

МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра МО ЭВМ

Отчет
по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Связывание классов

Студент гр. 3382

Яковлев Д. С.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы

Реализовать класс игры, сохранение и загрузку состояния игры.

Задание

а. Создать класс игры, который реализует следующий игровой цикл:

- i. Начало игры
- ii. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
- iii. В случае проигрыша пользователь начинает новую игру
- iv. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

- b. Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

- Класс игры может знать о игровых сущностях, но не наоборот
- Игровые сущности не должны сами порождать объекты состояния
- Для управления самой игрой можно использовать обертки над командами
- При работе с файлом используйте идиому RAII.

Выполнение работы

Описание архитектуры лабораторной работы: (см. приложение А, полная архитектура в приложении С).

Был создан класс *Game* и *GameState*. Класс *Game* хранит в себе менеджер кораблей и поля для игрока и бота, а также менеджер способностей игрока. Логика игры была помещена в методы класса, «говорящие сами за себя»: *new_game*, *play_round*, *play_next_level* (при победе для пересоздания поля компьютера), *save*, *load*. Для работы с файлами была использована идиома RAII. *GameState* – это оболочка над *Game*, используемая для упрощения создания цикла игры (в частности загрузки и сохранения).

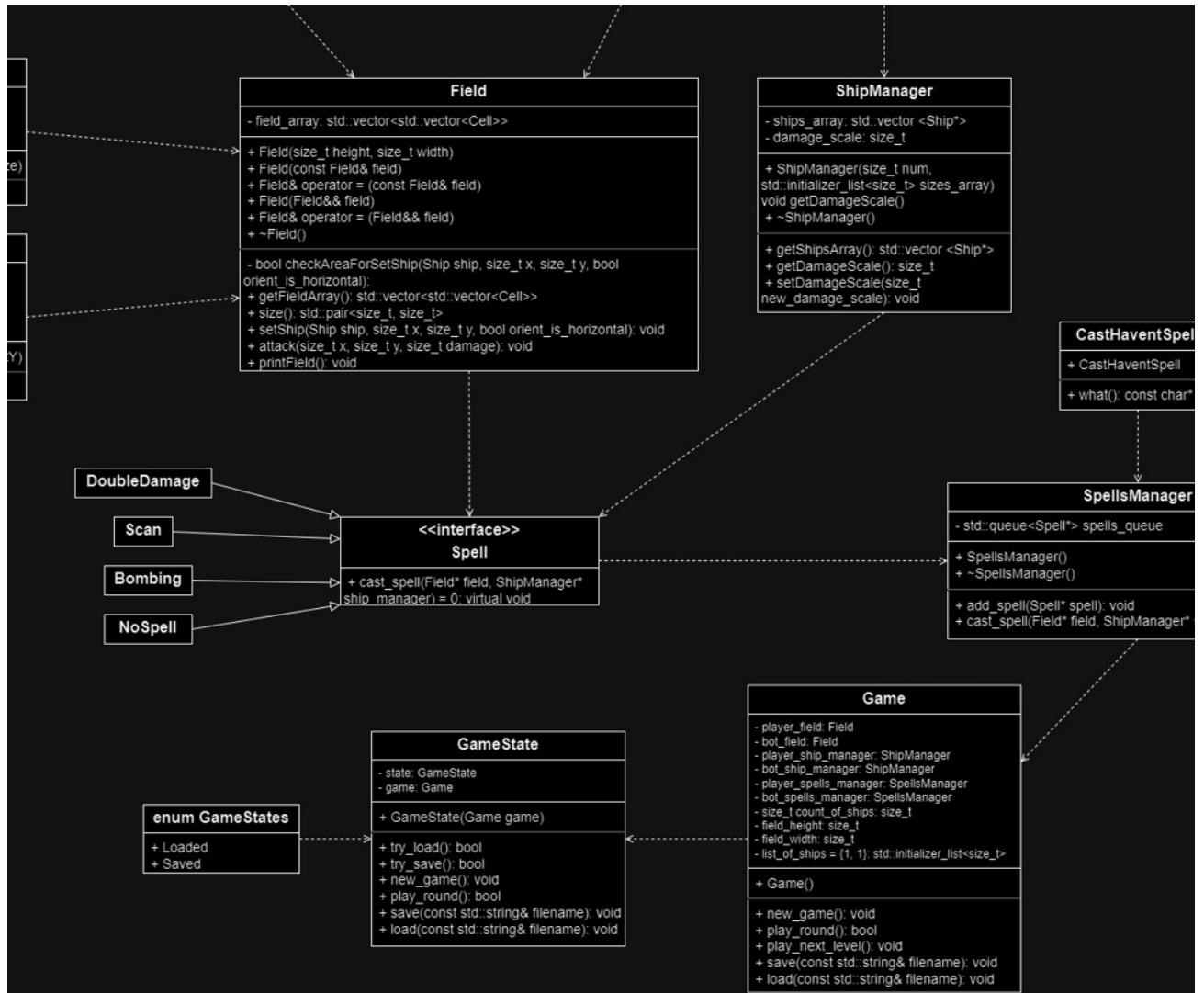
Были реализованы сохранение и загрузка, причём сохранение всегда начинается при расставлении кораблей. Были написаны операторы ввода и вывода в поток, однако они нигде не используются, поэтому их логика – «заглушка».

