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»

Студентка гр. 3344 _____ Якимова Ю.А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы

Изучить связывание классов путём усовершенствования программы из предыдущей лабораторной работы. Необходимо создать: класс игры и класс состояния игры.

Задание

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

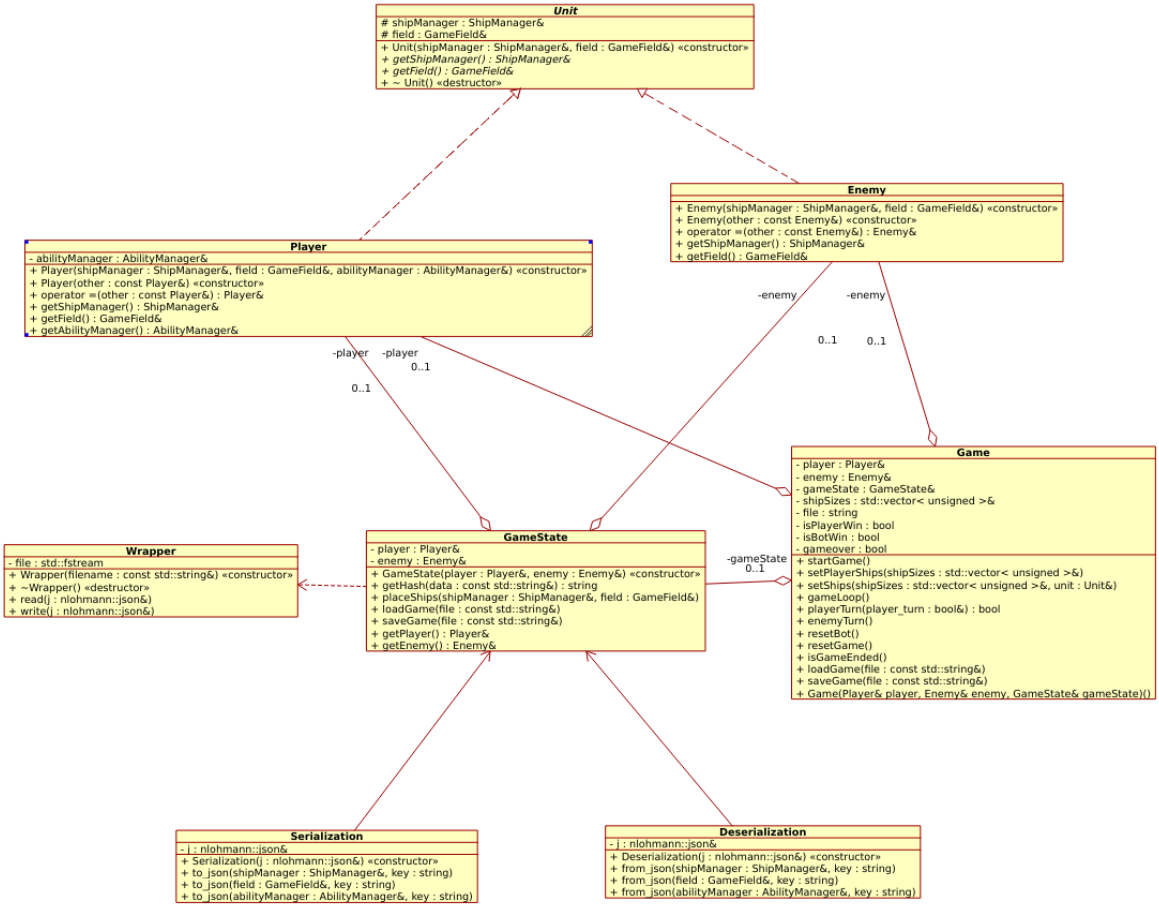

Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Game*, *Unit*, *Player*, *Enemy*, *Serialization*, *Deserialization*, *Wrapper* и *GameState*.

Классы *Game* и *GameState* были добавлены согласно заданию. *Game* связывает классы и работает с ними, описывает игровой цикл и выполнение ходов. Класс *GameState* отвечает за связывание классов *Serialization*, *Deserialization* и *Wrapper*, которые в сумме дают возможность работать с json файлом и совершать загрузку/сохранение игры. В нём также происходит хэширование json файла для его защиты от внешнего вмешательства.

