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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 4384 _____ Овчаренко Я.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить и применить принципы объектно-ориентированного программирования для разработки модульной системы управления, визуализации и логирования игры. Реализовать систему, основанную на шаблонах проектирования, обеспечивающую разделение ответственности между компонентами, расширяемость и гибкость конфигурации.

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 логирования, а только лишь информировать о событии. Запись может идти как в файл, так и в терминал, способ логирования определяется пользователем через параметры запуска программы.

Выполнение работы.

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

- Класс считывания ввода
- Шаблонный класс управления игрой
- Класс отрисовки игры
- Шаблонный класс визуализации
- Класс для конфигурируемого управления
- Систему логирования событий с классами и их реализациями

Архитектура программы.

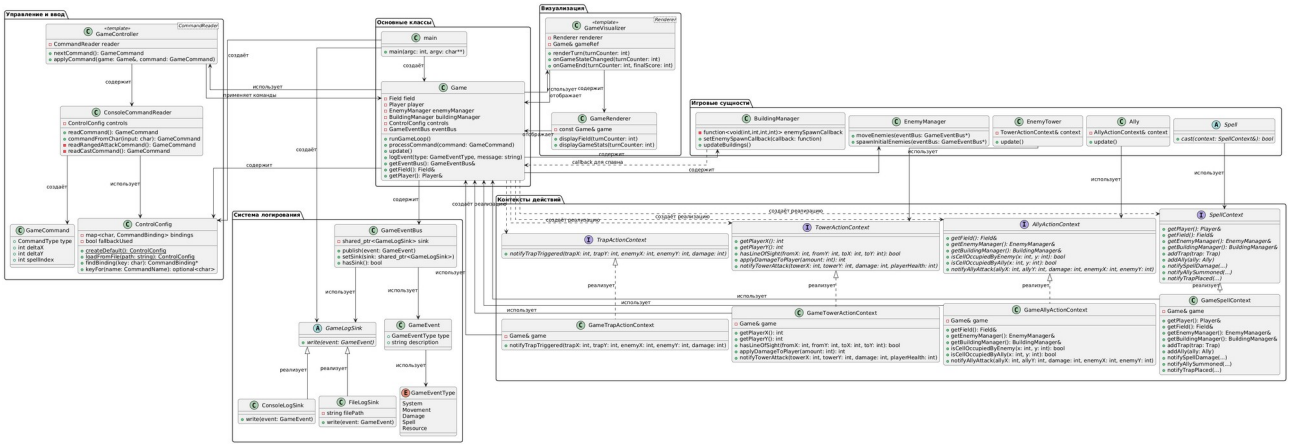

Рисунок 1 - Диаграмма взаимосвязи классов

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- ConsoleCommandReader - считывает ввод из консоли и преобразует в команды
- GameController<CommandReader> - шаблонный класс управления игрой
- GameRenderer - реализует консольную отрисовку игры
- GameVisualizer<Renderer> - шаблонный класс управления визуализацией
- ControlConfig - управляет конфигурацией привязки клавиш
- GameCommand - представляет игровую команду как объект
- GameEventBus - шина событий для публикации и подписки на события
- GameLogSink - абстрактный класс для обработчиков логирования
- ConsoleLogSink - вывод событий в консоль
- FileLogSink - запись событий в файл
- GameEvent - структура для представления игрового события
- GameEventType - перечисление типов событий

Описание классов.

Класс ConsoleCommandReader

Назначение: Класс, считывающий ввод пользователя из консоли и преобразующий символы в объекты команд (GameCommand).

Поля класса:

controls: ControlConfig - конфигурация управления для преобразования символов

Методы класса:

- readCommand(): GameCommand - считывает команду из консоли
- commandFromChar(char input): GameCommand - преобразует символ в команду
- readRangedAttackCommand(): GameCommand - считывает команду дальней атаки
- readCastCommand(): GameCommand - считывает команду каста заклинания

Логика работы:

Считывает символ из std::cin

Ищет привязку в ControlConfig

Для простых команд сразу создаёт GameCommand

Для команд с дополнительным вводом запрашивает дополнительные данные

Связи с другими классами:

- ControlConfig (ассоциация) - использует для преобразования символов
- GameCommand (ассоциация) - создаёт объекты команд

Шаблонный класс GameController<CommandReader>

Назначение: Шаблонный класс управления игрой. Принимает класс считывания ввода как параметр шаблона, создаёт его экземпляр и использует для получения команд.

Поля класса:

- reader: CommandReader - объект считывателя ввода (создаётся из параметра шаблона

Методы класса:

- nextCommand(): GameCommand - получает следующую команду через reader
- applyCommand(Game& game, const GameCommand& command): void - применяет команду к игре

Масштабируемость: Можно легко подставить альтернативные реализации:

- GameController<ConsoleCommandReader> - консольный ввод
- GameController<NetworkCommandReader> - сетевой ввод
- GameController<FileCommandReader> - чтение из файла

Класс GameRenderer

Назначение: Класс, определяющий способ отображения игры. Реализует консольную отрисовку игрового поля и статистики.

Поля класса:

- game: const Game& - ссылка на игровой объект для доступа к состоянию

Методы класса:

- displayField(int turnCounter): void - отображает поле и статистику

- displayGameStats(int turnCounter): void - отображает статистику игры

Принцип работы:

- Отображает поле в виде символов (P - игрок, E - враг, B - здание и т.д.)
- Выводит статистику: ход, HP, мана, счёт, режим боя, количество заклинаний

Шаблонный класс GameVisualizer<Renderer>

Назначение: Шаблонный класс, управляющий визуализацией игры. Принимает класс отрисовки как параметр шаблона, создаёт его экземпляр и реагирует на изменения в игре.

