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

a. 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.

b. 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.

c. 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.

d. 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

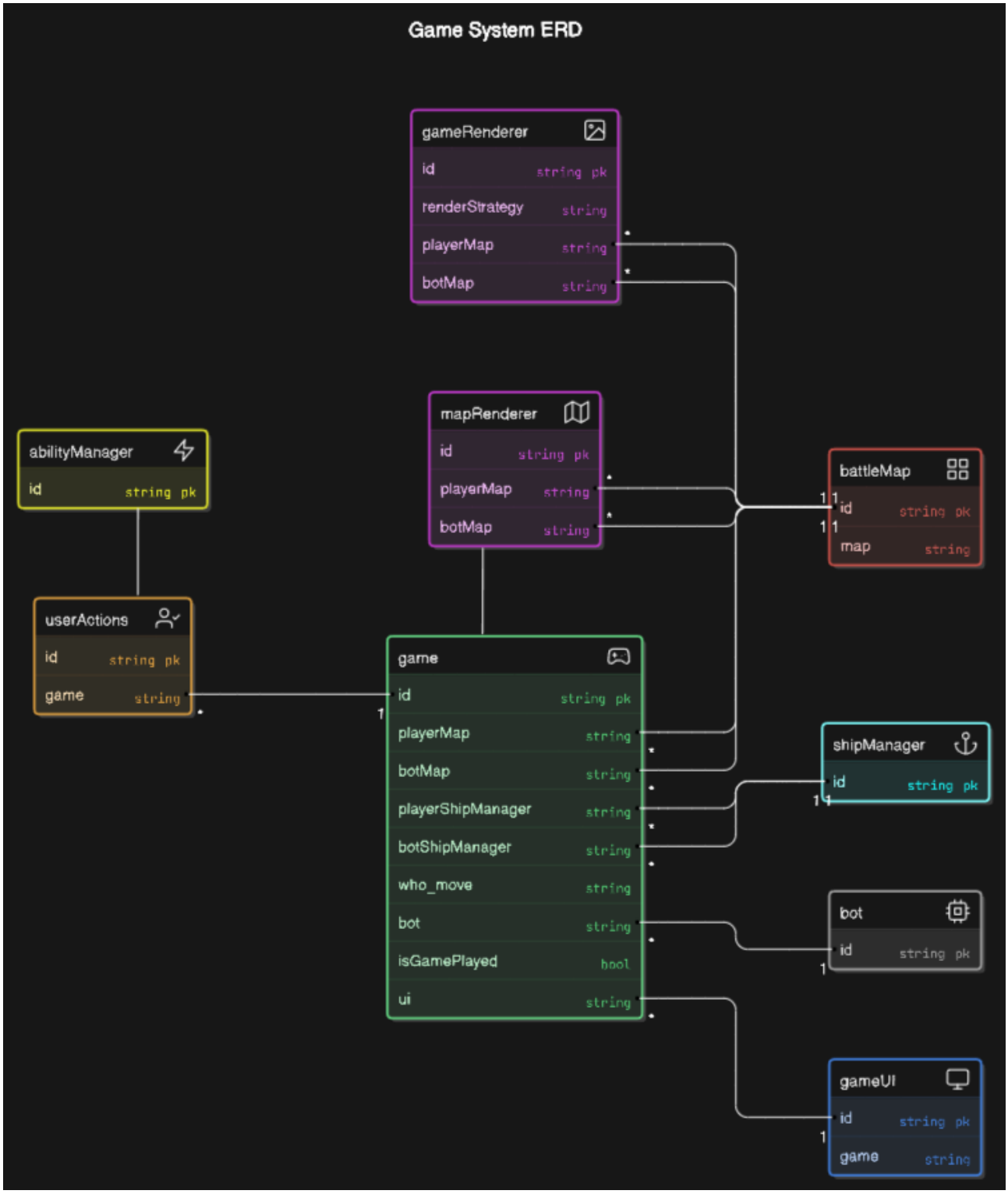

Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Game, MapRenderer, UserActions, GameRenderer*. 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 управления игрой. 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, LoadGame Action, 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.