Классы *Unit, Player* и *Enemy* являются дата-классами, *Unit* – абстрактный класс, который хранит общие для игрока и бота поля и методы; *Player* и *Enemy* – наследуемые от *Unit* классы, представляющие собой игрока и его врага (бота) соответственно, могут только возвращать значения полей.

Классы *Serialization* и *Deserialization* отвечают за считывание и запись из json файла. Прописаны методы для менеджера кораблей, поля и менеджера способностей, чтобы реализовать загрузку и сохранение игры. Обработка json файла организована с использованием библиотеки nlohmann/json.

Класс *Wrapper* реализован как обёртка над файлом с использованием идиомы RAII для более удобной работы. В конструкторе происходит открытие файла, а в деструкторе его закрытие.

Помимо обозначенных классов, реализованы и интегрированы в код новые классы-исключения для обработки различных исключительных случаев работы с файлом и игрой.

Game является классом для реализации логики игры. Он имеет следующие поля:

- *Player& player* – класс игрока.
- *Enemy& enemy* – класс бота.
- *GameState& gameState* – класс состояния игры.
- std::vector<unsigned>& shipSizes – вектор длин кораблей.
- *bool isPlayerWin* – выиграл ли игрок.
- *bool isBotWin* – выиграл ли бот.
- *bool gameover* – закончилась ли игра.

И следующие методы:

- *setPlayerShips* – установить корабли игрока.
- *setShips* – установить корабли.
- *gameLoop* – начать игровой цикл (ход игрока, ход бота и т.д.)
- *startGame* – метод начала игры.
- *playerTurn* – очередь игрока в игровом цикле (игроку предлагаются варианты хода: attack/ability/save/load/quit).
- *enemyTurn* – очередь бота.
- *resetBot* – обнуление бота (при выигрыше раунда).

- *resetGame* – обнуление всей игры (после проигрыша и при желании продолжить игру).
- *isGameEnded* – проверка, завершилась ли игра и требуется ли продолжение.
- *loadGame* – вызов загрузки игры у класса состояния.
- *saveGame* – вызов сохранения игры у класса состояния.

Класс *Unit* является абстрактным классом для игрока и бота. Он имеет следующие protected поля:

- *ShipManager& shipManager* – ссылка на менеджер кораблей.
- *Field& field* – ссылка на поле.

И следующие виртуальные методы:

- *virtual ShipManager& getShipManager() = 0* – возвращает ссылку на менеджер кораблей.
- *virtual Field& getField() = 0* – возвращает ссылку на поле.

Класс *Player* является реализацией дата-класса игрока, который наследуется от класса *Unit*. Он имеет следующие поля:

- *ShipManager& shipManager* – ссылка на менеджер кораблей, наследуется от *Unit*.
- *Field& field* – ссылка на поле, наследуется от *Unit*.
- *AbilityManager& abilityManager* – ссылка на менеджер способностей.

И соответствующие методы для получения полей.

Класс *Enemy* является реализацией дата-класса бота, он тоже наследуется от класса *Unit*. Он имеет следующие поля:

- *ShipManager& shipManager* – ссылка на менеджер кораблей, наследуется от *Unit*.
- *Field& field* – ссылка на поле, наследуется от *Unit*.

И соответствующие методы для получения полей.

Класс *Serialization* служит для записи информации в json файл с использованием библиотеки nlohmann/json. Он имеет следующее поле:

- *nlohmann::json& j* – ссылка на структуру данных для работы с json.

Он имеет три одинаковых по структуре метода (*to_json*) для подготовки к записи в файл менеджера кораблей, поля и менеджера способностей.

Класс *Deserialization* служит для загрузки информации из json файла. Он имеет следующее поле:

- *nlohmann::json& j* – ссылка на структуру данных для работы с json.

Он имеет три одинаковых по структуре метода (*from_json*) для загрузки из файла менеджера кораблей, поля и менеджера способностей.

Класс *Wrapper* является обёрткой над файлом с использованием идиомы RAII. Он имеет следующее поле:

- *fstream file* – поток для работы с файлом.

