemy.h"
#include <cmath>

Enemy::Enemy(const Position& position)
    : GameObject(position,
        GameConstants::INITIAL_ENEMY_HEALTH,
        GameConstants::INITIAL_ENEMY_DAMAGE) {
}

bool Enemy::canAttackPlayer(const Position& playerPosition) const {
    return currentPosition == playerPosition;
}
```

Название файла: position.h

```
#ifndef POSITION_H
#define POSITION_H

class Position {
public:
    Position(int x = 0, int y = 0);

    int getX() const;
    int getY() const;
    void setX(int x);
    void setY(int y);
    void setPosition(int x, int y);

    bool operator==(const Position& other) const;
    bool operator!=(const Position& other) const;

private:
    int xCoordinate;
    int yCoordinate;
};

#endif
```

Название файла: position.cpp

```
#include "position.h"

Position::Position(int x, int y) : xCoordinate(x), yCoordinate(y) {}

int Position::getX() const {
    return xCoordinate;
}

int Position::getY() const {
    return yCoordinate;
}

void Position::setX(int x) {
    xCoordinate = x;
}

void Position::setY(int y) {
    yCoordinate = y;
}

void Position::setPosition(int x, int y) {
    xCoordinate = x;
    yCoordinate = y;
}

bool Position::operator==(const Position& other) const {
    return xCoordinate == other.xCoordinate && yCoordinate ==
other.yCoordinate;
}

bool Position::operator!=(const Position& other) const {
    return !(*this == other);
}
```

Название файла: cell.h

```
#ifndef CELL_H
#define CELL_H
```

```
enum class CellType {
    EMPTY,
    PLAYER,
    ENEMY
};

class Cell {
public:
    Cell();
    CellType getType() const;
    void setType(CellType type);
    bool isEmpty() const;

private:
    CellType cellType;
};

#endif
```

Название файла: cell.cpp

```
#include "cell.h"

Cell::Cell() : cellType(CellType::EMPTY) {}

CellType Cell::getType() const {
    return cellType;
}

void Cell::setType(CellType type) {
    cellType = type;
}

bool Cell::isEmpty() const {
    return cellType == CellType::EMPTY;
}
```

Название файла: main.cpp

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

```
int main() {
    try {
        GameController gameController(15, 15, 3);
        gameController.runGame();

    }
    catch (const std::exception& exception) {
        std::cerr << "Error: " << exception.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Название файла: game_field.cpp

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

GameField::GameField(int width, int height)
    : fieldWidth(width), fieldHeight(height) {

    if (width < GameConstants::MIN_FIELD_SIZE ||
        width > GameConstants::MAX_FIELD_SIZE ||
        height < GameConstants::MIN_FIELD_SIZE ||
        height > GameConstants::MAX_FIELD_SIZE) {
        throw std::invalid_argument("Invalid field dimensions");
    }

    initializeGrid();
}

GameField::GameField(const GameField& other) //конструктор
копирования
    : fieldWidth(other.fieldWidth),
  fieldHeight(other.fieldHeight) {
    copyFrom(other);
}
```

```
    GameField::GameField(GameField&& other) noexcept { //конструктор перемещения
        moveFrom(std::move(other));
    }

    GameField& GameField::operator=(const GameField& other) { //оператор присваивания с копированием
        if (this != &other) {
            fieldWidth = other.fieldWidth;
            fieldHeight = other.fieldHeight;
            copyFrom(other);
        }
        return *this;
    }

    GameField& GameField::operator=(GameField&& other) noexcept { //оператор присваивания с перемещением
        if (this != &other) {
            moveFrom(std::move(other));
        }
        return *this;
    }

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

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

    CellType GameField::getCellType(const Position& position) const {
        if (!isValidPosition(position)) {
            throw std::out_of_range("Position is out of field bounds");
        }
        return grid[position.getY()][position.getX()].getType();
    }
```

```
bool GameField::isValidPosition(const Position& position) const {
  return position.getX() >= 0 && position.getX() < fieldWidth &&
        position.getY() >= 0 && position.getY() < fieldHeight;
}

bool GameField::isPositionEmpty(const Position& position) const {
    return isValidPosition(position) &&
        grid[position.getY()][position.getX()].isEmpty();
}

void GameField::setCellType(const Position& position, CellType type)
{
    if (!isValidPosition(position)) {
        throw std::out_of_range("Position is out of field bounds");
    }
    grid[position.getY()][position.getX()].setType(type);
}

void GameField::clearCell(const Position& position) {
    setCellType(position, CellType::EMPTY);
}

void GameField::initializeGrid() {
    grid.resize(fieldHeight);
    for (int y = 0; y < fieldHeight; y++) {
        grid[y].resize(fieldWidth);
        for (int x = 0; x < fieldWidth; x++) {
            grid[y][x] = Cell();
        }
    }
}

void GameField::copyFrom(const GameField& other) {
    grid = other.grid;//копирование и указателей и элементов
}

void GameField::moveFrom(GameField&& other) noexcept {
    fieldWidth = other.fieldWidth;
    fieldHeight = other.fieldHeight;
```

```
    grid = std::move(other.grid);

    other.fieldWidth = 0;
    other.fieldHeight = 0;
}
```

Название файла: game_field.h

```
#ifndef GAME_FIELD_H
#define GAME_FIELD_H

#include "cell.h"
#include "position.h"
#include "game_constants.h"
#include <vector>
#include <memory>

class GameField {
public:
    GameField(int width = GameConstants::DEFAULT_FIELD_SIZE,
        int height = GameConstants::DEFAULT_FIELD_SIZE);

    GameField(const GameField& other);
    GameField(GameField&& other) noexcept;

    GameField& operator=(const GameField& other);
    GameField& operator=(GameField&& other) noexcept;

    int getWidth() const;
    int getHeight() const;
    CellType getCellType(const Position& position) const;

    bool isValidPosition(const Position& position) const;
    bool isPositionEmpty(const Position& position) const;

    void setCellType(const Position& position, CellType type);
    void clearCell(const Position& position);

private:
    int fieldWidth;
```

```
    int fieldHeight;
    std::vector<std::vector<Cell>> grid;

    void initializeGrid();
    void copyFrom(const GameField& other);
    void moveFrom(GameField&& other) noexcept;
};

#endif
```