**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. 3342 |  | Лапшов К.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать класс игры, который реализует игровой цикл с чередованием ходов пользователя и компьютерного врага, включая управление игрой, сохранение и загрузку состояния игры, а также переопределение операторов ввода и вывода для состояния игры.

## **Задание**

1. Создать класс игры, который реализует следующий игровой цикл:
2. Начало игры.
3. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
4. В случае проигрыша пользователь начинает новую игру.
5. 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.

**Примечания:**

* 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**

Класс ‘Game’ представляет собой основной класс игры "Морской бой". Он управляет основным циклом игры, включая начало игры, ходы игрока и врага, сохранение и загрузку игры.

Поля класса:

* GameState game\_state - Состояние игры, включает в себе поля игрока и врага, менеджеры кораблей, менеджер способности.
* int round - Текущий раунд игры.
* bool is\_double\_damage - Флаг, указывающий на активацию способности "Двойной урон".
* bool is\_player\_turn - Флаг, указывающий, чей сейчас ход (игрока или врага).

Методы класса:

* Game() – Конструктор класса, инициализирует начальные значения полей.
* void start() – Начинает игру, предлагает игроку выбрать между началом новой игры или загрузкой сохраненной.
* void play() – Основной цикл игры, управляет ходами игрока и врага, проверяет условия победы или поражения.
* void newRound() - Начинает новый раунд игры, сбрасывает состояние врага.
* void playerTurn(int choice) - Обрабатывает ход игрока, включая атаку и использование способностей.
* void enemyTurn() - Обрабатывает ход врага, случайным образом выбирает координаты для атаки.
* void inputPlayerData() - Запрашивает у игрока координаты для размещения своих кораблей.

Класс ‘GameState’ представляет собой состояние игры, включая поля игрока и врага, менеджеры кораблей, менеджеры способности. Он отвечает за размещение кораблей, использование способностей, вывод полей игры на экран, а также сериализацию и десериализацию состояния игры.

Поля класса:

* AbilityManager\* player\_am - Менеджер способностей игрока.
* GameField\* player\_gf – Игровое поле игрока.
* ShipManager\* player\_sm - Менеджер кораблей игрока.
* GameField\* enemy\_gf - Игровое поле врага.
* ShipManager\* enemy\_sm - Менеджер кораблей врага.
* std::vector<Point> player\_cords - Координаты кораблей игрока.
* std::vector<int> player\_orient - Ориентации кораблей игрока.
* std::vector<Point> enemy\_cords - Координаты кораблей врага.
* std::vector<int> enemy\_orient - Ориентация кораблей врага.

Методы класса:

* GameState() - Конструктор класса, инициализирует начальные значения полей.
* ~GameState() - Деструктор класса, освобождает выделенную память.
* bool placePlayerShip(Point point, int orientation, int index) - Размещает корабль игрока на поле.
* void placeEnemyShips() - Размещает корабли врага на поле.
* void useAbility(bool& is\_double\_damage) - Использует способность игрока.
* void printBattleField() - Выводит на экран поля игрока и врага.
* void resetEnemy() - Сбрасывает состояние врага для нового раунда.
* int getAbilitiesCount() - Возвращает количество доступных способностей игрока.
* std::string getLastAbilityName() - Возвращает название последней способности в очереди.
* GameField& getPlayerField() - Возвращает ссылку на игровое поле игрока.
* GameField& getEnemyField() - Возвращает ссылку на игровое поле врага.
* ShipManager& getPlayerShipManager() - Возвращает ссылку на менеджер кораблей игрока.
* ShipManager& getEnemyShipManager() - Возвращает ссылку на менеджер кораблей врага.
* void serialize(FileWrapper& file) - Сериализует состояние игры в файл.
* void deserialize(FileWrapper& file) - Десериализует состояние игры из файла.

Класс ‘GameSaveLoader’ создан для работы с сохранениями и загрузкой состояния игры.

Поля класса:

* std::string filename – имя файла, с которым предстоит работа.

