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»
Тема: Шаблонные классы

Студент гр. 3388 _____ Лексин М.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Разработать систему пользовательского интерфейса, которая будет обеспечивать гибкое и удобное взаимодействие пользователя с игрой. Создать комплекс взаимосвязанных компонентов, включающих систему отображения игрового состояния, обработку пользовательского ввода, абстрактный слой для поддержки различных типов визуализации и конфигурируемое управление.

Выполнение работы.

1. **IGameObserver** – Интерфейс наблюдателя за состоянием игры.

1.1. onFieldUpdate() - вызывается при любом изменении состояния игрового поля (выстрел, размещения корабля).

1.2. onAbilityUsed() - вызывается после использования специальной способности игроком.

1.3. onGameOver() - вызывается при завершении игры (победа/ поражение).

1.4. onShipDestroyed() - вызывается при уничтожении корабля.

1.5. renderShipPlacement(int shipLength, int shipNumber) – вызывается при размещении кораблей в начале игры.

1.6. RenderStartMenu() – вызывается при отображении главного меню игры.

2. **IGameRender** – Интерфейс для реализации различных способов отображения игры.

2.1. renderUserField(const GameField& field) - отображает поле игрока со всеми кораблями.

2.2. renderComputerField(const GameField& field) - отображает поле компьютера (скрывая неатакованные корабли).

2.3. renderAbilityStatus(const AbilityManager& manager) - показывает доступные способности игрока.

2.4. renderGameStatus(const Game& game) - отображает общее состояние игры.

2.5. renderShipPlacement(int shipLength, int shipNumber) - показывает процесс размещения кораблей.

2.6. renderStartMenu() - отображает стартовое меню.

3. **GameDisplay** – Шаблонный класс, соединяющий логику игры с системой отображения.

3.1. GameDisplay(std::shared_ptr<Game> game) - конструктор, инициализирует связь с игрой.

3.2. updateDisplay() - обновляет все элементы интерфейса: поле игрока, поле компьютера, статус способностей, общий статус игры.

3.3. refresh() - принудительное обновление всего интерфейса.

4. **TerminalRender** – Реализация консольного интерфейса.

4.1. renderUserField(const GameField& field) - отображает поле игрока с полной информацией.

4.2. renderComputerField(const GameField& field) - отображает поле компьютера, скрывая корабли.

4.3. renderAbilityStatus(const AbilityManager& manager) - показывает доступные способности.

4.4. renderStartMenu() - отображает главное меню.

4.5. renderShipPlacement(int shipLength, int shipNumber) - показывает процесс размещения кораблей.

4.6. renderAttackResult(int x, int y, bool isHit, bool isSunk, bool isComputerAttack) - отображает результаты атаки.

4.7. renderField(const GameField& field, bool isComputer) - базовый метод отрисовки поля.

4.8. getFieldSymbol(const GameField& field, int x, int y, bool isComputer) - определяет символ для клетки поля.

4.9. renderHeader() - отрисовка заголовка игры.

4.10. renderInstructions() - отображение условных обозначений.

5. **GameController** – Шаблонный класс управления игрой.

5.1. GameController(std::shared_ptr<Game> game, std::shared_ptr<ICommandHandler> handler) - инициализация контроллера.

5.2. processInput() - обработка пользовательского ввода и передача команд обработку.

5.3. isGameOver() - проверка завершения игры.

5.2. run() - основной цикл игры.

6. **ICommandHandler** – Интерфейс обработки команд.

6.1. handleCommand(Command cmd, Game& game) - обработка игровых команд.

6.2. handleStartMenu(Game& game) - обработка команд главного меню.

7. **TerminalInputProcessor** – Обработчик консольного ввода.

7.1. TerminalInputProcessor(const std::string& configFile) - создает процессор ввода с указанной конфигурацией.

7.2. getCommand() - получает и интерпретирует команду пользователя.

7.3. getKeymap() - возвращает текущий маппинг клавиш.

7.4. getCommandDescription(char key) - возвращает описание команды для клавиши.

