FМИНОБРНАУКИ 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.

Студентка гр. 3385 _____ Дорогушина А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

По правилам ООП разработать и реализовать систему управления и отображения игры типа «Морской бой», состоящую из шаблонных классов, которые обеспечивают гибкость и модульность в обработке пользовательского ввода и визуализации игрового процесса.

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 - параметром 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. Класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку без изменения отслеживания.

После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа 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

1. Класс *CommandMapper*

Для чего: для управления сопоставлением клавиш и команд в игре. Он позволяет устанавливать команды по умолчанию, загружать их из файла, проверять на корректность и предоставлять доступ к командам через пользовательский ввод.

Описание класса:

private:

std::unordered_map<char, Command> key_to_command; - хранит информацию о командах: сопоставляет символ клавиши на команду, приватное поле

std::unordered_map<Command, char> command_to_key; - хранит информацию о командах: сопоставляет команду на символ клавиши, приватное поле

bool ValidateMapping(char key, Command command); - проверка пустого символа и дублирования при задании клавиши, приватный метод

public:

CommandMapper(); - конструктор, вызывает метод ниже, публичный метод

void set_default_commands(); - инициализация команд по умолчанию, публичный метод

void load_from_file(const std::string& filename); - очистка команд по умолчанию и загрузка команд из файла, если это возможно, публичный метод

void check_full_mapping(); - проверка всех основных команд для функционирования игры, если не все команды имеют ключ, создает его рандомно, публичный метод

Command string_to_command(const std::string& command_str); - препреобразование строки в команду, используется при считывании из файла, публичный метод

Command get_command(char key) const; - получение команды по ключу, публичный метод

void print_available_commands() const; - вывод списка доступных команд, публичный метод

Как и почему связаны классы: Та самая рекомендуемая «прослойка», сопоставляющая команду и считываемый символ. Используется в классе *InputProcessor*.

2. Класс *FieldRendering*

Для чего: для отображения полей.

Описание класса:

public:

void draw_fields(User& user, Bot& bot) – отображает поля пользователя и врага, публичный метод

Как и почему связаны классы: будет использоваться в классе *GameDisplay* для отображения полей при изменении.

3. Класс *GameDisplay*

Для чего: шаблонный класс, принимает параметр типа *FieldRendering*, наследует от класса *Observer*, что позволяет ему получать уведомления об изменениях в игре, за которой он наблюдает.

Описание класса:

private:

Game& game; - сама игра, приватное поле

FieldRendering field_render; - отвечает за отображение, приватное поле

public:

explicit GameDisplay(Game& new_game, FieldRendering& new_fieldrender); - конструктор, инициализирует поля и добавляет наблюдателя, публичный метод

void update() override; - переопределение виртуального метода класса наблюдателя, отображает поля при изменении, публичный метод

Как и почему связаны классы: создан через наследование с *Observer* для реализации метода *update* и использует параметр для отображения полей. Так как является шаблонным, позволяет использовать разные способы отображения полей.

4. Класс *GameManage*

Для чего: для управления игровым процессом, шаблонный класс, принимает параметр типа *InputProcessor*, что позволяет совершать обработку ввода команд пользователя.

Описание класса:

private:

Game& game; - сама игра, приватное поле

InputProcessor& data; - отвечает за обработку команд, приватное поле

public:

GameManage(Game& new_game, InputProcessor& input_processor); - конструктор, инициализирует поля, публичный метод

void CallCommand(); - выполнение команды, полученной от пользователя с помощью параметра класса, публичный метод

Как и почему связаны классы: использует параметр для получения команд пользователя. Так как является шаблонным, позволяет использовать разные способы реализации вода пользователя. В зависимости от них вызывает методы класса *Game*.

5. Класс *InputProcessor*

Для чего: для обработки ввода команд пользователя, использует *CommandMapper* для сопоставления ключей и команд.

Описание класса:

private:

std::string filename; - имя файла, приватное поле

CommandMapper mapper; - отвечает за сопоставление ключей и команд, приватное поле

public:

InputProcessor(const std::string& new_filename = "command.txt"); - конструктор, проверяет файл и загружает команды, публичный метод

Command get_command(); - считывает ввод пользователя, пока тот не будет правильным, и возвращает команду

Как и почему связаны классы: используется в шаблонном классе *GameManage* и для сопоставления команд и ввода пользователя использует *CommandMapper*.

