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. 3342 _____ Галеев А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Создание шаблонного класса для управления игрой, так же создание класса определяющего способ ввода команды и переводящий введенную информацию в команду. Создать шаблонный класс отображения игры.

Задание

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.

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.

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.

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 считывания 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

Класс GameControl

GameControl — это шаблонный класс, предназначенный для управления процессом игры. Он объединяет различные компоненты игры, такие как ввод команд от игрока, вывод сообщений и отображение игровых полей. Этот класс отвечает за обработку ввода, управление игровым процессом и взаимодействие с объектом Game.

Шаблонные параметры:

• InputHandler — класс, обрабатывающий ввод игрока (например, команды, координаты для атаки).

• OutputHandler — класс для вывода сообщений пользователю.

• Drawer — класс для отображения игрового поля.

Поля класса:

• Game* game — указатель на объект игры, содержащий основную логику и данные.

• InputHandler* inputter — объект, обрабатывающий ввод пользователя.

• OutputHandler displayer — объект для вывода сообщений.

• bool isNewGame — флаг, указывающий, началась ли новая игра.

Конструктор и деструктор:

1. GameControl(Game* game) noexcept

Конструктор, инициализирующий объект GameControl. Создает объект для обработки ввода (inputter) и связывает объект игры (game).

2. ~GameControl() noexcept

Деструктор, освобождающий память, занятую объектом inputter.

Основные методы:

void initNewGame(bool& processGameFlag)

Инициализирует новую игру.

• Загружает команды из файла commands.json. Если файл недоступен, используются команды по умолчанию.

• Отображает доступные команды (START_NEW_GAME, LOAD, EXIT) и позволяет пользователю выбрать одну из них.

• Управляет состоянием игры на основе команды игрока.

void start()

Запускает игровой процесс.

• Инициализирует новую игру или загружает сохранение.

• Управляет игровыми раундами: игрок и противник поочередно совершают действия (атаки или использование способностей).

• Обрабатывает команды, вводимые игроком, такие как атака, использование способности, сохранение, загрузка или выход.

• Обеспечивает переход между раундами и завершение игры, если один из игроков победил.

void save()

Сохраняет текущее состояние игры, вызывая метод saveGame объекта Game.

void load()

Загружает сохраненное состояние игры, вызывая метод loadGame объекта Game.

Логика игрового процесса:

1. Команды для старта игры

Игрок выбирает, начать новую игру, загрузить сохранение или выйти.

2. Процесс хода

В каждом раунде игрок:

• Выбирает действие (например, атаковать, использовать способность).

• Вводит координаты для атаки или способности.

• Наблюдает за действиями противника, которые происходят автоматически.

3. Использование способностей

Игрок может использовать доступные способности.

