**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: Связывание классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Песчатский С. Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Цель задания:

Разработать гибкую систему управления и отображения игры с использованием шаблонных классов. Создать модульную архитектуру, позволяющую легко изменять способы ввода команд и отрисовки, не затрагивая основную логику игры. Реализовать шаблонный класс управления, преобразующий ввод в команды и вызывающий методы игры, а также шаблонный класс отображения, реагирующий на изменения игры. Дополнительно разработать класс обработки ввода, считывающий символы из терминала и преобразующий их в команды, с проверкой корректности настроек. Создать класс отрисовки поля и обеспечить модульность системы, разделяя управление и отрисовку. Цель — создать расширяемую систему с удобной заменой способов ввода и отображения.

## **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**

Был реализован шаблонный класс контроля игры GameContr, который отвечает за ввод и вывод. Имеет поля:

IHandler Inner – класс для ввода

OHandler Outer – класс для вывода

Game\* game – ссылка на класс игры

а так же методы:

GameContr(Game\* NewGame) – инициализатор класса

void Start() – метод для считывания информации о полях и кораблях и начала игры.

void MainCycle() – вызывает PlayerMove и выводит информацию о победе или поражении в раунде.

void PlayerMove() – считывает необходимую информацию для действий игрока и вызывает соответствующие метода класса Game

Класс ConsoleIn, который подаётся в GameContr как класс ввода

Поля:

std::map<char, Command> commands – словарь хранящий команды и ключи к ним

std::map<Command, char> reversedComms – словарь, хранящий символы и команды, которые им соответсвуют

Методы:

void setDefaultComms() – устанавливает базовые ключи для команд, если не получилось загрузить их из файла

Command strToComm(const std::string& str) – преобразует строку в команду

void Init(const std::string Settings) – устанавливает ключи для каждой команды

Command readCommand() – считывает команду, введённую игроком

int requestSize() – считывает число

std::map<Command, char> SendComms() – возвращает поле reversedComms для класса ConsoleOut

std::string requestFileName() – считывает название файла

std::tuple<int,int> requestCords() – считывает координаты или размеры поля

std::tuple<int, char,int,int> requestShip() – считывает информацию, необходимую для корабля

Аналогично классу ConsoleIn, реализован класс ConsoleOut

Поля:

std::map<Command, char> Commands – словарь, который хранит символы для каждой команды

Методы:

    void GetComms(std::map<Command, char> Comms) – получает словарь команд

    void Attack() – выводит сообщение об атаке

    void File() – выводит сообщение о файле

    void ShowFields(Game\* game) – выводит поля

    void FieldSize() – выводит сообщение о размерах поля

    void ShipsInitFirst() – вывод сообщения о первой части ввода кораблей

    void ShipsInitSecond() – вывод сообщения о второй части ввода кораблей

    void ImproperShip(int a) – выводит сообщение, если корабль имеет некорректное положение

    void Start() – выводит начальное сообщение

    void Spell(Game\* game) – выводит информацию о способности

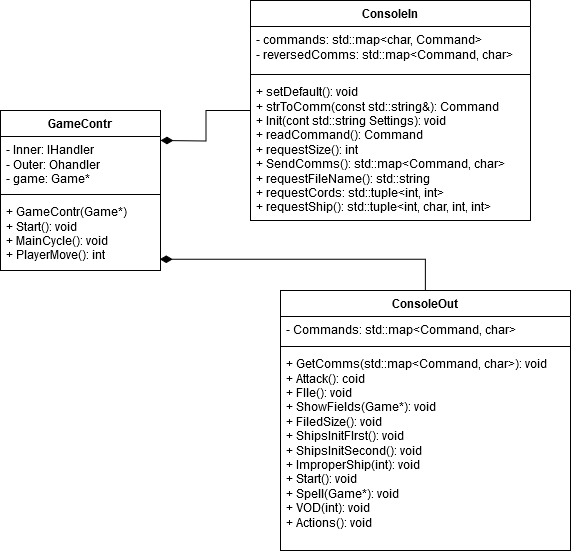
    void VOD(int state) – выводит сообщение о победе или поражении

    void Actions() – выводит сообщение о действии игрока

Разработанный программный код см. в приложении А.

