**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. 3344 |  | Тукалкин.В.А |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создать класс игры, класс состояния игры, реализовать сохранение и загрузку 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 самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

## Выполнение работы

При выполнении работы было создано три класса: игры, состояния игры и игрока.

1) Класс Game:

В приватном поле хранит поля игрока и врага, менеджеры кораблей игрока и врага, менеджеры способностей игрока и врага, хранители информации игрока и врага, игрока, врага, состояние игры, флаг хода игрока, флаг конца игры и метод проверки на конец игры.

1) bool checkGameOver(GameField& field, ShipManager& shipManager);

2) Game() – конструктор

3) void startGame() – начало игры

4) void saveGame(const std::string& filename) – сохранение игры в файл

5) void loadGame(const std::string& filename) – загрузка игры из файла

6) void playerTurn() – ход игрок, показывает игровые поля и позволяет сделать ход

7) void enemyTurn() – ход врага, делает выстрел по случайным координатам

8) void startRound() – начало раунда

9) void generateEnemyField() – создаёт поле врага случайным образом

2) Класс GameState:

В приватном поле хранит поля игрока и врага, менеджеры кораблей игрока и врага, менеджеры способностей игрока и врага.

1) GameState(GameField& playerField, GameField& enemyField, ShipManager& playerShipManager, ShipManager& enemyShipManager, AbilityManager& playerAbilityManager, AbilityManager& enemyAbilityManager) – конструктор

2) GameField& getPlayerField() – получить поле игрока

3) GameField& getEnemyField() – получить поле врага

4) AbilityManager& getPlayerAbilityManager() – получить менеджер способностей игрока

5) AbilityManager& getEnemyAbilityManager() – получить менеджер способностей врага

6) ShipManager& getPlayerShipManager() – получить менеджер кораблей игрока

7) ShipManager& getEnemyShipManager() – получить менеджер кораблей врага

8) void operator«(const std::string& filename) – переопределение оператора ввода для сохранения

9) void operator»(const std::string& filename) – переопределение оператора ввода для загрузки

10) nlohmann::json serializeField(GameField& field) – сериализация поля

11) nlohmann::json serializeShipManager(ShipManager& manager) – сериализация менеджера кораблей

12) void deserializeField(const nlohmann::json& field\_json, GameField& field, ShipManager& ship\_manager) – десериализация поля

13) void deserializeShipManager(const nlohmann::json& manager\_json, ShipManager& manager) - десериализация менеджера кораблей

3) Класс Player:

В приватном поле хранит игровое поле, менеджер кораблей, менеджер способностей и хранитель информации.

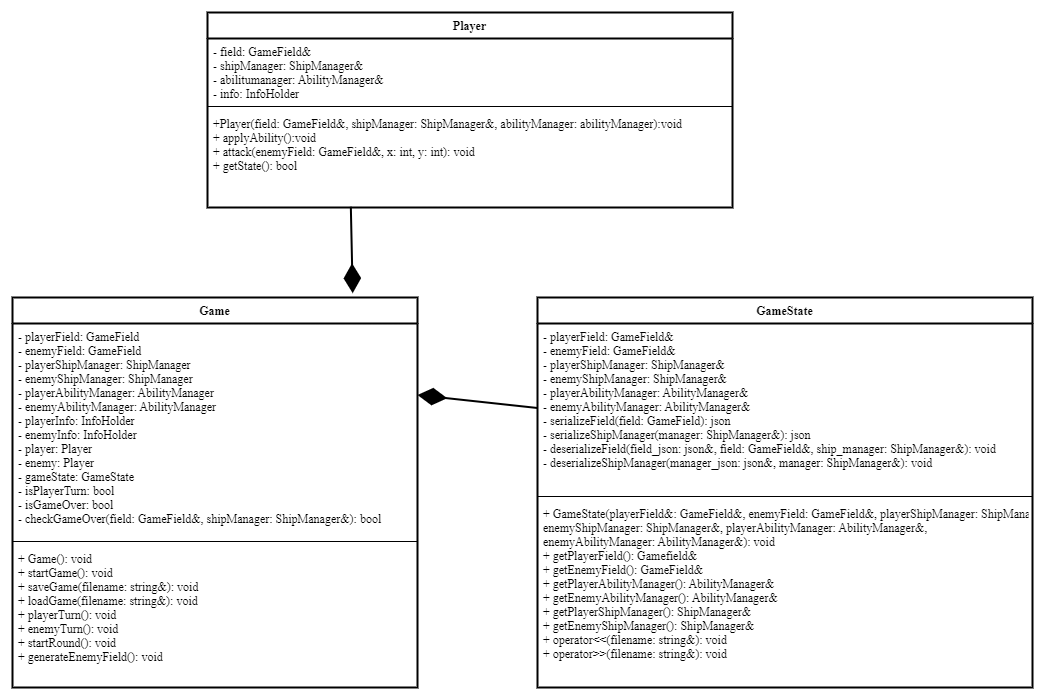
1) Player(GameField& field, ShipManager& shipManager, AbilityManager& abilityManager) – конструктор