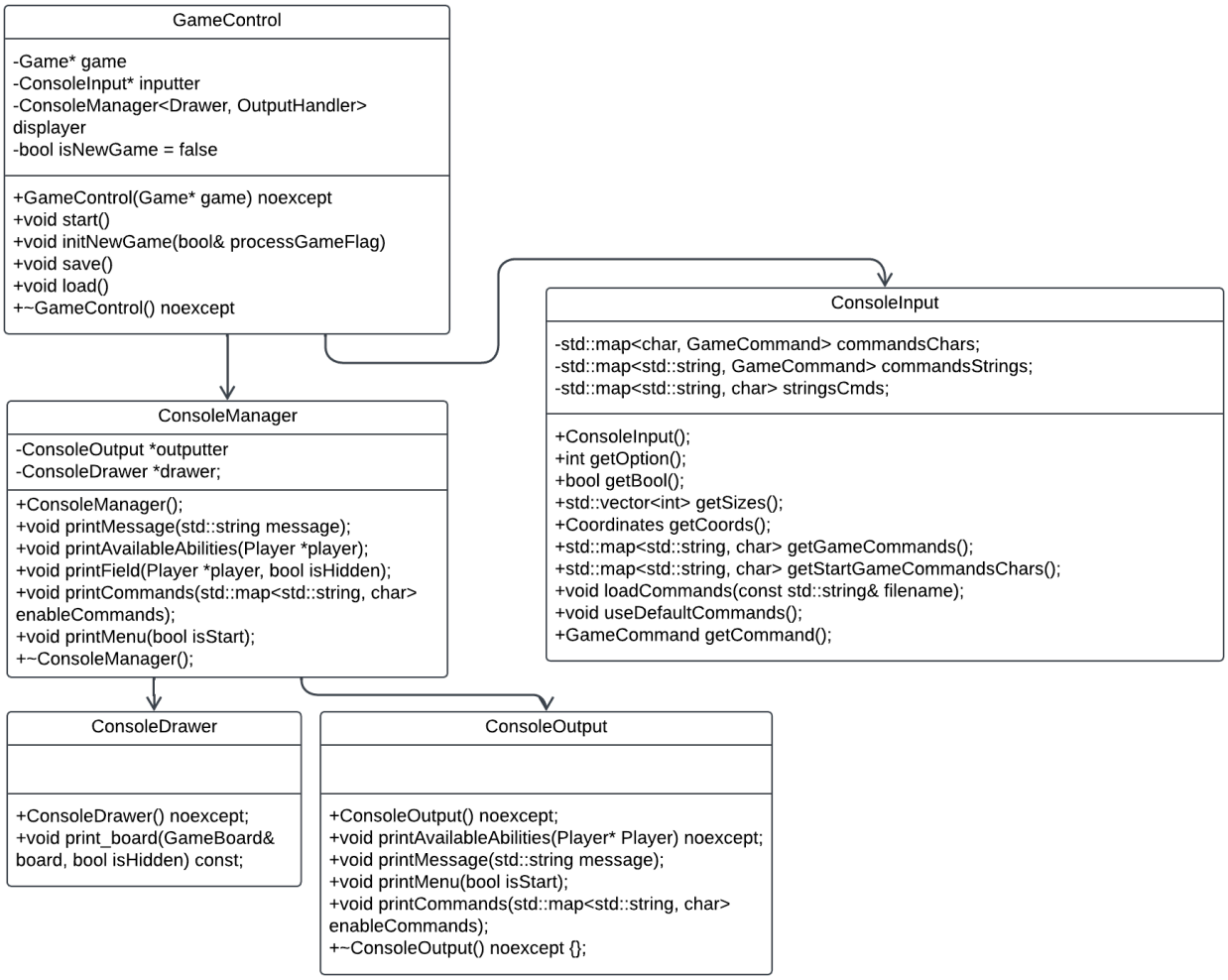
• Способности управляются через объект игрока.

4. Завершение раунда

После выполнения действий игрока и противника раунд заканчивается, и начинается следующий.

Класс GameControl служит посредником между игроком и основным объектом Game, обрабатывая пользовательский ввод, отображая игровую информацию и управляя игровым процессом.

UML-диаграмма классов:



Тестирование

Сборка через cmake:

```
amir@Noutbuk-Amir build % cmake ..
-- The C compiler identification is AppleClang 16.0.0.16000026
-- The CXX compiler identification is AppleClang 16.0.0.16000026
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.7s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/amir/Desktop/battle/build
amir@Noutbuk-Amir build %
```

Далее нас просят ввести все необходимые для игры данные. В main содержится только вызов меню.

```
Select option:
E - EXIT
L - LOAD
N - START_NEW_GAME
N

Enter height and width:
10 10

Enter sizes of ships (1 to 4 ships, each size between 1 and 4):
1 2

Enter coordinates (x y)
for ship of size 1 (Ship №1):
0 0

Enter orientation (0 for horizontal, 1 for vertical) for ship of size
for ship of size 1 (Ship №1):
0

Enter coordinates (x y)
for ship of size 2 (Ship №2):
0 2

Enter orientation (0 for horizontal, 1 for vertical) for ship of size
for ship of size 2 (Ship №2):
0
```

Выводы

Были созданы классы необходимые для управления игрой, выводом и вводом в консоль информации необходимой для игры в морской бой. Был создан класс управления игрой.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: GameController.h

```
#ifndef GAME_CONTROLLER_H
#define GAME_CONTROLLER_H

#include "Game.h"
#include "ConsoleInput.h"
#include "ConsoleManager.h"
#include "ConsoleDrawer.h"
#include "ConsoleOutput.h"
#include "Enemy.h"

#include <iostream>


template <typename InputHandler, typename OutputHandler, typename Drawer>
class GameControl{
private:
    Game* game;
    InputHandler* inputter;
    DisplayManager<Drawer, OutputHandler> displayer;
    bool isNewGame = false;

public:
    GameControl(Game* game) noexcept;
    void start();
    void initNewGame(bool& processGameFlag);
    void save();
    void load();
    ~GameControl() noexcept;
};

#endif
```