Поля класса:

- renderer: Renderer - объект отрисовки (создаётся из параметра шаблона)
- gameRef: Game& - ссылка на игровой объект

Методы класса:

- renderTurn(int turnCounter): void - отрисовка нового хода
- onGameStateChanged(int turnCounter): void - реакция на изменение состояния
- onGameEnd(int turnCounter, int finalScore): void - отрисовка завершения игры

Масштабируемость: Можно легко подставить альтернативные способы отрисовки:

- GameVisualizer<GameRenderer> - консольная отрисовка
- GameVisualizer<WebRenderer> - веб-отрисовка
- GameVisualizer<GUIRenderer> - графический интерфейс

Класс ControlConfig

Назначение: Управляет настройкой привязки клавиш к командам. Поддерживает загрузку конфигурации из файла с валидацией.

Поля класса:

- bindings: map<char, CommandBinding> - карта привязок клавиш
- fallbackUsed: bool - флаг использования настроек по умолчанию

Методы класса:

- createDefault(): static ControlConfig - создаёт конфигурацию по умолчанию
- loadFromFile(const string& path): static ControlConfig - загружает из файла
- findBinding(char key): const CommandBinding* - находит привязку по клавише
- keyFor(CommandName name): optional<char> - находит клавишу по команде

Система логирования событий

Архитектура: Используется паттерн "Шина событий" (Event Bus). Игровые сущности публикуют события через GameEventBus, а шина передаёт их выбранному GameLogSink.

Компоненты системы:

1. GameEventType - перечисление типов событий:
- System - системные события
- Movement - движение игрока
- Damage - урон
- Spell - заклинания
- Resource - ресурсы
2. GameEvent - структура события:
- type: GameEventType - тип события
- description: string - описание события

3. GameLogSink - абстрактный класс обработчика логирования:
- write(const GameEvent& event): virtual void - запись события
4. ConsoleLogSink - вывод в консоль с временными метками и префиксами
5. FileLogSink - запись в файл с временными метками и префиксами
6. GameEventBus - шина событий:
- publish(const GameEvent& event): void - публикация события
- setSink(shared_ptr<GameLogSink> sink): void - установка обработчика

Интеграция с игровыми сущностями: Сущности уведомляют о событиях через контексты действий:

- TowerActionContext::notifyTowerAttack() - для атак башен
- AllyActionContext::notifyAllyAttack() - для атак союзников
- SpellContext::notifySpellDamage() - для урона заклинаний
- И другие методы уведомления

Разработанный программный код см. в приложении А.

Выводы.

В ходе выполнения лабораторной работы была успешно реализована система управления, визуализации и логирования игры, построенная на принципах объектно-ориентированного программирования. Архитектура системы позволяет легко добавлять новые способы ввода (сетевой, файловый), новые способы визуализации (графический интерфейс, веб-интерфейс) и новые обработчики логирования без изменения основной игровой логики, что подтверждает правильность выбранных проектных решений.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: console_command_reader.h

```
#ifndef CONSOLE_COMMAND_READER_H
#define CONSOLE_COMMAND_READER_H

#include "game_command.h"
#include "control_config.h"

class ConsoleCommandReader {
public:
	explicit ConsoleCommandReader(ControlConfig config = ControlConfig::createDefault());

	GameCommand readCommand() const;

	GameCommand commandFromChar(char input) const;

private:
	GameCommand readRangedAttackCommand() const;
	GameCommand readCastCommand() const;

private:
	ControlConfig controls;
};

#endif
```

Название файла: console_command_reader.cpp

```
#include "console_command_reader.h"
#include <cctype>
#include <iostream>
#include <limits>


ConsoleCommandReader::ConsoleCommandReader(ControlConfig config)
	: controls(std::move(config)) {}


GameCommand ConsoleCommandReader::readCommand() const {
	std::cout << "Enter command: ";
	char input;
	std::cin >> input;
	return commandFromChar(input);
}
```

```
GameCommand ConsoleCommandReader::commandFromChar(char input) const
{
    const CommandBinding* binding = controls.findBinding(input);
    if (!binding) {
        return {CommandType::Invalid, 0, 0, -1};
    }

    if (binding->requiresDirection) {
        return readRangedAttackCommand();
    }
    if (binding->requiresSpellIndex) {
        return readCastCommand();
    }

    GameCommand command;
    command.type = binding->type;
    command.deltaX = binding->deltaX;
    command.deltaY = binding->deltaY;
    command.spellIndex = -1;

    return command;
}

GameCommand ConsoleCommandReader::readRangedAttackCommand() const {
    GameCommand command;
    command.type = CommandType::Invalid;
    command.deltaX = 0;
    command.deltaY = 0;
    command.spellIndex = -1;

    std::cout << "Enter direction for ranged attack (w/a/s/d): ";
    char direction;
    std::cin >> direction;

    switch (direction) {
        case 'w': case 'W':
            command.type = CommandType::RangedAttack;
            command.deltaY = -1;
            break;
```

```
            case 's': case 'S':
                  command.type = CommandType::RangedAttack;
                  command.deltaY = 1;
                  break;
            case 'a': case 'A':
                  command.type = CommandType::RangedAttack;
                  command.deltaX = -1;
                  break;
            case 'd': case 'D':
                  command.type = CommandType::RangedAttack;
                  command.deltaX = 1;
                  break;
            default:
                  std::cout << "Invalid direction!\n";
                  break;
      }

      return command;
}

GameCommand ConsoleCommandReader::readCastCommand() const {
      GameCommand command;
      command.type = CommandType::CastSpell;
      command.deltaX = 0;
      command.deltaY = 0;
      command.spellIndex = -1;
      return command;
}
```