Выводы

Была разработана описанная выше архитектура.

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА КЛАССОВ



ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: game.h

```
#pragma once

#include "spellsManager.h"
#include <fstream>

class Game{
    private:
        Field player_field, bot_field;
        ShipManager player_ship_manager, bot_ship_manager;
        SpellsManager player_spells_manager, bot_spells_manager;
        size_t count_of_ships, field_height, field_width;
        std::initializer_list<size_t> list_of_ships = {1, 1};

    public:
        Game();
        void new_game();
        bool play_round();
        void play_next_level();
        void save(const std::string& filename);
        void load(const std::string& filename);
};
```

Название файла: game.cpp

```
#include "game.h"

Game::Game() {
    /*count_of_ships = 4;
    list_of_ships = {1,2,3,4};
```

```

    field_height = 10;
    field_width = 10;*/

    count_of_ships = 2;
    field_height = 4;
    field_width = 4;
}

void Game::new_game() {
    player_ship_manager      =      ShipManager(count_of_ships,
list_of_ships);
    bot_ship_manager = ShipManager(count_of_ships, list_of_ships);

    player_field = Field(field_height, field_width);
    bot_field = Field(field_height, field_width);

    player_spells_manager = SpellsManager();

    std::vector<Ship*>      player_ships_array      =
player_ship_manager.getShipsArray();
    size_t i = 0, x, y;
    bool is_horizontal;

    std::ofstream file("setting_player.log");
    while(i < count_of_ships){
        std::cout << "How to setup ship with length = " <<
player_ships_array[i]->size() << "?\n";
        std::cout << " x and y coordinates: ";
        std::cin >> x >> y;

        char s = 'a';
        while(s != 'y' && s != 'n'){
            std::cout << "Is ship horisontal (y - yes, n - no): ";

```

```

        std::cin >> s;
    }
    if(s == 'y'){
        is_horizontal = true;
    } else{
        is_horizontal = false;
    }

    try{
        //player_field.setShip(*player_ships_array[i], x, y,
is_horizontal);
    }
    catch(WrongSettingShip& exception){
        std::cout << exception.what();
        std::cout << "Try again\n";
        continue;
    }

    file << x << "\n";
    file << y << "\n";
    file << (size_t)is_horizontal << "\n";

    i++;
}
file.close();

std::cout << "Creating bot's field...\n";
        std::vector<Ship*>        bot_ships_array        =
bot_ship_manager.getShipsArray();
    i = 0;
    std::ofstream file1("setting_bot.log");
    while(i < count_of_ships){
        x = rand()%field_width;