Название файла: GameController.cpp

```
#include "GameController.h"

template <typename InputHandler, typename OutputHandler, typename
Drawer>
GameControl<InputHandler, OutputHandler, Drawer>::GameControl(Game*
game) noexcept : game(game) {
    inputter = new InputHandler();
}

template <typename InputHandler, typename OutputHandler, typename
Drawer>
GameControl<InputHandler, OutputHandler, Drawer>::~GameControl()
noexcept{
    delete inputter;
}
```

```
     template <typename InputHandler, typename OutputHandler, typename
Drawer>
     void GameControl<InputHandler, OutputHandler,
Drawer>::initNewGame(bool& processGameFlag){
          try{
              this->inputter->loadCommands("commands.json");
          } catch(const std::runtime_error& e) {
              displayer.printMessage("Commnads loading error, default
settings will be applied");
              this->inputter->useDefaultCommands();
          }

          displayer.printMessage("\nSelect option:");

displayer.printCommands(this->inputter->getStartGameCommandsChars());

          bool optionSelected = false;
          while (!optionSelected){
              GameCommand command;

              try {
                  command = this->inputter->getCommand();
              }catch (const std::out_of_range& e) {
                  displayer.printMessage("Incorrect command");
                  continue;
              } catch (const std::exception& e){
                  displayer.printMessage(e.what());
                  continue;
              }

              switch (command) {
              case GameCommand::START_NEW_GAME:
                  isNewGame = true;
                  optionSelected = true;
                  break;
              case GameCommand::LOAD:
                  if (this->game->getState().hasSave()) {
                      displayer.printMessage("Loading");
                      this->load();
                      optionSelected = true;
                  } else {
                      displayer.printMessage("You have not saves");
                  }
                  break;
              case GameCommand::EXIT:
                  processGameFlag = false;
                  optionSelected = true;
                  displayer.printMessage("Exit");
                  break;

              default:
                  displayer.printMessage("Incorrect command");
                  break;
              }
          }
     }
```

```
    template <typename InputHandler, typename OutputHandler, typename
Drawer>
    void GameControl<InputHandler, OutputHandler, Drawer>::start() {
        bool processGameFlag = true;
        this->initNewGame(processGameFlag);

        while (processGameFlag) {
            if (isNewGame){
                this->game->newGame();
                isNewGame = false;
            }

            displayer.printMessage("\nSelect option:");
            displayer.printCommands(this->inputter->getGameCommands());

            while (!game->getPlayer(true)->allShipsSunk()
&& !game->getPlayer(false)->allShipsSunk()) {
                std::string roundMessage = "\nCurrent round: " +
std::to_string(this->game->getCurrentRound());
                displayer.printMessage(roundMessage);

                displayer.printField(this->game->getPlayer(false),
false);
                displayer.printField(this->game->getPlayer(true),
true);


displayer.printAvailableAbilities(this->game->getPlayer(false));

                bool optionSelected = false;
                GameCommand option;

                while(!optionSelected){
                    try {
                        option = inputter->getCommand();
                        optionSelected = true;
                    } catch (const std::out_of_range& e) {
                        displayer.printMessage("Incorrect command");
                    } catch (const std::exception& e){
                        displayer.printMessage(e.what());
                    }
                }
                switch (option) {
                case GameCommand::ATTACK: {
                    displayer.printMessage("\nSelected attack, enter
coordinates (x, y)");
                    Coordinates coords(0,0);
                    bool check = false;

                    while(!check){
                        try {
                            coords = this->inputter->getCoords();
                            check = true;
                        } catch (const std::out_of_range& e) {
                            displayer.printMessage("Incorrect
