**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. 3343 |  | Калиберов Н. И. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

# Цель работы

Изучить работу шаблонов, путём усовершенствования программы из предыдущей лабораторной работы. Необходимо создать: шаблонный класс управления игрой, шаблонный класс отображения игры (наблюдатель), класс считывания ввода из терминала и класс отрисовки.

# Задание

1. Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.
2. Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.
3. Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.
4. Реализовать класс, отвечающий за отрисовку поля.

**Примечание:**

* Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
* После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
* Для представления команды можно разработать системы классов или использовать перечисление enum.
* Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
* При считывании управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

## Выполнение работы



Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Game, MapRenderer, UserActions, GameRenderer, GameController, и GameUI*. Эти классы реализуют основные функции игры "Морской бой", включая управление игрой, обработку пользовательского ввода, отрисовку игрового поля, а также управление действиями игрока и бота.

Класс *Game*:

* + **Роль**: Основной класс игры. Он отвечает за инициализацию игрового процесса, обработку ходов игрока и бота, а также за сохранение и загрузку игры.
  + **Поля**:
    - *BattleMap playerMap и BattleMap botMap* – игровые поля игрока и бота.
    - WhoMove who\_move – перечисление, определяющее, чей ход (игрока или бота).
    - *bool isGamePlayed* – флаг, указывающий, идет ли игра.
  + **Методы**:
    - *void runGame()* – запускает игру, вызывая меню и основной игровой цикл.
    - *void game\_loop()* – реализует основной игровой цикл, обрабатывая ходы игрока и бота.
    - *void call\_action(UserActions::UserAction\*)* – вызывает действие, выбранное игроком.

Класс *MapRenderer:*

* + **Роль**: Отвечает за отрисовку игровых полей игрока и бота.
  + **Поля**:
    - *const BattleMap& playerMap и const BattleMap& botMap* – ссылки на игровые поля игрока и бота.
  + **Методы**:
    - *void render() const* – отрисовывает игровые поля.
    - *char getDisplayChar(BattleMap::CellStatus status, bool hideShips = false) const* – возвращает символ, соответствующий статусу ячейки на поле.

Класс *UserActions*:

* + **Роль**: Обрабатывает пользовательский ввод и преобразует его в действия, такие как атака, сохранение игры, загрузка игры, использование способностей и т.д.
  + **Классы**:
    - *UserAction* – базовый класс для всех действий.
    - *NewGameAction, ExitGameAction, SaveGameAction, LoadGameAction, AttackAction, AbilityAction* – конкретные действия, которые могут быть выполнены игроком.
  + **Методы**:
    - *void read()* – считывает ввод от пользователя для выполнения действия.
    - *void print()* – выводит описание действия.
    - *UserAction\* menu(std::map<char, UserAction\*>& actions)* – отображает меню действий и возвращает выбранное действие.

Класс *GameRender*:

* + **Роль**: Шаблонный класс для отслеживания изменений в *playerMap* и *botMap*.
  + **Поля**:
    - *const BattleMap& playerMap* и *const BattleMap& botMap* – ссылки на игровые поля игрока и бота.
    - *RenderStrategy renderStrategy* – стратегия отрисовки.
  + **Методы**:
    - *void render() const* – отрисовывает игровые поля с использованием *MapRenderer*.

Класс *GameUI*:

* + **Роль**: Класс, отвечающий за пользовательский интерфейс и взаимодействие с игроком. Он предоставляет методы для отображения меню, вывода сообщений и обработки пользовательского ввода.
  + **Поля**:
    - *Game\* game* – указатель на объект игры, который позволяет взаимодействовать с игровым процессом.
  + **Методы**:
    - *void print\_main\_menu()* – отображает главное меню игры, где игрок может выбрать действия, такие как начало новой игры, загрузка игры, сохранение игры или выход.
    - *void print\_inner\_menu()* – отображает внутреннее меню во время игры, где игрок может выбрать действия, такие как использование способностей, сохранение игры, загрузка игры или выход.
    - *void print\_message(const std::string& msg)* – выводит сообщение в консоль. Этот метод может быть использован для отображения информации или диалоговых сообщений.
    - *void print\_sstream(std::istream& sstr)* – выводит содержимое потока ввода в консоль.

