**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. 3381 |  | Марков М.М. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать гибкую систему управления и отображения игры, обеспечивающую модульность и легкость замены компонентов. Реализовать механизм считывания и обработки команд, а также проверку корректности их назначения.

## **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**

В этой лабораторной работе была разработана архитектура для управления игрой с использованием шаблонных классов. Основной целью было создание гибкой системы, которая разделяет логику игры, обработку ввода и отображение состояния игры. В процессе работы были реализованы следующие ключевые компоненты:

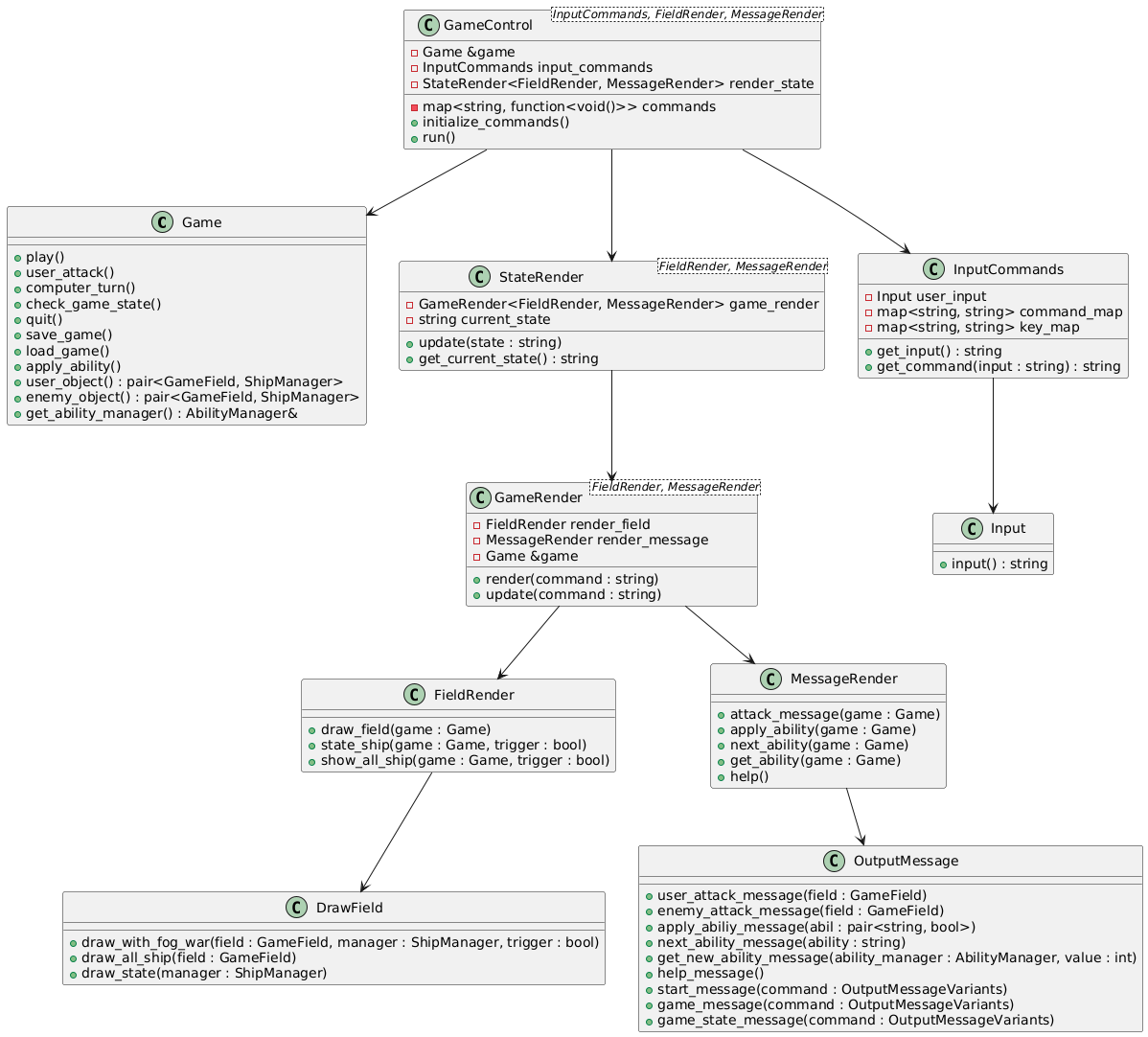
1. Шаблонный класс управления игрой (GameControl):  
   Этот класс служит связующим звеном между пользователем и игрой. Он принимает команды от пользователя, которые были преобразованы в объекты команд через обработку ввода, и вызывает соответствующие методы игры для выполнения этих команд. В качестве шаблонного параметра класс принимает обработчик ввода, что позволяет легко подменить способ ввода (например, использование консоли или графического интерфейса). После получения команды, класс GameControl вызывает методы игрового класса для выполнения нужных действий, например, для перемещения объектов или изменения состояния игры.
2. Классы отображения игры (StateRender, FieldRender, MessageRender):  
   В процессе отрисовки игры были реализованы три отдельных класса, каждый из которых отвечает за свой аспект отображения:
   * StateRender: Этот класс отвечает за отображение текущего состояния игры, например, информацию о том, чья сейчас очередь, статус игры (победа/проигрыш), и другие данные, которые описывают текущую ситуацию в игре. Это позволяет пользователю легко отслеживать, что происходит в игре в любой момент.
   * FieldRender: Класс, который отвечает за отрисовку игрового поля. В нем отображаются позиции объектов (например, кораблей), их текущее состояние (повреждения, уничтожение и т.д.). Он обновляется в ответ на действия игрока, такие как атаки или перемещения объектов, и отображает актуальное состояние поля.
   * MessageRender: Класс, предназначенный для отображения сообщений и подсказок для игрока, таких как ошибки ввода, успешные или неудачные действия и другие сообщения, которые информируют игрока о происходящем в игре.