command");
                        }
                    }
```

```
this->game->getPlayer(true)->attack_player(coords.x, coords.y);
                    Coordinates crd = this->game->enemyTurn();
                    this->game->getPlayer(false)->attack_player(crd.x,
crd.y);
                    displayer.printMessage("\nEnemy attacked the point
" + std::to_string(crd.x) + " " +  std::to_string(crd.y));
                    this->game->nextRound();
                    break;
                }
                case GameCommand::USE_ABILITY: {
                    if
(this->game->getPlayer(false)->get_ability_type() == 2) {
                        displayer.printMessage("\nScanner, enter
coordinates (x, y)");
                        Coordinates coords(0,0);
                        bool check = false;

                        while(!check){
                            try {
                                coords = this->inputter->getCoords();
                                check = true;
                            } catch (const std::out_of_range& e) {
                                displayer.printMessage("Incorrect
command");
                            }
                        }

this->game->getPlayer(false)->get_coords(coords.x, coords.y);
                    }
                    if
(this->game->getPlayer(false)->get_ability_type() == 3) {
                        displayer.printMessage("\nBombardment");
                    }


this->game->getPlayer(false)->use_ability_on_player(*this->game->getPlaye
r(true));

                    if (this->game->getPlayer(true)->get_flag_dd()) {
                        displayer.printMessage("\nDouble attack, enter
coordinates (x, y)");
                        Coordinates coords(0,0);
                        bool check = false;

                        while(!check){
                            try {
                                coords = this->inputter->getCoords();
                                check = true;
                            } catch (const std::out_of_range& e) {
                                displayer.printMessage("Incorrect
command");
                            }
                        }

this->game->getPlayer(true)->attack_player(coords.x, coords.y);
                    }
```

```
                    Coordinates crd = this->game->enemyTurn();
                    this->game->getPlayer(false)->attack_player(crd.x,
crd.y);
                    displayer.printMessage("Enemy attacked the point "
+ std::to_string(crd.x) + " " +  std::to_string(crd.y));
                    this->game->nextRound();
                    continue;
                }
                case GameCommand::SAVE: {
                    this->save();
                    displayer.printMessage("Saved");
                    continue;
                }
                case GameCommand::LOAD: {
                    if (this->game->getState().hasSave()) {
                        this->load();
                    } else {
                        displayer.printMessage("You have not saves");
                    }

                    break;
                }
                case GameCommand::START_NEW_GAME: {
                    this->game->newGame();
                    continue;
                }
                case GameCommand::EXIT: {
                    displayer.printMessage("Exit");
                    return;
                }

                default:{
                    displayer.printMessage("Incorrect command");
                    break;
                }

                }
            }
            if (game->getPlayer(false)->allShipsSunk()) {
                displayer.printMessage("Enemy won!");
                displayer.printMessage("Game Over");
                displayer.printMessage("Starting new game");
                isNewGame = true;
            } else if (game->getPlayer(true)->allShipsSunk()) {
                displayer.printMessage("Player won!");
                displayer.printMessage("Continue game");
                exit(0);
            }
        }
    }

    template <typename InputHandler, typename OutputHandler, typename
Drawer>
    void GameControl<InputHandler, OutputHandler, Drawer>::save() {
        game->saveGame();
    }
```

```
    template <typename InputHandler, typename OutputHandler, typename
Drawer>
    void GameControl<InputHandler, OutputHandler, Drawer>::load() {
        game->loadGame();
    }

template class GameControl<ConsoleInput, ConsoleOutput, ConsoleDrawer>;
```

