**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 |  | Пименов П.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Создать шаблонные классы управления вводом и выводом игры. Дать возможность конфигурировать управление игры из файла. Сделать консольный ввод-вывод.

**Задание.**

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

Примечание:

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

Необходимое задание успешно выполнено. Описание реализованных классов:

* ConsoleInput – класс, отвечающий за работу с консольным вводом. Используется в качестве параметра шаблона для класса контроллера ввода игры. Отвечает за считывание символа, сопоставление его с конфигурацией и выдачу игровой команды. Также содержит несколько других методов, например, считывание выбора (bool) или считывание целого числа.
* ConsoleOutput – класс, отвечающий за работу с консольным выводом. Используется в качестве параметра шаблона для класса контроллера вывода игры. Также используется классом-контроллером ввода игры (об этом подробнее рассказано в части про него). Содержит методы печати: поля, сообщения, ошибки.
* GameCommand – enum класс, представляющий игровую команду (далее, Команда). Среди команд есть команды перемещения (вперед-назад, вправо-влево), команда атаки, команда использования способности, команда загрузки, команда сохранения и другие. Enum класс выбран для удобной десериализации в файл. Также создана статическая функция преобразования Команды в строку.
* GameInputController – класс-контроллер ввода игры. Отвечает за подготовку игры и управление игрой. Сопоставляет Команду с методом игры. Осуществляет этап инициализации игровой сессии. Это шаблонный класс, в качестве параметров которого указаны классы, отвечающие за ввод и вывод. Класс вывода пришлось добавить, поскольку при реализации сценария подготовки игры необходимо полноценное взаимодействие с пользователем (получение информации, некоторая обработка, вывод сообщения (например, в случае ошибки)).
* GameObserver – интерфейс Наблюдатель за классом игры. Реализует паттерн Наблюдатель. Содержит методы-триггеры для вызова отрисовки или печати сообщений при том или ином событии в игре. Сам класс игры содержит в себе вектор указателей на экземпляры наблюдателей, уведомляя последних о событиях в игре (вызывая соответствующие метода). Таким образом, слой вывода информации был отделен от слоя игры.
* GameOutputController – класс-контроллер вывода игры. Реализует интерфейс GameObserver. Это шаблонный класс, параметром которого является класс, определяющий способ отрисовки и вывода. Реализует определенные интерфейсом методы-триггеры, выводя релевантную информацию на данный момент игры.

Конфигурация управления консольным вводом сделана через:

1. Ассоциативный массив в классе ConsoleInput, а также его инициализацию управлением по умолчанию. Ассоциативный массив содержит пары вида символ-Команда, сопоставляя при вводе первый второму.
2. В классе FileReader создан метод, считывающий файл конфигурации в JSON объект. Формат JSON выбран так как это устоявшийся стандарт, плюс файлы сохранения игры также являются JSON файлами.
3. В классе Deserializer создан метод, осуществляющий трансляцию JSON объекта в ассоциативный массив. В этом методе также реализованы все необходимые проверки корректности файла конфигурации.
4. В классе ConsoleInput создан метод загрузки новой конфигурации, заменяющий предыдущий ассоциативный массив новым.

UML-диаграмму реализованных классов см. в Приложении A. Исходный код программы см. в Приложении B.