7.5. loadDefaultKeymap() - загружает стандартную конфигурацию клавиш.

7.6. loadKeymap(const std::string& filename) - загружает конфигурацию из файла.

7.7. validateKeymap() - проверяет корректность конфигурации.

7.8. stringToCommand(const std::string& str) - преобразует строку в команду.

7.9. CommandToString(command cmd) - преобразует команду в строку.

8. **DefaultCommandHandler** – стандартная реализация обработка команд.

8.1. handleCommand(Command cmd, Game& game) - обрабатывает команды.

8.2 handleStartMenu(Game& game) - обработка главного меню.

8.3 showStartMenu() - отображение и обработка выбора в главном меню.

UML-диаграмма.

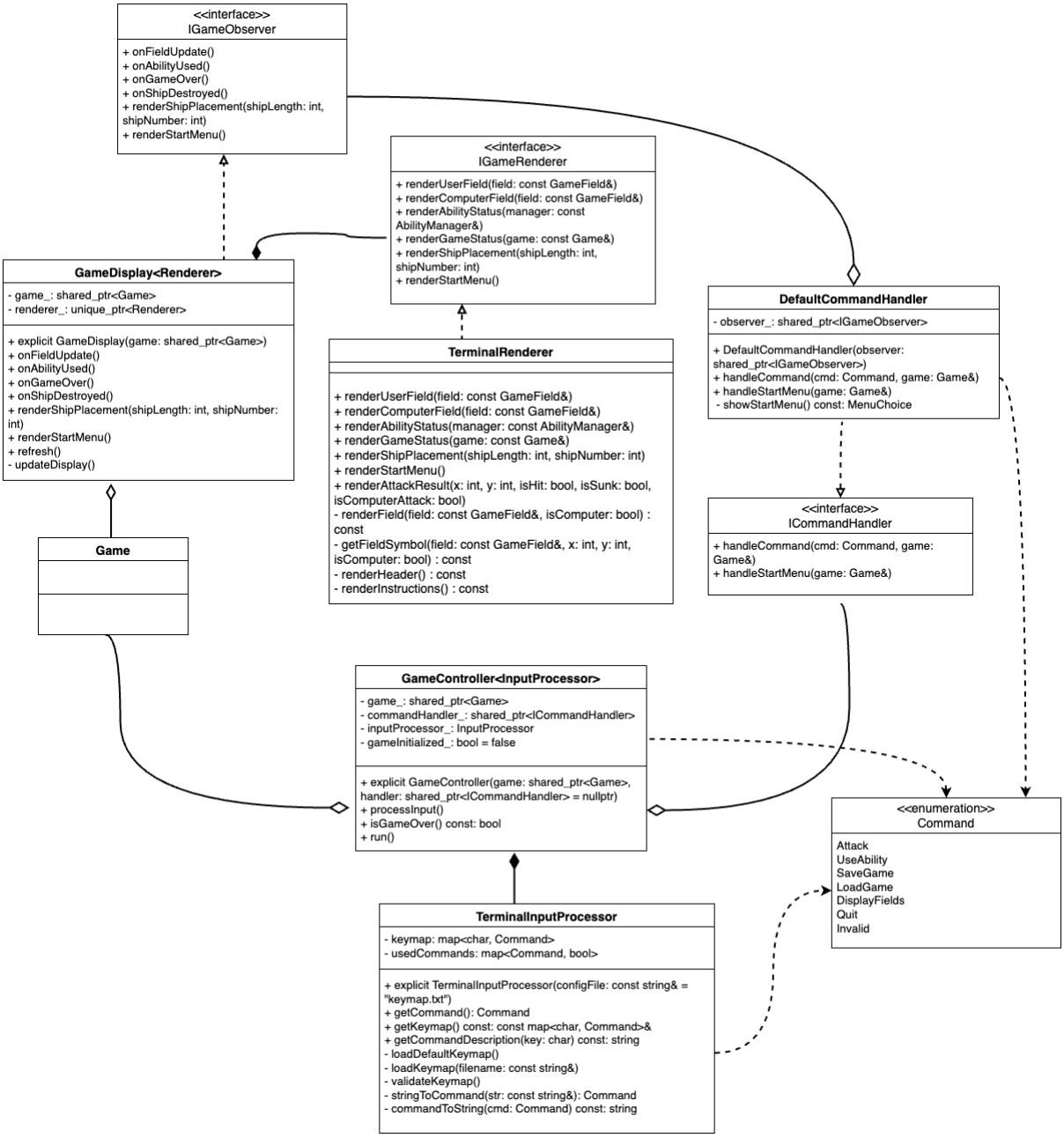

Рисунок 1 – UML-диаграмма

Вывод.

В ходе выполнения лабораторной работы была разработана система пользовательского интерфейса, которая обеспечивает гибкое и удобное взаимодействие пользователя с игрой. Создан комплекс взаимосвязанных компонентов, включающих систему отображения игрового состояния, обработку пользовательского ввода, абстрактный слой для поддержки различных типов визуализации и конфигурируемое управление.

ПРИЛОЖЕНИЕ

А. Исходный программный код

1. game_controller.h

```
#ifndef GAME_CONTROLLER_H
#define GAME_CONTROLLER_H

#include "game.h"
#include <memory>

enum class Command {
    Attack,
    UseAbility,
    SaveGame,
    LoadGame,
    DisplayFields,
    Quit,
    Invalid
};

enum class MenuChoice {
    NewGame,
    LoadGame,
    Exit
};

class ICommandHandler {
public:
    virtual ~ICommandHandler() = default;
    virtual void handleCommand(Command cmd, Game& game) = 0;
    virtual void handleStartMenu(Game& game) = 0;
};

class DefaultCommandHandler : public ICommandHandler {