И следующие методы:

- *read(nlohmann::json& j)* – записывает содержимое файла в структуру json.
- *write(nlohmann::json& j)* – записывает содержимое структуры json в файл.

Класс *GameState* является классом состояния для связывания других классов и для реализации полной логики загрузки/сохранения игры. Он имеет следующие поля:

- *Player& player* – ссылка на игрока.
- *Enemy& enemy* – ссылка на бота.

И следующие методы:

- *Wrapper& operator<<(Wrapper& fileWrapper, GameState& state)* – переопределяет оператор << следующим образом: сначала происходит сериализация и вся необходимая информация по кораблям, полям и способностям сохраняется в библиотечную структуру, которая потом переносится в обёртку и она возвращается.

- *Wrapper& operator>>(Wrapper& fileWrapper, GameState& state)* – переопределяет оператор >> следующим образом: сначала происходит считывание информации из обёртки в структуру json, затем десериализация, информация записывается в временные объекты и позже переносится на используемые, в конце возвращается обёртка.
- *void placeShips(ShipManager& shipManager, Field& field)* – расставляет корабли обратно после загрузки из файла.
- *void loadGame(const string& file)* – создаёт обертку и заполняет объект класса информацией из файла.
- void saveGame(const string file) – очищает файл, создаёт обёртку и загружает в неё информацию из объекта класса.

Тестирование:

Происходит симуляция игры между игроком и врагом (ботом), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. В начале хода игрок может использовать одну случайную способность; перейти к атаке вражеского поля; загрузить игру, получив состояния кораблей, поля и способностей; сохранить игру, уже записав состояния игровых сущностей; выйти из игры.

При победе игроку предлагается продолжить игру с сохранением его поля и с новым противником. В случае победы бота, игру можно продолжить, обнулив её полностью.