## **Тестирование**

Для проверки корректной работы программы, была проверена работоспособность класса игры и класса состояния игры. Было протестировано несколько раундов игры с загрузкой и сохранением.



## **Выводы**

Была разработана модульная система управления и отображения игры с использованием шаблонных классов, обеспечивающую гибкость и легкость расширения. Система должна поддерживать изменение способов ввода команд и отрисовки без изменения основной логики игры, а также включать обработку пользовательского ввода, проверку настроек и независимую отрисовку игрового поля.

**Приложение А  
Исходный код программы**

Название файла: main.cpp

#include "Game.h"

int main(){

    Game Match;

    while(true){

        if(Match.newGame() != 0){

            break;

        }

        Match.Restart();

    }

}

Название файла: Map.cpp

#include "Map.h"

        void Map::Fill\_Map(const int\* cors){

            size=new int[2];

            size[0]=cors[0];

            size[1]=cors[1];

            std::vector<Sector> tmp;

            for(int j=0; j < cors[0]; j++){

                for(int i = 0; i < cors[1]; i++){

                    tmp.push\_back(unknown);

                }

                field.push\_back(tmp);

                FoggedField.push\_back(tmp);

                tmp.clear();

            }

        }

        void Map::LoadMapLine(std::string line, bool which, int index){

            if(which){

                for(int i=0; i<line.length(); i++){

                    switch (line[i])

                    {

                    case '1':

                        field[i][index] = ship;

                        break;

                    case '0':

                        field[i][index] =  blank;

                    case '\*':

                        field[i][index] = unknown;

                    default:

                        break;

                    }

                }

            }

            else{

                 for(int i=0; i<line.length(); i++){

                    switch (line[i])

                    {

                    case '1':

                        FoggedField[i][index] = ship;

                        break;

                    case '0':

                        FoggedField[i][index] =  blank;

                    case '\*':

                        FoggedField[i][index] = unknown;

                    default:

                        break;

                    }

                }}

        }

        void Map::Show(){

            for(int i=0; i<size[1]; i++){

                std::cout<<getMapLine(i, 1);

            }

        }

        void Map::ShowFogged(){

            for(int i=0; i<size[1]; i++){

                std::cout<<getMapLine(i, 0);

            }

        }

        bool Map::CheckShip(ShipInfo boat, int length){

            int X=boat.x;

            int Y=boat.y;

            for(int i=0; i<length; i++){

                if(boat.ori == 'r'){

                    if (i+X>=size[0] or i+X<0 or Y>=size[1] or Y<0){

                        return false;

                    }

                    if (0<=X+1+i and X+1+i<size[0]){

                        if(field[X+1+i][Y]==ship){return false;}}

                    if (i==0){

                        if(X-1>=0){

                            if(field[X-1][Y]==ship){return false;}}}

                    if (0<=X-1 and X-1<size[0]){

                        if(field[X+i][Y-1]==ship){return false;}}

                    if (0<=X+1 and X+1<size[0]){

                        if(field[X+i][Y+1]==ship){return false;}}

                }

                else{

                    if (i+Y>=size[1] or i+Y<0 or X>=size[0] or X<0){

                        return false;

                    }

                    if (0<=Y+1+i and Y+1+i<size[1]){

                        if(field.at(X).at(Y+i+1)==ship){return false;}}

                    if (i==0){

                        if(X-1>=0){

                            if(field.at(X)[Y-1]==ship){return false;}}}

                    if (0<=X-1 and X-1<size[0]){

                        if(field[X-1][Y+i]==ship){return false;}}

                    if (0<=X+1 and X+1<size[0]){

                        if(field[X+1][Y+i]==ship){return false;}}

                }

            }

            return true;

        }

        void Map::ShipIsNoMore(std::vector<int> info){

            int X=info[0];

            int Y=info[1];

            for(int i=0; i<info[2]; i++){

                if(info[3]==100){

                    if (i+X>=size[0] or i+X<0 or Y>=size[1] or Y<0){

                        return;

