**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 |  | Лапшов К.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Изучить основы шаблонных классов и функций. Используя эту парадигму, разработать дополнения к игре: классы для считывания ввода и отрисовки в терминале, а также класс управления игрой, который принимает в качестве параметров шаблона классы, определяющие способ ввода команд и вывода информации. Интегрировать эти классы с ранее созданными сущностями.

## **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.

## **Выполнение работы**

Перечисление ‘InputCommand’ представляет собой логику представления доступных команд: Attack, Ability, Save, Load, Exit, Input, None

Класс ‘CliInput’ представляет собой реализацию класса считывающего ввод пользователя из терминала.

Поля класса:

* std::map<int, InputCommand> commands\_map - словарь реализующий логику сопоставления номера команды и ее тип. Ключ – номер команды в игре, значение – тип комманды
* std::map<std::string, InputCommand> string\_map – словарь перевода строкового представления команды в необходимый тип. Необходим для загрузки команд из файла.

Методы класса:

* CliInput() – Конструктор класса, инициализирует поле string\_map необходимыми данными.
* void loadCommands() – Метод считывания клавиши команды и самой команды из файла с необходимыми значениями. С помощью реализованных словарей проверяет данные на дубликаты и ошибки в исходном файле. В случае ошибки выбрасывает исключение и устанавливает команды по дефолту с помощью метода fill\_default().
* void fillDefault() – Метод для заполнения словаря commands\_map значениями по умолчанию.
* Point getCords() – Метод для получения координат из терминала. Реализует логику с помощью getValidInput().
* int getValidInput() – Метод для получения валидных целочисленных значений из терминала. Работает до тех пор, пока пользователь не введет правильное значение.
* InputCommand readCommand() – Чтение команды от пользователя и возвращение соответствующего типо.
* std::map<int, InputCommand> getCommands() – Возвращает значение словаря commands\_map.

Класс ‘CliOutput’ представляет собой реализацию класса выводящего необходимую текстовую информацию в терминал.

Поля класса:

* std::map<InputCommand, std::string> commands\_to\_string - словарь перевода типа команды в строковое представление.

Методы класса:

* CliOutput() – Конструктор класса, инициализирует поле string\_map необходимыми данными.
* void display(std::string\_view message) – Вывод сообщения message в тконсоль.
* void print\_command(int symbol, InputCommand command) – Метод отрисовки номера и значения команды.
* void printWelcomeCommands(std::map<int, InputCommand> commands\_map) - вывод в консоль команд типа Load, Exit, Input.
* void printGameCommands(std::map<int, InputCommand> commands\_map) - вывод в консоль команд типа : Attack, Ability, Save, Load, Exit.
* void clear() – Метод для очищения консоли.
* void wait() – Метод для приостановки работы ввода консоли на 1.5 сек.

Класс ‘Printer’ создан для отрисовки полей игрока и противника.

Метод класса:

* void printBattleField(GameField& player\_gf, GameField& enemy\_gf) – Реализует логику отрисовки полей игроков в консоль.

Класс GameController <typename InputType, typename OutputType, typename PaintType> представляет собой реализацию шаблонного класса управления игрой. Принимает, в качестве шаблонов классы, с помощью которых определяется способ ввода и вывода информации.

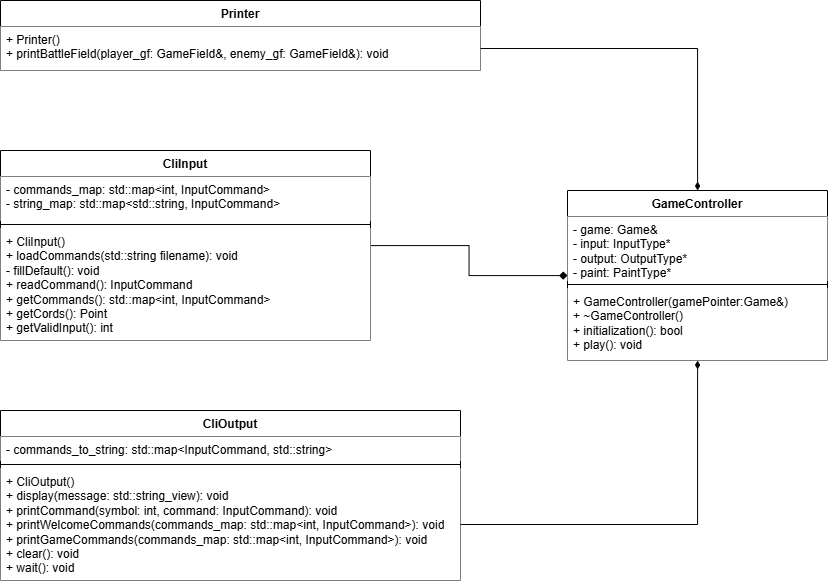
Поля класса:

* Game& game – Ссылка на класс игры.
* InputType\* input – тип класса для ввода.
* OutputType\* output – тип класса для вывода.
* PaintType\* paint – тип класса для отрисовки.

