**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

1. Создать класс игры, который реализует следующий игровой цикл:
   1. Начало игры
   2. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
   3. В случае проигрыша пользователь начинает новую игру
   4. 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.

1. 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 самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

## Выполнение работы

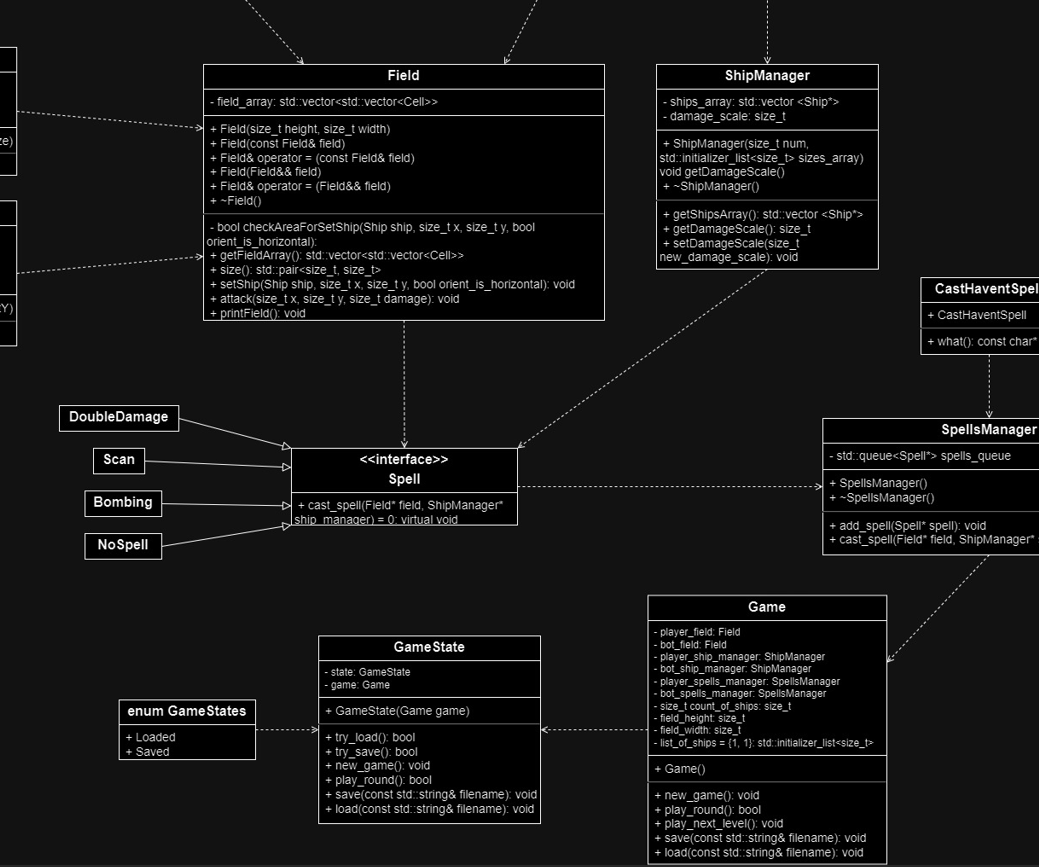
Описание архитектуры лабораторной работы: (см. приложение A, полная архитектура в приложении C).

Был создан класс *Game* и *GameState*. Класс *Game* хранит в себе мэнеджер кораблей и поля для игрока и бота, а также мэнеджер способностей игрока. Для работы с файлами была использована идиома RAII. *GameState* – это оболочка над *Game*, используемая для работы с загрузкой и сохранением. Было реализовано сохранение (загрузки ещё нет). Операторы не были реализованы, так как я не разобрался.

## Выводы

Была разработана описанная выше архитектура.

# ПРИЛОЖЕНИЕ A **UML-ДИАГРАММА КЛАССОВ**



# ПРИЛОЖЕНИЕ B **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]->getArray()[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 C **UML-ДИАГРАММА КЛАССОВ**