Название файла: ConsoleManager.h

```
#ifndef CONSOLE_MANAGER_H
#define CONSOLE_MANAGER_H

#include "ConsoleDrawer.h"
#include "ConsoleOutput.h"
#include "ConsoleInput.h"
#include "Player.h"

#include <string>

template <typename Drawer, typename MessageHandler>
class DisplayManager
{
private:
    MessageHandler *outputter;
    Drawer *drawer;

public:
    DisplayManager();
    void printMessage(std::string message);
    void printAvailableAbilities(Player *player);
    void printField(Player *player, bool isHidden);
    void printCommands(std::map<std::string, char> enableCommands);
    void printMenu(bool isStart);
    ~DisplayManager();
};

#endif
```

Название файла: ConsoleManager.cpp

```
#include "ConsoleManager.h"



template <typename Drawer, typename MessageHandler>
DisplayManager<Drawer, MessageHandler>::DisplayManager(){
    this->outputter = new MessageHandler();
    this->drawer = new Drawer();


}
template <typename Drawer, typename MessageHandler>
DisplayManager<Drawer, MessageHandler>::~DisplayManager(){
    delete outputter;
```

```
        delete drawer;
    }

    template <typename Drawer, typename MessageHandler>
    void DisplayManager<Drawer, MessageHandler>::printMessage(std::string message){
        outputter->printMessage(message);
    }

    template <typename Drawer, typename MessageHandler>
    void DisplayManager<Drawer, MessageHandler>::printAvailableAbilities(Player* player){
        outputter->printAvailableAbilities(player);
    }

    template <typename Drawer, typename MessageHandler>
    void DisplayManager<Drawer, MessageHandler>::printField(Player* player, bool isHidden){
        drawer->print_board(player->getBoard(), isHidden);
    }

    template <typename Drawer, typename MessageHandler>
    void DisplayManager<Drawer, MessageHandler>::printMenu(bool isStart){
        outputter->printMenu(isStart);
    }

    template <typename Drawer, typename OutputHandler>
    void DisplayManager<Drawer, OutputHandler>::printCommands(std::map<std::string, char> enableCommands){
        outputter->printCommands(enableCommands);
    }

template class DisplayManager<ConsoleDrawer, ConsoleOutput>;
```

Название файла: ConsoleInput.h

```
#ifndef CONSOLE_INPUT_H
#define CONSOLE_INPUT_H
```

```
#include "Structures.h"
#include "Exceptions.h"
#include "JsonHandler.h"
#include "nlohmann/json.hpp"

#include <iostream>
#include <limits>
#include <sstream>
#include <set>
#include <vector>
#include <map>


class ConsoleInput {
private:
    std::map<char, GameCommand> commandsChars;
    std::map<std::string, GameCommand> commandsStrings;
    std::map<std::string, char> stringsCmds;
public:
    ConsoleInput();
    int getOption();
    bool getBool();
    std::vector<int> getSizes();
    Coordinates getCoords();
    std::map<std::string, char> getGameCommands();
    std::map<std::string, char> getStartGameCommandsChars();
    void loadCommands(const std::string& filename);
    void useDefaultCommands();
    GameCommand getCommand();
};

#endif
```