Класс *GameController:*

* **Роль**: Шаблонный класс управления игрой, который управляет обработкой пользовательского ввода и передачей действий в основной класс игры.
* **Поля**:
* *Game\* game* – указатель на объект игры.
* *InputController\* inputController* – указатель на контроллер ввода, который обрабатывает действия пользователя.
* **Методы**:
  + *void processInput()* – обрабатывает ввод пользователя и передает соответствующее действие в игру.

# Тестирование:

Происходит симуляция игры между игроком (слева) и ботом (справа), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. В начале хода игрок может использовать одну случайную способность или сразу перейти к атаке вражеского поля.

В классе *GameController* реализована логика игры, которая позволяет выбирать действия в зависимости от команд пользователя и вызывать методы *Game*. Класс управления игрой с помощью команд может: провести обычную атаку, использовать способность и атаковать, загрузить игру, получив состояния кораблей, поля и способностей; сохранить игру, уже записав состояния игровых сущностей; выйти из игры.

При победе игрок продолжает игру с сохранением его поля и с новым противником. В случае победы бота, игру можно продолжить, обнулив вообще всё.

# 

Рисунок 2 – Начало игры

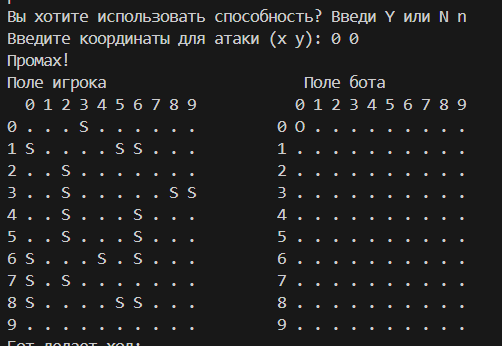


Рисунок 3 – Выполнение атаки

# 

Рисунок 4 – Игра сохранена и продолжается.

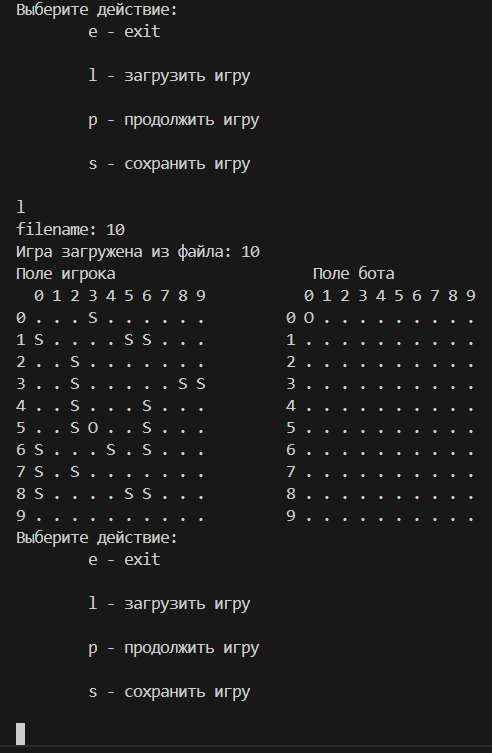
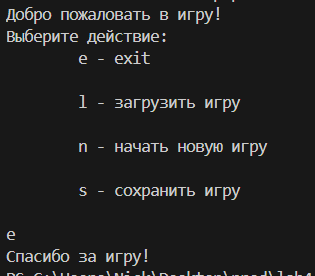