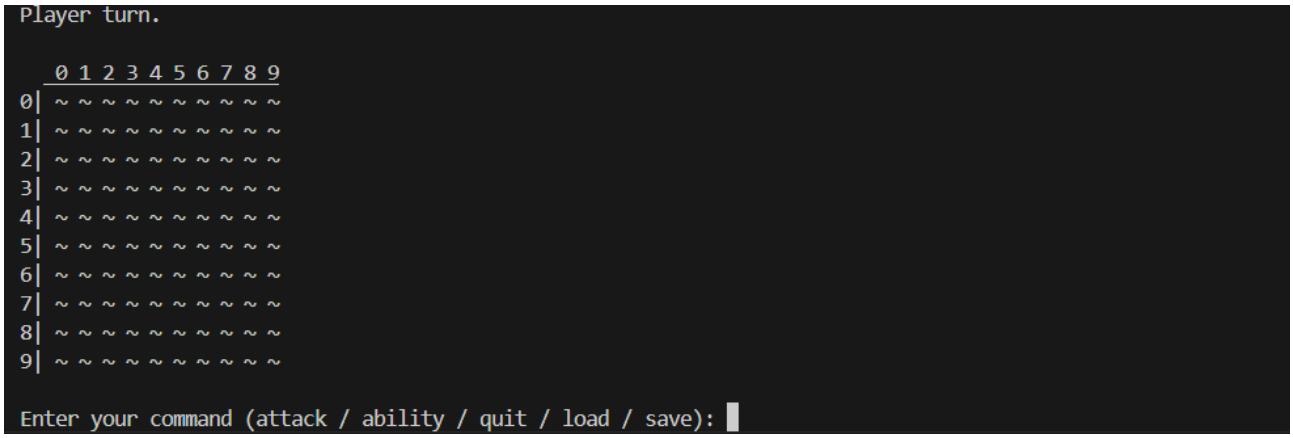

Рисунок 2 – Начало игры

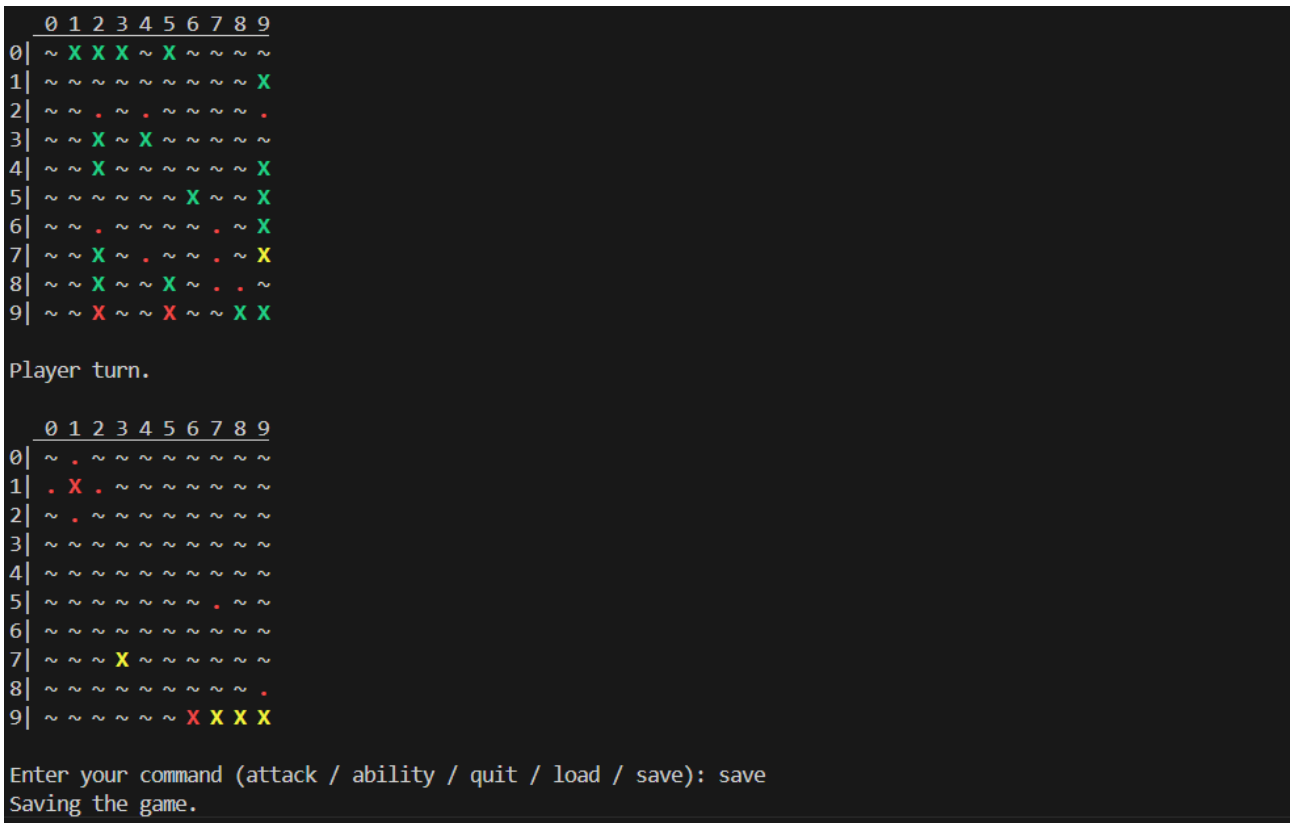

Рисунок 3 – Игра сохранена и закрыта.