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.

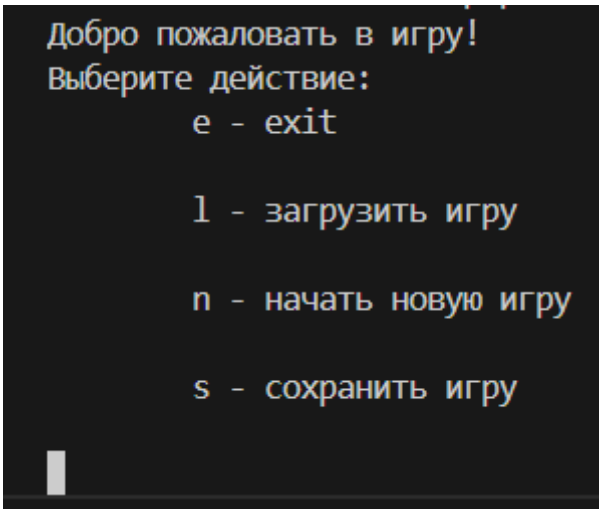

Рисунок 2 – Начало игры

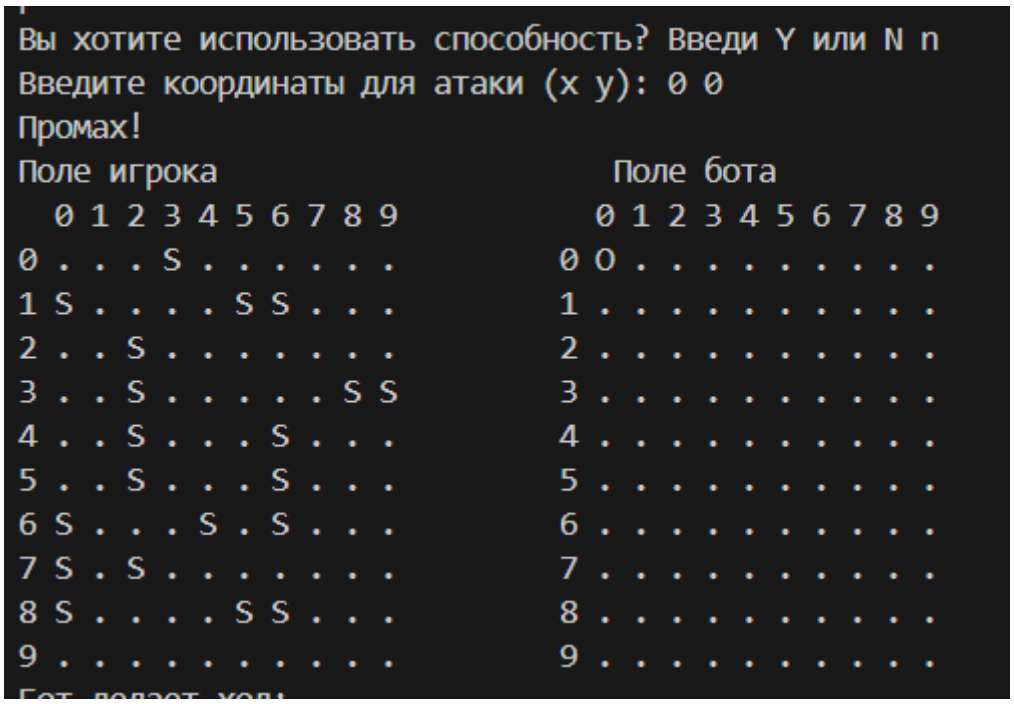

Рисунок 3 – Выполнение атаки

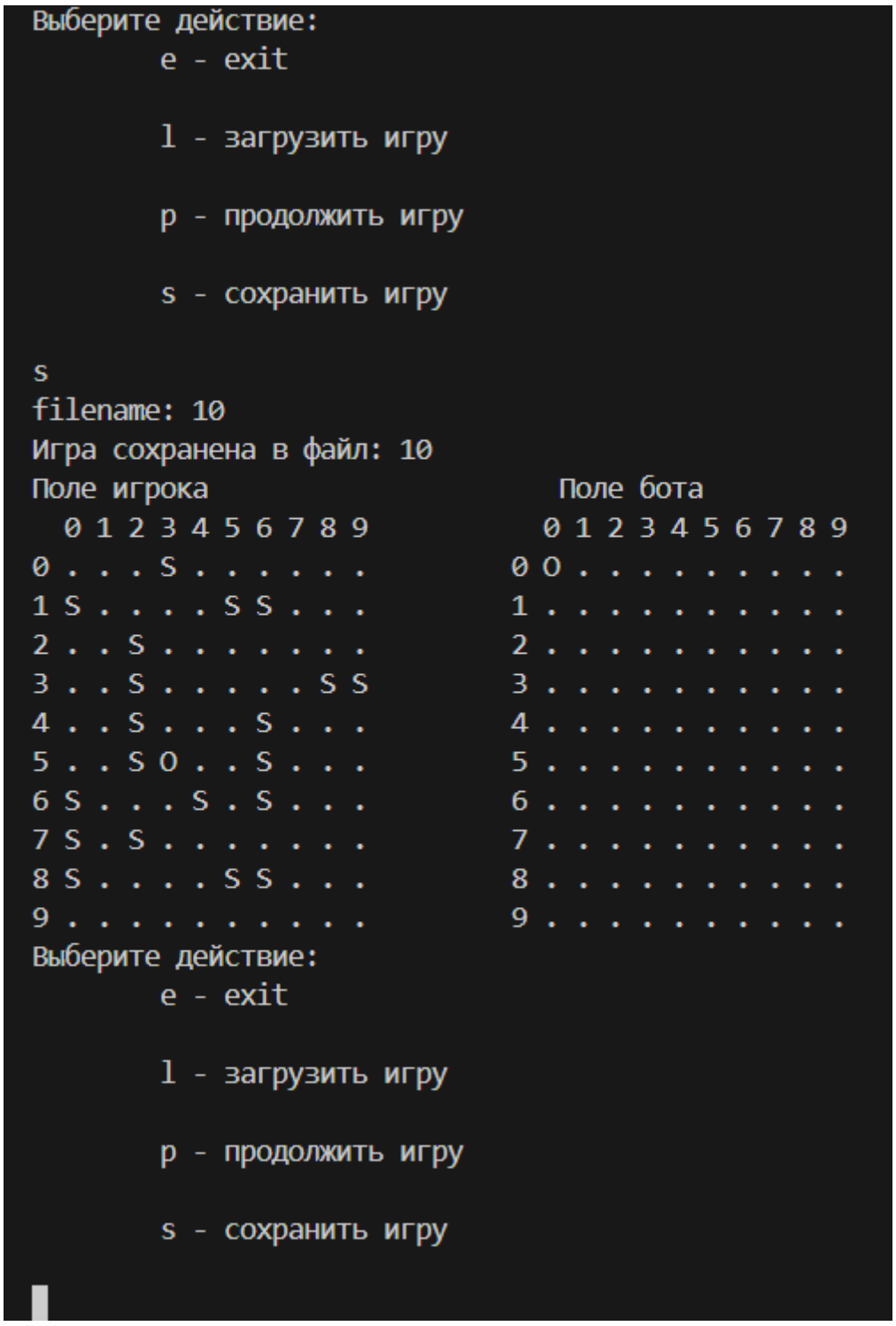

Рисунок 4 – Игра сохранена и продолжается.