Рисунок 5 – Игра загружена



# Рисунок 6 – Выход из игры, до её начала

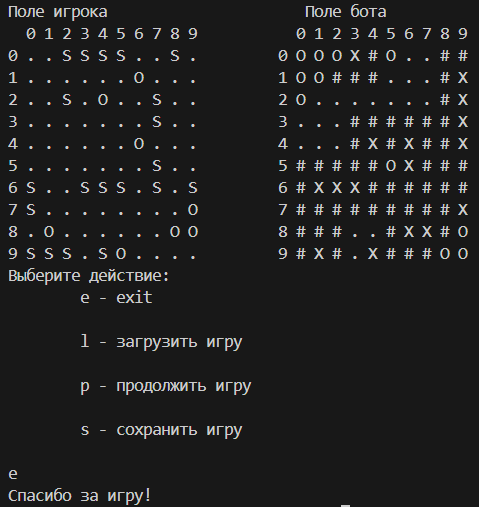
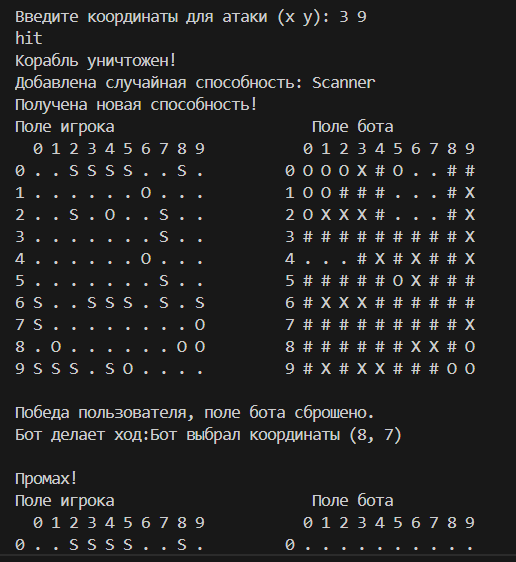


Рисунок 7 – Выход из игры, в процессе игры



# Рисунок 8 – Победа пользователя

# Выводы

Во время выполнения лабораторной работы, была изучена работа шаблонных классов и созданы соответствующие заданию классы.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Game.cpp

#include "Game.h"

#include <cstdlib>

#include <ctime>

#include "AbilityManager.h"

#include "GameState.h"

#include "useraction.h"

#include "GameRender.h"

#include "MapRender.h"

#include <sstream>

Game::Game()

    : playerMap(10, 10),

      playerShipManager(),

      botMap(10, 10),

      botShipManager(),

      bot(botMap, botShipManager),

      who\_move(WhoMove::Player),

    isGamePlayed(true), ui(this) {

    srand(static\_cast<unsigned int>(time(0)));

    playerShipManager.placeShipsRandomly(playerMap);

    botShipManager.placeShipsRandomly(botMap);

}

void Game::runGame() {

    ui.print\_main\_menu();

}

void Game::game\_loop(){

    while (isGamePlayed) {

        displayMap();

        if (playerMap.allShipsDestroyed()) {

            ui.print\_message("О нет, бот победил! Начинаем новую игру...");

            playerMap = BattleMap(10, 10);

            playerShipManager = ShipManager();

            playerShipManager.placeShipsRandomly(playerMap);

        }

        if (botMap.allShipsDestroyed()) {

            botMap = BattleMap(10, 10);

            botShipManager = ShipManager();

            botShipManager.placeShipsRandomly(botMap);

            who\_move = WhoMove::Bot;

        }

        if (who\_move == WhoMove::Bot) {

            botTurn();

            continue;

        }

        ui.print\_inner\_menu();

    }

}

void Game::call\_action(UserActions::UserAction\* action) {

    UserActions::ExitGameAction\* exit\_action = dynamic\_cast<UserActions::ExitGameAction\*>(action);

    UserActions::LoadGameAction\* load\_action = dynamic\_cast<UserActions::LoadGameAction\*>(action);

    UserActions::SaveGameAction\* save\_action = dynamic\_cast<UserActions::SaveGameAction\*>(action);

    UserActions::NewGameAction\* newgame\_action = dynamic\_cast<UserActions::NewGameAction\*>(action);

    UserActions::AbilityAction\* ability\_action = dynamic\_cast<UserActions::AbilityAction\*>(action);

    UserActions::AttackAction\* attack\_action = dynamic\_cast<UserActions::AttackAction\*>(action);

    if (exit\_action) {

        isGamePlayed = false;

    }

    if (load\_action) {

        loadGame(load\_action->path);

    }

    if (save\_action) {

        saveGame(save\_action->path);