Методы класса:

* GameSaveLoader(std::string\_view filename) – Конструктор класса
* void save(const GameState& game\_state, int round) – Метод для сохранения игры. Открывает файл, записывая состояние игры. Так же, с помощью getSum, записывает контрольную сумму
* void load(GameState& game\_state, int& round) – Метод для загрузки сохранения. Перед загрузкой сверяет контрольные суммы нынешнего состояния файла и прошлого. При несоответствии выбрасывается исключение.
* int getSum() - Метод для получения контрольной суммы файла.

Класс ‘FileWrapper’ представляет собой обертку над файловым потоком, которая упрощает работу с файлами для сериализации и десериализации данных, учитывая идиому RAII.

Поля класса:

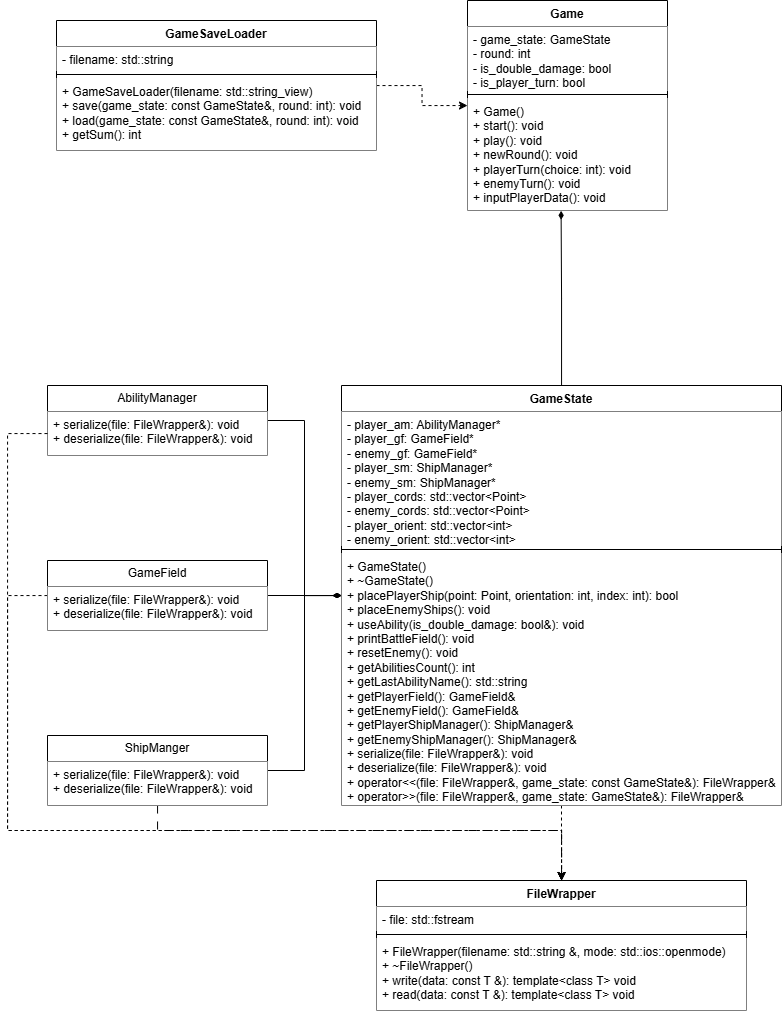
* std::fstream file – Файловый поток для работы с файлом.

Методы класса:

* FileWrapper(std::string &filename, std::ios::openmode mode) – Конструктор класса, открывает файл в указанном режиме.
* ~FileWrapper() – Деструктор класса, закрывает файл, если он открыт.
* template<class T> void write(const T &data) – Записывает данные в файл.
* template<class T> void read(T &data) - Считывает данные из файла.

Также, для сокращения кода, в классы ‘ShipManager’, ‘GameField’, ‘AbilityManager’, были добавлены методы сериализации и десериализации для сохранения и загрузки конкретных данных соответственно.

Разработанный программный код см. в приложении А.



## **Выводы**

В процессе выполнения задания был разработан класс игры, который реализует игровой цикл с чередованием ходов пользователя и компьютерного врага.

Для лучшего понимания структуры игры была создана UML-диаграмма классов.