Название файла: game_controller.h

```
#ifndef GAME_CONTROLLER_H
#define GAME_CONTROLLER_H

#include "game_command.h"

class Game;

template <typename CommandReader>
```

```
class GameController {
public:
    explicit GameController(CommandReader reader = CommandReader{})
        : reader(std::move(reader)) {}

    GameCommand nextCommand() {
        return reader.readCommand();
    }

    void applyCommand(Game& game, const GameCommand& command) {
        game.processCommand(command);
    }

private:
    CommandReader reader;
};

#endif
```

Название файла: gamerenderer.h

```
#ifndef GAME_RENDERER_H
#define GAME_RENDERER_H

class Game;

class GameRenderer {
public:
    explicit GameRenderer(const Game& game);

    void displayField(int turnCounter) const;
    void displayGameStats(int turnCounter) const;

private:
    const Game& game;
};

#endif
```

Название файла: gamerenderer.cpp

```
#include "gamerenderer.h"
#include "game.h"
#include "field.h"
#include "player.h"
#include "enemymanager.h"
#include "buildingmanager.h"
#include "ally.h"
#include <iostream>

GameRenderer::GameRenderer(const Game& game) : game(game) {}

void GameRenderer::displayGameStats(int turnCounter) const {
    const Player& player = game.getPlayer();
    const EnemyManager& enemyManager = game.getEnemyManager();
    const BuildingManager& buildingManager = game.getBuildingManager();
    const std::vector<Ally>& allies = game.getAllies();

    std::cout << "\nTurn " << turnCounter;
    std::cout << " | HP: " << player.getHealth() << "/" << player.getMaxHealth();
    std::cout << " | Mana: " << player.getMana() << "/" << player.getMaxMana();
    std::cout << " | Score: " << player.getScore();
    std::cout << " | Mode: " << (player.getCombatMode() == CombatMode::MELEE ? "Melee" : "Ranged");
    std::cout << " | Enhancement: " << player.getCurrentEnhancementLevel();
    std::cout << " | Slowed: " << (player.isSlowed() ? "Yes" : "No");
    std::cout << " | Spells: " << player.getHand().getSpellCount() << "/" << player.getHand().getMaximumSize();
    std::cout << " | Enemies: " << enemyManager.getEnemies().size();
    std::cout << " | Allies: " << allies.size();
```

```
          std::cout      <<      "      |      Towers:      "      <<
buildingManager.getTowers().size();
          std::cout << "\n";

          for (size_t i = 0; i < allies.size(); ++i) {
               if (allies[i].isAlive()) {
                    std::cout  <<  "Ally  "  <<  i  <<  "  HP:  "  <<
allies[i].getHealth() << "  ";
               }
          }
          if (!allies.empty()) {
               std::cout << "\n";
          }
     }

     void GameRenderer::displayField(int turnCounter) const {
          displayGameStats(turnCounter);

          const Field& field = game.getField();
          const Player& player = game.getPlayer();
          const EnemyManager& enemyManager = game.getEnemyManager();
          const        BuildingManager&        buildingManager        =
game.getBuildingManager();
          const std::vector<Ally>& allies = game.getAllies();

          int playerX = player.getX();
          int playerY = player.getY();

          for (int y = 0; y < field.getHeight(); ++y) {
               for (int x = 0; x < field.getWidth(); ++x) {
                    if (x == playerX && y == playerY) {
                         std::cout << "P ";
                         continue;
                    }

                    bool hasAlly = false;
                    for (const auto& ally : allies) {
                         if (ally.isAlive() && ally.getPositionX() == x
&& ally.getPositionY() == y) {
```

```
                    std::cout << "A ";
                    hasAlly = true;
                    break;
                }
            }
            if (hasAlly) continue;

            bool hasEnemy = false;
            for (const auto& enemy : enemyManager.getEnemies())
{
                if (enemy.isAlive() && enemy.getX() == x &&
enemy.getY() == y) {
                    std::cout << "E ";
                    hasEnemy = true;
                    break;
                }
            }
            if (hasEnemy) continue;

            bool hasTower = false;
            for (const auto& tower :
buildingManager.getTowers()) {
                if (tower.getX() == x && tower.getY() == y) {
                    std::cout << "T ";
                    hasTower = true;
                    break;
                }
            }
            if (hasTower) continue;

            bool hasBuilding = false;
            for (const auto& building :
buildingManager.getBuildings()) {
                if (building.getX() == x && building.getY() ==
y) {
                    std::cout << "B ";
                    hasBuilding = true;
                    break;
                }
```

```
                    }
                    if (hasBuilding) continue;

                    switch (field.getCellType(x, y)) {
                        case CellType::WALL: std::cout << "# "; break;
                        case CellType::SLOW: std::cout << "~ "; break;
                        case  CellType::EMPTY:  std::cout  <<  ".  ";
break;
                    }
                }
                std::cout << "\n";
            }
        }
```