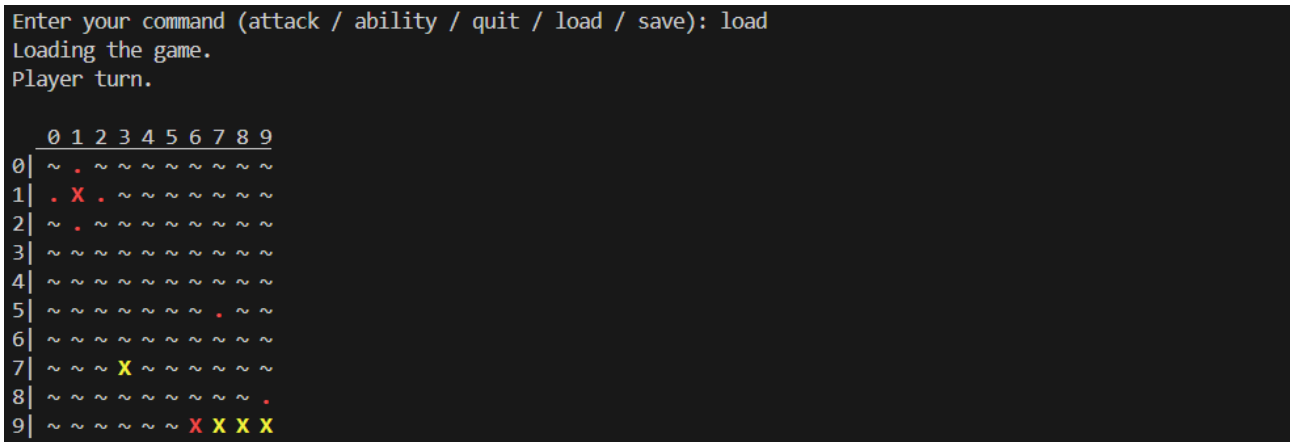

Рисунок 4 – Игра загружена

```
Enter your command (attack / ability / quit / load / save): attack 6 5
Please enter the coordinates (x, y): You won!
Do you want to continue playing? y/n
y
```

```
  0 1 2 3 4 5 6 7 8 9
0| ~ X X X . X . . . .
1| . ~ . . . ~ . . . X
2| . . . ~ . . . . . .
3| . . X . X ~ ~ ~ ~ .
4| . ~ X ~ . ~ . ~ ~ X
5| . ~ . ~ . . X . ~ X
6| . . . . . . ~ . ~ X
7| ~ . X ~ . . ~ . . X
8| ~ . X . . X . . . .
9| . . X ~ ~ X . . X X

Player turn.

  0 1 2 3 4 5 6 7 8 9
0| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
3| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
5| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
6| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9| ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Enter your command (attack / ability / quit / load / save):
```