2) void applyAbility() – использовать способность из менеджера способностей

3) int attack(GameField& enemyField, int x, int y) – атака по полю врага

4) bool getState() – получение состояния жив или нет

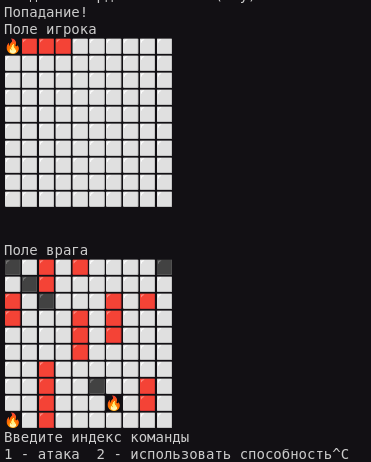
UML-диаграмма классов программы на данный момент:



## Тестирование.

Результаты тестирования представлены в табл. 1.

После некоторого количества действий программа имеет данное состояние.



## Выводы

Были созданы классы игры, состояния игры и игрока, также были связаны все классы.

# Приложение А Исходный код программы

Название файла: main.cpp

}

Название файла: Ability.h

#ifndef ABILITY\_H

#define ABILITY\_H

#include <string>

#include "GameField.h"

class Ability{

public:

virtual ~Ability() = default;

virtual void apply(GameField& field, int x, int y) = 0;

virtual std::string getName() const = 0;

};

#endif // ABILITY\_H

Название файла: DoubleDamageAbility.cpp

#include "DoubleDamageAbility.h"

void DoubleDamageAbility::apply(GameField& field, int x, int y){

field.attack(x,y);

field.attack(x,y);

}

std::string DoubleDamageAbility::getName() const{

return "DoubleDamage";

}

Название файла: DoubleDamageAbility.h

#ifndef DOUBLEDAMAGEABILITIES\_H

#define DOUBLEDAMAGEABILITIES\_H

#include "Ability.h"

class DoubleDamageAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // DOUBLEDAMAGEABILITIES\_H

Название файла: ScannerAbility.cpp

#include "ScannerAbility.h"

#include <iostream>

void ScannerAbility::apply(GameField& field, int x, int y){

int height = field.getHeight();

int width = field.getWidth();

if(x >= width) x = width - 1;

if(y >= height) y = height - 1;

if(field.OpenCell(x, y) || field.OpenCell(x+1, y) || field.OpenCell(x+1, y+1) || field.OpenCell(x, y+1)){

std::cout << "Ship in sector!" << '\n';

}

}

std::string ScannerAbility::getName() const{

return "Scanner";

}

Название файла: ScannerAbility.h

#ifndef SCANNERABILITIES\_H

#define SCANNERABILITIES\_H

#include "Ability.h"

class ScannerAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // SCANNERABILITIES\_H

Название файла: ShootingAbility.cpp

#include "ShootingAbility.h"

#include <random>

void ShootingAbility::apply(GameField& field, int x, int y){

auto ships = field.getShips();

srand(time(0));

int index = rand()%ships.size();

x = ships[index].first.first;

y = ships[index].first.second;

int len = ships[index].second.getLength();

int segmentShip = len;

int orientation = ships[index].second.getOrientationShip();

len = rand()%len;

while(ships[index].second.getSegmentHP(segmentShip) == ShipStateHP::dead){

segmentShip++;

if(segmentShip > len){

segmentShip = 0;