# **Приложение А Исходный код программы**

Название файла: main.cpp

#include <windows.h>

#include "seagame/src/game/core/game.h"

int main() {

SetConsoleOutputCP(CP\_UTF8);

Game current\_game = Game();

current\_game.start();

return 0;

}

Название файла: game.h

#ifndef OOP\_GAME\_H

#define OOP\_GAME\_H

#include "iostream"

#include "game\_state.h"

#include "../errors/place\_error.h"

class Game {

private:

GameState game\_state;

int round;

bool is\_double\_damage;

bool is\_player\_turn;

public:

Game();

void start();

void play();

void newRound();

void load();

void save();

void playerTurn(int choice);

void enemyTurn();

void inputPlayerData();

};

#endif //OOP\_GAME\_H

Название файла: game.cpp

#include "game.h"

#include "Windows.h"

Game::Game() {

this->is\_player\_turn = true;

this->is\_double\_damage = false;

this->round = 1;

}

void Game::start() {

int choice = 0;

std::cout << "Welcome to the sea battle!" << "\n";

std::cout << "Choose the option: 0 - Start new game; 1 - Load game" << '\n';

std::cin >> choice;

system("cls");

if (choice){

this->load();

}else{

this->inputPlayerData();

this->game\_state.placeEnemyShips();

}

this->play();

}

void Game::inputPlayerData() {

std::cout << "To continue, select the coordinates for the ships\n";

std::cout << "Coordinates are entered in this way: X (0-9) Y (0-9) orientation (0 - horizontal, 1 - vertical)\n";

ShipManager& player\_sm = game\_state.getPlayerShipManager();

for (int i = 0; i < player\_sm.getShipCount(); ++i) {

Point point = {0, 0};

int orientation = 0;

Ship\* ship = player\_sm.getShip(i);

bool placed = false;

while(!placed){

std::cout << "Enter the coordinates for the ship of length " << ship->getShipLength() << ": ";

std::cin >> point.x >> point.y >> orientation;

placed = game\_state.placePlayerShip(point, orientation, i);

}

}

system("cls");

}

void Game::play() {

std::cout << "Everything was successful!\n";

while (true){

if(is\_player\_turn){

int choice = 0;

std::cout << "Current round: " << this->round << "\n";

game\_state.printBattleField();

std::cout << "\n";

std::cout << "Available abilities: " << game\_state.getAbilitiesCount() << "\n";

std::cout << "The first available ability: " << game\_state.getLastAbilityName() << "\n";

std::cout << "Choose the action: 0 - Attack the enemy; 1 - Use ability; 2 - Save game and exit" << '\n';

std::cin >> choice;

if(choice == 2){

save();

return;

}

playerTurn(choice);

if(game\_state.getEnemyShipManager().allAreDestroyed()){

system("cls");

std::cout << "Congratulations! You won!!!!!\n";

std::cout << "Creating new round, wait pls....\n";

Sleep(2000);

this->newRound();

continue;

}

is\_player\_turn = false;

}else{

this->enemyTurn();

if(game\_state.getPlayerShipManager().allAreDestroyed()){

system("cls");

std::cout << "You lose(\n";

std::cout << "You lasted for " << this->round - 1 << " round!\n";

std::cout << "They'll take you next time) \n";

Sleep(2000);

return;

}

is\_player\_turn = true;

}

system("cls");

}

}

void Game::playerTurn(int choice){

Point point = {0, 0};

if(choice == 1){

game\_state.useAbility(is\_double\_damage);

}

std::cout << "Choose the coords to attack: ";

std::cin >> point.x >> point.y;

game\_state.getEnemyField().attackField(point, this->is\_double\_damage);

}

void Game::enemyTurn(){

bool fake\_dd = false;

GameField& player\_gf = game\_state.getPlayerField();

while (true){

Point point = {0, 0};

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis\_width(0, player\_gf.getFieldWidth() - 1);

std::uniform\_int\_distribution<> dis\_height(0, player\_gf.getFieldHeight() - 1);

point.x = dis\_width(gen);

point.y = dis\_height(gen);

if(player\_gf.getSellState(point) != CellState::Empty){

player\_gf.attackField(point, fake\_dd);

break;