                    }

                    if (0<=X+1+i and X+1+i<size[0] and i==info[2]-1){field[X+1+i][Y]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (i==0 and X-1>=0){field[X-1][Y]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (0<=X-1 and X-1<size[0]){field[X+i][Y-1]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (0<=X+1 and X+1<size[0]){field[X+i][Y+1]=blank;FoggedField[X+1+i][Y]=blank;}

                }

                else{

                    if (i+Y>=size[1] or i+Y<0 or X>=size[0] or X<0){

                        return;

                    }

                    if (0<=Y+1+i and Y+1+i<size[1] and i==info[2]-1){field[X][Y+1+i]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (i==0 and Y-1>=0){field[X][Y-1]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (0<=Y-1 and Y-1<size[1]){field[X-1][Y+i]=blank;FoggedField[X+1+i][Y]=blank;}

                    if (0<=Y+1 and Y+1<size[1]){field[X+1][Y+i]=blank;FoggedField[X+1+i][Y]=blank;}

                }

            }

        }

        bool Map::PlaceShip(Ship boat, ShipInfo info){

            if(CheckShip(info, boat.getSize())){

                if(info.ori == 'r'){

                    int j=info.y;

                    for(int k=0; k<boat.getSize() and 0<=k<size[0] and 0<=j<size[1]; k++){

                        field[k+info.x][j]=ship;

                    }

                }

                else{

                    int j=info.x;

                    for(int k=0; k<boat.getSize() and 0<=k<size[0] and 0<=j<size[1]; k++){

                        field[j][k+info.y]=ship;

                    }

                }

                }

                else{

                    return 0;

                }

            return 1;

        }

        bool Map::SectorChecker(int \*cords){

            if (field[cords[0]][cords[1]]==ship){

                return true;

            }

            else{

                return false;

            }

        }

        void Map::ActivateDD(){

            DD=1;

        }

        bool Map::Hit(int \*coords, Manager\* manage){

            try{

            if((0<=coords[0] and coords[0]<size[0]) and (0<=coords[1] and coords[1]<size[1])){

                if(field[coords[0]][coords[1]]==ship){

                    if (DD==1){manage->Hit(coords);DD=0;}

                    if(manage->Hit(coords)){

                        ShipIsNoMore(manage->getShipInfo(coords));

                        return 1;

                    }

                }

                if(field[coords[0]][coords[1]]==unknown){

                    FoggedField[coords[0]][coords[1]]=blank;

                    field[coords[0]][coords[1]]=blank;

                }

                else{

                    FoggedField[coords[0]][coords[1]] = field[coords[0]][coords[1]];

                }

            }

            else{throw ShotOOB();}

            }

            catch(const ShotOOB& e){

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

            }

            return 0;

        }

        Map& Map::operator=(const Map& source){

            if(this==&source){

                return \*this;

            }

            size=source.size;

            if (source.field.empty()){

                std::vector<std::vector<Sector>> field;

                std::vector<Sector> tmp;

                for(int j=0; j < size[0]; j++){

                    for(int i = 0; i < size[1]; i++){

                        tmp.push\_back(source.field[j][i]);

                    }

                    field.push\_back(tmp);

                }

            }

    return \*this;

}

int \* Map::getSize(){

    return size;

}

std::string Map::getMapLine(int index, bool which){

    std::string result;

    if(which){

        for(int j=0; j<size[0]; j++){

            if(field.at(j).at(index)==unknown){result = result + "\*";}

            if(field.at(j).at(index)==ship){result = result + "1";}

            if(field.at(j).at(index)==blank){result = result + "0";}

        }

    }

    else{

        for(int j=0; j<size[0]; j++){

            if(FoggedField.at(j).at(index)==unknown){result = result + "\*";}

            if(FoggedField.at(j).at(index)==ship){result = result + "1";}

            if(FoggedField.at(j).at(index)==blank){result = result + "0";}

        }

    }

    return result + "\n";

}

Map& Map::operator=(Map&& moved){

    if(this==&moved){

        return \*this;