Методы класса:

* GameController(Game& gamePointer) – Конструктор класса, принимающий ссылку на игру. Создает объекты классов InputType, OutputType, PaintType.
* ~GameController() – Деструктор класса, освобождает память из под input, output, paint.
* bool initialization() – Метод отвечает за подготовку игры к началу: загрузку команд, вывод приветственного сообщения, обработку выбора игрока (создание новой игры, загрузка сохранения или выход), а также размещение кораблей игрока при выборе создания новой игры.
* bool play() - Метод отвечает за основной игровой цикл, в котором происходит взаимодействие с игроком, обработка команд, выполнение ходов и проверка условий победы или поражения.

Разработанный программный код см. в приложении А.



## **Выводы**

Были изучены шаблонные классы. В дополнение к игре "Морской бой" были созданы отдельные классы для обработки ввода, вывода и управления игрой. Эти классы были интегрированы с ранее разработанными сущностями, что позволило завершить игру с возможностью дальнейшего расширения, включая добавление графического интерфейса пользователя (GUI).

# **Приложение А Исходный код программы**

Название файла: game\_conroller.h

#ifndef OOP\_GAME\_CONTROLLER\_H

#define OOP\_GAME\_CONTROLLER\_H

#include "game.h"

#include "../IOlogic/cli\_input.h"

#include "../IOlogic/cli\_output.h"

#include "../IOlogic/printer.h"

template <typename InputType, typename OutputType, typename PaintType>

class GameController {

private:

Game& game;

InputType\* input;

OutputType\* output;

PaintType\* paint;

public:

GameController(Game& gamePointer);

~GameController();

bool initialization();

void play();

};

#endif

Название файла: game\_conroller.cpp

template <typename InputType, typename OutputType, typename PaintType>

bool GameController<InputType, OutputType, PaintType>::initialization() {

input->loadCommands("../attribs");

output->display("Welcome to the sea battle!\n");

output->printWelcomeCommands(input->getCommands());

bool correct\_input = false;

while(!correct\_input){

InputCommand choice = input->readCommand();

if(choice == InputCommand::Input){

output->clear();

output->display("To continue, select the coordinates for the ships\n");

output->display("Coordinates are entered in this way: X (0-9) Y (0-9) orientation (0 - horizontal, 1 - vertical)\n");

ShipManager& player\_sm = game.getGameState().getPlayerShipManager();

for (int i = 0; i < player\_sm.getShipCount(); ++i) {

Point point = {0, 0};

int orientation = 0;

Ship\* ship = player\_sm.getShip(i);

bool placed = false;

while(!placed){

try{

output->display("Enter the coordinates for the ship of length " + std::to\_string(ship->getShipLength()) + ": ");

point = input->getCords();

orientation = input->getValidInput();

placed = game.getGameState().placePlayerShip(point, orientation, i);

}catch (const std::invalid\_argument& e) {

std::cin.clear();

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

output->display(e.what());

output->display("\n");

}

}

}

game.getGameState().placeEnemyShips();

correct\_input = true;

}

if(choice == InputCommand::Load){

GameSaveLoader saverLoader = GameSaveLoader("../game.txt");

try{

saverLoader.load(game);

}catch (const std::invalid\_argument& e){

output->display(e.what());

output->display(" Try to creat the new one\n");

continue;

}

correct\_input = true;

}

if(choice == InputCommand::Exit){

return false;

}

if(!correct\_input){

output->display("Unknown command!\n");

}

}

return true;

}

template <typename InputType, typename OutputType, typename PaintType>

void GameController<InputType, OutputType, PaintType>::play() {

bool is\_end = this->initialization();

if(!is\_end){

return;

}

bool game\_running = true;

output->display("Everything was successful!\n");

while (game\_running){

output->clear();

output->display("Current round: " + std::to\_string(game.getCurrentRound()) + "\n");

paint->printBattleField(game.getGameState().getPlayerField(), game.getGameState().getEnemyField());

output->display("\nAvailable abilities: " + std::to\_string(game.getGameState().getAbilitiesCount()) + "\n");

output->display("The first available ability: " + game.getGameState().getLastAbilityName() + "\n");

output->printGameCommands(input->getCommands());