Эти классы обеспечивают разделение логики отображения, позволяя легко адаптировать вывод на различные устройства или интерфейсы, а также облегчают замену или расширение способа отображения, например, переход от текстового консольного вывода к графическому интерфейсу (GUI), не затрагивая саму логику игры.

1. Класс обработки ввода (InputCommands):  
   Класс InputCommands отвечает за обработку ввода пользователя. Он считывает символы с клавиатуры, преобразует их в команды и передает в GameControl. Введённые команды сопоставляются с соответствующими действиями в игре через файл конфигурации, где прописаны клавиши для каждой команды. В случае отсутствия конфигурации по умолчанию используются стандартные клавиши. Важно, что перед обработкой команды выполняется проверка на уникальность сопоставления клавиш и команд, а также на отсутствие конфликтов между клавишами.
2. Архитектура взаимодействия:  
   Вся система взаимодействует через четко определенные интерфейсы, что позволяет легко заменять или модифицировать отдельные компоненты, не влияя на основную логику игры. Класс GameControl принимает команды от обработчика ввода (например, через InputCommands) и передает их в класс игры для выполнения. После этого данные обновляются и передаются в рендеры (например, StateRender, FieldRender, MessageRender) для отображения текущего состояния игры пользователю. Это позволяет обеспечить чистую и модульную архитектуру.
3. Шаблонные классы и их роль:  
   Шаблонные классы играют ключевую роль в гибкости и расширяемости программы. Например, класс GameControl может работать с любым классом ввода и отображения, который соответствует нужному интерфейсу. Это упрощает замену и расширение отдельных частей системы, таких как методы обработки ввода или рендеринг. Это также позволяет сделать систему легко адаптируемой к разным устройствам ввода/вывода.
4. Применение принципов ООП:  
   В ходе реализации работы активно использовались принципы объектно-ориентированного программирования: инкапсуляция (сокрытие реализации в каждом классе), абстракция (представление действий через интерфейсы) и наследование/полиморфизм (позволяют расширять функционал классов и заменять компоненты системы без изменения основной логики).

Разработанный программный код см. в приложении А

**Диаграмма классов.**

****

**Выводы**

В результате выполнения лабораторной работы была создана гибкая и модульная система управления игрой, которая позволяет легко изменять способы ввода и отображения, не затрагивая основную логику игры. Использование шаблонных классов обеспечило необходимую гибкость и расширяемость, а также позволило реализовать четкую и понятную архитектуру. Разделение системы на классы, такие как StateRender, FieldRender и MessageRender, позволяет легко адаптировать программу под различные интерфейсы и устройства, повышая её гибкость и удобство в дальнейшем расширении.