    }

    if (newgame\_action) {

        who\_move = WhoMove::Player;

        playerMap = BattleMap(10, 10);

        playerShipManager = ShipManager();

        botMap = BattleMap(10, 10);

        botShipManager = ShipManager();

        playerShipManager.placeShipsRandomly(playerMap);

        botShipManager.placeShipsRandomly(botMap);

    }

    if (ability\_action) {

        if (ability\_action->is\_active == true) {

            playerMap.abilityManager->applyAbility(playerMap, botMap, playerShipManager, botShipManager);

        }

        delete action;

        auto secondAction = new UserActions::AttackAction(this);

        secondAction->read();

        call\_action(secondAction);

    }

    if (attack\_action) {

        if (false == botMap.shoot(attack\_action->x, attack\_action->y)) {

            who\_move = WhoMove::Bot;

        }

        else {

            if (botMap.isShipDestroyed(attack\_action->x, attack\_action->y)) {

                ui.print\_message("Корабль уничтожен!");

                if (playerMap.abilityManager) { //TODO: а менеджера может не быть?

                    playerMap.abilityManager->assignRandomAbility();

                    ui.print\_message("Получена новая способность!");

                }

            }

        }

    }

    if (action)

        delete action;

    game\_loop();

}

Ability\* Game::first\_userAbility() {

    return playerMap.abilityManager->first\_ability();

}

void Game::displayMap() {

    GameRenderer<MapRenderer> renderer(playerMap, botMap);

    stringstream messages;

    renderer.render(messages);

    ui.print\_sstream(messages);

}

void Game::botTurn() {

    stringstream messages;

    auto [x, y] = bot.makeMove();

    messages << "Бот делает ход:"

             << "Бот выбрал координаты (" << x << ", " << y << ")" << std::endl;

    ui.print\_sstream(messages);

    if (!playerMap.shoot(x, y)) {

        who\_move = WhoMove::Player;

    }

}

// Сохранение состояния игры

void Game::saveGame(const std::string& filename) {

    GameState gameState(playerMap, botMap, playerShipManager, botShipManager, bot);

    gameState.saveGame(filename);

    stringstream messages;

    messages << "Игра сохранена в файл: " << filename << std::endl;

    ui.print\_sstream(messages);

}

// Загрузка состояния игры

void Game::loadGame(const std::string& filename) {

    try {

        GameState gameState(playerMap, botMap, playerShipManager, botShipManager, bot);

        gameState.loadGame(filename);

        playerMap = gameState.playerMap;

        playerShipManager = gameState.playerShipManager;

        botMap = gameState.botMap;

        botShipManager = gameState.botShipManager;

        stringstream messages;

        messages << "Игра загружена из файла: " << filename << std::endl;

        ui.print\_sstream(messages);

    }

    catch(...) {

    }

}

Название файла: Game.h

#ifndef GAME\_H

#define GAME\_H

#include "BattleMap.h"

#include "ShipManager.h"

#include "Bot.h"

#include "useraction.h"

#include "GameRender.h"

#include "MapRender.h"

#include "gameui.h"

class Ability;

class Game {

public:

    Game();

    void runGame();

    void saveGame(const std::string& filename); // Метод сохранения игры

    void loadGame(const std::string& filename); // Метод загрузки игры

    Ability \*first\_userAbility();

private:

    void botTurn();

    void displayMap();

    void handleAbility();

    void call\_action(UserActions::UserAction\*);

    void game\_loop();

    BattleMap playerMap;

    ShipManager playerShipManager;

    BattleMap botMap;

    ShipManager botShipManager;

    Bot bot;

    enum class WhoMove { Player, Bot };

    WhoMove who\_move;

    bool isGamePlayed;

    friend class GameUI;

    GameUI ui;

};

#endif

Название файла: useraction.cpp

#include "useraction.h"

#include <iostream>

#include <limits>

#include <algorithm>

#include <ios>

#include "Check.h"

#include "Ability.h"