public:
        DefaultCommandHandler(std::shared_ptr<IGameObserver> observer) :
observer_(observer) {}
    void handleCommand(Command cmd, Game& game) override;
    void handleStartMenu(Game& game) override;
private:
    MenuChoice showStartMenu() const;
    std::shared_ptr<IGameObserver> observer_;
};

template<typename InputProcessor>
class GameController {
public:
    explicit GameController(std::shared_ptr<Game> game,
                            std::shared_ptr<ICommandHandler> handler =
std::make_shared<DefaultCommandHandler>());
    void processInput();
    bool isGameOver() const;
    void run();

private:
    std::shared_ptr<Game> game_;
    std::shared_ptr<ICommandHandler> commandHandler_;
    InputProcessor inputProcessor_;
```

```
    bool gameInitialized_ = false;
};

template<typename InputProcessor>
GameController<InputProcessor>::GameController(std::shared_ptr<Game>
game,

std::shared_ptr<ICommandHandler> handler)
    : game_(game), commandHandler_(handler) {}

template<typename InputProcessor>
void GameController<InputProcessor>::processInput() {
    if (!gameInitialized_) {
        commandHandler_->handleStartMenu(*game_);
        gameInitialized_ = true;
        return;
    }

    Command cmd = inputProcessor_.getCommand();
    if (cmd != Command::Invalid) {
        commandHandler_->handleCommand(cmd, *game_);
    }
}

template<typename InputProcessor>
bool GameController<InputProcessor>::isGameOver() const {
    return game_ ? game_->isGameOver() : true;
}

template<typename InputProcessor>
void GameController<InputProcessor>::run() {
    while (!isGameOver()) {
        processInput();
    }
}

#endif
```