    }

    field=moved.field;

    size=moved.size;

    return \*this;

}

Map::~Map(){}

Название файла: Map.h

#ifndef MAP\_H

#define MAP\_H

#include <iostream>

#include <vector>

#include "Manager.h"

#include "ImproperShipPlacement.h"

#include "ShotOOB.h"

enum Sector{blank, ship, unknown};

class Map{

    private:

        std::vector<std::vector<Sector>> field;

        std::vector<std::vector<Sector>> FoggedField;

        int \*size;

        bool DD=0;

    public:

        Map() = default;

        void Fill\_Map(const int\* cors);

        void LoadMapLine(std::string line ,bool which, int index);

        void Show();

        void ShowFogged();

        bool CheckShip(ShipInfo boat, int length);

        void ShipIsNoMore(std::vector<int> info);

        bool PlaceShip(Ship ship, ShipInfo info);

        bool SectorChecker(int \*cords);

        bool Hit(int \*coords, Manager\* manage);

        void ActivateDD();

        Map& operator=(const Map& source);

        Map& operator=(Map&& moved);

        int\* getSize();

        std::string getMapLine(int index, bool which);

        Map(const Map& source) : size(source.size), field(source.field){}

        Map(Map&& moved) noexcept: size(moved.size), field(moved.field){}

        ~Map();

};

#endif

Название файла:Ship.cpp

#include "Ship.h"

Ship::Ship(int len){

    for (int i = 0; i<len; i++){

        segHp.push\_back(untouched);

    }

}

bool Ship::damage(int index){

        if(segHp.at(index)==damaged){segHp.at(index)=destroyed;}

        if(segHp.at(index)==untouched){segHp.at(index)=damaged;}

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

            if(segHp.at(i)!=destroyed){

                return 0;

            }

        }

        if(cond==1){return 0;}

        cond=1;

        return cond;

    }

void Ship::showShip(){

    if (cond){std::cout<<"alive"<<" len-"<<std::to\_string(segHp.size())<<" position-";}

    else{std::cout<<"dead"<<" len-"<<std::to\_string(segHp.size())<<" position-";}

    }

int Ship::getSize(){

        return segHp.size();

    }

bool Ship::isDead(){return cond;}

Ship::~Ship(){}

Название файла: Ship.h

#ifndef SHIP\_H

#define SHIP\_H

#include <iostream>

#include <vector>

enum Health{untouched, damaged, destroyed};

class Ship{

    private:

        bool cond=0;

        std::vector <Health> segHp;

    public:

        Ship(int len);

        bool damage(int index);

        void showShip();

        int getSize();

        bool isDead();

        ~Ship();

};

#endif //SHIP.H

Название файла: Shipinfo.cpp

#include "ShipInfo.h"

ShipInfo::ShipInfo(int hor, int ver){

    x=hor;

    y=ver;

}

void ShipInfo::Set\_ori(char dir){

    ori=dir;

}

ShipInfo::~ShipInfo(){}

Название файла: Shipinfo.h

#ifndef SHIPINFP\_H

#define SHIPINFO\_H

class ShipInfo{

    public:

    int x;

    int y;

    char ori;

    ShipInfo(int hor, int ver);

    void Set\_ori(char dir);

    ~ShipInfo();

};

#endif //SHIPINFO\_H

Название файла: Manager.cpp

#include "Manager.h"

Manager::Manager(){}

void Manager::get\_Fleet(std::vector<int> Sizes){

    std::cout<<"ENTER DIRECTION AND POSITION OF EACH SHIP\n";

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

        get\_Ship(Sizes[i]);

    }

}

void Manager::get\_Ship(int size){

        char tempCord; std::cin>>tempCord;

        int tempPos[2];

        for(int j=0; j<2; j++){std::cin>>tempPos[j];}

            ShipInfo info(tempPos[0], tempPos[1]);

            info.Set\_ori(tempCord);

            Report.push\_back(info);

            Ship newborn=Ship(size);

            Fleet.push\_back(newborn);