#include "Game.h"

void UserActions::clear\_input() {

    std::cin.clear();

    if (std::cin.peek() != std::ios::traits\_type::eof())

        std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

void print\_actions(std::map<char, UserActions::UserAction\*>& actions) {

    std::cout << "Выберите действие:" << std::endl;

    for (auto& action : actions) {

        std::cout << "\t" << action.first << " - ";

        action.second->print();

        std::cout << "\n";

    }

}

std::string current\_actions\_symbols(std::map<char, UserActions::UserAction\*>& actions) {

    std::string symbols;

    for (auto& action : actions) {

        symbols += action.first;

    }

    return symbols;

}

void UserActions::NewGameAction::print() {

    std::cout << "начать новую игру" << std::endl;

}

void UserActions::ExitGameAction::print() {

    std::cout << "exit" << std::endl;

}

void UserActions::SaveGameAction::read() {

    std::cout << "filename: ";

    while (true) {

        std::getline(std::cin, path);

        if (path.empty() == false)

            break;

    }

}

void UserActions::SaveGameAction::print() {

    std::cout << "сохранить игру" << std::endl;

}

void UserActions::LoadGameAction::read() {

    std::cout << "filename: ";

    while (true) {

        std::getline(std::cin, path);

        if (path.empty() == false)

            break;

    }

}

void UserActions::LoadGameAction::print() {

    std::cout << "загрузить игру" << std::endl;

}

void UserActions::AttackAction::read() {

    while (true) {

        std::cout << "Введите координаты для атаки (x y): ";

        std::cin >> x >> y;

        if (false == Check::checkAttackBounds(x, y, 10, 10)) { //TODO: magic numbers

            std::cout << "плохие координаты\n";

            continue;

        }

        if (false == std::cin.fail() && std::cin.peek() == '\n') {

            return;

        }

        clear\_input();

    }

}

void UserActions::AbilityAction::read() {

    Ability\* ability = this->game->first\_userAbility();

    is\_active = false;

    if (nullptr == ability) {

        return;

    }

    std::cout << "Вы хотите использовать способность? Введи Y или N ";

    if (tolower(read\_symbol\_of("YyNn")) != 'y') {

        return;

    }

    cout << "Доступная способность: " << ability->getName() << "\n";

    std::cout << "Точно будем использовать? Введи Y или N ";

    is\_active = tolower(read\_symbol\_of("YyNn")) == 'y';

}

void UserActions::AbilityAction::print() {

    std::cout << "продолжить игру" << std::endl;

}

char UserActions::read\_symbol\_of(const std::string& symbols) {

    char c;

    while (true) {

        std::cin >> c;

        if (std::cin.peek() != '\n') {

            continue;

        }

        if (std::find(symbols.begin(), symbols.end(), c) != symbols.end()) {

            return c;

        }

        clear\_input();

    }

}

UserActions::UserAction\* UserActions::menu(std::map<char, UserAction\*>& actions) {

    print\_actions(actions);

    char choice = read\_symbol\_of(current\_actions\_symbols(actions));

    actions[choice]->read();

    return actions[choice];

}

Название файла: useraction.h

#ifndef USERACTION\_H

#define USERACTION\_H

#include <string>

#include <map>

class Game;

namespace UserActions {

void clear\_input();

struct UserAction {

    virtual ~UserAction() = default;

    virtual void read() {}

    virtual void print() {}

    UserAction(Game\* game\_) : game(game\_){

    }

    Game\* game;

};

struct NewGameAction : public UserAction {

    NewGameAction(Game\* game\_) : UserAction(game\_){}

    void print() override;

};

struct ExitGameAction : public UserAction {

    ExitGameAction(Game\* game\_) : UserAction(game\_){}

    void print() override;

};

struct SaveGameAction : public UserAction {

    std::string path;

    SaveGameAction(Game\* game\_) : UserAction(game\_){}

    void print() override;

    void read() override;

};

struct LoadGameAction : public UserAction {

    std::string path;

    LoadGameAction(Game\* game\_) : UserAction(game\_){}

    void read() override;

    void print() override;

};

struct AttackAction : public UserAction {

    int x, y;

    AttackAction(Game\* game\_) : UserAction(game\_){}

    void read() override;