2. game_controller.cpp

```
#include "game_controller.h"
#include "ship_placement_handler.h"
#include <iostream>
#include <iomanip>
#include <limits>

MenuChoice DefaultCommandHandler::showStartMenu() const {
    std::cout << "\n=== Морской бой ===\n";
    std::cout << "1. Новая игра\n";
    std::cout << "2. Загрузить игру\n";
    std::cout << "3. Выход\n";
    std::cout << "\nВыберите действие (1-3): ";

    int choice;
    while (!(std::cin >> choice) || choice < 1 || choice > 3) {
        std::cin.clear();
             std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        std::cout << "Пожалуйста, введите число от 1 до 3: ";
    }
```

```
    switch (choice) {
        case 1: return MenuChoice::NewGame;
        case 2: return MenuChoice::LoadGame;
        case 3: return MenuChoice::Exit;
        default: return MenuChoice::Exit;
    }
}

void DefaultCommandHandler::handleStartMenu(Game& game) {
    MenuChoice choice = showStartMenu();

    switch (choice) {
        case MenuChoice::NewGame:
            game.initializeGame();
            break;

        case MenuChoice::LoadGame: {
            std::string filename;
            std::cout << "Введите имя файла для загрузки: ";
             std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            std::getline(std::cin, filename);

            try {
                game.loadGame(filename);
            } catch (const std::exception& e) {
                     std::cout << "Ошибка при загрузке: " << e.what() <<
"\n";
                std::cout << "Начинаем новую игру...\n";
                game.initializeGame();
            }
            break;
        }

        case MenuChoice::Exit:
            game.setGameOver(true);
            std::cout << "Спасибо за игру!\n";
            break;
    }
}

void DefaultCommandHandler::handleCommand(Command cmd, Game& game) {
    try {
        switch(cmd) {
            case Command::Attack:
            {
                int x, y;
                    std::cout << "Введите координату X для атаки (0-" <<
GameField::DEFAULT_WIDTH - 1 << "): ";
                             while (!(std::cin >> x) || x < 0 || x >=
GameField::DEFAULT_WIDTH) {
                    std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                             std::cout << "Пожалуйста, введите корректную
координату X (0-"
                             << GameField::DEFAULT_WIDTH - 1 << "): ";
                }
```

```
                    std::cout << "Введите координату Y для атаки (0-" <<
GameField::DEFAULT_HEIGHT - 1 << "): ";
                                while (!(std::cin >> y) || y < 0 || y >=
GameField::DEFAULT_HEIGHT) {
                    std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                                std::cout << "Пожалуйста, введите корректную
координату Y (0-"
                                << GameField::DEFAULT_HEIGHT - 1 << "): ";
                }

                GameField& computerField = game.getComputerField();
                                        ShipManager* computerShipManager =
game.getComputerShipManager();

                            bool shipSunk = computerField.attackCell(x, y,
*computerShipManager);
                if (shipSunk) {
                    game.getUserAbilityManager()->addRandomAbility();
                }

                    game.computerTurn();
                    break;
            }

            case Command::UseAbility:
            {
                if (game.getUserAbilityManager()->hasAbilities()) {
                                            game.getUserAbilityManager()-
>useAbility(game.getComputerField());
                    std::cout << "Способность использована успешно.\n";
                } else {
                    std::cout << "У вас нет доступных способностей.\n";
                }
                break;
            }

            case Command::SaveGame:
            {
                std::string filename;
                std::cout << "Введите имя файла для сохранения: ";
                std::getline(std::cin, filename);

                if (filename.empty()) {
                    std::cout << "Имя файла не может быть пустым.\n";
                    break;
                }

                if (filename.find(".sav") == std::string::npos) {
                    filename += ".sav";
                }

                try {
                    game.saveGame(filename);
                       std::cout << "Игра успешно сохранена в файл: " <<
filename << "\n";
                } catch (const std::exception& e) {
                      std::cerr << "Ошибка при сохранении: " << e.what()
<< "\n";
```

```
                }
                break;
            }

            case Command::LoadGame:
            {
                std::string filename;
                std::cout << "Введите имя файла для загрузки: ";
                std::getline(std::cin, filename);

                if (filename.empty()) {
                    std::cout << "Имя файла не может быть пустым.\n";
                    break;
                }

                if (filename.find(".sav") == std::string::npos) {
                    filename += ".sav";
                }

                try {
                    game.loadGame(filename);
                      std::cout << "Игра успешно загружена из файла: " <<
filename << "\n";
                } catch (const std::exception& e) {
                      std::cerr << "Ошибка при загрузке: " << e.what() <<
"\n";
                }
                break;
            }

            case Command::DisplayFields:
            {
                observer_->onFieldUpdate();
                break;
            }

            case Command::Quit:
                game.setGameOver(true);
                std::cout << "Спасибо за игру!\n";
                break;

            default:
                std::cout << "Неизвестная команда\n";
                break;
        }
    } catch (const std::exception& e) {
          std::cerr << "Ошибка при выполнении команды: " << e.what() <<
"\n";
    }
}
```

