**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 |  | Иванов Д. М. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

## Цель работы

Изучить основы шаблонных классов и функций. Через эту парадигму написать дополнения к игре: классы считывания и отрисовки в терминале, а также класс управления игры, принимающий в качестве параметра шаблона классы, которые определяет способ ввода команда и вывода информации. Связать данные классы с раннее написанными 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

1) Реализация класса, считывающего ввод пользователя из терминала

В классе будет сначала считывание клавиш для команд, а затем получение информации от пользователя и передача этих данных контроллеру.

class Input\_from\_terminal

std::map<char, COMMAND> commands\_map — словарь, который будет заполняться по ходу считывания файла. Ключ — клавиша для считывния из терминала, значение — соответствовующая ей команда

std::map<std::string, COMMAND> string\_commands — словарь перевода строковго представления команды в сам тип команды

Input\_from\_terminal(); - конструктор, заполняет string\_commands

void load\_commands(const std::string& filename); - загрузка данных из файла, содержащего клавиши для команд. Считываются циклом данные и передаются в словарь. Если появляются проблемы: или одинаковые клавиши, или одинковые команды, или неверное название команд, или неверный формат текстового файла — будет возвращено исключение и заполнение словаря значениями по умолчанию

void default\_arguments(); - заполнение словаря значениями по умолчанию

void read\_sizes(int& width, int& height); - чтение размеров поля

bool check\_input\_numbers(int x, int y); - проверка, что числовые значения (для кординат и размеров поля) введены верно

COMMAND read\_command(); - чтение команды от пользователя и вовзрат соответсвующего типа

Coords read\_coords(); - чтение координат целочисленного типа. Если ввод неверный, то происходит соответсвующая обработка

void read\_ships(std::vector<Length\_of\_the\_ship>& length\_of\_ships, std::vector<Coords>& coords\_of\_ships, std::vector<Orientation>& orientations\_of\_ships); - чтение данных для кораблей. Передаются ссылки необходимых векторов (ориентация, координаты и длины) и идет их заполнение.

std::map<char, COMMAND> get\_map\_of\_commands() const; - возврат словаря с командами и их клавишами

2) Реализация классов отрисовки игры

Для отображения всех изменений игры в консоль реализуем 3 класса:

Paint — класс отрисовки поля. Имеет 3 метода

void print\_your\_ground(Playground& p); - отрисовка поля игрока. Это значит, что будет отображаться жизнь каждого сегмента корабля.

void print\_enemy\_ground(Playground& p); - отрисовка поля врага. Все сегменты будут изначально скрыты. И будет отображения его жизни только после 1-ого попадания.

void print\_ground(Playground& p, const char c); - основная логика отрисовки поля. Жизнь сегмента выводится в зависмости от того, какой из 2-ух методов выше вызвал print\_ground.

Output\_to\_terminal — класс, выводящий текстовую информацию в терминал.

std::map<COMMAND, std::string> commands\_to\_string; - словарь для перевода команды в ее соответсвующее строковое представление

std::map<Name\_of\_builder, std::string> builder\_to\_string; - словарь для перевода способности в ее соответсвующее строковое представление

Display<PaintType, OutputType> - шаблонный класс отображения игры. Производит отрисовку игры для ее нынешнего состояния. То, как происходит отрисовка игры определяется классами переданном в качестве параметра шаблона (как будут выводится сообщения, и как будет выглядеть поля).

OutputType\* output; - переданный класс вывода информации

PaintType painter; - переданный класс отрисовки поля

std::map<std::string, std::function<void()>> commands\_to\_function; - словарь, необходимый для выполнения нужного вывода информации, для которой не требуются аргументы. То есть подается строка, что нужно сделать. И идет вызов нужного функционала.

void execute\_message(std::string command);

void print\_player\_ground(Playground& playground);

void print\_enemy\_ground(Playground& playground);

void print\_commands(std::map<char, COMMAND> commands\_map);

void print\_builder(Name\_of\_builder builder);

void print\_player\_shoot\_result(bool flag, Coords coord);

void print\_enemy\_turn(Coords coord);

void print\_round(int number\_of\_round);

void print\_scanner\_result(bool result); - методы для вывода пользователю текущего состояния игры. Все они принимают необходимые аргументы и по ним в нужный момент времени выводят данные.

3) Реализация класса управления игры.

Game\_controller<InputType, OutputType, PaintType> - шаблонный класс управления игрой. Принимает классы, определяющие способ ввода команды и вывода информации.

Game\* game; - сслыка на игру

InputType\* input; - тип ввода данных

Display<PaintType, OutputType> display; - класс отрисовки игры, написанный ранее и принимающий в качестве шаблонный параметров тип вывода и ввода.

