**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**

a. 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.

**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:

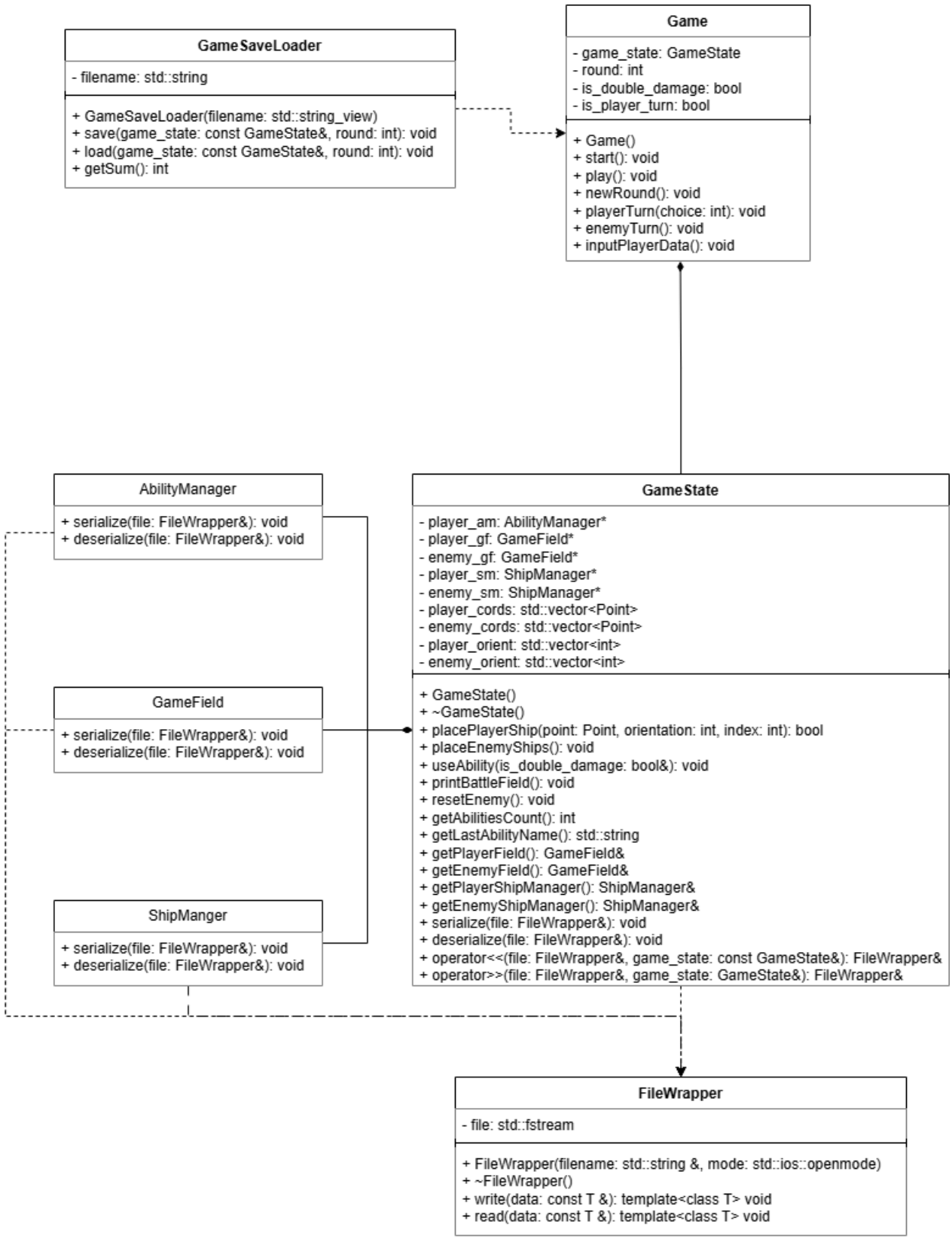
- std::fstream file – Файловый поток для работы с файлом.

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.


GameSaveLoader
- filename: std::string
+ GameSaveLoader(filename: std::string_view)
+ save(game_state: const GameState&, round: int): void
+ load(game_state: const GameState&, round: int): void
+ getSum(): int
Game
- game_state: GameState
- round: int
- is_double_damage: bool
- is_player_turn: bool
+ Game()
+ start(): void
+ play(): void
+ newRound(): void
+ playerTurn(choice: int): void
+ enemyTurn(): void
+ inputPlayerData(): void
AbilityManager
+ serialize(file: FileWrapper&): void
+ deserialize(file: FileWrapper&): void
GameState
- player_am: AbilityManager*
- player_gf: GameField*
- enemy_gf: GameField*
- player_sm: ShipManager*
- enemy_sm: ShipManager*
- player_cords: std::vector<Point>
- enemy_cords: std::vector<Point>
- player_orient: std::vector<int>
- enemy_orient: std::vector<int>
+ GameState()
+ ~GameState()
+ placePlayerShip(point: Point, orientation: int, index: int): bool
+ placeEnemyShips(): void
+ useAbility(is_double_damage: bool&): void
+ printBattleField(): void
+ resetEnemy(): void
+ getAbilitiesCount(): int
+ getLastAbilityName(): std::string
+ getPlayerField(): GameField&
+ getEnemyField(): GameField&
+ getPlayerShipManager(): ShipManager&
+ getEnemyShipManager(): ShipManager&
+ serialize(file: FileWrapper&): void
+ deserialize(file: FileWrapper&): void
+ operator<<(file: FileWrapper&, game_state: const GameState&): FileWrapper&
+ operator>>(file: FileWrapper&, game_state: GameState&): FileWrapper&
GameField
+ serialize(file: FileWrapper&): void
+ deserialize(file: FileWrapper&): void
ShipManger
+ serialize(file: FileWrapper&): void
+ deserialize(file: FileWrapper&): void
FileWrapper
- file: std::fstream
+ FileWrapper(filename: std::string &, mode: std::ios::openmode)
+ ~FileWrapper()
+ write(data: const T &): template<class T> void
+ read(data: const T &): template<class T> void

**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 спос
обность
        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 игрок
a
        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));
                }
```

```
            }
        }
    }
```