3. game_display_impl.h

```
#ifndef GAME_DISPLAY_IMPL_H
#define GAME_DISPLAY_IMPL_H

#include "game_display.h"
#include "game.h"
#include "game_field.h"
#include "ability_manager.h"
```

```
template<typename Renderer>
GameDisplay<Renderer>::GameDisplay(std::shared_ptr<Game> game)
    : game_(game), renderer_(std::make_unique<Renderer>()) {
}

template<typename Renderer>
void GameDisplay<Renderer>::onFieldUpdate() {
    if (game_) {
        updateDisplay();
    }
}

template<typename Renderer>
void GameDisplay<Renderer>::onAbilityUsed() {
    if (game_) {
        renderer_->renderAbilityStatus(*game_->getUserAbilityManager());
    }
}

template<typename Renderer>
void GameDisplay<Renderer>::onGameOver() {
    if (game_) {
        renderer_->renderGameStatus(*game_);
    }
}

template<typename Renderer>
void GameDisplay<Renderer>::onShipDestroyed() {
    if (game_) {
        updateDisplay();
    }
}

template<typename Renderer>
void GameDisplay<Renderer>::renderStartMenu() {
    renderer_->renderStartMenu();
}

template<typename Renderer>
void  GameDisplay<Renderer>::renderShipPlacement(int  shipLength,  int
shipNumber) {
    renderer_->renderShipPlacement(shipLength, shipNumber);
}

template<typename Renderer>
void GameDisplay<Renderer>::refresh() {
    updateDisplay();
}

template<typename Renderer>
void GameDisplay<Renderer>::updateDisplay() {
    if (!game_) return;

    const GameField& userField = game_->getUserField();
    const GameField& computerField = game_->getComputerField();

    renderer_->renderUserField(userField);
    renderer_->renderComputerField(computerField);
    renderer_->renderAbilityStatus(*game_->getUserAbilityManager());
```

```
    renderer_->renderGameStatus(*game_);
}

#endif
```

4. game_display.h

```
#ifndef GAME_DISPLAY_H
#define GAME_DISPLAY_H

#include <memory>

class Game;
class GameField;
class AbilityManager;

class IGameObserver {
public:
    virtual ~IGameObserver() = default;
    virtual void onFieldUpdate() = 0;
    virtual void onAbilityUsed() = 0;
    virtual void onGameOver() = 0;
    virtual void onShipDestroyed() = 0;
      virtual void renderShipPlacement(int shipLength, int shipNumber) =
0;
    virtual void renderStartMenu() = 0;
};

class IGameRenderer {
public:
    virtual ~IGameRenderer() = default;
    virtual void renderUserField(const GameField& field) = 0;
    virtual void renderComputerField(const GameField& field) = 0;
    virtual void renderAbilityStatus(const AbilityManager& manager) = 0;
    virtual void renderGameStatus(const Game& game) = 0;
      virtual void renderShipPlacement(int shipLength, int shipNumber) =
0;
    virtual void renderStartMenu() = 0;
};

template<typename Renderer>
class GameDisplay : public IGameObserver {
public:
    explicit GameDisplay(std::shared_ptr<Game> game);
    void onFieldUpdate() override;
    void onAbilityUsed() override;
    void onGameOver() override;
    void onShipDestroyed() override;
    void renderShipPlacement(int shipLength, int shipNumber) override;
    void renderStartMenu() override;
    void refresh();

private:
    std::shared_ptr<Game> game_;
    std::unique_ptr<Renderer> renderer_;
    void updateDisplay();
};

#endif
```