}

}

}

void Game::newRound() {

this->round++;

std::cout << "New round started\n";

game\_state.resetEnemy();

}

Название файла: game\_state.h

#ifndef OOP\_GAME\_STATE\_H

#define OOP\_GAME\_STATE\_H

#include "game\_field.h"

#include "ship\_manager.h"

#include "ship.h"

#include "../../abilities/core/ability\_manager.h"

#include "../errors/place\_error.h"

class GameState {

private:

AbilityManager\* player\_am;

GameField\* player\_gf;

ShipManager\* player\_sm;

GameField\* enemy\_gf;

ShipManager\* enemy\_sm;

std::vector<Point> player\_cords;

std::vector<int> player\_orient;

std::vector<Point> enemy\_cords;

std::vector<int> enemy\_orient;

public:

GameState();

~GameState();

bool placePlayerShip(Point point, int orientation, int index);

void placeEnemyShips();

void useAbility(bool& is\_double\_damage);

void printBattleField();

void resetEnemy();

int getAbilitiesCount();

std::string getLastAbilityName();

GameField& getPlayerField();

GameField& getEnemyField();

ShipManager& getPlayerShipManager();

ShipManager& getEnemyShipManager();

void serialize(FileWrapper& file);

void deserialize(FileWrapper& file);

friend FileWrapper& operator<<(FileWrapper& file, const GameState& game\_state);

friend FileWrapper& operator>>(FileWrapper& file, GameState& game\_state);

};

#endif

Название файла: game\_state.cpp

#include "game\_state.h"

#include "../../abilities/core/add\_new\_ability.h"

#include "../../abilities/errors/ability\_extract\_error.h"

GameState::GameState() {

this->player\_gf = new GameField(10, 10);

this->enemy\_gf = new GameField(10, 10);

this->player\_sm = new ShipManager(4, {1, 2, 3, 4});

this->enemy\_sm = new ShipManager(4, {1, 2, 3, 4});

this->player\_am = new AbilityManager();

// При уничтожении кораблей врага, + 1 способность

std::shared\_ptr<AddNewAbility> reaction = std::make\_shared<AddNewAbility>(\*player\_am);

enemy\_gf->setCommand(reaction);

// При уничтожении кораблей пользователя ничего не происходит

player\_gf->setCommand(nullptr);

}

GameState::~GameState() {

delete player\_gf;

delete enemy\_gf;

delete player\_sm;

delete enemy\_sm;

delete player\_am;

}

bool GameState::placePlayerShip(Point point, int orientation, int index){

Ship\* ship = player\_sm->getShip(index);

try{

player\_gf->placeShip(ship, point, orientation);

this->player\_cords.push\_back(point);

this->player\_orient.push\_back(orientation);

return true;

}catch(PlaceError &err){

std::cout << err.what() << "\n";

return false;

}

}

void GameState::placeEnemyShips(){

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis\_width(0, this->enemy\_gf->getFieldWidth() - 1);

std::uniform\_int\_distribution<> dis\_height(0, this->enemy\_gf->getFieldHeight() - 1);

std::uniform\_int\_distribution<> dis\_orientation(0, 1);

int index = 0;

while (index < enemy\_sm->getShipCount()){

int x = dis\_width(gen);

int y = dis\_height(gen);

int orientation = dis\_orientation(gen);

Ship\* current\_ship = enemy\_sm->getShip(index);

try{

enemy\_gf->placeShip(current\_ship, {x, y}, orientation);

this->enemy\_cords.push\_back({x, y});

this->enemy\_orient.push\_back(orientation);

}catch(PlaceError &err){

continue;

}

index++;

}

}

void GameState::useAbility(bool& is\_double\_damage) {

try{

std::shared\_ptr<AbilityFactory> ab = player\_am->extractAbility();

if(ab->getName() == "Scanner"){

bool flag = false;

Point point = {0, 0};

std::cout << "Choose the coords to scanner: ";

std::cin >> point.x >> point.y;

enemy\_gf->setAbility(ab->create(point, [&flag](bool is\_there\_ship){flag = is\_there\_ship;}));

if(flag){

std::cout << "Ship is detected" << "\n";