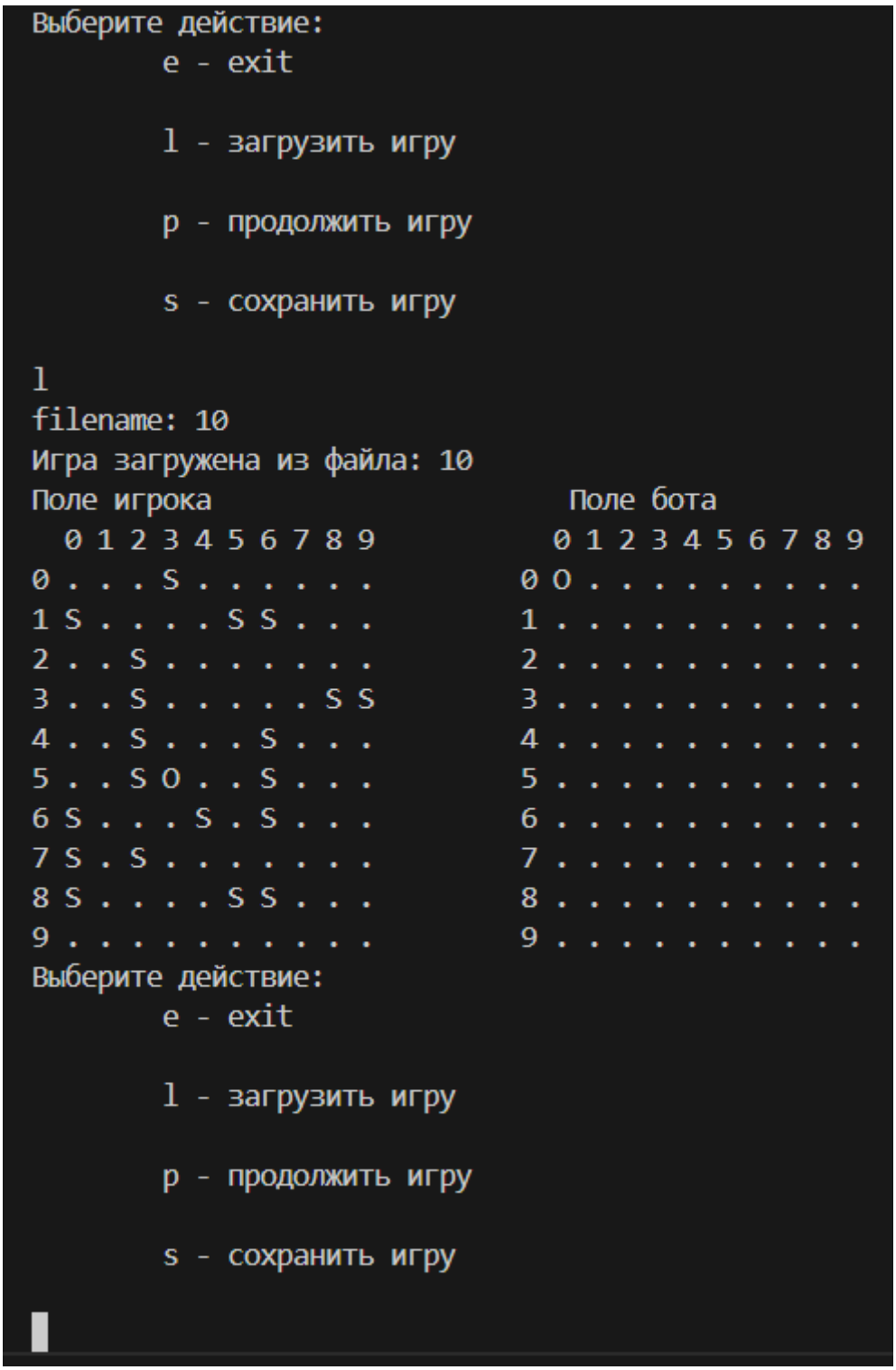

Рисунок 5 – Игра загружена

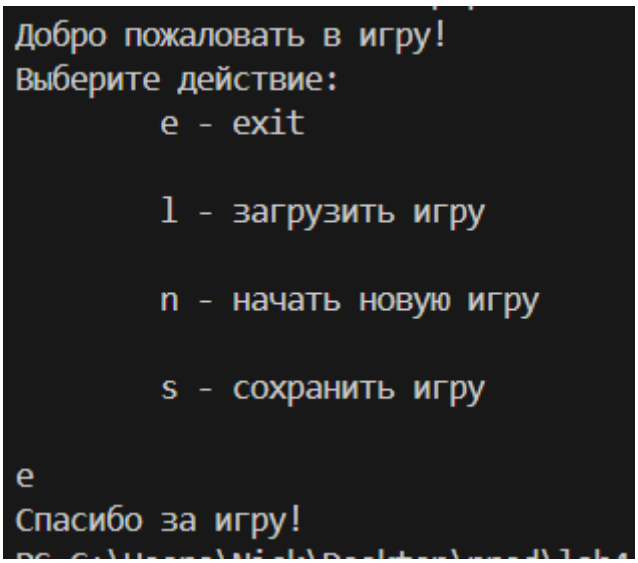