4. terminal_input.cpp

```
#include "terminal_input.h"
#include <fstream>
#include <iostream>
#include <sstream>
#include <algorithm>

TerminalInputProcessor::TerminalInputProcessor(const std::string&
configFile) {
    loadDefaultKeymap();
    try {
        loadKeymap(configFile);
    } catch (const std::exception& e) {
            std::cerr << "Warning: Could not load keymap from file: " <<
e.what() << "\n";
        std::cerr << "Using default keymap instead.\n";
    }
    validateKeymap();
}

Command TerminalInputProcessor::getCommand() {
    char input;
    std::cout << "Введите команду: ";
    std::cin >> input;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    input = std::tolower(static_cast<unsigned char>(input));

    auto it = keymap.find(input);
    if (it != keymap.end()) {
        return it->second;
    }

    std::cout << "Неизвестная команда. Используйте:\n";
    for (const auto& [key, cmd] : keymap) {
        std::cout << key << " - " << commandToString(cmd) << "\n";
    }
    return Command::Invalid;
}

const std::map<char, Command>& TerminalInputProcessor::getKeymap() const
{
    return keymap;
}

std::string TerminalInputProcessor::getCommandDescription(char key)
const {
    auto it = keymap.find(key);
    if (it != keymap.end()) {
        return commandToString(it->second);
    }
    return "Неизвестная команда";
}

void TerminalInputProcessor::loadDefaultKeymap() {
    keymap = {
        {'a', Command::Attack},
        {'s', Command::UseAbility},
        {'v', Command::SaveGame},
```

```
        {'l', Command::LoadGame},
        {'d', Command::DisplayFields},
        {'q', Command::Quit}
    };
}

void TerminalInputProcessor::loadKeymap(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
            throw std::runtime_error("Could not open keymap file: " +
filename);
    }

    keymap.clear();
    std::string line;
    int lineNumber = 0;

    while (std::getline(file, line)) {
        lineNumber++;
        if (line.empty() || line[0] == '#') continue;

        line.erase(0, line.find_first_not_of(" \t"));
        line.erase(line.find_last_not_of(" \t") + 1);

        size_t pos = line.find('=');
        if (pos == std::string::npos) {
                    std::cerr << "Warning: Invalid format in line " <<
lineNumber << ": " << line << "\n";
            continue;
        }

        std::string keyStr = line.substr(0, pos);
        std::string commandStr = line.substr(pos + 1);

                    keyStr.erase(std::remove_if(keyStr.begin(),
keyStr.end(), ::isspace), keyStr.end());
                    commandStr.erase(std::remove_if(commandStr.begin(),
commandStr.end(), ::isspace), commandStr.end());

        if (keyStr.length() != 1) {
              std::cerr << "Warning: Invalid key in line " << lineNumber
<< ": " << keyStr << "\n";
            continue;
        }

        char key = std::tolower(static_cast<unsigned char>(keyStr[0]));
        Command cmd = stringToCommand(commandStr);

        if (cmd != Command::Invalid) {
            keymap[key] = cmd;
        } else {
                    std::cerr << "Warning: Invalid command in line " <<
lineNumber << ": " << commandStr << "\n";
        }
    }
}

void TerminalInputProcessor::validateKeymap() {
    std::map<Command, bool> commandCovered;
    for (const auto& [key, cmd] : keymap) {
```

```
        if (commandCovered[cmd]) {
             throw std::runtime_error("Duplicate command mapping detected
for command: " +
                                      commandToString(cmd));
        }
        commandCovered[cmd] = true;
    }

    std::vector<Command> requiredCommands = {
        Command::Attack,
        Command::UseAbility,
        Command::SaveGame,
        Command::LoadGame,
        Command::DisplayFields,
        Command::Quit
    };

    for (Command cmd : requiredCommands) {
        if (!commandCovered[cmd]) {
                 throw std::runtime_error("Missing required command: " +
commandToString(cmd));
        }
    }

    std::map<char, Command> keyUsage;
    for (const auto& [key, cmd] : keymap) {
        if (keyUsage.find(key) != keyUsage.end()) {
             throw std::runtime_error("Multiple commands mapped to key: "
+ std::string(1, key));
        }
        keyUsage[key] = cmd;
    }
}

Command TerminalInputProcessor::stringToCommand(const std::string& str)
{
    std::string upperStr = str;
          std::transform(upperStr.begin(), upperStr.end(),
upperStr.begin(), ::toupper);

    if (upperStr == "ATTACK") return Command::Attack;
    if (upperStr == "ABILITY") return Command::UseAbility;
    if (upperStr == "SAVE") return Command::SaveGame;
    if (upperStr == "LOAD") return Command::LoadGame;
    if (upperStr == "DISPLAY") return Command::DisplayFields;
    if (upperStr == "QUIT") return Command::Quit;
    return Command::Invalid;
}

std::string TerminalInputProcessor::commandToString(Command cmd) const {
    switch (cmd) {
        case Command::Attack:
            return "attack";
        case Command::UseAbility:
            return "ability";
        case Command::SaveGame:
            return "save";
        case Command::LoadGame:
            return "load";
        case Command::DisplayFields:
```

```
            return "display";
        case Command::Quit:
            return "quit";
        default:
            return "INVALID";
    }
}
```