}

}

if(orientation == OrientationShip::Horizontal){

field.attack(x + segmentShip, y);

}else{

field.attack(x, y + segmentShip);

}

}

std::string ShootingAbility::getName() const{

return "Shooting";

}

Название файла: ShootingAbility.h

#ifndef SHOOTINGABILITIES\_H

#define SHOOTINGABILITIES\_H

#include "Ability.h"

class ShootingAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // SHOOTINGABILITIES\_H

Название файла: InvalidPlacementShipExceptions.h

#ifndef INVALIDPLACEMENTSHIPEXCEPTIONS\_H

#define INVALIDPLACEMENTSHIPEXCEPTIONS\_H

#include <stdexcept>

class InvalidPlacementShipExceptions : public std::runtime\_error{

public:

InvalidPlacementShipExceptions() : std::runtime\_error("Invalid placement ship") {}

};

#endif // INVALIDPLACEMENTSHIPEXCEPTIONS\_H

Название файла: NoAbilityExceptions.h

#ifndef NOABILITYEXCEPTIONS\_H

#define NOABILITYEXCEPTIONS\_H

#include <stdexcept>

class NoAbilityExceptions : public std::runtime\_error{

public:

NoAbilityExceptions() : std::runtime\_error("No abilities") {}

};

#endif // NOABILITYEXCEPTIONS\_H

Название файла: OutOfFieldAttackExceptions.h

#ifndef OUTOFFIELDATTACKEXCEPTIONS\_H

#define OUTOFFIELDATTACKEXCEPTIONS\_H

#include <stdexcept>

class OutOfFieldAttackExceptions : public std::runtime\_error{

public:

OutOfFieldAttackExceptions() : std::runtime\_error("Attack out of field") {}

};

#endif // OUTOFFIELDATTACKEXCEPTIONS\_H

Название файла: AbilityManager.h

#ifndef ABILITYMANAGER\_H

#define ABILITYMANAGER\_H

#include "./Abilities/ShootingAbility.h"

#include "./Abilities/ScannerAbility.h"

#include "./Abilities/DoubleDamageAbility.h"

#include <vector>

#include <memory>

class AbilityManager{

private:

std::vector<std::unique\_ptr<Ability>> abilities;

public:

AbilityManager();

void applyAbility(GameField& field, int x, int y);

void addRandomAbility();

};

#endif // ABILITYMANAGER\_H

Название файла: AbilityManager.cpp

#include "AbilityManager.h"

#include "Exceptions/NoAbilityException.h"

#include <random>

#include <algorithm>

#include <memory>

AbilityManager::AbilityManager(){

abilities.push\_back(std::make\_unique<DoubleDamageAbility>());

abilities.push\_back(std::make\_unique<ScannerAbility>());

abilities.push\_back(std::make\_unique<ShootingAbility>());

std::random\_device rd;

std::mt19937 g(rd());

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

}

void AbilityManager::applyAbility(GameField& field, int x, int y){

if(abilities.size()<=0){

throw NoAbilityExceptions();

}

abilities.back()->apply(field, x, y);

abilities.pop\_back();

}

void AbilityManager::addRandomAbility(){

srand(time(0));

switch (rand()%3)

{

case 0:

abilities.insert(abilities.begin(), std::make\_unique<DoubleDamageAbility>());

break;

case 1:

abilities.insert(abilities.begin(), std::make\_unique<ScannerAbility>());

break;

case 2:

abilities.insert(abilities.begin(), std::make\_unique<ShootingAbility>());

break;

default:

break;

}

}

Название файла: CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(NavalBattle)

set(CMAKE\_CXX\_STANDARD 17)

#aux\_source\_directory(./src tree)

add\_executable(NavalBattle

main.cpp

Ship.cpp

ShipManager.cpp

GameField.cpp

AbilityManager.cpp

./Abilities/ShootingAbility.cpp

./Abilities/ScannerAbility.cpp

./Abilities/DoubleDamageAbility.cpp

)

target\_include\_directories(NavalBattle PUBLIC ${CMAKE\_SOURCE\_DIR})