**Приложение А  
Исходный код программы**

Название файла: main.cpp

#include <iostream>

#include <stdexcept>

#include "game.h"

#include "input/inputCommands.h"

#include "input/gameControl.h"

#include "output/fieldRender.h"

#include "output/messageRender.h"

class Game;

int main() {

try {

Game game;

GameControl<InputCommands, FieldRender, MessageRender> controller(game);

controller.run();

} catch (const std::exception& e) {

std::cerr << "Exception: " << e.what() << std::endl;

}

return 0;

}

Название файла: gameControl.h

#ifndef GAME\_CONTROL\_H

#define GAME\_CONTROL\_H

#include <iostream>

#include <map>

#include <functional>

#include "../output/drawField.h"

#include "../output/outputMessage.h"

#include "../output/gameRender.h"

#include "../output/stateRender.h"

#include "../state/gameState.h"

#include "../gameField.h"

#include "../game.h"

template<typename InputCommands, typename FieldRender, typename MessageRender>

class GameControl {

private:

Game &game;

InputCommands input\_commands;

StateRender<FieldRender, MessageRender> render\_state;

std::map<std::string, std::function<void()>> commands;

public:

GameControl(Game &g) : game(g), render\_state(g) { initialize\_commands(); }

void initialize\_commands();

void run();

};

template<typename InputCommands, typename FieldRender, typename MessageRender>

void GameControl<InputCommands, FieldRender, MessageRender>::initialize\_commands() {

commands["attack"] = [&]() {

game.user\_attack();

render\_state.update("get\_new\_ability\_message");

render\_state.update("new\_game");

};

commands["state\_my\_ships"] = [&]() { render\_state.update("state\_my\_ships"); };

commands["state\_enemy\_ships"] = [&]() { render\_state.update("state\_enemy\_ships"); };

commands["quit"] = [&]() { game.quit(); };

commands["fields"] = [&]() { render\_state.update("fields"); };

commands["show\_enemy\_ships"] = [&]() { render\_state.update("show\_enemy\_ships"); };

commands["show\_my\_ships"] = [&]() { render\_state.update("show\_my\_ships"); };

commands["next\_ability"] = [&]() { render\_state.update("next\_ability"); };

commands["save\_game"] = [&]() { game.save\_game(); };

commands["load\_game"] = [&]() { game.load\_game(); };

commands["apply\_ability"] = [&]() {

game.apply\_ability();

render\_state.update("apply\_ability");

};

commands["help"] = [&]() { render\_state.update("help"); };

}

template<typename InputCommands, typename FieldRender, typename MessageRender>

void GameControl<InputCommands, FieldRender, MessageRender>::run() {

game.play();

while (true) {

std::string input = input\_commands.get\_input();

if (commands.find(input) != commands.quit()) {

commands[input]();

} else {

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

}

if (!game.trigger\_bot\_attack) continue;

game.computer\_turn();

game.check\_game\_state();

render\_state.update("attack\_message");

render\_state.update("fields");

}

}

#endif

Название файла: input.h

#ifndef INPUT\_H

#define INPUT\_H

#include <iostream>

#include <string>

class Input {

public:

std::string input();

};

#endif

Название файла: input.cpp

#include "input.h"

std::string Input::input() {

std::string user\_input;

std::getline(std::cin, user\_input);

return user\_input;

}

Название файла: inputCommands.h

#ifndef INPUT\_COMMANDS\_H

#define INPUT\_COMMANDS\_H

#include <iostream>

#include <map>

#include <functional>

#include <utility>

#include "../shipManager.h"

#include "../ship.h"

#include "../state/gameState.h"

#include "../gameField.h"

#include "input.h"

#include "../exception.h"

#include "../nlohmann/json.hpp"

using json = nlohmann::json;

class InputCommands {

private:

Input user\_input;

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

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

public:

InputCommands();

std::string get\_input();

std::string get\_command(const std::string& input);