Название файла: game_visualizer.h

```
#ifndef GAME_VISUALIZER_H
#define GAME_VISUALIZER_H

class Game;

template <typename Renderer>
class GameVisualizer {
public:
    explicit GameVisualizer(Game& game)
        : renderer(game), gameRef(game) {
    }

    void renderTurn(int turnCounter) {
        renderer.displayField(turnCounter);
    }

    void onGameStateChanged(int turnCounter) {
        renderer.displayField(turnCounter);
    }

    void onGameEnd(int turnCounter, int finalScore) {
        renderer.displayField(turnCounter);
    }
```

```
private:
     Renderer renderer;
     Game& gameRef;
};

#endif
```

Название файла: control_config.h

```
#ifndef CONTROL_CONFIG_H
#define CONTROL_CONFIG_H

#include "game_command.h"
#include <map>
#include <optional>
#include <string>

// Перечень поддерживаемых команд для назначения на клавиши
enum class CommandName {
     MoveUp,
     MoveDown,
     MoveLeft,
     MoveRight,
     SwitchMode,
     RangedAttack,
     CastSpell,
     BuySpell,
     ShowInfo,
     Save,
     Load,
     Quit,
     Help
};

struct CommandBinding {
     CommandName name;
     CommandType type;
     int deltaX = 0;
```

```
        int deltaY = 0;
        bool requiresDirection = false;
        bool requiresSpellIndex = false;
    };

    // Класс инкапсулирует настройку управления: сопоставление клавиши
и действия.
    class ControlConfig {
    public:
        static ControlConfig createDefault();
        static ControlConfig loadFromFile(const std::string& path);

        const CommandBinding* findBinding(char key) const;
        std::optional<char> keyFor(CommandName name) const;
        bool isUsingDefault() const { return fallbackUsed; }

    private:
        std::map<char, CommandBinding> bindings;
        bool fallbackUsed = true;
    };

    #endif
```

Название файла: control_config.cpp

```
#include "control_config.h"
#include <algorithm>
#include <cctype>
#include <fstream>
#include <iostream>
#include <sstream>
#include <unordered_map>

namespace
{

    // Имя команды -> перечисление
    const std::unordered_map<std::string, CommandName> kNameMap = {
```

```
            {"move_up", CommandName::MoveUp},
            {"move_down", CommandName::MoveDown},
            {"move_left", CommandName::MoveLeft},
            {"move_right", CommandName::MoveRight},
            {"switch_mode", CommandName::SwitchMode},
            {"ranged_attack", CommandName::RangedAttack},
            {"cast_spell", CommandName::CastSpell},
            {"buy_spell", CommandName::BuySpell},
            {"show_info", CommandName::ShowInfo},
            {"save", CommandName::Save},
            {"load", CommandName::Load},
            {"quit", CommandName::Quit},
            {"help", CommandName::Help}};

        // Базовый набор команд с привязками по умолчанию
        const std::map<char, CommandBinding> kDefaultBindings = {
                {'w', {CommandName::MoveUp, CommandType::Move, 0, -1,
false, false}},
                {'s', {CommandName::MoveDown, CommandType::Move, 0, 1,
false, false}},
                {'a', {CommandName::MoveLeft, CommandType::Move, -1, 0,
false, false}},
                {'d', {CommandName::MoveRight, CommandType::Move, 1, 0,
false, false}},
                {'m', {CommandName::SwitchMode, CommandType::SwitchMode, 0,
0, false, false}},
                                        {'f',    {CommandName::RangedAttack,
CommandType::RangedAttack, 0, 0, true, false}},
                {'c', {CommandName::CastSpell, CommandType::CastSpell, 0,
0, false, true}},
                {'b', {CommandName::BuySpell, CommandType::BuySpell, 0, 0,
false, false}},
                {'i', {CommandName::ShowInfo, CommandType::ShowInfo, 0, 0,
false, false}},
                {'p', {CommandName::Save, CommandType::Save, 0, 0, false,
false}},
                {'o', {CommandName::Load, CommandType::Load, 0, 0, false,
false}},
```

```
            {'q', {CommandName::Quit, CommandType::Quit, 0, 0, false,
false}},
            {'h', {CommandName::Help, CommandType::Help, 0, 0, false,
false}}};

        std::string trim(const std::string &value)
        {
            auto begin = value.find_first_not_of(" \t\r\n");
            if (begin == std::string::npos)
            {
                return "";
            }
            auto end = value.find_last_not_of(" \t\r\n");
            return value.substr(begin, end - begin + 1);
        }

    } // namespace

    ControlConfig ControlConfig::createDefault()
    {
        ControlConfig cfg;
        cfg.bindings = kDefaultBindings;
        cfg.fallbackUsed = true;
        return cfg;
    }

    ControlConfig ControlConfig::loadFromFile(const std::string &path)
    {
        std::ifstream input(path);
        if (!input.is_open())
        {
            return createDefault();
        }

        std::map<char, CommandBinding> result;
        std::unordered_map<CommandName, char> reverseIndex;
        std::string line;
        bool invalid = false;
```

```
        while (std::getline(input, line))
        {
            if (line.empty())
            {
                continue;
            }

            // Пропускаем комментарии (строки, начинающиеся с #)
            std::string trimmedLine = trim(line);
            if (trimmedLine.empty() || trimmedLine[0] == '#')
            {
                continue;
            }

            auto delimiter = line.find('=');
            if (delimiter == std::string::npos)
            {
                invalid = true;
                break;
            }

            std::string namePart = trim(line.substr(0, delimiter));
            std::string keyPart = trim(line.substr(delimiter + 1));
                      std::transform(namePart.begin(),  namePart.end(),
namePart.begin(), [](unsigned char c)
                           { return std::tolower(c); });

            auto nameIt = kNameMap.find(namePart);
            if (nameIt == kNameMap.end() || keyPart.size() != 1)
            {
                invalid = true;
                break;
            }

            char key = static_cast<char>(std::tolower(keyPart[0]));
            CommandName commandName = nameIt->second;

            if (result.find(key) != result.end())
            {
```

```
                invalid = true;
                break;
            }
            if (reverseIndex.find(commandName) != reverseIndex.end())
            {
                invalid = true;
                break;
            }

              CommandBinding binding{commandName, CommandType::Invalid,
0, 0, false, false};
            bool foundTemplate = false;
            // Берем параметры из шаблона по имени, а не по клавише.
                    for (const auto &[defaultKey, defaultBinding] :
kDefaultBindings)
            {
                if (defaultBinding.name == commandName)
                {
                    binding = defaultBinding;
                    foundTemplate = true;
                    break;
                }
            }

            if (!foundTemplate)
            {
                invalid = true;
                break;
            }

            result.insert({key, binding});
            reverseIndex.insert({commandName, key});
        }

        // Проверяем, что каждая команда настроена ровно один раз
        if (!invalid && reverseIndex.size() == kNameMap.size())
        {
            ControlConfig cfg;
            cfg.bindings = std::move(result);
```

```
            cfg.fallbackUsed = false;
            return cfg;
        }

          std::cout << "Invalid control mapping file. Using default
controls.\n";
        return createDefault();
    }

    const CommandBinding *ControlConfig::findBinding(char key) const
    {
        auto it = bindings.find(static_cast<char>(std::tolower(key)));
        if (it == bindings.end())
        {
            return nullptr;
        }
        return &it->second;
    }

    std::optional<char> ControlConfig::keyFor(CommandName name) const
    {
        for (const auto &[key, binding] : bindings)
        {
            if (binding.name == name)
            {
                return key;
            }
        }
        return std::nullopt;
    }
```