6. Классы *Observer, Observable*

Для чего: для реализации наблюдателя и наблюдаемого объекта.

Описание класса:

Observer:

public:

virtual void update() = 0; - виртуальная функция, переопределяется в подклассах-наблюдателях, публичный метод

virtual ~Observer() = default; - деструктор по умолчанию, публичный метод

Observable:

private:

std::vector<Observer> observers;* - список наблюдателей, приватное поле

public:

void add_observer(Observer observer);* - добавление наблюдателя, публичный метод

void notify_observers(); - уведомления об изменении, вызывает *update*, публичный метод

virtual ~Observable() = default; - деструктор по умолчанию, публичный метод

Как и почему связаны классы: используется в классе *Game* при игровом процессе и является родительским классом для *GameDisplay*, который реализует *update* через отображение полей.

7. UML-диаграмма созданных классов

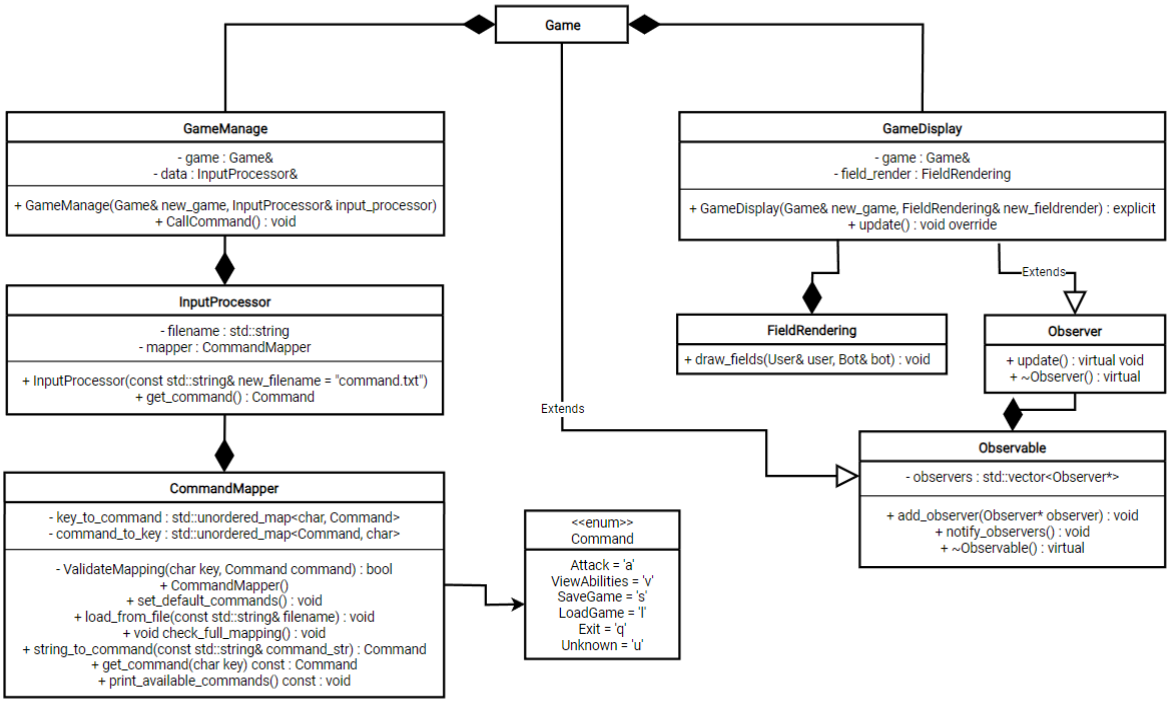

Рис. 1 – *UML*-диаграмма созданных классов

Выводы

По правилам ООП была разработана и реализована система управления и отображения игры типа «Морской бой».

Были созданы необходимые классы (шаблонные и их параметры) по заданию: *GameManage, InputProcessor, GameDisplay, FieldRendering*.