}else{

std::cout << "Ship is not detected" << "\n";

}

std::cout << "\n";

}else if(ab->getName() == "DoubleDamage"){

enemy\_gf->setAbility(ab->create({},[&is\_double\_damage](bool is\_dd){is\_double\_damage = is\_dd;}));

}else if(ab->getName() == "RandomShot"){

enemy\_gf->setAbility(ab->create());

std::cout << "Our artillery has successfully covered them!" << "\n";

}

}catch(AbilityExtractError(& err)){

std::cerr<<"Error: " << err.what() << "\n";

}

}

int GameState::getAbilitiesCount(){

return player\_am->queue\_size();

}

void GameState::printBattleField() {

std::cout << " Player Field\t\t\t Enemy Field\n";

std::cout << " ";

for (int j = 0; j < player\_gf->getFieldWidth(); ++j) {

std::cout << j << " ";

}

std::cout << "\t\t ";

for (int j = 0; j < enemy\_gf->getFieldWidth(); ++j) {

std::cout << j << " ";

}

std::cout << "\n";

for (int i = 0; i < player\_gf->getFieldHeight(); ++i) {

std::cout << i << " ";

for (int j = 0; j < player\_gf->getFieldWidth(); ++j) {

FieldCell\* state = player\_gf->getCellInfo({j, i});

if (state->cell\_state == CellState::Unknown) {

if(state->ship\_pointer != nullptr){

std::cout << "■ ";

continue;

}

std::cout << "□ ";

}else if(state->cell\_state == CellState::Ship){

if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Damaged){

std::cout << "◧ ";

}else if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Destroyed){

std::cout << "▧ ";

}

}else if (state->cell\_state == CellState::Empty) {

std::cout << "● ";

}

}

std::cout << "\t\t" << i << " ";

for (int j = 0; j < enemy\_gf->getFieldWidth(); ++j) {

FieldCell\* state = enemy\_gf->getCellInfo({j, i});

if (state->cell\_state == CellState::Unknown) {

std::cout << "□ ";

} else if (state->cell\_state == CellState::Empty) {

std::cout << "● ";

} else if (state->cell\_state == CellState::Ship) {

if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Damaged){

std::cout << "◧ ";

}else if(state->ship\_pointer->getSegment(state->segment\_index) == SegmentState::Destroyed){

std::cout << "▧ ";

}

}

}

std::cout << "\n";

}

}

void GameState::resetEnemy() {

const int old\_width = enemy\_gf->getFieldWidth();

const int old\_height = enemy\_gf->getFieldHeight();

GameField\* new\_enemy\_gf = new GameField(old\_width, old\_height);

ShipManager\* new\_enemy\_sm = new ShipManager(4, {1, 2, 3, 4});

new\_enemy\_gf->setCommand(enemy\_gf->getCommand());

delete enemy\_gf;

delete enemy\_sm;

enemy\_gf = new\_enemy\_gf;

enemy\_sm = new\_enemy\_sm;

this->placeEnemyShips();

}

GameField &GameState::getPlayerField() {

return \*player\_gf;

}

GameField &GameState::getEnemyField(){

return \*enemy\_gf;

}

ShipManager &GameState::getPlayerShipManager() {

return \*player\_sm;

}

ShipManager &GameState::getEnemyShipManager() {

return \*enemy\_sm;

}

std::string GameState::getLastAbilityName() {

return player\_am->getLastName();

}

void GameState::serialize(FileWrapper &file) {

file << \*this;

}

void GameState::deserialize(FileWrapper &file) {

file >> \*this;

}

FileWrapper& operator<<(FileWrapper& file, const GameState& game\_state) {

game\_state.player\_sm->serialize(file);

for (int i = 0; i < game\_state.player\_sm->getShipCount(); ++i) {

file.write(game\_state.player\_cords[i].x);

file.write(" ");

file.write(game\_state.player\_cords[i].y);

file.write(" ");

file.write(game\_state.player\_orient[i]);

file.write("\n");