Рисунок 6 – Выход из игры, до её начала

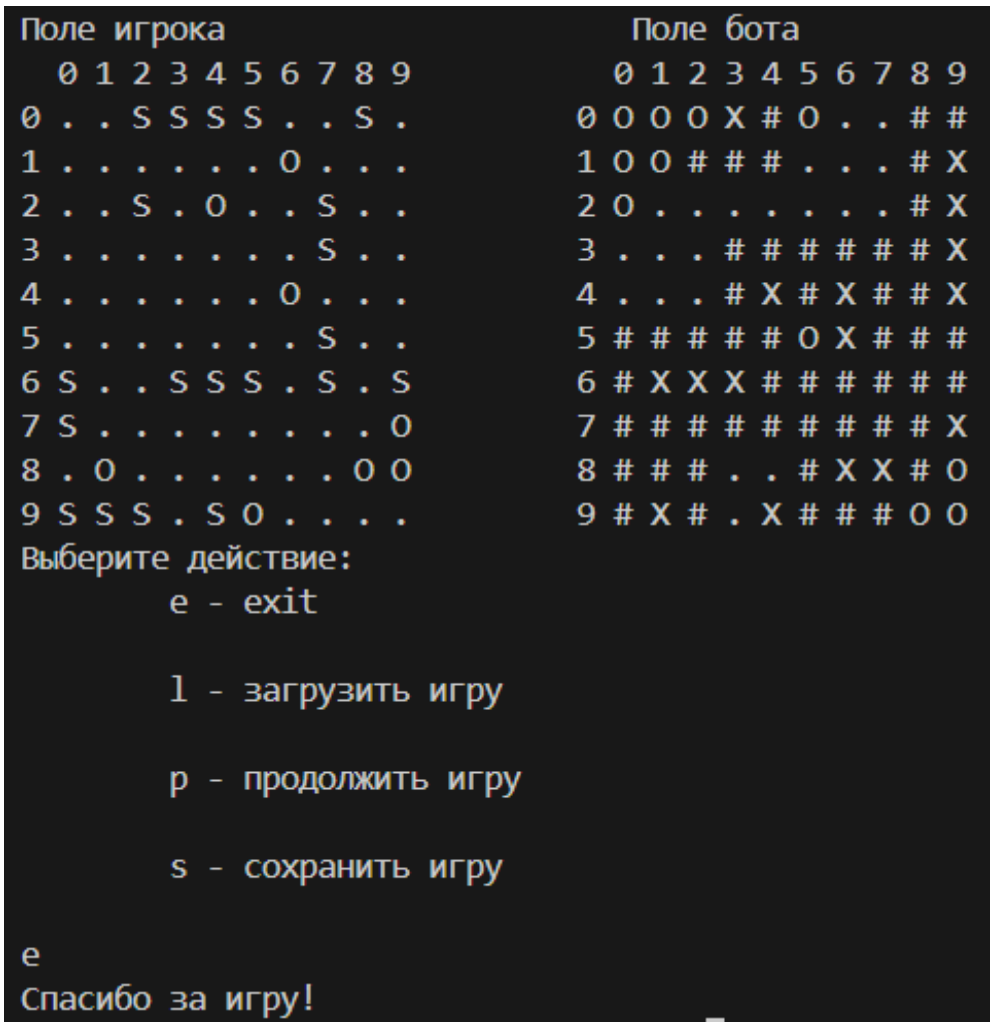

Рисунок 7 – Выход из игры, в процессе игры