}

void Manager::loadShip(int size, char tempCord, int pos1, int pos2){

    int tempPos[2]; tempPos[0] = pos1; tempPos[1] = pos2;

    ShipInfo info(tempPos[0], tempPos[1]);

    info.Set\_ori(tempCord);

    Report.push\_back(info);

    Ship newborn=Ship(size);

    Fleet.push\_back(newborn);

}

void Manager::show\_Fleet(){

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

        std::cout<<"Stats of ship number "<<i+1<<": ";Fleet.at(i).showShip();

        std::cout<<Report[i].x<<"."<<Report[i].y<<" orientation-";

        if(Report[i].ori=='d'){std::cout<<"vertical";}

        else{std::cout<<"horizontal";}

        std::cout<<'\n';

    }

}

void Manager::Bot\_Ship(int size, int\* borders){

        srand(time(0));

        int tempCord; tempCord = rand()%2;

        int tempPos[2];

        tempPos[0] = rand()%borders[0];

        tempPos[1] = rand()%borders[1];

        ShipInfo info(tempPos[0], tempPos[1]);

            if(tempCord==0){

                info.Set\_ori('d');

            }

            else if (tempCord==1)

            {

                info.Set\_ori('r');

            }else{

                std::cout<<"ERROR";

            }

            Report.push\_back(info);

            Ship newborn=Ship(size);

            Fleet.push\_back(newborn);

}

bool Manager::Hit(int \* coords){

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

        for(int j=0; j<Fleet.at(i).getSize(); j++){

            if(Report[i].ori=='r'){

                if(Report[i].x+j==coords[0] and Report[i].y==coords[1]){

                    if(Fleet.at(i).damage(j)){

                        return 1;

                    }

                }

            }

            else{

                if(Report[i].y+j==coords[1] and Report[i].x==coords[0]){

                    if(Fleet.at(i).damage(j)){

                        return 1;

                    }

                }

            }

        }

    }

    return 0;

}

bool Manager::CheckAll(){

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

        if(!Fleet[i].isDead()){

            return false;

        }

    }

    return true;

}

std::vector<int> Manager::getShipInfo(int\* coords){

    std::vector<int> val;

    std::vector<int> No;

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

        for(int j=0; j<Fleet.at(i).getSize(); j++){

            if(Report[i].ori=='r'){

                if(Report[i].x+j==coords[0] and Report[i].y==coords[1]){

                    val.push\_back(Report[i].x); val.push\_back(Report[i].y); val.push\_back(Fleet[i].getSize()); val.push\_back(Report[i].ori);

                    return val;

                }

            }

            else{

                if(Report[i].y+j==coords[1] and Report[i].x==coords[0]){

                    val.push\_back(Report[i].x); val.push\_back(Report[i].y); val.push\_back(Fleet[i].getSize()); val.push\_back(Report[i].ori);

                    return val;

                }

            }

        }

    }

    return No;

}

void Manager::Auto\_Fleet(std::vector<int> Sizes, int\* borders){

    srand(time(0));

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

        int tempCord; tempCord = rand()%2;

        int tempPos[2];

        tempPos[0] = rand()%borders[0];

        tempPos[1] = rand()%borders[1];

        ShipInfo info(tempPos[0], tempPos[1]);

            if(tempCord==0){

                info.Set\_ori('d');

            }

            else if (tempCord==1)

            {

                info.Set\_ori('r');

            }else{

                std::cout<<"ERROR";

            }

            Report.push\_back(info);

            Ship newborn=Ship(Sizes[i]);

            Fleet.push\_back(newborn);

    }

}

std::vector <Ship>& Manager::send\_Fleet(){

    return Fleet;

}

std::vector <ShipInfo>& Manager::send\_Coordinates(){

    return Report;

}

std::string Manager::getShipInfo(int index){

    std::string result = std::to\_string(Fleet[index].getSize())+" "+Report[index].ori +" "+ std::to\_string(Report[index].x) + " " + std::to\_string(Report[index].y) + "\n";

    return result;

}

void Manager::delete\_ship(int index){

    Fleet.erase(Fleet.begin()+index);

    Report.erase(Report.begin()+index);