Название файла: game_logging.h

```
#ifndef GAME_LOGGING_H
#define GAME_LOGGING_H

#include <memory>
#include <string>
```

```
enum class GameEventType {
    System,
    Movement,
    Damage,
    Spell,
    Resource
};

struct GameEvent {
    GameEventType type = GameEventType::System;
    std::string description;
};

class GameLogSink {
public:
    virtual ~GameLogSink() = default;
    virtual void write(const GameEvent& event) = 0;
};

class ConsoleLogSink : public GameLogSink {
public:
    void write(const GameEvent& event) override;
};

class FileLogSink : public GameLogSink {
public:
    explicit FileLogSink(const std::string& path);
    void write(const GameEvent& event) override;

private:
    std::string filePath;
};

 // Простая "шина" событий: сущности уведомляют об игровых событиях,
// а шина передает их выбранному обработчику.
class GameEventBus {
public:
```

```
          explicit   GameEventBus(std::shared_ptr<GameLogSink>   sink   =
nullptr);

          void setSink(std::shared_ptr<GameLogSink> sink);
          void publish(const GameEvent& event) const;
          bool hasSink() const { return static_cast<bool>(sink); }

      private:
          std::shared_ptr<GameLogSink> sink;
      };

      #endif
```

Название файла: game_logging.cpp

```
#include "game_logging.h"
#include <ctime>
#include <fstream>
#include <iostream>

namespace
{

          std::string timestamp()
          {
                std::time_t now = std::time(nullptr);
                char buffer[32]{};
                std::strftime(buffer,  sizeof(buffer),  "%Y-%m-%d  %H:%M:
%S", std::localtime(&now));
                return buffer;
          }

          std::string prefix(GameEventType type)
          {
                switch (type)
                {
                case GameEventType::Movement:
                      return "[MOVE]";
```

```
                case GameEventType::Damage:
                     return "[DMG ]";
                case GameEventType::Spell:
                     return "[SPELL]";
                case GameEventType::Resource:
                     return "[RES ]";
                case GameEventType::System:
                default:
                     return "[SYS ]";
                }
           }

     } // namespace

     void ConsoleLogSink::write(const GameEvent &event)
     {
           std::cout << prefix(event.type) << " " << timestamp() << " "
<< event.description << "\n";
     }

     FileLogSink::FileLogSink(const std::string &path) : filePath(path)
{}

     void FileLogSink::write(const GameEvent &event)
     {
           std::ofstream out(filePath, std::ios::app);
           if (!out.is_open())
           {
                return;
           }
           out << prefix(event.type) << " " << timestamp() << " " <<
event.description << "\n";
     }

     GameEventBus::GameEventBus(std::shared_ptr<GameLogSink> sink)
           : sink(std::move(sink)) {}

     void GameEventBus::setSink(std::shared_ptr<GameLogSink> newSink)
     {
```

```
        sink = std::move(newSink);
    }

    void GameEventBus::publish(const GameEvent &event) const
    {
        if (sink)
        {
            sink->write(event);
        }
    }
```