Также по примечаниям была реализована «прослойка» *CommandMapper* для *InputProcessor*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: CommandMapper.cpp

```
#include "CommandMapper.h"

// Проверка на дурака и дублирование
bool CommandMapper::ValidateMapping(char key, Command command) {
    if (key == '\0') {
        std::cerr << "Error: Key cannot be null or empty." <<
std::endl;
        return false;
    }

    if (key_to_command.find(key) != key_to_command.end()) {
        return false;
    }

    if (command_to_key.find(command) != command_to_key.end()) {
        return false;
    }

    return true;
}

// По умолчанию
CommandMapper::CommandMapper() {
    set_default_commands();
}

// Если файлик пустой
void CommandMapper::set_default_commands(){
    key_to_command['a'] = Command::Attack;
    key_to_command['v'] = Command::ViewAbilities;
    key_to_command['s'] = Command::SaveGame;
    key_to_command['l'] = Command::LoadGame;
    key_to_command['q'] = Command::Exit;

    for (const auto& [key, cmd] : key_to_command){
        command_to_key[cmd] = key;
    }
}

// Загрузка из файла, если это возможно
void CommandMapper::load_from_file(const std::string& filename){
    std::ifstream file(filename);
    if (!file || !file.is_open() || file.peek() ==
std::ifstream::traits_type::eof()) {
        std::cout << "The command file cannot be opened or is
empty. The default commands are used." << std::endl;
        set_default_commands();
        return;
    }

    // Если возможно, очистка клавиш по умолчанию и загрузка
    key_to_command.clear();
```

```
    command_to_key.clear();
    char key;
    std::string command_str;
    while (file >> key >> command_str){
        Command command = string_to_command(command_str);
        if (ValidateMapping(key, command)){
            key_to_command[key] = command;
            command_to_key[command] = key;
        }else{
            std::cerr << "Invalid mapping in file. Skipping: " <<
key << " -> " << command_str << std::endl;
            continue;
        }
    }
    check_full_mapping();
}

// Все ли клавиши заполнены
void CommandMapper::check_full_mapping() {
    std::vector<Command> all_commands = {Command::Attack,
Command::ViewAbilities, Command::SaveGame,
                                         Command::LoadGame,
Command::Exit};
    for (Command cmd : all_commands) {
        if (command_to_key.find(cmd) == command_to_key.end()) {
            std::cerr << "Base command is missing a key, it will be
random." << std::endl;
            char key_char = rand();
            key_to_command[key_char] = cmd;
            command_to_key[cmd] = key_char;
        }
    }
}

// Очев
Command CommandMapper::string_to_command(const std::string&
command_str){
    if (command_str == "Attack" || command_str == "attack") return
Command::Attack;
    if (command_str == "ViewAbilities" || command_str ==
"viewabilities" || command_str == "view_abilities") return
Command::ViewAbilities;
    if (command_str == "SaveGame" || command_str == "savegame" ||
command_str == "save_game") return Command::SaveGame;
    if (command_str == "LoadGame" || command_str == "loadgame" ||
command_str == "load_game") return Command::LoadGame;
    if (command_str == "Exit" || command_str == "exit") return
Command::Exit;
    return Command::Unknown;
}

// Получаем команду
Command CommandMapper::get_command(char key) const{
    auto it = key_to_command.find(key);
    if (it != key_to_command.end()) {
        return it->second;
    }
    return Command::Unknown;
```

```
}

// Вывод доступных
void CommandMapper::print_available_commands() const {
    std::cout << "Available commands:" << std::endl;
    for (const auto& [key, cmd] : key_to_command) {
        std::cout << "Key: " << key << " -> Command: ";
        switch (cmd) {
            case Command::Attack: std::cout << "Attack"; break;
            case Command::ViewAbilities: std::cout <<
"ViewAbilities"; break;
            case Command::SaveGame: std::cout << "SaveGame"; break;
            case Command::LoadGame: std::cout << "LoadGame"; break;
            case Command::Exit: std::cout << "Exit"; break;
            default: std::cout << "Unknown";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}
```