**Тестирование.**

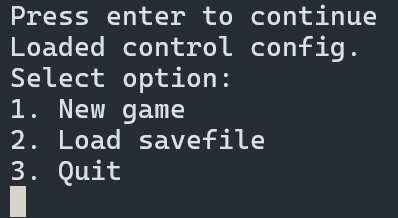


Рисунок 1 – Успешная загрузка конфигурации

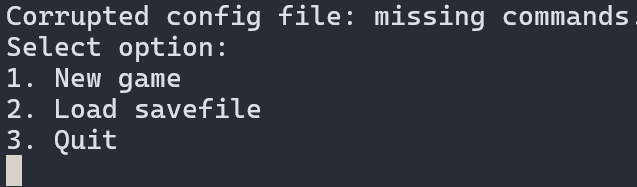


Рисунок 2 – Загрузка некорректной конфигурации 1

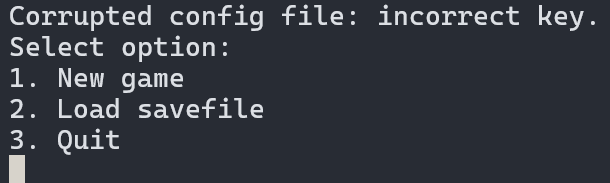


Рисунок 3 – Загрузка некорректной конфигурации 2

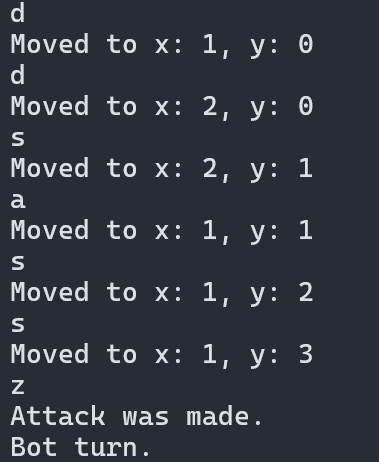


Рисунок 4 – Управление игрой

**Выводы.**

Созданы шаблонные классы управления вводом и выводом игры. Реализована возможность конфигурировать управление игры из файла. Создан консольный ввод-вывод.

Приложение A

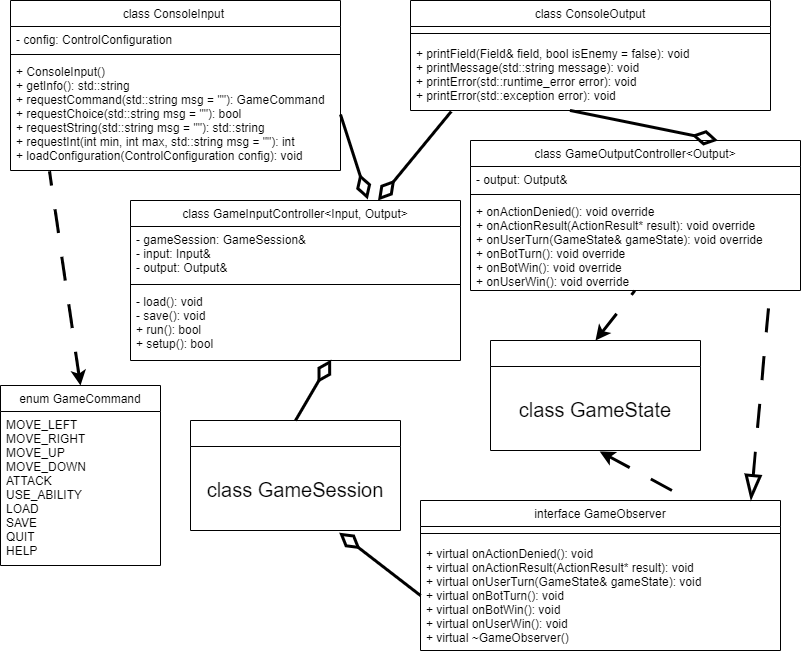


Рисунок 5 – UML-диаграмма

Приложение B

**Файл ConsoleInput.hpp:**

#ifndef CONSOLEINPUT\_HPP // CONSOLEINPUT\_HPP

#define CONSOLEINPUT\_HPP

#include <map>

#include <string>

#include "Coordinates.hpp"

#include "GameCommand.hpp"

#define COMMANDS\_COUNT 10

using ControlConfiguration = std::map<char, GameCommand>;

class ConsoleInput {

private:

ControlConfiguration config;

public:

ConsoleInput();

std::string getInfo();

GameCommand requestCommand(std::string msg = "");

bool requestChoice(std::string msg = "");

std::string requestString(std::string msg = "");

int requestInt(int min, int max, std::string msg = "");

void loadConfiguration(ControlConfiguration config);

};

#endif // CONSOLEINPUT\_HPP

**Файл ConsoleOutput.hpp:**

#ifndef CONSOLEOUTPUT\_HPP // CONSOLEOUTPUT\_HPP

#define CONSOLEOUTPUT\_HPP

#include <stdexcept>

#include <string>

#include "Field.hpp"

class ConsoleOutput {

public:

void printField(Field& field, bool isEnemy = false);

void printMessage(std::string message);

void printError(std::runtime\_error error);

void printError(std::exception error);

};