};

#endif  
  
Название файла: inputCommands.cpp

#include "inputCommands.h"

InputCommands::InputCommands() {

const std::string& filename = "commands.json";

if (std::ifstream file{filename}) {

json j;

file >> j;

for (auto& element : j.items()) {

const std::string command = element.key();

const std::string key = element.value();

if (key\_map.find(key) != key\_map.end()) {

throw std::invalid\_argument("Duplicate key assignment: " + key);

}

if (command\_map.find(command) != command\_map.end()) {

throw std::invalid\_argument("Duplicate command assignment: " + command);

}

command\_map[command] = key;

key\_map[key] = command;

}

} else {

std::cerr << "Error opening " << filename << ", loading default commands." << std::endl;

if (std::ifstream default\_file{"default.json"}) {

json j;

default\_file >> j;

for (auto& element : j.items()) {

const std::string command = element.key();

const std::string key = element.value();

if (key\_map.find(key) != key\_map.end()) {

throw std::invalid\_argument("Duplicate key assignment in default commands: " + key);

}

if (command\_map.find(command) != command\_map.end()) {

throw std::invalid\_argument("Duplicate command assignment in default commands: " + command);

}

command\_map[command] = key;

key\_map[key] = command;

}

} else {

throw std::runtime\_error("Error opening default command file");

}

}

}

std::string InputCommands::get\_input() {

while (true) {

std::cout << "Enter command: ";

std::string input = user\_input.input();

try {

return get\_command(input);

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

std::cerr << e.what() << ". Please try again." << std::endl;

}

}

}

std::string InputCommands::get\_command(const std::string& input) {

if (command\_map.find(input) == command\_map.end()) {

throw std::invalid\_argument("Unknown command: " + input);

}

return command\_map[input];

}

Название файла: gameRender.h

#ifndef GAME\_RENDER\_H

#define GAME\_RENDER\_H

#include <iostream>

#include "../game.h"

class Game;

template<typename FieldRender, typename MessageRender>

class GameRender {

private:

FieldRender render\_field;

MessageRender render\_message;

Game &game;

public:

GameRender(Game &g) : game(g) {}

void render(const std::string &command);

void update(const std::string &command);

};

template<typename FieldRender, typename MessageRender>

void GameRender<FieldRender, MessageRender>::render(const std::string &command) {

if (command == "fields") {

render\_field.draw\_field(game);

} else if (command == "attack\_message") {

render\_message.attack\_message(game);

} else if (command == "state\_my\_ships") {

render\_field.state\_ship(game, 1);

} else if (command == "state\_enemy\_ships") {

render\_field.state\_ship(game, 0);

} else if (command == "show\_enemy\_ships") {

render\_field.show\_all\_ship(game, 0);

} else if (command == "show\_my\_ships") {

render\_field.show\_all\_ship(game, 1);

} else if (command == "apply\_ability") {

render\_message.apply\_ability(game);

} else if (command == "next\_ability") {

render\_message.next\_ability(game);

} else if (command == "get\_new\_ability\_message") {

render\_message.get\_ability(game);

} else if (command == "help") {

render\_message.help();

}

}

template<typename FieldRender, typename MessageRender>

void GameRender<FieldRender, MessageRender>::update(const std::string &command) {

render(command);

}

#endif  
  
Название файла: fieldRender.h

#ifndef FIELD\_RENDERER\_H

#define FIELD\_RENDERER\_H

#include <iostream>

#include <utility>

#include "../game.h"

class FieldRender {

public:

void draw\_field(Game &game);

void state\_ship(Game &game, bool trigger);

void show\_all\_ship(Game &game, bool trigger);

};

#endif

Название файла: fieldRender.cpp

#include "fieldRender.h"

void FieldRender::draw\_field(Game &game) {

DrawField field;

auto user\_data = game.user\_object();

auto enemy\_data = game.enemy\_object();

field.draw\_with\_fog\_war(user\_data.first, user\_data.second, 0);

field.draw\_with\_fog\_war(enemy\_data.first, enemy\_data.second, 1);

}

void FieldRender::state\_ship(Game &game, bool trigger) {

DrawField field;

auto user\_data = game.user\_object();

auto enemy\_data = game.enemy\_object();

if(trigger == 0) {

field.draw\_state(enemy\_data.second);