}

Manager::~Manager(){}

Название файла: Manager.h

#ifndef MANAGER\_H

#define MANAGER\_H

#include <iostream>

#include <vector>

#include "Ship.h"

#include "ShipInfo.h"

class Manager{

    private:

        std::vector <Ship> Fleet;

        std::vector <ShipInfo> Report;

    public:

        Manager();

        void get\_Fleet(std::vector<int> Sizes);

        void get\_Ship(int size);

        void loadShip(int size, char tempCord, int pos1, int pos2);

        void show\_Fleet();

        void Bot\_Ship(int num, int\* borders);

        bool Hit(int \* coords);

        bool CheckAll();

        std::vector<int> getShipInfo(int\* coords);

        void Auto\_Fleet(std::vector<int> Sizes, int\* borders);

        std::vector <Ship>& send\_Fleet();

        std::vector <ShipInfo>& send\_Coordinates();

        std::string getShipInfo(int index);

        void delete\_ship(int index);

        ~Manager();

};

#endif

Название файла: Barrage.cpp

#include "Barrage.h"

bool Barrage::cast(Map\* cords, Manager\* forThisToWorkProperly){

    std::vector<Ship> tmp = forThisToWorkProperly->send\_Fleet();

    if(tmp.size()>0){

        int a = rand()%tmp.size(); int b = rand()%tmp.at(a).getSize();

        if(tmp.at(a).damage(b)){

            return 1;

        }

    }

    std::cout<<"RANDOM DAMAGE\n";

    return 0;

}

std::string Barrage::saySpell(){return "Barrage\n";}

Название файла: Barrage.h

#ifndef BARRAGE\_H

#define BARRAGE\_H

#include "Spell.h"

class Barrage:public Spell{

public:

    bool cast(Map\* cords, Manager\* forThisToWorkProperly) override;

    std::string saySpell() override;

};

#endif

Название файла: Scaner.cpp

#include "Scanner.h"

bool Scanner::cast(Map\* cords, Manager\* forThisToWorkProperly){

    int shot[2];std::cin>>shot[0]>>shot[1];

    int \*tmp = cords->getSize();

    if(shot[0]<0){shot[0]=0;}

    if(shot[0]>=tmp[0]){shot[0]=tmp[0]-2;}

    if(shot[1]<0){shot[1]=0;}

    if(shot[1]>tmp[1]){shot[1]=tmp[1]-2;}

    int tmp1[2]; tmp1[0]=shot[0]+1; tmp1[1] = shot[1];

    int tmp2[2]; tmp2[1]=shot[1]+1; tmp2[0] = shot[0];

    int tmp3[2]; tmp3[0]=shot[0]+1; tmp3[1]=shot[1]+1;

    std::cout<<"AREA IS SCANNED\n";

    if(cords->SectorChecker(shot) || cords->SectorChecker(tmp1) || cords->SectorChecker(tmp2) || cords->SectorChecker(tmp3)){

        std::cout<<"SHIP DETECTED\n";

    }

    else{

        std::cout<<"NO SHIP";

    }

    return 0;

}

std::string Scanner::saySpell(){return "Scanner\n";}

Название файла: Scaner.h

#ifndef SCANNER\_H

#define SCANNER\_H

#include "Spell.h"

class Scanner:public Spell{

public:

    bool cast(Map\* cords, Manager\* forThisToWorkProperly) override;

    std::string saySpell() override;

};

#endif

Название файла: DoubleDamage.cpp

#include "DoubleDamage.h"

bool DoubleDamage::cast(Map\* cords, Manager\* forThisToWorkProperly){

        std::cout<<"DAMAGE INCREASED";

        cords->ActivateDD();

        return 0;

    }

std::string DoubleDamage::saySpell(){return "DoubleDamage\n";}

Название файла: DoubleDamage.h

#ifndef DOUBLEDAMAGE\_H

#define DOUBLEDAMAGE\_H

#include "Spell.h"

class DoubleDamage:public Spell{

public:

    bool cast(Map\* cords, Manager\* forThisToWorkProperly) override;

    std::string saySpell()override;