Название файла: game_loop.cpp

```
    #include "game.h"
    #include "game_controller.h"
    #include "game_visualizer.h"
    #include "console_command_reader.h"
    #include "gamerenderer.h"
    #include <iostream>

    void Game::runGameLoop() {
        displayHelp();

         GameController<ConsoleCommandReader>
controller{ConsoleCommandReader(controls)};
         GameVisualizer<GameRenderer> visualizer(*this);

        while (campaignActive) {
            while (isGameRunning()) {
                    visualizer.renderTurn(turnCounter);
                GameCommand command = controller.nextCommand();
                controller.applyCommand(*this, command);
                update();
            }

            if (!campaignActive) {
                break;
            }
```

```
            if (!player.isAlive()) {
                onPlayerDefeat();
            } else if (!levelInProgress && !awaitingUpgradeSelection) {
                startLevel(currentLevelIndex);
            } else if (!gameRunning) {
                break;
            }
        }
    }
```

Название файла: game_input.cpp

```
#include "game.h"
#include "console_command_reader.h"
#include <iostream>
#include <limits>
#include <sstream>

void Game::processCommand(const GameCommand& command) {
    playerActionTaken = false;

    int x = player.getX();
    int y = player.getY();
    int newX = x + command.deltaX;
    int newY = y + command.deltaY;

    if (player.isSlowed() && command.type == CommandType::Move) {
        std::cout << "You are slowed and cannot move this turn!
\n";
        player.setSlowed(false);
        playerActionTaken = true;
        return;
    }

    switch (command.type) {
        case CommandType::Move:
            if (enemyManager.attackEnemyAtPosition(newX, newY,
player, &eventBus)) {
                playerActionTaken = true;
```

```
                    giveSpellForEnemyKill();
                } else if (buildingManager.isCellOccupiedByBuilding(newX, newY)) {
                    std::cout << "Cannot move there - building in the way!\n";
                } else if (field.canMoveTo(newX, newY)) {
                    player.setPosition(newX, newY);
                    processCellEffects(newX, newY);
                    playerActionTaken = true;
                    std::ostringstream ss;
                    ss << "Player moved to (" << newX << ", " << newY << ")";
                    logEvent(GameEventType::Movement, ss.str());
                } else {
                    std::cout << "Cannot move there!\n";
                }
                break;

            case CommandType::SwitchMode:
                player.switchCombatMode();
                std::cout << "Switched to "
                    << (player.getCombatMode() == CombatMode::MELEE ? "Melee" : "Ranged")
                    << " mode (turn consumed)\n";
                playerActionTaken = true;
                logEvent(GameEventType::System, "Player switched combat mode");
                break;

            case CommandType::RangedAttack:
                if (player.getCombatMode() == CombatMode::RANGED && (command.deltaX != 0 || command.deltaY != 0)) {
                    enemyManager.performRangedAttack(field, player, command.deltaX, command.deltaY, &eventBus);
                    playerActionTaken = true;
                    giveSpellForEnemyKill();
                    std::ostringstream ss;
                    ss << "Ranged attack from (" << x << ", " << y << ") direction (" << command.deltaX << ", " << command.deltaY << ")";
```

```
                    logEvent(GameEventType::Damage, ss.str());
                } else {
                    std::cout << "You can only use ranged attacks
in ranged mode!\n";
                }
                break;

            case CommandType::CastSpell: {
                displaySpells();
                if (player.getHand().getSpellCount() > 0) {
                    int selectedIndex = command.spellIndex;
                    if (selectedIndex < 0) {
                        std::cout  <<  "Select  spell  to  cast
(index): ";
                        if (!(std::cin >> selectedIndex)) {
                            std::cout << "Invalid input! Please
enter a number.\n";
                            std::cin.clear();

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                            break;
                        }
                    }

                    if  (selectedIndex  >=  0  &&  selectedIndex  <
player.getHand().getSpellCount()) {
                        if (castSpell(selectedIndex)) {
                            playerActionTaken = true;
                            logEvent(GameEventType::Spell,
"Spell cast from hand slot " + std::to_string(selectedIndex));
                        }
                    } else {
                        std::cout << "Invalid spell index!\n";
                    }
                }
                break;
            }

            case CommandType::BuySpell:
```

```
                    if (buySpell()) {
                        playerActionTaken = true;
                        logEvent(GameEventType::Resource,      "Player
bought a spell");
                    }
                    break;

                case CommandType::ShowInfo:
                    displaySpells();
                    std::cout << "Mana: " << player.getMana() << "/" <<
player.getMaxMana() << "\n";
                    std::cout << "Score: " << player.getScore() << " |
Next spell at: "
                        <<            (ENEMIES_PER_SPELL            -
enemiesKilledSinceLastSpell) << " kills\n";
                    break;

                case CommandType::Save:
                    saveGameState();
                    logEvent(GameEventType::System,     "Manual     save
requested");
                    break;

                case CommandType::Load:
                    loadGameState();
                    logEvent(GameEventType::System,     "Manual     load
requested");
                    break;

                case CommandType::Quit:
                    gameRunning = false;
                    campaignActive = false;
                    std::cout << "Game ended by player.\n";
                    logEvent(GameEventType::System, "Game terminated by
player");
                    break;

                case CommandType::Help:
                    displayHelp();
```

```
                    break;

                case CommandType::None:
                case CommandType::Invalid:
                default:
                    std::cout << "Unknown command. Press 'h' for help.
\n";
                    break;
            }
        }

        void Game::processInput(char input) {
            ConsoleCommandReader reader(controls);
            processCommand(reader.commandFromChar(input));
        }
```