}

game\_state.player\_gf->serialize(file);

game\_state.player\_am->serialize(file);

game\_state.enemy\_sm->serialize(file);

for (int i = 0; i < game\_state.enemy\_sm->getShipCount(); ++i) {

file.write(game\_state.enemy\_cords[i].x);

file.write(" ");

file.write(game\_state.enemy\_cords[i].y);

file.write(" ");

file.write(game\_state.enemy\_orient[i]);

file.write("\n");

}

game\_state.enemy\_gf->serialize(file);

return file;

}

FileWrapper &operator>>(FileWrapper &file, GameState &game\_state) {

//Десериализация менджера кораблей игрока

game\_state.player\_sm->deserialize(file);

game\_state.player\_cords.clear();

game\_state.player\_orient.clear();

for (int i = 0; i < game\_state.player\_sm->getShipCount(); ++i) {

Point point;

int orientation;

file.read(point.x);

file.read(point.y);

file.read(orientation);

game\_state.player\_cords.push\_back(point);

game\_state.player\_orient.push\_back(orientation);

}

//Десериализация поля игрока

game\_state.player\_gf->deserialize(file);

for (int i = 0; i < game\_state.player\_sm->getShipCount(); ++i) {

Ship\* ship = game\_state.player\_sm->getShip(i);

game\_state.player\_gf->placeShip(ship, game\_state.player\_cords[i], game\_state.player\_orient[i]);

}

//Десериализация способностей

game\_state.player\_am->deserialize(file);

game\_state.enemy\_sm->deserialize(file);

game\_state.enemy\_cords.clear();

game\_state.enemy\_orient.clear();

for (int i = 0; i < game\_state.enemy\_sm->getShipCount(); ++i) {

Point point;

int orientation;

file.read(point.x);

file.read(point.y);

file.read(orientation);

game\_state.enemy\_cords.push\_back(point);

game\_state.enemy\_orient.push\_back(orientation);

}

//Десериализация поля игрока

game\_state.enemy\_gf->deserialize(file);

for (int i = 0; i < game\_state.enemy\_sm->getShipCount(); ++i) {

Ship\* ship = game\_state.enemy\_sm->getShip(i);

game\_state.enemy\_gf->placeShip(ship, game\_state.enemy\_cords[i], game\_state.enemy\_orient[i]);

}

return file;

}

Название файла: FileWrapper.h

#ifndef OOP\_FILEWRAPPER\_H

#define OOP\_FILEWRAPPER\_H

#include <iostream>

#include "fstream"

class FileWrapper {

private:

std::fstream file;

public:

FileWrapper(std::string &filename, std::ios::openmode mode);

~FileWrapper();

template<class T>

void write(const T &data);

template<class T>

void read(T &data);

};

template<class T>

void FileWrapper::write(const T &data) {

if(!file.is\_open()){

throw std::runtime\_error("File not opened in output mode");

}

file << data;

}

template<class T>

void FileWrapper::read(T &data) {

if(!file.is\_open()){

throw std::runtime\_error("File not opened in output mode");

}

file >> data;

}

#endif

Название файла: FileWrapper.cpp

#include "FileWrapper.h"

FileWrapper::FileWrapper(std::string &filename, std::ios::openmode mode): file(filename, mode) {}

FileWrapper::~FileWrapper() {

if(file.is\_open()){

file.close();

}

}

Название файла: ship\_manager.cpp

bool ShipManager::allAreDestroyed(){

bool all\_destroyed = true;

for(auto ship: ships){

if (!ship->isDestroyed()){

all\_destroyed = false;

}

}

return all\_destroyed;

}

int ShipManager::getShipCount() {

return shipsCount;

}

void ShipManager::serialize(FileWrapper& file){

file.write(shipsCount);

file.write('\n');

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

file.write(ships[i]->getShipLength());

file.write(' ');

}

file.write('\n');

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

for (int j = 0; j < ships[i]->getShipLength(); ++j) {

file.write(static\_cast<int>(ships[i]->getSegment(j)));

file.write(' ');

}

file.write('\n');