};

#endif

Название файла: SpellBook.cpp

#include "SpellBook.h"

void SpellBook::getSpells(Map\* newMap, Manager\* newManager){

    queue.push\_back(std::make\_unique<DoubleDamage>());

    queue.push\_back(std::make\_unique<Scanner>());

    queue.push\_back(std::make\_unique<Barrage>());

    std::random\_device rd;

    std::mt19937 g(rd());

    std::shuffle(queue.begin(), queue.end(), g);

    manage = newManager;

    field = newMap;

}

void SpellBook::addSpell(){

    srand(time(0));

    switch (rand()%3)

    {

    case 0:

        queue.insert(queue.begin(), std::make\_unique<DoubleDamage>());

        break;

    case 1:

        queue.insert(queue.begin(), std::make\_unique<Scanner>());

        break;

    case 2:

        queue.insert(queue.begin(), std::make\_unique<Barrage>());

        break;

    default:

        break;

    }

}

std::string SpellBook::AllSpellsInStr(){

    std::string result;

    for(int i=queue.size()-1; i>-1; i--){result.append(queue[i]->saySpell());}

    return result;

}

void SpellBook::useSpell(){

    try{

    if(queue.size()==0){throw NoSpells();}

    if(queue.back()->cast(field, manage)){

        addSpell();

    }

    queue.pop\_back();

    }

    catch(const NoSpells& e){

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

    }

}

void SpellBook::nextSpell(){

    std::cout<<queue.back()->saySpell();

}

int SpellBook::AmountoSpells(){

    return queue.size();

}

void SpellBook::LoadSpell(std::string name){

    if(name == "DoubleDamage"){

        queue.push\_back(std::make\_unique<DoubleDamage>());

    }

    if(name == "Scanner"){

        queue.push\_back(std::make\_unique<Scanner>());

    }

    if(name == "Barrage"){

        queue.push\_back(std::make\_unique<Barrage>());

    }

}

void SpellBook::update(Map\* newMap, Manager\* newManager){

    manage = newManager;

    field = newMap;

}

Название файла: SpellBook.h

#ifndef SPELLBOOK\_H

#define SPELLBOOK\_H

#include "DoubleDamage.h"

#include "Scanner.h"

#include "Barrage.h"

#include "NoSpells.h"

#include "Map.h"

#include <iterator>

#include <random>

#include <vector>

#include <memory>

#include <algorithm>

class SpellBook{

private:

    std::vector<std::unique\_ptr<Spell>> queue;

    Manager\* manage;

    Map\* field;

public:

    SpellBook() = default;

    void getSpells(Map\* newMap, Manager\* newManager);

    void addSpell();

    std::string AllSpellsInStr();

    void nextSpell();

    void useSpell();

    int AmountoSpells();

    void LoadSpell(std::string name);

    void update(Map\* newMap, Manager\* newManager);

};

#endif

Название файла: ShotOOB.h

#ifndef SHOTOOB\_H

#define SHOTOOB\_H

#include <exception>

#include <string>

#include <iostream>

class ShotOOB/\*Short of Shot out of bounds\*/ : public std::exception {

private:

    std::string Error = "Shot is out of bounds";

public:

    explicit ShotOOB();

    const char\* what() const noexcept override;

};

#endif

Название файла: ImproperShipPlacement.h

#ifndef IMPROPERSHIPPLACEMENT\_H

#define IMPROPERSHIPPLACEMENT\_H

#include <exception>

#include <string>

#include <iostream>

class ImproperShipPlacement : public std::exception {

private:

    std::string Error;

    std::string first="Ship number ";

    std::string second = " Overlaps with another ship or goes beyond the field";

public:

    explicit ImproperShipPlacement(int index);

    const char\* what() const noexcept override;

};

#endif

Название файла NoSpells.h

#ifndef NOSPELLS\_H

#define NOSPELLS\_H

#include <exception>

#include <string>

class NoSpells : public std::exception {

private:

    std::string Error="No available abilities";

public:

    explicit NoSpells();

    const char\* what() const noexcept override;

};

#endif