Название файла: main.cpp

```
        #include <iostream>
        #include <memory>
        #include <string>
        #include "game.h"
        #include "control_config.h"
        #include "game_logging.h"

        int main(int argc, char* argv[]) {
            std::string controlFile;
            std::string logMode = "console";
            std::string logFilePath = "game.log";

            for (int i = 1; i < argc; ++i) {
                std::string arg = argv[i];
                if (arg == "--controls" && i + 1 < argc) {
                    controlFile = argv[++i];
                } else if (arg == "--log" && i + 1 < argc) {
                    logMode = argv[++i];
                } else if (arg == "--log-file" && i + 1 < argc) {
                    logFilePath = argv[++i];
                }
```

```
 }

 ControlConfig controls = controlFile.empty()
      ? ControlConfig::createDefault()
      : ControlConfig::loadFromFile(controlFile);

 std::shared_ptr<GameLogSink> sink;
 if (logMode == "file") {
      sink = std::make_shared<FileLogSink>(logFilePath);
 } else if (logMode == "silent") {
      sink = nullptr;
 } else {
      sink = std::make_shared<ConsoleLogSink>();
 }

int menuChoice = 0;
std::cout << "1. Start New Game\n";
std::cout << "2. Load Saved Game\n";
std::cout << "Select option: ";
std::cin >> menuChoice;

int width = 15;
int height = 15;
if (menuChoice == 1) {
    std::cout << "Enter field width (10-25): ";
    std::cin >> width;
    if (width < 10) width = 10;
    if (width > 25) width = 25;
    std::cout << "Enter field height (10-25): ";
    std::cin >> height;
    if (height < 10) height = 10;
    if (height > 25) height = 25;
}

Game game(width, height, controls, sink);

if (menuChoice == 2) {
    game.loadCampaign();
}
```

```
    game.runGameLoop();

    return 0;
}
```

Название файла: game.h

```
#ifndef GAME_H
#define GAME_H

#include "field.h"
#include "player.h"
#include "enemymanager.h"
#include "buildingmanager.h"
#include "game_command.h"
#include "ally.h"
#include "trap.h"
#include "game_state.h"
#include "game_persistence.h"
#include "spell.h"
#include "control_config.h"
#include "game_logging.h"
#include <iostream>
#include <vector>
#include <memory>
#include <string>
#include <map>

class Spell;

class Game {
private:
     struct LevelDefinition {
          int fieldWidth = 0;
          int fieldHeight = 0;
          int initialEnemies = 0;
          int buildingCount = 0;
          int towerCount = 0;
```

```
        int enemyHealth = 30;
        int enemyDamage = 10;
        int buildingHealth = 100;
        int towerHealth = 150;
        int towerRange = 4;
        int towerDamage = 8;
    };

    Field field;
    Player player;
    EnemyManager enemyManager;
    BuildingManager buildingManager;
    bool gameRunning;
    int turnCounter;
    bool playerActionTaken;
    std::vector<Ally> allies;
    std::vector<Trap> traps;
    std::vector<std::unique_ptr<Spell>> availableSpells;
    int enemiesKilledSinceLastSpell;
    const int ENEMIES_PER_SPELL = 3;
    const int SPELL_COST = 50;
     int baseFieldWidth;
     int baseFieldHeight;
     int currentLevelIndex;
     bool awaitingUpgradeSelection;
     bool awaitingRestartDecision;
     bool campaignActive;
     bool levelInProgress;
     std::string saveFilePath;
     GamePersistence persistence;
     std::map<SpellType, int> permanentSpellEnhancements;
     int currentEnemyHealth;
     int currentEnemyDamage;
     ControlConfig controls;
     GameEventBus eventBus;
     void applyPermanentEnhancementsToSpell(Spell& spell) const;
    void spawnPlayer();
    void processCellEffects(int x, int y);
    void displayHelp() const;
```

```
        void checkTraps();
        void updateAllies();
         void startLevel(int levelIndex, bool fromLoad = false);
         void resetWorldState();
         LevelDefinition buildLevelDefinition(int levelIndex) const;
         bool isLevelCleared() const;
         void handleLevelCompletion();
         void promptUpgradeSelection();
         void applyUpgradeChoice(int choice);
         void removeHalfOfPlayerSpells();
         void onPlayerDefeat();
         void saveGameState();
         void loadGameState();
         GameState buildGameState() const;
         void restoreFromState(const GameState& state);
         std::vector<SpellState> captureAvailableSpellStates() const;
         void   restoreAvailableSpells(const   std::vector<SpellState>&
states);

     public:
               Game(int   fieldWidth,   int   fieldHeight,   ControlConfig
controlConfig = ControlConfig::createDefault(),
                std::shared_ptr<GameLogSink> sink = nullptr);

        bool isGameRunning() const;
        bool isCampaignActive() const { return campaignActive; }
        int getTurnCounter() const { return turnCounter; }
        void displayField() const;
        void processCommand(const GameCommand& command);
        void processInput(char input);
        void update();
        void runGameLoop();
         void saveCampaign();
         void loadCampaign();
         const ControlConfig& getControls() const { return controls; }
         GameEventBus& getEventBus() { return eventBus; }

         // Метод для логирования событий (доступен для контекстов и
других компонентов)
```

```
        void logEvent(GameEventType type, const std::string& message);

        // Геттеры для доступа к приватным полям
        Field& getField() { return field; }
        const Field& getField() const { return field; }

        Player& getPlayer() { return player; }
        const Player& getPlayer() const { return player; }

        EnemyManager& getEnemyManager() { return enemyManager; }
            const EnemyManager& getEnemyManager() const { return
enemyManager; }

        BuildingManager& getBuildingManager() { return buildingManager;
}
          const BuildingManager& getBuildingManager() const { return
buildingManager; }

        std::vector<Ally>& getAllies() { return allies; }
        const std::vector<Ally>& getAllies() const { return allies; }

        std::vector<Trap>& getTraps() { return traps; }
        const std::vector<Trap>& getTraps() const { return traps; }

        void addAlly(const Ally& ally);
        void addTrap(const Trap& trap);
        void removeTrap(int index);
        void initializeSpells();
        void displaySpells() const;
        bool castSpell(int spellIndex);
        void giveRandomSpell();
        void giveSpellForEnemyKill();
        bool buySpell();
         bool isCellOccupiedByAlly(int x, int y) const;
         bool isCellOccupiedByEnemy(int x, int y) const;
         bool damageEntitiesAtPosition(int x, int y, int damageAmount);
    };

    #endif
```