Название файла: CommandMapper.h

```
#ifndef COMMANDMAPPER_H
#define COMMANDMAPPER_H

#include <vector>
#include <fstream>
#include <unordered_map>
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <algorithm>

// По умолчанию
enum class Command {Attack = 'a',
                    ViewAbilities = 'v',
                    SaveGame = 's',
                    LoadGame = 'l',
                    Exit = 'q',
                    Unknown = 'u'};

// Класс комманд: установка по умолчанию, загрузка из файла,
проверки, вывод доступных
class CommandMapper{
private:
    std::unordered_map<char, Command> key_to_command;
    std::unordered_map<Command, char> command_to_key;
    bool ValidateMapping(char key, Command command);

public:
    CommandMapper();
    void set_default_commands();
    void load_from_file(const std::string& filename);
    void check_full_mapping();
    Command string_to_command(const std::string& command_str);
    Command get_command(char key) const;
    void print_available_commands() const;
```

```
};

#endif //COMMANDMAPPER_H
```

Название файла: FieldRendering.cpp

```
#include "FieldRendering.h"

void FieldRendering::draw_fields(User& user, Bot& bot) {
    int field_size = user.GetField().GetFieldSize();
    cout << "\n"<< string((field_size * 2), ' ') << "Playing
fields\n" << endl;
    cout << "User's Field" << "  " << string(((field_size-4) * 3),
' ') << "Enemy's Field" << endl;

    for (int y = 0; y < field_size; y++) {
        // Поле юзера
        for (int x = 0; x < field_size; x++) {
            auto& cell = user.GetField().GetAllCell()[y][x];
            int index_seg =
user.GetField().GetAllCell()[y][x].GetIndexSegment();

            if (cell.GetStatus() == CellState::Unknown) {
                cout << "~  ";
            } else if (cell.GetStatus() == CellState::Ship) {
                int id =  cell.GetShipId();
                Ship* ship =
user.GetShipManager().SearchShipId(id);
                cell.SetShip(ship);
                if (ship != nullptr){
                    if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::HealthyShipSegment){
                        cout << "2  ";
                    }else if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::DamagedShipSegment){
                        cout << "1  ";
                    }else if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::DestroyedShipSegment){
                        cout << "x  ";
                    }
                }
            } else if (cell.GetStatus() == CellState::No_ship) {
                cout << "o  ";
            }
        }

        cout << "  ";

        // Поле бота
        for (int x = 0; x < field_size; x++) {
            int index_seg =
bot.GetField().GetAllCell()[y][x].GetIndexSegment();
            auto& cell = bot.GetField().GetAllCell()[y][x];
```

```
            if (cell.GetStatus() == CellState::Unknown) {
                cout << "~  ";
            } else if (cell.GetStatus() == CellState::Ship){
                int id = cell.GetShipId();
                Ship* ship = bot.GetShipManager().SearchShipId(id);
                cell.SetShip(ship);
                if (ship != nullptr){
                    if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::HealthyShipSegment){
                        cout << "~  ";
                    }else if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::DamagedShipSegment){
                        cout << "1  ";
                    }else if (ship-
>GetSegmentState(index_seg).GetStatus() ==
ConditionState::DestroyedShipSegment){
                        cout << "x  ";
                    }
                }
            } else if (cell.GetStatus() == CellState::No_ship) {
                cout << "o  ";
            }
        }
        cout << endl;
    }
    cout << endl;
}
```

Название файла: FieldRendering.h

```
#ifndef FIELDRENDERIND_H
#define FIELDRENDERIND_H

#include "User.h"
#include "Bot.h"

#include <iostream>

// Рисуем поля
class FieldRendering{
public:
    void draw_fields(User& user, Bot& bot);
};

#endif //FIELDRENDERIND_H
```

Название файла: GameDisplay.h

```
#ifndef GAMEDISPLAY_H
#define GAMEDISPLAY_H

#include "Game.h"
#include "Observer.h"
#include "FieldRendering.h"
```

```
// Шаблонныый класс
template <typename FieldRendering>
// Класс чекер изменений
class GameDisplay: public Observer{
private:
    Game& game;
    FieldRendering field_render;
public:
    explicit GameDisplay(Game& new_game, FieldRendering&
new_fieldrender)
        : game(new_game), field_render(new_fieldrender) {
            game.add_observer(this);
        }
    void update() override{
        field_render.draw_fields(game.GetUser(), game.GetBot());
    }
};

#endif // GAMEDISPLAY_H
```