};

struct AbilityAction : public UserAction {

    bool is\_active;

    AbilityAction(Game\* game\_) : UserAction(game\_){}

    void read() override;

    void print() override;

};

char read\_symbol\_of(const std::string& symbols);

UserAction\* menu(std::map<char, UserAction\*>& actions);

}

#endif // USERACTION\_H

Название файла: MapRender.cpp

#include "MapRender.h"

#include <ostream>

using namespace std;

MapRenderer::MapRenderer(const BattleMap& playerMap, const BattleMap& botMap)

    : playerMap(playerMap), botMap(botMap) {}

void MapRenderer::render(ostream &ost) const {

    ost << "Поле игрока" << string(22, ' ') << "Поле бота" << endl;

    ost << "  ";

    for (int col = 0; col < playerMap.getWidth(); ++col) {

        ost << col << " ";

    }

    ost << string(8, ' ');

    ost << "  ";

    for (int col = 0; col < botMap.getWidth(); ++col) {

        ost << col << " ";

    }

    ost << endl;

    for (int row = 0; row < playerMap.getHeight(); ++row) {

        ost << row << " ";

        for (BattleMap::CellStatus cell : playerMap.map[row]) {

            ost << getDisplayChar(cell) << " ";

        }

        ost << string(8, ' ');

        ost << row << " ";

        for (BattleMap::CellStatus cell : botMap.map[row]) {

            ost << getDisplayChar(cell, true) << " ";

        }

        ost << endl;

    }

}

char MapRenderer::getDisplayChar(BattleMap::CellStatus status, bool hideShips) const {

    switch (status) {

        case BattleMap::CellStatus::Unknown: return '.';

        case BattleMap::CellStatus::Empty: return ' ';

        case BattleMap::CellStatus::Ship: return hideShips ? '.' : 'S';

        case BattleMap::CellStatus::Hit: return 'X';

        case BattleMap::CellStatus::Miss: return 'O';

        case BattleMap::CellStatus::Destroyed: return '#';

        default: return '?';

    }

}

// красивая отрисовка

// MapRenderer::MapRenderer(const BattleMap& playerMap, const BattleMap& botMap)

//     : playerMap(playerMap), botMap(botMap) {}

// void MapRenderer::render() const {

//     // Заголовок

//     ost << "Поле игрока" << string(15, ' ') << "Поле бота" << endl;

//     // Верхняя строка с координатами столбцов

//     ost << "  ";

//     for (int col = 0; col < playerMap.getWidth(); ++col) {

//         ost << "| " << col << " ";

//     }

//     ost << "|" << string(8, ' ');

//     ost << "  ";

//     for (int col = 0; col < botMap.getWidth(); ++col) {

//         ost << "| " << col << " ";

//     }

//     ost << "|" << endl;

//     // Верхняя граница

//     ost << "  ";

//     for (int col = 0; col < playerMap.getWidth(); ++col) {

//         ost << "+---";

//     }

//     ost << "+" << string(8, ' ');

//     ost << "  ";

//     for (int col = 0; col < botMap.getWidth(); ++col) {

//         ost << "+---";

//     }

//     ost << "+" << endl;

//     // Основная часть карты

//     for (int row = 0; row < playerMap.getHeight(); ++row) {

//         // Левая граница

//         ost << row << " ";

//         // Содержимое карты игрока

//         for (BattleMap::CellStatus cell : playerMap.map[row]) {

//             ost << "| " << getDisplayChar(cell) << " ";

//         }

//         ost << "|" << string(8, ' ');

//         // Содержимое карты бота

//         ost << row << " ";

//         for (BattleMap::CellStatus cell : botMap.map[row]) {

//             ost << "| " << getDisplayChar(cell, true) << " ";

//         }

//         ost << "|" << endl;

//         // Нижняя граница

//         ost << "  ";

//         for (int col = 0; col < playerMap.getWidth(); ++col) {

//             ost << "+---";

//         }

//         ost << "+" << string(8, ' ');

//         ost << "  ";

//         for (int col = 0; col < botMap.getWidth(); ++col) {

//             ost << "+---";