```



```

        y = rand()%field_height;
        bool is_horizontal = rand()%2;
        try{
            //bot_field.setShip(*bot_ships_array[i], x, y,
is_horizontal);
        }
        catch(WrongSettingShip& exception){
            continue;
        }
        file1 << x << "\n";
        file1 << y << "\n";
        file1 << (size_t)is_horizontal << "\n";
        i++;
    }
    file1.close();
    std::cout << "Done.\n";
}

bool isChanged(std::vector<bool> a, std::vector<bool> b){
    for(size_t i = 0; i < a.size(); i++){
        if(a[i] != b[i]){
            return true;
        }
    }
    return false;
}

bool isVictory(ShipManager ship_manager){
    for(size_t i = 0; i < ship_manager.getShipsArray().size(); i++){
        if(ship_manager.getShipsArray()[i]->isShipDestroyed() ==
false){
            return false;
        }
    }
}

```

```

    }
    return true;
}

bool Game::play_round() {
    try{
        player_spells_manager.cast_spell(&bot_field,
&bot_ship_manager);

        std::cout << "Enter the coordinates of the attack " <<
"(damage = " << player_ship_manager.getDamageScale() << "):\n";
        size_t x, y;
        std::cin >> x >> y;
        std::cout << '\n';

        std::vector<bool> start_array, finish_array;
        for(size_t i = 0; i <
bot_ship_manager.getShipsArray().size(); i++){
            start_array.push_back(bot_ship_manager.getShipsArray()[
i]->isShipDestroyed());
        }

        bot_field.attack(x, y, bot_ship_manager.getDamageScale());
        bot_ship_manager.setDamageScale(1);

        for(size_t i = 0; i <
bot_ship_manager.getShipsArray().size(); i++){
            finish_array.push_back(bot_ship_manager.getShipsArray()[
i]->isShipDestroyed());
        }
        if(isChanged(start_array, finish_array)){
            size_t n = rand();
            if(n%3 == 1){

```

```

        player_spells_manager.add_spell(new Scan);
    } else if(n%3 == 2){
        player_spells_manager.add_spell(new Bombing);
    } else{
        player_spells_manager.add_spell(new DoubleDamage);
    }
}

std::cout<< "\n Bot's Field: \n";
bot_field.printField();
}
catch(CastHaventSpell& exception){
    std::cout << exception.what();
}
catch(OutOfRangeAttack& exception){
    std::cout << exception.what();
}

if(isVictory(bot_ship_manager)){
    std::cout << "It is victory!\n";
    play_next_level();
    return true;
}

//-----
Bot's attack -----
-----

    player_field.attack(rand()%field_width,  rand()%field_height,
player_ship_manager.getDamageScale());

std::cout<< "\n Your Field: \n";
player_field.printField();

```

```

        if(isVictory(player_ship_manager)){
            std::cout << "It is defeat.\n";
            return false;
        } else{
            return true;
        }
    }
}

void Game::play_next_level(){
    std::cout << "Creating bot's field...\n";
    bot_field = Field(field_height, field_width);
    bot_ship_manager = ShipManager(count_of_ships, list_of_ships);
    std::vector<Ship*> bot_ships_array =
bot_ship_manager.getShipsArray();
    size_t i = 0;
    while(i < count_of_ships){
        size_t x = rand()%field_width;
        size_t y = rand()%field_height;
        bool is_horizontal = rand()%2;
        try{
            bot_field.setShip(*bot_ships_array[i], x, y,
is_horizontal);
        }
        catch(WrongSettingShip& exception){
            continue;
        }
        i++;
    }
    std::cout << "Done.\n";
    play_round();
}

```

```

void Game::save(const std::string& filename){
    std::ofstream file(filename);

    file << player_ship_manager.getDamageScale() << "\n";
    file << player_ship_manager.getShipsArray().size() << "\n";
    for(size_t i = 0; i < player_ship_manager.getShipsArray().size();
i++){
        file << player_ship_manager.getShipsArray()[i]->size() <<
"\n";
        for(size_t j = 0; j < player_ship_manager.getShipsArray()[i]-
>size(); j++){
            file << player_ship_manager.getShipsArray()[i]-
>getArray()[j]->getHealth() << "\n";
        }
    }

    file << bot_ship_manager.getDamageScale() << "\n";
    for(size_t i = 0; i < bot_ship_manager.getShipsArray().size();
i++){
        file << bot_ship_manager.getShipsArray()[i]->size() << "\n";
        for(size_t j = 0; j < bot_ship_manager.getShipsArray()[i]-
>size(); j++){
            file << bot_ship_manager.getShipsArray()[i]-
>getArray()[j]->getHealth() << "\n";
        }
    }

    file << field_height << "\n";
    file << field_width << "\n";
    std::vector<std::pair<size_t, size_t>> visiable_cords;
    for(size_t i = 0; i < field_height; i++){
        for(size_t j = 0; j < field_width; j++){
            if(!player_field.getFieldArray()[i][j].isHidden()){