InputCommand choice = input->readCommand();

switch (choice) {

case InputCommand::Ability:{

AbilityUsageInfo info;

if(game.getGameState().getLastAbilityName() == "Scanner"){info.needsCoordinates = true;}

if(info.needsCoordinates){

output->display("Choose the coords to scanner: ");

info.coordinates = input->getCords();

}

game.usePlayerAbility(info);

if (info.errorOccurred) {

output->display("Error using ability.\n");

} else if (info.noAbilities) {

output->display("No abilities available.\n");

}else if (info.isDoubleDamage) {

output->display("Double damage activated!\n");

} else if (info.randomShotPerformed) {

output->display("Random shot performed!\n");

}

if (info.resultOfScanner) {

output->display("Ship has been detected on this cords!\n");

}else if(info.needsCoordinates && !info.resultOfScanner){

output->display("Ship hasn`t been detected on this cords!\n");

}

}

case InputCommand::Attack:{

bool correct\_shoot = false;

output->display("Choose the coords to attack: ");

while(!correct\_shoot){

Point cords\_for\_shoot = input->getCords();

try{

game.playerShoot(cords\_for\_shoot);

correct\_shoot = true;

}catch (const std::invalid\_argument& e) {

std::cin.clear();

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

output->display(e.what());

output->display("\n");

}

}

if(game.isPlayerWon()){

output->clear();

output->display("Congratulations! You won!!!!!\n");

output->display("Creating new round, wait pls....\n");

game.newRound();

output->wait();

continue;

}

break;

}

case InputCommand::Save:{

GameSaveLoader saverLoader("../game.txt");

saverLoader.save(game);

output->clear();

output->display("Game has been saved!\n");

continue;

}

case InputCommand::Load:{

GameSaveLoader saverLoader = GameSaveLoader("../game.txt");

try{

saverLoader.load(game);

}catch (const std::invalid\_argument& e){

output->clear();

output->display(e.what());

output->display("\n");

output->display("Can't load the save :(\n");

continue;

}

output->clear();

output->display("Game has been loaded!\n");

continue;

}

case InputCommand::Exit:{

output->clear();

output->display("Thank you for playing!\n");

output->wait();

game\_running = false;

continue;

}

default:{

output->display("There is no such command :(\n");

output->wait();

continue;

}

}

game.enemyTurn();

if(game.isPlayerLose()){

output->clear();

output->display("You lose(\n");

output->display("You lasted for " + std::to\_string(game.getCurrentRound() - 1) + " round!\n");

output->display("They'll take you next time) \n");

output->wait();

game\_running = false;

continue;

}

}

}

Название файла: cli\_input.h

#ifndef OOP\_CLI\_INPUT\_H

#define OOP\_CLI\_INPUT\_H

#include "iostream"

#include "map"

#include "set"

#include "vector"

#include "commands.h"

#include "../utils/point.h"

#include "../../common/FileWrapper.h"

class CliInput {

private:

std::map<int, InputCommand> commands\_map;

std::map<std::string, InputCommand> string\_map;

void fillDefault();

public:

CliInput();

void loadCommands(std::string filename);

InputCommand readCommand();

std::map<int, InputCommand> getCommands();

Point getCords();

int getValidInput();

};

#endif

Название файла: cli\_input.cpp

#include "cli\_output.h"

CliOutput::CliOutput() {

commands\_to\_string[InputCommand::Attack] = "Attack the enemy";

commands\_to\_string[InputCommand::Ability] = "Use ability";

commands\_to\_string[InputCommand::Save] = "Save game";

commands\_to\_string[InputCommand::Load] = "Load game";

commands\_to\_string[InputCommand::Exit] = "Exit game";

commands\_to\_string[InputCommand::Input] = "Start new game";

}

void CliOutput::printWelcomeCommands(std::map<int, InputCommand> commands\_map){

this->display("Choose the option: ");

for (const auto& pair: commands\_map){

if(pair.second == InputCommand::Input || pair.second == InputCommand::Load || pair.second == InputCommand::Exit){

this->printCommand(pair.first, pair.second);

}

}

this->display("\n");

}

void CliOutput::printGameCommands(std::map<int, InputCommand> commands\_map){

this->display("Choose the option: ");

for (const auto& pair: commands\_map){

if(pair.second != InputCommand::Input){

this->printCommand(pair.first, pair.second);