Название файла: ConsoleInput.cpp

```
#include "ConsoleInput.h"

ConsoleInput::ConsoleInput() {
    this->commandsStrings["ATTACK"] = GameCommand::ATTACK;
    this->commandsStrings["USE_ABILITY"] = GameCommand::USE_ABILITY;
```

```
        this->commandsStrings["SAVE"] = GameCommand::SAVE;
        this->commandsStrings["LOAD"] = GameCommand::LOAD;
        this->commandsStrings["START_NEW_GAME"]                          =
GameCommand::START_NEW_GAME;
        this->commandsStrings["EXIT"] = GameCommand::EXIT;
    }

    int ConsoleInput::getOption() {
        int number;
        std::cin >> number;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            throw InvalidInputType();
        }

        if (number <= 0) {
            throw NegativeNumberEntering();
        }

        return number;
    }

    Coordinates ConsoleInput::getCoords() {
        int number1, number2;

        std::cin >> number1 >> number2;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            throw InvalidCoordinatesFormat();
        }

        if (number1 < 0 || number2 < 0) {
            throw NegativeNumberEntering();
```

```
        }

        return Coordinates{number1, number2};
    }

    std::vector<int> ConsoleInput::getSizes() {
        std::vector<int> sizes;
        std::string input;
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        std::getline(std::cin, input);

        std::istringstream stream(input);
        int value;

        while (stream >> value) {
            if (value < 1 || value > 4) {
                throw InvalidInputType();
            }
            sizes.push_back(value);
        }

        if (sizes.size() < 1 || sizes.size() > 4) {
            throw InvalidInputType();
        }

        return sizes;
    }

    bool ConsoleInput::getBool() {
        int input;
        std::cin >> input;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            throw InvalidInputType();
        }
```

```
        if (input != 0 && input != 1) {
            throw InvalidInputType();
        }

        return static_cast<bool>(input);
    }

    void ConsoleInput::loadCommands(const std::string& filename) {
        JsonFileHandler fileHandler(filename);
        fileHandler.load();
        json& data = fileHandler.getData();

        std::set<char> keys;
        std::set<std::string> stringCommands;

        for (const auto& item : data.items()) {
            char key = item.key()[0];
            std::string value = item.value();

            if (keys.find(key) != keys.end()) {
                throw std::runtime_error("Duplicate key assignment: " +
std::string(1, key));
            }

            if (stringCommands.find(value) != stringCommands.end()) {
                throw std::runtime_error("Duplicate command assignment:
" + value);
            }

            if (commandsStrings.find(value) == commandsStrings.end()) {
                throw std::runtime_error("Unknown command: " + value);
            }

            this->commandsChars[key] = this->commandsStrings[value];
            keys.insert(key);
            stringCommands.insert(value);
        }
```

```
        if (commandsChars.size() < commandsStrings.size()) {
            throw std::runtime_error("Few commands!");
        }
    }

    void ConsoleInput::useDefaultCommands() {
        for (int i = 0; i < static_cast<int>(GameCommand::ENUM_END); i++)
{
            this->commandsChars[static_cast<char>(static_cast<int>(i   +
48))] = static_cast<GameCommand>(i);
        }
    }

    std::map<std::string, char> ConsoleInput::getGameCommands() {
        this->stringsCmds.clear();
        for (const auto& pair : this->commandsStrings) {
            for (const auto& otherPair : this->commandsChars) {
                if (pair.second == otherPair.second) {
                    this->stringsCmds[pair.first] = otherPair.first;
                    break;
                }
            }
        }

        return this->stringsCmds;
    }

    std::map<std::string, char> ConsoleInput::getStartGameCommandsChars()
{
        this->stringsCmds.clear();
        std::set<std::string> startCommands;

        startCommands.insert("START_NEW_GAME");
        startCommands.insert("LOAD");
        startCommands.insert("EXIT");

        for (const auto& pair : this->commandsStrings) {
            for (const auto& otherPair : this->commandsChars) {
```

```
                if     ((pair.second     ==     otherPair.second)     &&
startCommands.count(pair.first)) {
                    this->stringsCmds[pair.first] = otherPair.first;
                    break;
                }
            }
        }

        return this->stringsCmds;
    }

    GameCommand ConsoleInput::getCommand() {
        char key;
        std::cin >> key;

        if (std::cin.fail()) {
            std::cin.clear();
            throw InvalidInputType();
        }

        return this->commandsChars.at(key);
}
```

Название файла: ConsoleOutput.h

```
#ifndef CONSOLE_OUTPUT_H
#define CONSOLE_OUTPUT_H

#include "AbilityManager.h"
#include "Player.h"


class ConsoleOutput {
public:
    ConsoleOutput() noexcept;
    void printAvailableAbilities(Player* Player) noexcept;
    void printMessage(std::string message);
    void printMenu(bool isStart);
    void printCommands(std::map<std::string, char> enableCommands);
```

```
        ~ConsoleOutput() noexcept {};
    };


#endif
```

Название файла: ConsoleOutput.cpp

```
    #include "ConsoleOutput.h"

    ConsoleOutput::ConsoleOutput() noexcept {}

    void ConsoleOutput::printAvailableAbilities(Player* player)
noexcept{
        std::vector<int> abilities = player->get_ability_list();
        std::unordered_map<int, std::string> abilityNames = {
            {1, "Double_damage"},
            {2, "Scanner"},
            {3, "Bombardment"}
        };

        std::cout << "Player Abilities" << std::endl;
        int index = 1;
        for (int ability : abilities) {
            if (abilityNames.find(ability) != abilityNames.end()) {
                std::cout << index << ". " << abilityNames[ability] <<
std::endl;
                ++index;
            } else {
                std::cout << index << ". Unknown Ability" << std::endl;
                ++index;
            }
        }
    }

    void ConsoleOutput::printMessage(std::string message) {
        std::cout << message << "\n";
    }

    void ConsoleOutput::printCommands(std::map<std::string, char>
enableCommands) {
        for (auto pair : enableCommands){
            std::cout << pair.second << " - " << pair.first << "\n";
        }
}
```