} else {

field.draw\_state(user\_data.second);

}

}

void FieldRender::show\_all\_ship(Game &game, bool trigger) {

DrawField field;

auto user\_data = game.user\_object();

auto enemy\_data = game.enemy\_object();

if(trigger == 0) {

field.draw\_all\_ship(enemy\_data.first);

} else {

field.draw\_all\_ship(user\_data.first);

}

}

Название файла: drawField.h

#ifndef DRAW\_FIELD\_H

#define DRAW\_FIELD\_H

#include <iostream>

#include <utility>

#include "../shipManager.h"

#include "../ship.h"

#include "../gameField.h"

class DrawField {

public:

void draw\_with\_fog\_war(GameField& field, ShipManager& manager, bool trigger);

void draw\_all\_ship(GameField& field);

void draw\_state(ShipManager& manager);

};

#endif

Название файла: drawField.cpp

#include "drawField.h"

void DrawField::draw\_with\_fog\_war(GameField& field, ShipManager& manager, bool trigger) {

std::cout << std::endl;

if(trigger == 0)

std::cout << "User field: " << std::endl;

else

std::cout << "Enemy field: " << std::endl;

std::cout << " ";

for (int x = 0; x < field.get\_width(); ++x) {

std::cout << x << " ";

}

std::cout << std::endl;

for (int y = 0; y < field.get\_height(); ++y) {

std::cout << y << " ";

for (int x = 0; x < field.get\_width(); ++x) {

bool is\_damaged = false;

bool is\_hit = false;

for (int i = 0; i < field.count\_ships; ++i) {

Ship& ship = manager.get\_ship(i);

int ship\_length = ship.get\_length();

bool is\_vertical = ship.is\_vertical\_orientation();

if (is\_vertical) {

if (x == ship.get\_x()) {

for (int j = 0; j < ship\_length; ++j) {

if (y == ship.get\_y() + j) {

if (ship.get\_segment\_state(j) == 2) {

is\_damaged = true;

} else if (ship.get\_segment\_state(j) == 1) {

is\_hit = true;

}

break;

}

}

}

} else {

if (y == ship.get\_y()) {

for (int j = 0; j < ship\_length; ++j) {

if (x == ship.get\_x() + j) {

if (ship.get\_segment\_state(j) == 2) {

is\_damaged = true;

} else if (ship.get\_segment\_state(j) == 1) {

is\_hit = true;

}

break;

}

}

}

}

if (is\_damaged || is\_hit) break;

}

if (field.get\_cell\_status(y, x) == 0) {

std::cout << ". ";

} else if (field.get\_cell\_status(y, x) == 1) {

std::cout << "~ ";

} else if (is\_damaged) {

std::cout << "D ";

} else if (is\_hit) {

std::cout << "H ";

} else {

std::cout << ". ";

}

}

std::cout << std::endl;

}

}

void DrawField::draw\_all\_ship(GameField& field) {

std::cout << std::endl;

std::cout << "Ships on the field: " << std::endl;

std::cout << " ";

for (int x = 0; x < field.get\_width(); ++x) {

std::cout << x << " ";

}

std::cout << std::endl;

for (int y = 0; y < field.get\_height(); ++y) {

std::cout << y << " ";

for (int x = 0; x < field.get\_width(); ++x) {

if (field.get\_cell\_status(y, x) == 2) {

std::cout << "S ";

} else if (field.get\_cell\_status(y, x) == 0 || field.get\_cell\_status(y, x) == 1) {

std::cout << "~ ";

}

}

std::cout << std::endl;

}

}

void DrawField::draw\_state(ShipManager& manager) {

std::vector<std::unique\_ptr<Ship>>& ships = manager.get\_ships();

for(int i = 0; i < ships.size(); i++) {

Ship& ship = \*ships[i];

int len\_ship = ship.get\_length();

int count\_destroy = 0;

std::cout << "Ship " << i+1 <<": ";

for(int i = 0; i < len\_ship; i++) {

int state = ship.get\_segment\_state(i);

if(state == 0) {

std::cout << "i"; // intact

} else if (state == 1) {

std::cout << "h"; // hit

} else if (state == 2) {

std::cout << "d"; // destroyed

count\_destroy++;