#endif // CONSOLEOUTPUT\_HPP

**Файл GameCommand.hpp:**

#ifndef GAMECOMMAND\_HPP // GAMECOMMAND\_HPP

#define GAMECOMMAND\_HPP

#include <string>

enum class GameCommand {

MOVE\_LEFT = 0,

MOVE\_RIGHT = 1,

MOVE\_UP = 2,

MOVE\_DOWN = 3,

ATTACK = 4,

USE\_ABILITY = 5,

LOAD = 6,

SAVE = 7,

QUIT = 8,

HELP = 9

};

static std::string commandToString(GameCommand command) {

{

switch (command) {

case GameCommand::MOVE\_UP:

return "Move up";

break;

case GameCommand::MOVE\_LEFT:

return "Move left";

break;

case GameCommand::MOVE\_DOWN:

return "Move down";

break;

case GameCommand::MOVE\_RIGHT:

return "Move right";

break;

case GameCommand::ATTACK:

return "Attack";

break;

case GameCommand::USE\_ABILITY:

return "Use ability";

break;

case GameCommand::LOAD:

return "Load";

break;

case GameCommand::SAVE:

return "Save";

break;

case GameCommand::QUIT:

return "Quit";

break;

case GameCommand::HELP:

return "Help";

break;

default:

return "Unknown command";

break;

}

}

}

#endif // GAMECOMMAND\_HPP

**Файл GameInputController.hpp:**

#ifndef GAMEINPUTCONTROLLER\_HPP // GAMEINPUTCONTROLLER\_HPP

#define GAMEINPUTCONTROLLER\_HPP

#include "Action.hpp"

#include "AttackAction.hpp"

#include "FileReader.hpp"

#include "FileWriter.hpp"

#include "GameCommand.hpp"

#include "GameOutputController.hpp"

#include "GameSession.hpp"

#include "ManualFieldProvider.hpp"

#include "MoveAction.hpp"

#include "RandomFieldProvider.hpp"

#include "UseAbilityAction.hpp"

template <class Input, class Output>

class GameInputController {

private:

GameSession& gameSession;

Input& input;

Output& output;

void load() {

std::string readPath =

input.requestString("Enter savefile path (by default: save.json):");

if (readPath.size() == 0) {

readPath = "save.json";

}

FileReader reader = FileReader(readPath);

reader.readGameState(gameSession.getGameState());

output.printMessage("Savefile loaded.");

}

void save() {

std::string loadPath =

input.requestString("Enter savefile path (by default: save.json):");

if (loadPath.size() == 0) {

loadPath = "save.json";

}

FileWriter writer = FileWriter(loadPath);

writer.writeGameState(gameSession.getGameState());

output.printMessage("Savefile written.");

}

public:

GameInputController(GameSession& gameSession, Input& input, Output& output)

: gameSession(gameSession), input(input), output(output) {}

bool run() {

GameCommand gameCommand = GameCommand::HELP;

while (gameCommand != GameCommand::QUIT) {

Action\* action = nullptr;

switch (gameCommand) {

case GameCommand::MOVE\_DOWN:

action = new MoveAction(0, 1);

break;

case GameCommand::MOVE\_UP:

action = new MoveAction(0, -1);

break;

case GameCommand::MOVE\_LEFT:

action = new MoveAction(-1, 0);

break;

case GameCommand::MOVE\_RIGHT:

action = new MoveAction(1, 0);

break;

case GameCommand::LOAD:

try {

load();

} catch (std::runtime\_error e) {

output.printError(e);

}

break;

case GameCommand::SAVE:

try {

save();

} catch (std::runtime\_error e) {

output.printError(e);

}

break;

case GameCommand::ATTACK:

action = new AttackAction();

break;

case GameCommand::USE\_ABILITY:

action = new UseAbilityAction();

break;

case GameCommand::HELP:

output.printMessage("Help:");

output.printMessage(input.getInfo());

break;

default:

break;

}

if (action != nullptr) {

gameSession.makeAction(action);

}

if (!gameSession.getStarted()) {

break;

}

gameCommand = input.requestCommand();

}

return gameCommand == GameCommand::QUIT;