5. terminal_input.h

```
#ifndef TERMINAL_INPUT_H
#define TERMINAL_INPUT_H

#include <map>
#include <string>
#include "game_controller.h"

class TerminalInputProcessor {
public:
        explicit TerminalInputProcessor(const std::string& configFile =
"keymap.txt");
    Command getCommand();
    const std::map<char, Command>& getKeymap() const;
    std::string getCommandDescription(char key) const;

private:
    std::map<char, Command> keymap;
    std::map<Command, bool> usedCommands;

    void loadDefaultKeymap();
    void loadKeymap(const std::string& filename);
    void validateKeymap();
    Command stringToCommand(const std::string& str);
    std::string commandToString(Command cmd) const;
};

#endif
```

6. terminal_render.cpp

```
#include "terminal_renderer.h"
#include <iostream>
#include <iomanip>

void TerminalRenderer::renderUserField(const GameField& field) {
    std::cout << "\nВаше поле:\n";
    renderField(field, false);
}

void TerminalRenderer::renderComputerField(const GameField& field) {
    std::cout << "\nПоле компьютера:\n";
    renderField(field, true);
}

void TerminalRenderer::renderAbilityStatus(const AbilityManager&
manager) {
    std::cout << "\nДоступные способности:\n";
    if (manager.hasAbilities()) {
                    std::cout << "Следующая способность: " <<
manager.getFirstAbilityName() << "\n";
```

```
    } else {
        std::cout << "У вас нет доступных способностей\n";
    }
}

void TerminalRenderer::renderGameStatus(const Game& /* game */) {
    renderHeader();
    renderInstructions();
}

void TerminalRenderer::renderStartMenu() {
    std::cout << "\n=== Морской бой ===\n";
    std::cout << "1. Новая игра\n";
    std::cout << "2. Загрузить игру\n";
    std::cout << "3. Выход\n";
    std::cout << "\nВыберите действие (1-3): ";
}

void TerminalRenderer::renderShipPlacement(int shipLength, int
shipNumber) {
     std::cout << "Размещение корабля " << shipNumber << " (Длина: " <<
shipLength << ")\n";
}

void TerminalRenderer::renderAttackResult(int x, int y, bool isHit, bool
isSunk, bool isComputerAttack) {
    std::string attacker = isComputerAttack ? "Компьютер" : "Вы";
    std::cout << attacker << " атакует клетку (" << x << ", " << y << ")
\n";

    if (isHit) {
          std::cout << attacker << " попал" << (isComputerAttack ? "" :
"и") << " в корабль!\n";
        if (isSunk) {
             std::cout << attacker << " уничтожил" << (isComputerAttack ?
"" : "и") << " корабль!\n";
        }
    } else {
          std::cout << attacker << " промахнул" << (isComputerAttack ?
"ся" : "ись") << "!\n";
    }
}

void TerminalRenderer::renderField(const GameField& field, bool
isComputer) const {
    std::cout << "   ";
    for (int x = 0; x < GameField::DEFAULT_WIDTH; ++x) {
        std::cout << x << " ";
    }
    std::cout << "\n";

    for (int y = 0; y < GameField::DEFAULT_HEIGHT; ++y) {
        std::cout << std::setw(2) << y << " ";
        for (int x = 0; x < GameField::DEFAULT_WIDTH; ++x) {
            char symbol = getFieldSymbol(field, x, y, isComputer);
            std::cout << symbol << " ";
        }
        std::cout << "\n";
    }
    std::cout << "\n";
```

```
}

char TerminalRenderer::getFieldSymbol(const GameField& field, int x, int
y, bool isComputer) const {
    try {
        CellStatus status = field.getCellStatus(x, y);

        if (isComputer) {
            switch (status) {
                case CellStatus::Unknown:
                case CellStatus::Empty:
                    return '.';
                case CellStatus::Miss:
                    return 'O';
                case CellStatus::Ship: {
                    Ship* ship = field.getShipAt(x, y);
                    if (!ship) return '.';

                    int segmentIndex = field.getSegmentIndexAt(x, y);
                                    SegmentStatus segmentStatus = ship-
>getSegmentStatus(segmentIndex);

                    switch (segmentStatus) {
                        case SegmentStatus::Intact:
                            return '.';
                        case SegmentStatus::Damaged:
                            return 'D';
                        case SegmentStatus::Destroyed:
                            return 'X';
                        default:
                            return '.';
                    }
                }
                default:
                    return '.';
            }
        } else {
            switch (status) {
                case CellStatus::Unknown:
                    return '?';
                case CellStatus::Empty:
                    return '.';
                case CellStatus::Miss:
                    return 'O';
                case CellStatus::Ship: {
                    Ship* ship = field.getShipAt(x, y);
                    if (!ship) return 'S';

                    int segmentIndex = field.getSegmentIndexAt(x, y);
                                    SegmentStatus segmentStatus = ship-
>getSegmentStatus(segmentIndex);

                    switch (segmentStatus) {
                        case SegmentStatus::Intact:
                            return 'S';
                        case SegmentStatus::Damaged:
                            return 'D';
                        case SegmentStatus::Destroyed:
                            return 'X';
                        default:
```

```
                            return 'S';
                        }
                    }
                    default:
                        return '?';
                }
            }
        } catch (const std::exception& e) {
            std::cerr << "Ошибка при получении символа поля: " << e.what()
<< "\n";
            return '?';
        }
}

void TerminalRenderer::renderHeader() const {
    std::cout << "\n=========================\n";
    std::cout << "     МОРСКОЙ БОЙ";
    std::cout << "\n=========================\n";
}

void TerminalRenderer::renderInstructions() const {
    std::cout << "\nОбозначения:\n";
    std::cout << ". - пустая клетка\n";
    std::cout << "O - промах\n";
    std::cout << "S - корабль\n";
    std::cout << "D - поврежденный корабль\n";
    std::cout << "X - уничтоженный корабль\n\n";
}
```

7. terminal_render.h

```
#ifndef TERMINAL_RENDERER_H
#define TERMINAL_RENDERER_H

#include "game_display.h"
#include "game_field.h"
#include "ability_manager.h"

class TerminalRenderer : public IGameRenderer {
public:
    void renderUserField(const GameField& field) override;
    void renderComputerField(const GameField& field) override;
    void renderAbilityStatus(const AbilityManager& manager) override;
    void renderGameStatus(const Game& game) override;
    void renderStartMenu() override;
    void renderShipPlacement(int shipLength, int shipNumber) override;
     void renderAttackResult(int x, int y, bool isHit, bool isSunk, bool
isComputerAttack);

private:
    void renderField(const GameField& field, bool isComputer) const;
       char getFieldSymbol(const GameField& field, int x, int y, bool
isComputer) const;
    void renderHeader() const;
    void renderInstructions() const;
};

#endif
```