}

std::cout << " ";

}

if(count\_destroy == len\_ship) {

std::cout << " " << "Ship destroy";

}

std::cout << std::endl;

}

}

Название файла: messageRender.h

#ifndef MESSAGE\_RENDERER\_H

#define MESSAGE\_RENDERER\_H

#include <iostream>

#include <utility>

#include "../game.h"

#include "../abilityManager.h"

class MessageRender {

public:

void attack\_message(Game &game);

void apply\_ability(Game &game);

void next\_ability(Game &game);

void get\_ability(Game &game);

void help();

};

#endif

class FieldRender {

public:

void draw\_field(Game &game);

void state\_ship(Game &game, bool trigger);

void show\_all\_ship(Game &game, bool trigger);

};

#endif

Название файла: messageRender.cpp

#include "messageRender.h"

void MessageRender::attack\_message(Game &game) {

OutputMessage message;

auto user\_data = game.user\_object();

auto enemy\_data = game.enemy\_object();

message.user\_attack\_message(enemy\_data.first);

message.enemy\_attack\_message(user\_data.first);

}

void MessageRender::apply\_ability(Game &game) {

OutputMessage message;

message.apply\_abiliy\_message(game.abil);

}

void MessageRender::next\_ability(Game &game) {

OutputMessage message;

AbilityManager& ability\_manager = game.get\_ability\_manager();

std::string name\_ability = ability\_manager.next\_abilities();

message.next\_ability\_message(name\_ability);

}

void MessageRender::get\_ability(Game &game) {

OutputMessage message;

AbilityManager& ability\_manager = game.get\_ability\_manager();

int value = ability\_manager.get\_is\_grand\_random\_ability();

message.get\_new\_ability\_message(ability\_manager, value);

}

void MessageRender::help() {

OutputMessage message;

message.help\_message();

}

Название файла: outputMessage.h

#ifndef OUTPUT\_MESSAGE\_H

#define OUTPUT\_MESSAGE\_H

#include <iostream>

#include <map>

#include <functional>

#include <utility>

#include "../shipManager.h"

#include "../ship.h"

#include "../abilityManager.h"

#include "../state/gameState.h"

#include "../gameField.h"

#include "../outputMessageEnum.h"

class OutputMessage {

public:

void user\_attack\_message(GameField& field);

void enemy\_attack\_message(GameField& field);

void apply\_abiliy\_message(std::pair<std::string, bool> abil);

void next\_ability\_message(std::string ability);

void get\_new\_ability\_message(AbilityManager& ability\_manager, int value);

void help\_message();

void start\_message(OutputMessageVariants command);

void game\_message(OutputMessageVariants command);

void game\_state\_message(OutputMessageVariants command);

};

#endif

Название файла: outputMessage.cpp

#include "outputMessage.h"

void OutputMessage::user\_attack\_message(GameField& field) {

if(field.get\_last\_attemp\_user() == 0) {

std::cout << "You miss!" << std::endl;

} else {

std::cout << "You hit!" << std::endl;

}

}

void OutputMessage::enemy\_attack\_message(GameField& field) {

if(field.get\_last\_attemp\_enemy() == 0) {

std::cout << "Bot miss!" << std::endl;

} else {

std::cout << "Bot hit!" << std::endl;

}

}

void OutputMessage::apply\_abiliy\_message(std::pair<std::string, bool> ability) {

if(ability.first == "DoubleDamage") {

std::cout << "DoubleDamage" << std::endl;

if(ability.second == 1) std::cout << "Next hit deals double damage" << std::endl;

} else if(ability.first == "Scanner") {

std::cout << "Scanner" << std::endl;

if(ability.second == 0) std::cout << "No ships in the area" << std::endl;

else if (ability.second == 1) std::cout << "Ship detected in the area" << std::endl;

} else if (ability.first == "Bombard") {

std::cout << "Bombard" << std::endl;

} else {

std::cout << "No abilities available." << std::endl;

}

}

void OutputMessage::next\_ability\_message(std::string ability) {

std::cout << "Next ability: ";

std::cout << ability << std::endl;