}

}

void ShipManager::deserialize(FileWrapper& file){

int new\_ships\_count;

file.read(new\_ships\_count);

for (auto ship: ships) {

delete ship;

}

ships.clear();

shipsCount = new\_ships\_count;

std::vector<int> shipLengths(shipsCount);

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

file.read(shipLengths[i]);

}

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

Ship\* ship = new Ship(shipLengths[i]);

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

int segmentStateInt;

file.read(segmentStateInt);

SegmentState segmentState = static\_cast<SegmentState>(segmentStateInt);

ship->setSegment(j, segmentState);

}

ships.push\_back(ship);

}

}

Название файла: game\_save\_loader.h

#ifndef OOP\_GAME\_SAVE\_LOADER\_H

#define OOP\_GAME\_SAVE\_LOADER\_H

#include "game\_state.h"

#include "../../common/FileWrapper.h"

class GameSaveLoader {

private:

std::string filename;

public:

GameSaveLoader(std::string\_view filename);

void save(const GameState& game\_state, int round);

void load(GameState& game\_state, int& round);

int getSum();

};

#endif

Название файла: game\_save\_loader.cpp

#include "game\_save\_loader.h"

GameSaveLoader::GameSaveLoader(std::string\_view filename) : filename(filename) {}

void GameSaveLoader::save(const GameState &game\_state, int round) {

FileWrapper\* file = new FileWrapper(filename, std::ios::out);

\*file << game\_state;

file->write(round);

delete file;

FileWrapper fileOfSum("../summary", std::ios::out);

fileOfSum.write(getSum());

}

void GameSaveLoader::load(GameState& game\_state, int& round) {

int oldSum, fileSum;

FileWrapper fileOfSum("../summary", std::ios::in);

fileOfSum.read(oldSum);

fileSum = getSum();

if (oldSum != fileSum){

throw std::invalid\_argument("Data has been crashed!");

}

FileWrapper file(filename, std::ios::in);

file >> game\_state;

file.read(round);

}

int GameSaveLoader::getSum(){

FileWrapper file(filename, std::ios::in);

int totalSum = 0;

char ch;

while(!file.isEof()) {

file.read(ch);

totalSum += static\_cast<int>(ch);

}

return totalSum;

}

Название файла: game\_field.cpp

std::shared\_ptr<Command> GameField::getCommand() {

return addNewAbility;

}

void GameField::serialize(FileWrapper &file) {

file.write(width);

file.write(" ");

file.write(height);

file.write("\n");

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

for (int j = 0; j < width; j++) {

file.write(static\_cast<int>(field[i][j].cell\_state));

file.write(' ');

}

file.write("\n");

}

}

void GameField::deserialize(FileWrapper& file){

int old\_width, old\_height;

file.read(old\_width);

file.read(old\_height);

field.clear();

field.assign(old\_height, std::vector<FieldCell>(old\_width, FieldCell {CellState::Unknown, nullptr}));

for (int i = 0; i < old\_height; i++) {

for (int j = 0; j < old\_width; j++) {

int cellStateInt;

file.read(cellStateInt);

CellState cellState = static\_cast<CellState>(cellStateInt);

field[i][j].cell\_state = cellState;

}

}

};

Название файла: ability\_manager.cpp

void AbilityManager::serialize(FileWrapper& file){

std::queue<std::shared\_ptr<AbilityFactory>> tempQue = this->abilities;

file.write(this->queueSize());

file.write("\n");

while(!tempQue.empty()){

std::string factName = tempQue.front()->getName();

file.write(factName);

file.write("\n");

tempQue.pop();

}

}

void AbilityManager::deserialize(FileWrapper& file){

this->abilities = std::queue<std::shared\_ptr<AbilityFactory>>();

int queue\_length;

file.read(queue\_length);

for (int i = 0; i < queue\_length; ++i) {

std::string factoryName;

file.read(factoryName);

for (int j = 0; j < this->abilityProduction.getAbilitiesQuantity(); ++j) {

if (factoryName == this->abilityProduction.getFactory(j)->getName()){

abilities.push(this->abilityProduction.getFactory(j));

}

}

}

}