}

bool setup() {

bool quit = false;

while (true) {

int option = input.requestInt(

1, 3, "Select option:\n1. New game\n2. Load savefile\n3. Quit");

if (option == 1) {

int rows = input.requestInt(1, 100, "Select field rows:");

int columns = input.requestInt(1, 100, "Select field columns:");

int s1 = input.requestInt(1, 100,

"Select count of ships length = 1:");

int s2 = input.requestInt(0, 100,

"Select count of ships length = 2:");

int s3 = input.requestInt(0, 100,

"Select count of ships length = 3:");

int s4 = input.requestInt(0, 100,

"Select count of ships length = 4:");

std::map<uint8\_t, uint8\_t> limits = {

{1, s1}, {2, s2}, {3, s3}, {4, s4}};

FieldProvider\* fieldProvider;

bool choice = input.requestChoice("Place ships randomly?");

if (choice) {

fieldProvider = new RandomFieldProvider();

} else {

fieldProvider =

new ManualFieldProvider<Input, Output>(input, output);

}

try {

gameSession.initialize(rows, columns, limits,

fieldProvider);

} catch (std::runtime\_error e) {

delete fieldProvider;

output.printError(e);

continue;

}

gameSession.start();

delete fieldProvider;

break;

}

if (option == 2) {

try {

load();

} catch (std::runtime\_error e) {

output.printError(e);

continue;

}

gameSession.start();

break;

}

if (option == 3) {

quit = true;

break;

}

}

return quit;

}

};

#endif // GAMEINPUTCONTROLLER\_HPP

**Файл GameObserver.hpp:**

#ifndef GAMEOBSERVER\_HPP // GAMEOBSERVER\_HPP

#define GAMEOBSERVER\_HPP

#include "ActionResult.hpp"

#include "GameState.hpp"

class GameObserver {

public:

virtual void onActionDenied() = 0;

virtual void onActionResult(ActionResult\* result) = 0;

virtual void onUserTurn(GameState& gameState) = 0;

virtual void onBotTurn() = 0;

virtual void onBotWin() = 0;

virtual void onUserWin() = 0;

};

#endif // GAMEOBSERVER\_HPP

**Файл GameOutputController.hpp:**

#ifndef GAMEOUTPUTCONTROLLER\_HPP // GAMEOUTPUTCONTROLLER\_HPP

#define GAMEOUTPUTCONTROLLER\_HPP

#include <iostream>

#include "GameObserver.hpp"

template <class Output>

class GameOutputController : public GameObserver {

private:

Output& output;

public:

GameOutputController(Output& output) : output(output) {}

void onActionDenied() override {

output.printMessage("Action usage denied.");

}

void onActionResult(ActionResult\* result) override {

output.printMessage(result->getResult());

}

void onUserTurn(GameState& gameState) override {

Player& user = gameState.getUser();

output.printMessage("Yout turn.");

output.printMessage("Your field:");

output.printField(user.getField());

output.printMessage("Enemy field:");

output.printField(user.getEnemy().getField(), true);

if (user.getAbilityContainer().getAbilitiesCount() != 0) {

output.printMessage("Available ability: " +

user.getAbilityContainer().peek());

}

}

void onBotTurn() override { output.printMessage("Bot turn."); }

void onBotWin() override { output.printMessage("Bot won. Game finished."); }

void onUserWin() override { output.printMessage("You won. Next round."); }

};

#endif // GAMEOUTPUTCONTROLLER\_HPP

**Файл ConsoleInput.cpp:**

#include <iostream>

#include <limits>

#include <sstream>

#include "../include/ConsoleInput.hpp"

ConsoleInput::ConsoleInput() {

config = {{'w', GameCommand::MOVE\_UP}, {'a', GameCommand::MOVE\_LEFT},

{'s', GameCommand::MOVE\_DOWN}, {'d', GameCommand::MOVE\_RIGHT},

{'z', GameCommand::ATTACK}, {'x', GameCommand::USE\_ABILITY},

{'l', GameCommand::LOAD}, {'p', GameCommand::SAVE},

{'h', GameCommand::HELP}, {'q', GameCommand::QUIT}};

}

GameCommand ConsoleInput::requestCommand(std::string msg) {

if (msg != "") {

std::cout << msg << std::endl;

}

std::string line;

while (true) {

if (std::cin.eof()) {

return GameCommand::QUIT;

}

std::getline(std::cin, line);

if (line.size() > 1 || config.count(line[0]) == 0) {

std::cout << "Unknown command." << std::endl;