```

```

        visiable_cords.push_back(std::make_pair(i, j));
    }
}

file << visiable_cords.size() << "\n";
for(size_t i = 0; i < visiable_cords.size(); i++){
    file << visiable_cords[i].first << "\n";
    file << visiable_cords[i].second << "\n";
}

visiable_cords.clear();

for(size_t i = 0; i < field_height; i++){
    for(size_t j = 0; j < field_width; j++){
        if(!bot_field.getFieldArray()[i][j].isHidden()){
            visiable_cords.push_back(std::make_pair(i, j));
        }
    }
}

file << visiable_cords.size() << "\n";
for(size_t i = 0; i < visiable_cords.size(); i++){
    file << visiable_cords[i].first << "\n";
    file << visiable_cords[i].second << "\n";
}

file << player_spells_manager.size() << "\n";
for(size_t i = 0; i < player_spells_manager.size(); i++){
    file << player_spells_manager.array()[i] << "\n";
}

file.close();
}

void Game::load(const std::string& filename){

```

```

std::ifstream file(filename);

size_t damage_scale, size_of_ship;

file >> damage_scale;
file >> count_of_ships;
for(size_t i = 0; i < count_of_ships; i++){
    file >> size_of_ship;
    for(size_t j = 0; j < player_ship_manager.getShipsArray()[i]-
>size(); j++){
        //file >> player_ship_manager.getShipsArray()[i]-
>toArray()[j]->getHealth();
    }
}

file.close();
}

```

Название файла: gameState.h

```

#pragma once

#include "game.h"
#include "gameStateExceptions.h"
#include <iostream>

class GameState {
enum GameStates{Loaded, Saved};

private:
    GameStates state;
    Game game;

public:
    GameState(Game game);

```

```

    bool try_load();
    bool try_save();
    void new_game();
    bool play_round();
    void save(const std::string& filename);
    void load(const std::string& filename);
};

```

Название файла: gameState.cpp

```

#include "gameState.h"

GameState::GameState(Game game): game(game) {
    state = GameStates::Loaded;
}

bool GameState::try_load() {
    char s = 'a';
    std::cout << "Do you want to load the game or start new game(y
- yes, load; n - no, start new game)?\n";
    while(s != 'y' && s != 'n') {
        std::cin >> s;
    }
    if(s == 'y') {
        try {
            load("saved_game.txt");
        }
        catch(NoSaves& exception) {
            std::cout << exception.what();
            return false;
        }
        return true;
    } else {

```



```

        return false;
    }
}

bool GameState::try_save(){
    char s = 'a';
    std::cout << "Do you want to save the game (y - yes, n - no)?\n";
    while(s != 'y' && s != 'n'){
        std::cin >> s;
    }
    if(s == 'y'){
        save("saved_game.txt");
        return true;
    } else{
        return false;
    }
}

void GameState::new_game(){
    game.new_game();
}

bool GameState::play_round(){
    return game.play_round();
}

void GameState::save(const std::string& filename){
    game.save(filename);
}

void GameState::load(const std::string& filename){
    /*char s[100];
    std::vector<std::vector<std::string>> save;

```

```

std::vector<std::string> local;
std::ifstream file(filename);

while(file.getline(s, 100)){
    char* pch = strtok(s, " ");
    while(pch != NULL){
        local.push_back(pch);
        pch = strtok (NULL, " ");
    }
    save.push_back(local);
    local.clear();
}

file.close();*/

game.load(filename);
}

std::ostream& operator << (std::ostream &os, const GameState
&game_state){

}

std::istream& operator >> (std::istream& in, GameState &game_state){

}

```

ПРИЛОЖЕНИЕ С

UML-ДИАГРАММА КЛАССОВ