//         }

//         ost << "+" << endl;

//     }

// }

// char MapRenderer::getDisplayChar(BattleMap::CellStatus status, bool hideShips) const {

//     switch (status) {

//         case BattleMap::CellStatus::Unknown: return '.';

//         case BattleMap::CellStatus::Empty: return ' ';

//         case BattleMap::CellStatus::Ship: return hideShips ? '.' : 'S';

//         case BattleMap::CellStatus::Hit: return 'X';

//         case BattleMap::CellStatus::Miss: return 'O';

//         case BattleMap::CellStatus::Destroyed: return '#';

//         default: return '?';

//     }

// }

Название файла: MapRender.h

#ifndef MAPRENDERER\_H

#define MAPRENDERER\_H

#include "BattleMap.h"

class MapRenderer {

public:

    MapRenderer(const BattleMap& playerMap, const BattleMap& botMap);

    void render(std::ostream& ost) const; // Метод для отрисовки игры

private:

    const BattleMap& playerMap;

    const BattleMap& botMap;

    char getDisplayChar(BattleMap::CellStatus status, bool hideShips = false) const;

};

#endif // MAPRENDERER\_H

Название файла: GameRender.h

#ifndef GAMERENDERER\_H

#define GAMERENDERER\_H

#include "BattleMap.h"

#include <ostream>

template <typename RenderStrategy>

class GameRenderer {

public:

    GameRenderer(const BattleMap& playerMap, const BattleMap& botMap)

        : playerMap(playerMap), botMap(botMap), renderStrategy(playerMap, botMap) {}

    void render(std::ostream& ost) const {

        renderStrategy.render(ost);

    }

private:

    const BattleMap& playerMap;

    const BattleMap& botMap;

    RenderStrategy renderStrategy;

};

#endif // GAMERENDERER\_H

Название файла: gameui.cpp

#include "gameui.h"

#include "Game.h"

GameUI::GameUI(Game\* game\_) : game(game\_) {}

void GameUI::print\_main\_menu() {

    std::map<char, UserActions::UserAction\*> actions = {

        {'e', new UserActions::ExitGameAction(game)},

        {'n', new UserActions::NewGameAction(game)},

        {'l', new UserActions::LoadGameAction(game)},

        {'s', new UserActions::SaveGameAction(game)}

    };

    UserActions::UserAction\* selected = menu(actions);

    game->call\_action(selected);

    game->game\_loop();

}

void GameUI::print\_inner\_menu() {

    std::map<char, UserActions::UserAction\*> actions = {

        {'e', new UserActions::ExitGameAction(game)},

        {'p', new UserActions::AbilityAction(game)},

        {'l', new UserActions::LoadGameAction(game)},

        {'s', new UserActions::SaveGameAction(game)}

    };

    UserActions::UserAction\* selected = menu(actions);

    game->call\_action(selected);

}

void GameUI::print\_message(const std::string& msg) {

    std::cout << msg << "\n";

}

void GameUI::print\_sstream(std::istream& sstr) {

    string str;

    while (sstr) {

        getline(sstr, str);

        cout << str << "\n";

    }

}

Название файла: gameui.h

#ifndef GAMEUI\_H

#define GAMEUI\_H

#include "useraction.h"

#include <ostream>

class Game;

class GameUI {

public:

    GameUI(Game\* game\_);

    void print\_main\_menu();

    void print\_inner\_menu();

    void print\_message(const std::string& msg); // мог бы быть вывод dialogBox

    void print\_sstream(std::istream& sstr);

    friend class Game;

    Game\* game;

};

#endif // GAMEUI\_H

Название файла: GameController.h

#ifndef GAMECONTROLLER\_H

#define GAMECONTROLLER\_H

#include "Game.h"

#include "useraction.h"

template <typename InputController>

class GameController {

public:

    GameController(Game\* game, InputController\* inputController)

        : game(game), inputController(inputController) {}

    void processInput() {

        auto action = inputController->UserAction();

        game->call\_action(action);

    }

private:

    Game\* game;

    InputController\* inputController;

};

#endif