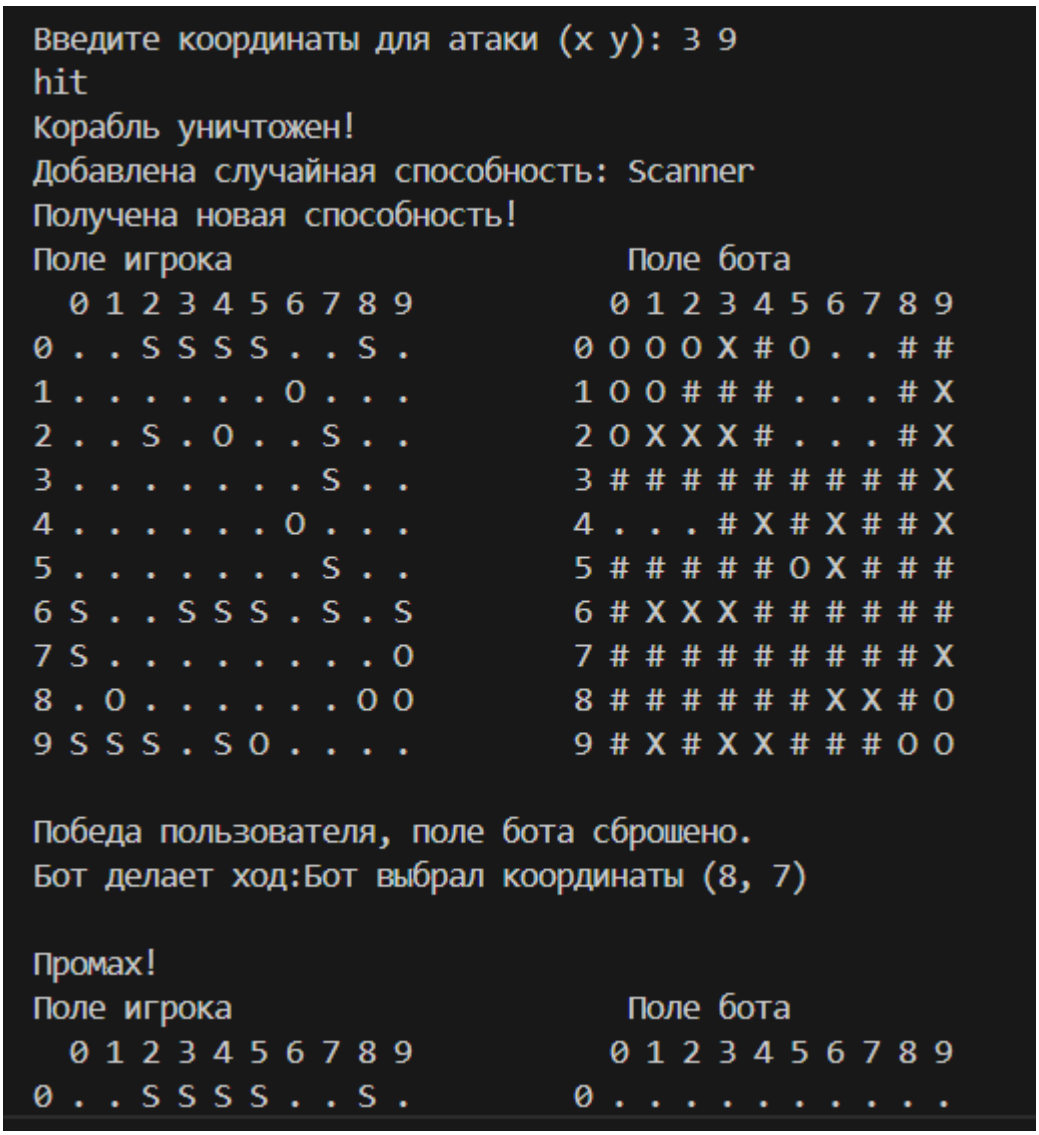

Рисунок 8 – Победа пользователя

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
```