Название файла: game_initialization.cpp

```
#include "game.h"
#include "ally.h"
#include "trap.h"
#include "directdamagespell.h"
#include "areadamagespell.h"
#include "trapspell.h"
#include "summonspell.h"
#include "enhancementspell.h"
#include "gamerenderer.h"
#include <iostream>
#include <random>
#include <sstream>

Game::Game(int fieldWidth, int fieldHeight, ControlConfig controlConfig, std::shared_ptr<GameLogSink> sink)
    : field(fieldWidth, fieldHeight),
      player(),
      gameRunning(true),
      turnCounter(0),
      playerActionTaken(false),
      enemiesKilledSinceLastSpell(0),
      baseFieldWidth(fieldWidth),
      baseFieldHeight(fieldHeight),
      currentLevelIndex(1),
      awaitingUpgradeSelection(false),
      awaitingRestartDecision(false),
      campaignActive(true),
      levelInProgress(false),
      saveFilePath("savegame.dat"),
      currentEnemyHealth(30),
      currentEnemyDamage(10),
       controls(std::move(controlConfig)),
       eventBus(std::move(sink)) {
     // Устанавливаем callback для уведомлений о спавне врагов от зданий
    // Сущности только информируют, Game решает, логировать ли
```

```
        buildingManager.setEnemySpawnCallback([this](int buildingX, int
buildingY, int spawnX, int spawnY) {
            std::ostringstream ss;
            ss << "Building at (" << buildingX << ", " << buildingY
                  << ") spawned new enemy at (" << spawnX << ", " <<
spawnY << ")";
            logEvent(GameEventType::System, ss.str());
        });

        initializeSpells();
        startLevel(currentLevelIndex);
    }

    void Game::spawnPlayer() {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> disX(1, field.getWidth() - 2);
        std::uniform_int_distribution<> disY(1, field.getHeight() - 2);

        int x, y;
        do {
            x = disX(gen);
            y = disY(gen);
                    }    while    (!field.isCellPassable(x,    y)    ||
buildingManager.isCellOccupiedByBuilding(x, y));

        player.setPosition(x, y);
    }

    void Game::processCellEffects(int x, int y) {
        if (field.isSlowCell(x, y)) {
            player.setSlowed(true);
             std::cout << "You stepped on a slow cell! You'll miss next
turn.\n";
        }
    }

    void Game::displayHelp() const {
         auto keyOrFallback = [&](CommandName name, char fallback) {
```

```
            auto value = controls.keyFor(name);
            return value.value_or(fallback);
        };

        std::cout << "Controls:\n";
        std::cout << "  " << keyOrFallback(CommandName::MoveUp, 'w')
                  << "/" << keyOrFallback(CommandName::MoveLeft,
'a')
                  << "/" << keyOrFallback(CommandName::MoveDown,
's')
                  << "/" << keyOrFallback(CommandName::MoveRight,
'd') << " - move\n";
         std::cout << "   " << keyOrFallback(CommandName::SwitchMode,
'm') << " - switch combat mode\n";
         std::cout << "   " << keyOrFallback(CommandName::RangedAttack,
'f') << " - ranged attack (ranged mode only)\n";
        std::cout << "  " << keyOrFallback(CommandName::CastSpell, 'c')
<< " - cast spell\n";
         std::cout << "  " << keyOrFallback(CommandName::BuySpell, 'b')
<< " - buy spell (cost: " << SPELL_COST << " score)\n";
        std::cout << "  " << keyOrFallback(CommandName::Save, 'p') << "
- save game\n";
        std::cout << "  " << keyOrFallback(CommandName::Load, 'o') << "
- load game\n";
         std::cout << "  " << keyOrFallback(CommandName::ShowInfo, 'i')
<< " - show spell information and stats\n";
        std::cout << "  " << keyOrFallback(CommandName::Help, 'h') << "
- show this help\n";
        std::cout << "  " << keyOrFallback(CommandName::Quit, 'q') << "
- quit\n";
         std::cout      <<      "Mapping      source:      "      <<
(controls.isUsingDefault() ? "default layout" : "custom layout") << "\n";
        std::cout << "Get new spells by killing " << ENEMIES_PER_SPELL
<< " enemies or buying them.\n";
    }

    bool Game::isGameRunning() const {
        return gameRunning && player.isAlive();
    }
```

```
void Game::displayField() const {
    GameRenderer renderer(*this);
    renderer.displayField(turnCounter);
}
```