Название файла: GameManage.h

```
#ifndef GAMEMANAGE_H
#define GAMEMANAGE_H

#include <iostream>
#include "InputProcessor.h"
#include "Game.h"

// Шаблонный класс
template <typename InputProcessor>
// Класс управления игрой
class GameManage{
private:
    Game& game;
    InputProcessor& data;
public:
    GameManage(Game& new_game, InputProcessor& input_processor)
        : game(new_game), data(input_processor){}

    void CallCommand(){
        switch (data.get_command()) {
            case Command::Attack:
                std::cout << "You chose to attack the opponent's
field.\n";
                game.AttackRound();
                break;
            case Command::ViewAbilities:
                std::cout << "The number of habilitation is being
viewed..." << std::endl;
                std::cout << "You have " <<
game.GetUser().GetAbilCount() << " abilities in your manager.\n";
                break;
            case Command::SaveGame:
                std::cout << "Saving game...\n";
                game.SaveGame();
                std::cout << "Game saved successfully.\n";
```

```
                break;
            case Command::LoadGame:
                std::cout << "Loading saved game...\n";
                game.LoadGame();
                std::cout << "Game loaded successfully.\n";
                break;
            case Command::Exit:
                char choice;
                do {
                    std::cout << "Are you sure you want to get out?
(y/n): ";
                    std::cin >> choice;
                    if (std::cin.fail()) {
                        std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                        std::cout << "Invalid input. Please enter
'y' or 'n'.\n";
                        break;
                        continue;
                    }
                    choice = std::tolower(choice);
                    if (choice != 'y' && choice != 'n') {
                        std::cout << "Invalid input. Please enter
'y' or 'n'.\n";
                    }
                } while (choice != 'y' && choice != 'n');

                if (choice == 'y' || choice == 'Y') {
                    std::cout << "Exiting the game...\n";
                    exit(0);
                }
                std::cout << "The game continues!\n";
                break;
            default:
                std::cout << "Unknown command cannot be
executed.\n";
                break;
        }
    }
};

#endif //GAMEMANAGE_H
```

Название файла: InputProcessor.cpp

```
#include "InputProcessor.h"

// Загрузка из файла, если возможно
InputProcessor::InputProcessor(const std::string& new_filename){
    filename = new_filename;
    if (!filename.empty()){
        mapper.load_from_file(filename);
    }else{
        std::cout << "The file is empty. Default commands will be
used." << std::endl;
    }
};
```

```
// Считываем команду
Command InputProcessor::get_command() {
    std::string user_input;
    while (true) {
        try {
            std::cout << "Enter command: ";
            std::cin >> user_input;
            if (user_input.length() != 1) {
                throw std::invalid_argument("Invalid input. Please
enter only a single character.");
            }
            char input = user_input[0];
            Command command = mapper.get_command(input);
            // Если команда не распознана
            if (command == Command::Unknown) {
                throw std::invalid_argument("The entered command is
not recognized. Please enter a valid command.");
            }
            // Если команда норм бро
            return command;
        } catch (const std::invalid_argument& e) {
            std::cerr << e.what() << std::endl;
            mapper.print_available_commands();
        }
    }
}
```

Название файла: InputProcessor.h

```
#ifndef INPUTPROCESSOR_H
#define INPUTPROCESSOR_H

#include <iostream>
#include <string>
#include "CommandMapper.h"

// Класс ввода
class InputProcessor{
private:
    std::string filename;
    CommandMapper mapper;
public:
    InputProcessor(const std::string& new_filename =
"command.txt");
    Command get_command();
};

#endif //INPUTPROCESSOR_H
```

Название файла: Observer.cpp

```
#include "Observer.h"

void Observable::add_observer(Observer* observer){
    observers.push_back(observer);
}
```

```
void Observable::notify_observers(){
    for(Observer* observer : observers){
        observer->update();
    }
}
```

Название файла: Observer.h

```
#ifndef OBSERVER_H
#define OBSERVER_H
#include <vector>

// Класс наблюдателя
class Observer{
public:
    virtual void update() = 0;
    virtual ~Observer() = default;
};

// Класс наблюдаемого
class Observable{
private:
    std::vector<Observer*> observers;
public:
    void add_observer(Observer* observer);
    void notify_observers();
    virtual ~Observable() = default;
};

#endif //OBSERVER_H
```