} else {

return config[line[0]];

}

}

}

bool ConsoleInput::requestChoice(std::string msg) {

std::cout << msg << " (y/n):" << std::endl;

std::string source;

std::getline(std::cin, source);

if (source == "" || source == "y" || source == "yes") {

return true;

}

return false;

}

std::string ConsoleInput::requestString(std::string msg) {

if (msg != "") {

std::cout << msg << std::endl;

}

std::string source;

std::getline(std::cin, source);

return source;

}

void ConsoleInput::loadConfiguration(ControlConfiguration config) {

this->config = config;

}

std::string ConsoleInput::getInfo() {

std::stringstream ss;

for (auto it = config.begin(); it != config.end(); ++it) {

char key = it->first;

GameCommand command = it->second;

ss << key << ": " << commandToString(command) << std::endl;

}

return ss.str();

}

int ConsoleInput::requestInt(int min, int max, std::string msg) {

std::string line;

int i;

while (true) {

if (msg != "") {

std::cout << msg << std::endl;

}

std::getline(std::cin, line);

try {

i = std::stoi(line);

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

continue;

}

if (i < min || i > max) {

continue;

}

break;

}

return i;

}

}

**Файл ConsoleOutput.cpp:**

#include <cmath>

#include <iomanip>

#include <iostream>

#include "../include/ConsoleOutput.hpp"

void ConsoleOutput::printField(Field& field, bool isEnemy) {

uint8\_t rowsWidth = int(std::log10(field.getRows()) + 1);

uint8\_t columnsWidth = int(std::log10(field.getColumns()) + 1);

std::cout << std::setw(columnsWidth) << " ";

for (uint8\_t i = 0; i < field.getColumns(); ++i) {

std::cout << " " << std::internal << std::setw(columnsWidth) << +i;

}

std::cout << std::endl;

for (uint8\_t y = 0; y < field.getRows(); ++y) {

std::cout << std::right << std::setw(rowsWidth) << +y;

for (uint8\_t x = 0; x < field.getColumns(); ++x) {

Tile tile = field.getTile(x, y);

std::cout << " " << std::internal << std::setw(columnsWidth);

bool hidden = tile.tileState == TileState::HIDDEN;

bool hasShip = tile.ship != nullptr;

if (hidden && isEnemy) {

std::cout << "H";

}

if (hidden && !isEnemy && !hasShip) {

std::cout << "~";

}

if (hasShip && ((hidden && !isEnemy) || (!hidden))) {

std::cout << static\_cast<int>(

tile.ship->getSegment(field.getSegmentIndex(x, y)));

}

if (!hidden && !hasShip) {

std::cout << "X";

}

}

std::cout << std::endl;

}

}

void ConsoleOutput::printMessage(std::string message) {

std::cout << message << std::endl;

}

void ConsoleOutput::printError(std::runtime\_error error) {

std::cout << error.what() << std::endl;

}

void ConsoleOutput::printError(std::exception error) {

std::cout << error.what() << std::endl;

}

**Файл main.cpp:**

#include "../include/ConsoleInput.hpp"

#include "../include/ConsoleOutput.hpp"

#include "../include/FileReader.hpp"

#include "../include/GameInputController.hpp"

#include "../include/GameOutputController.hpp"

#include "../include/GameSession.hpp"

int main() {

ConsoleInput consoleInput = ConsoleInput();

ConsoleOutput consoleOutput = ConsoleOutput();

try {

Json::Value configObject = FileReader("config.json").readConfigFile();

ControlConfiguration config =

Deserializer().deserializeControlConfiguration(configObject);

consoleInput.loadConfiguration(config);

consoleOutput.printMessage("Loaded control config.");

} catch (std::runtime\_error e) {

consoleOutput.printError(e);

}

GameSession gameSession = GameSession();

GameOutputController<ConsoleOutput>\* outputController =

new GameOutputController<ConsoleOutput>(consoleOutput);

gameSession.addObserver(outputController);

GameInputController<ConsoleInput, ConsoleOutput> inputController =

GameInputController<ConsoleInput, ConsoleOutput>(

gameSession, consoleInput, consoleOutput);

bool quit = inputController.setup();

while (!quit) {

quit = inputController.run();

if (quit) {

break;

}

quit = inputController.setup();

}

delete outputController;

return 0;

}