}

}

this->display("\n");

}

void CliOutput::printCommand(int symbol, InputCommand command){

std::cout << symbol << " - " << commands\_to\_string[command] << "; ";

}

void CliOutput::display(std::string\_view message) {

std::cout << message;

}

void CliOutput::clear() {

system("cls");

}

void CliOutput::wait() {

Sleep(1500);

}

Название файла: cli\_output.h

#ifndef OOP\_CLI\_OUTPUT\_H

#define OOP\_CLI\_OUTPUT\_H

#include "iostream"

#include "map"

#include "commands.h"

#include "Windows.h"

class CliOutput {

private:

std::map<InputCommand, std::string> commands\_to\_string;

public:

CliOutput();

void display(std::string\_view message);

void printCommand(int symbol, InputCommand command);

void printWelcomeCommands(std::map<int, InputCommand> commands\_map);

void printGameCommands(std::map<int, InputCommand> commands\_map);

void clear();

void wait();

};

#endif

Название файла: cli\_output.cpp

#include "cli\_output.h"

CliOutput::CliOutput() {

commands\_to\_string[InputCommand::Attack] = "Attack the enemy";

commands\_to\_string[InputCommand::Ability] = "Use ability";

commands\_to\_string[InputCommand::Save] = "Save game";

commands\_to\_string[InputCommand::Load] = "Load game";

commands\_to\_string[InputCommand::Exit] = "Exit game";

commands\_to\_string[InputCommand::Input] = "Start new game";

}

void CliOutput::printWelcomeCommands(std::map<int, InputCommand> commands\_map){

this->display("Choose the option: ");

for (const auto& pair: commands\_map){

if(pair.second == InputCommand::Input || pair.second == InputCommand::Load || pair.second == InputCommand::Exit){

this->printCommand(pair.first, pair.second);

}

}

this->display("\n");

}

void CliOutput::printGameCommands(std::map<int, InputCommand> commands\_map){

this->display("Choose the option: ");

for (const auto& pair: commands\_map){

if(pair.second != InputCommand::Input){

this->printCommand(pair.first, pair.second);

}

}

this->display("\n");

}

void CliOutput::printCommand(int symbol, InputCommand command){

std::cout << symbol << " - " << commands\_to\_string[command] << "; ";

}

void CliOutput::display(std::string\_view message) {

std::cout << message;

}

void CliOutput::clear() {

system("cls");

}

void CliOutput::wait() {

Sleep(1500);

}

Название файла: printer.h

#ifndef OOP\_PRINTER\_H

#define OOP\_PRINTER\_H

#include "../core/game\_field.h"

class Printer {

public:

Printer() = default;

void printBattleField(GameField& player\_gf, GameField& enemy\_gf);

};

#endif

Название файла: printer.cpp

#include "printer.h"

void Printer::printBattleField(GameField &player\_gf, GameField &enemy\_gf) {

std::cout << " Player Field\t\t\t Enemy Field\n";

std::cout << " ";

for (int j = 0; j < player\_gf.getFieldWidth(); ++j) {

std::cout << j << " ";

}

std::cout << "\t\t ";

for (int j = 0; j < enemy\_gf.getFieldWidth(); ++j) {

std::cout << j << " ";

}

std::cout << "\n";

for (int i = 0; i < player\_gf.getFieldHeight(); ++i) {

std::cout << i << " ";

for (int j = 0; j < player\_gf.getFieldWidth(); ++j) {

FieldCell\* state = player\_gf.getCellInfo({j, i});

if (state->cell\_state == CellState::Unknown) {

if(state->ship\_pointer != nullptr){

std::cout << "■ ";

continue;

}

std::cout << "□ ";

}else if(state->cell\_state == CellState::Ship){

if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Damaged){

std::cout << "◧ ";

}else if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Destroyed){

std::cout << "▧ ";

}

}else if (state->cell\_state == CellState::Empty) {

std::cout << "● ";

}

}

std::cout << "\t\t" << i << " ";

for (int j = 0; j < enemy\_gf.getFieldWidth(); ++j) {

FieldCell\* state = enemy\_gf.getCellInfo({j, i});

if (state->cell\_state == CellState::Unknown) {

std::cout << "□ ";

} else if (state->cell\_state == CellState::Empty) {

std::cout << "● ";

} else if (state->cell\_state == CellState::Ship) {

if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Damaged){

std::cout << "◧ ";

}else if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Destroyed){

std::cout << "▧ ";

}

}

}

std::cout << "\n";

}

}