}

void OutputMessage::get\_new\_ability\_message(AbilityManager& ability\_manager, int value) {

if(value == 1) {

std::cout << "New ability gained." << std::endl;

ability\_manager.set\_is\_grand\_random\_ability(0);

}

}

void OutputMessage::help\_message() {

std::cout << "attack - a" << "\n"

<< "state\_my\_ships - ms" << "\n"

<< "state\_enemy\_ships - es" << "\n"

<< "quit - q" << "\n"

<< "show\_enemy\_ships - hack to view enemy ships" << "\n"

<< "show\_my\_ships - view just my ships" << "\n"

<< "fields - view the playing fields" << "\n"

<< "abilities - look next ability" << "\n"

<< "stop\_game - the command to control serialization" << "\n"

<< "apply\_ability - apply ability" << std::endl;

}

void OutputMessage::start\_message(OutputMessageVariants command) {

switch(command) {

case OutputMessageVariants::INPUT\_WIDTH:

std::cout << "Enter field width: ";

break;

case OutputMessageVariants::INPUT\_HEIGHT:

std::cout << "Enter field height: ";

break;

case OutputMessageVariants::INVALID\_INPUT:

std::cout << "Invalid input. Please enter a positive integer." << std::endl;

break;

case OutputMessageVariants::INPUT\_COORDINATE:

std::cout << "Enter coordinates of the ship and orientation: ";

break;

case OutputMessageVariants::INVALID\_COORDINATE:

std::cout << "Invalid input for coordinates. Please enter valid integers." << std::endl;

break;

case OutputMessageVariants::RE\_ENTER\_COORDINATE:

std::cout << "Please re-enter coordinates of the ship and orientation." << std::endl;

break;

}

}

void OutputMessage::game\_message(OutputMessageVariants command) {

switch(command) {

case OutputMessageVariants::ATTACK\_COORDINATE:

std::cout << "Coordinate for attack: ";

break;

case OutputMessageVariants::SCANNER\_COORDINATE:

std::cout << "Coordinate for ability: ";

break;

case OutputMessageVariants::OPTION\_LOAD\_GAME:

std::cout << "Do you want load game?" << std::endl;

break;

case OutputMessageVariants::COORDINATE\_OUT\_OF\_RANGE:

std::cout << "Attack coordinates are out of range. Please enter valid coordinates." << std::endl;

break;

case OutputMessageVariants::INVALID\_COORDINATE\_FOR\_ATTACK:

std::cout << "Invalid input. Please enter valid integers for coordinates." << std::endl;

break;

}

}

void OutputMessage::game\_state\_message(OutputMessageVariants command) {

switch(command) {

case OutputMessageVariants::LOST:

std::cout << "You lost!" << std::endl;

break;

case OutputMessageVariants::QUES\_START\_NEW\_GAME:

std::cout << "Do you want to start a new game: " << std::endl;

break;

case OutputMessageVariants::START\_NEW\_GAME:

std::cout << "Starting a new game." << std::endl;

break;

case OutputMessageVariants::WIN:

std::cout << "You won this round!" << std::endl;

break;

case OutputMessageVariants::QUES\_START\_NEW\_ROUND:

std::cout << "Do you want to start a new round: ";

break;

case OutputMessageVariants::START\_NEW\_ROUND:

std::cout << "Starting the next round." << std::endl;

break;

}

}

Название файла: stateRender.h

#ifndef STATE\_RENDER\_H

#define STATE\_RENDER\_H

#include <string>

#include "../game.h"

template<typename FieldRender, typename MessageRender>

class StateRender {

private:

GameRender<FieldRender, MessageRender> game\_render;

std::string current\_state;

public:

StateRender(Game &game) : game\_render(game), current\_state("") {}

void update(const std::string &state);

std::string get\_current\_state() const;

};

template<typename FieldRender, typename MessageRender>

void StateRender<FieldRender, MessageRender>::update(const std::string &state) {

current\_state = state;

game\_render.update(state);

}

template<typename FieldRender, typename MessageRender>

std::string StateRender<FieldRender, MessageRender>::get\_current\_state() const {

return current\_state;

}

#endif