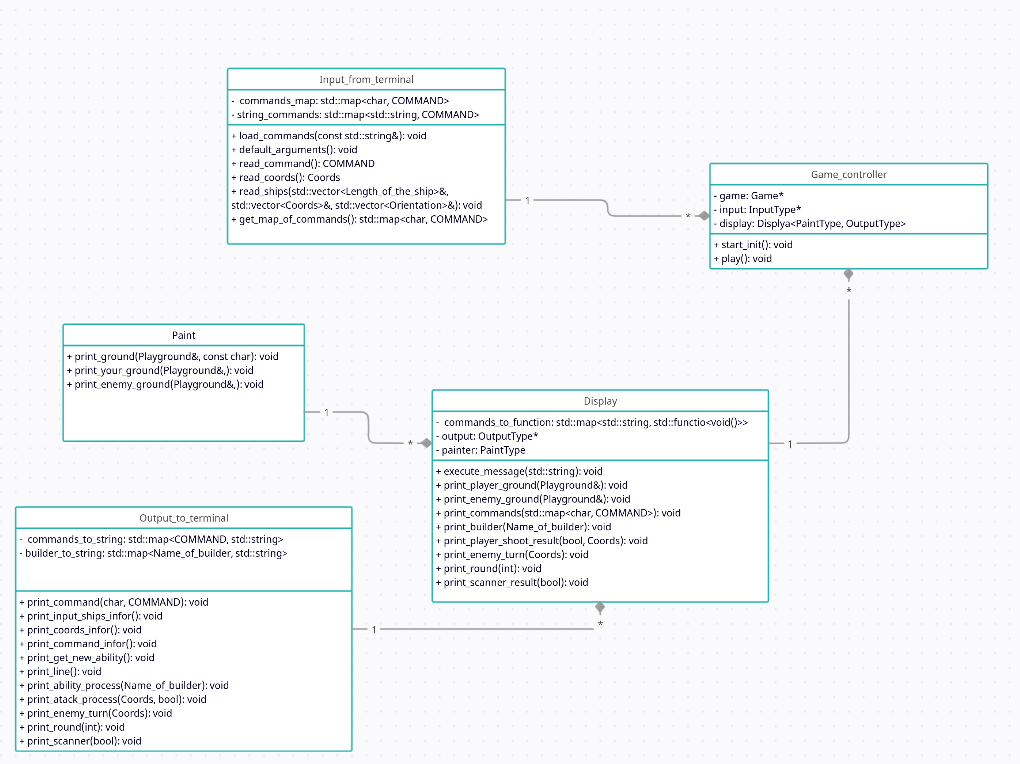
void start\_init(); - стартовая иницализация игры. Сначала читается в input команды и их клавиши из файла. Потом выводится информация пользователю о доступном функционале и идет ожидание ввода команды. Сначала необъодимо выбрать, либо игроку самому заполнить корабли и начать новую игру. Или загрузить игру из файла и продолжить предыдущую игру. При вводе других клавиш ввод перезапрашивается. Для ввода команды и вывода пользователю соответсвующих состояний используются методы раннее написанных классов input и output.

void play(); - после стартовой иницализаиции идет сам цикл игры. Принимается команда от пользователя и вызов соответсвующих методов из game. После каждого хода через output выводятся текцщие состояния полей игрока и врага.

При выборе атаки ожидается ввод координат от игрока (input->read\_coords()) и выполнение атаки в методе game. При выборе применения способности берется из игры следующая способность, если она есть. И определяется, нужны ли координаты для нее или нет. Выводится пользователю название выполняющейся способности и переход к атаке. Для сохранения вызывается метод игры с переданным названием файла.

## Представление в UML-диаграмме

Рисунок 1 — UML диаграмма



## Выводы

Была изучена шаблонные классы. В дополнение к игре морского боя были добавлены отдельные классы для ввода, вывода и контроля игры. Связав эти классы с раннее написанными сущностями, была полность доделана игра с возможность дополнения в виде GUI.

# Приложение А Исходный код программы

Название файла: Game\_controller.h

#ifndef GAME\_CONTROLLER\_H

#define GAME\_CONTROLLER\_H

#include <iostream>

#include "Input\_from\_terminal.h"

#include "Output\_to\_terminal.h"

#include "Game.h"

#include "Display.h"

template <typename InputType, typename OutputType, typename PaintType>

class Game\_controller{

private:

Game\* game;

InputType\* input;

//OutputType\* output;

Display<PaintType, OutputType> display;

public:

Game\_controller(Game\* game);

~Game\_controller();

void play();

void start\_init();

};

#endif

Название файла: Input\_from\_terminal\_state.h

#ifndef TREMINAL\_INPUT\_H

#define TREMINAL\_INPUT\_H

#include <iostream>

#include <map>

#include <limits>

#include <set>

#include "Commands.h"

#include "FileWrapper.h"

#include "Playground.h"

class Input\_from\_terminal{

private:

std::map<char, COMMAND> commands\_map;

std::map<std::string, COMMAND> string\_commands;

public:

Input\_from\_terminal();

void load\_commands(const std::string& filename);

void default\_arguments();

COMMAND read\_command();

Coords read\_coords();

void read\_ships(std::vector<Length\_of\_the\_ship>& length\_of\_ships, std::vector<Coords>& coords\_of\_ships, std::vector<Orientation>& orientations\_of\_ships);

std::map<char, COMMAND> get\_map\_of\_commands() const;

};

#endif

Название файла: Output\_to\_terminal.h

#ifndef OUTPUT\_TERMINAL\_H

#define OUTPUT\_TERMINAL\_H

#include <iostream>

#include <map>

#include "Commands.h"

#include "Paint.h"

#include "Playground.h"

class Output\_to\_terminal{

private:

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

std::map<Name\_of\_builder, std::string> builder\_to\_string;

public:

Output\_to\_terminal();

void print\_command(char symbol, COMMAND command);

void print\_input\_ships\_infor();

void print\_coords\_infor();

void print\_command\_infor();

void print\_get\_new\_ability();

void print\_line();

void print\_ability\_process(Name\_of\_builder builder);

void print\_atack\_process(Coords coord, bool is\_good\_hit);

void print\_enemy\_turn(Coords coord);

void print\_round(int number);

void print\_scanner(bool result);

};

#endif