Рисунок 5 – Победа игрока, обнуление поля бота

Выводы

Во время выполнения лабораторной работы, было изучено связывание классов и созданы соответствующие заданию классы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: game.cpp

```
#include "game.h"

void Game::startGame() {
    setShips(shipSizes, player);
    setShips(shipSizes, enemy);
    gameLoop();
}

void Game::setShips(std::vector<unsigned>& shipSizes, Unit& unit) {
    // Получение случайной последовательности координат поля
    std::vector<Coords> unit_coords;
    for(int y=0; y<unit.getField().getHeight(); y++) {
        for(int x=0; x<unit.getField().getWidth(); x++) {
            Coords coords = {x, y};
            unit_coords.push_back(coords);
        }
    }
    std::random_device rd;
    std::mt19937 gen(rd());
    std::shuffle(unit_coords.begin(), unit_coords.end(), gen);

    // Размещение кораблей
    Ship* ship_ptr;
    bool pos_flag;
    bool ship_is_placed;
    for (size_t i=0; i < shipSizes.size(); i++) {
        ship_ptr = unit.getShipManager().getShip(i);
        ship_is_placed = false;
        std::uniform_int_distribution<> distrib(0, 1);
        pos_flag = (distrib(gen) == 1) ? true : false;

        for (size_t j=0; j < unit_coords.size(); j++) {
            ship_is_placed = true;
            try {
                unit.getField().setShip(ship_ptr,  unit_coords[j].x,
unit_coords[j].y, pos_flag);
            } catch(const ShipOutsideTheFieldException& e) {
                ship_is_placed = false;
            } catch(const ShipsClosePlacementException& e) {
                ship_is_placed = false;
            }
            if(ship_is_placed) break;

            ship_is_placed = true;
            try {
                unit.getField().setShip(ship_ptr,  unit_coords[j].x,
unit_coords[j].y, !pos_flag);
            } catch(const ShipOutsideTheFieldException& e) {
                ship_is_placed = false;
            } catch(const ShipsClosePlacementException& e) {
```

```
                    ship_is_placed = false;
                }
                if(ship_is_placed) break;
            }
        }
    }

    void Game::gameLoop() {
        bool player_turn = false;
        while(!gameover) {
            std::cout << "Player turn.\n";
            enemy.getField().printField(true);
            while(!playerTurn(player_turn)) {
                std::cout << "Try again.\n";
            }
            if(gameover) break;

            if(player_turn) continue;
            enemy.getField().printField(true);
            std::cout << "Enemy turn.\n";
            player.getField().printField();
            enemyTurn();
            player.getField().printField();
            if(gameover) break;
        }
        std::cout << "Game Over!\n";
    }

    bool Game::playerTurn(bool& player_turn) {
        std::string command;
        int x, y;

        if (enemy.getShipManager().anotherShipDestroyed()) {
            player.getAbilityManager().addAbility();
        }

        std::cout << "Enter your command (attack / ability / quit / load
/ save): ";
        std::cin >> command;

        if (command == "attack") {
            std::cout << "Please enter the coordinates (x, y): ";
            std::cin >> x >> y;

            if (std::cin.fail()) {
                std::cerr << "Bad input.\n" << std::endl;
                std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                return false;
            }

            try {
                enemy.getField().attack(x, y);
                player_turn = false;
            } catch(const AttackOutsideTheFieldException& e) {
                std::cerr << e.what() << '\n';
                return false;
```

```
            }
        } else if (command == "ability") {

            try {
                player.getAbilityManager().useAbility(enemy.getField());
                player_turn = false;
            } catch(const EmptyAbilitiesException& e) {
                std::cerr << e.what() << '\n';
                return false;
            }
        } else if (command == "quit") {
            gameover = true;
        } else if (command == "load") {
            player_turn = true;
            std::cout << "Loading the game.\n";
            this->loadGame(file);
        } else if (command == "save") {
            player_turn = true;
            std::cout << "Saving the game." << std::endl;
            this->saveGame(file);
        } else {
            std::cerr << "Incorrect command." << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            return false;
        }
        if(enemy.getShipManager().areDestroyed()) {
            this->isPlayerWin = true;
            std::cout << "You won!\n";
            isGameEnded();
        }
        return true;
    }

    void Game::enemyTurn() {
        std::vector<Coords> cells_coords;
        player.getField().getDamagedCellsCoords(cells_coords);

        if(cells_coords.empty()) {
            player.getField().getNotDestroyedCellsCoords(cells_coords);
        }

        if (cells_coords.empty()) {
            isGameEnded();
            return;
        }

        std::random_device rd;
        std::mt19937 gen(rd());
        std::shuffle(cells_coords.begin(), cells_coords.end(), gen);

        auto firstPair = cells_coords.front();
        int randomX = firstPair.x;
        int randomY = firstPair.y;
        player.getField().attack(randomX, randomY);
```

```
        GameField::Cell   cell   =   player.getField().getCell(randomX,
randomY);
        if (cell.getStatus() == GameField::Cell::CellStatus::occupied) {
            int index = cell.getShipSegIndex();
            if(player.getShipManager().areDestroyed()) {
                this->isBotWin = true;
                std::cout << "You lost.\n";
                isGameEnded();
            }
        }
        return;
    }

    void Game::resetBot() {
        std::vector<unsigned> shipSizes = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1};

        GameField newField = GameField(10, 10);
        ShipManager newShips(shipSizes);
        this->enemy = Enemy(newShips, newField);

        setShips(shipSizes, enemy);
    }

    void Game::resetGame() {
        resetBot();
        std::vector<unsigned> shipSizes = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1};

        GameField newField = GameField(10, 10);
        ShipManager newShips(shipSizes);
        AbilityManager newAbilities;
        this->player = Player(newShips, newField, newAbilities);

        setShips(shipSizes, player);
    }

    void Game::isGameEnded() {
        if (!this->isPlayerWin && !this->isBotWin) {
            this->gameover = false;
            return;
        }

        std::cout << "Do you want to continue playing? y/n" << std::endl;
        std::string line;
        std::cin >> line;
        if (line == "n" || line == "N") {
            this->gameover = true;
            return;
        }

        if (this->isPlayerWin) {
            resetBot();
            this->isPlayerWin = false;
        }
        if (this->isBotWin) {
            resetGame();
            this->isBotWin = false;
        }
    }
```

```
void Game::loadGame(const std::string& file) {
    try {
        this->gameState.loadGame(file);
    } catch (nlohmann::json::exception& e) {
        std::cerr << "Error parsing JSON: " << e.what() << std::endl;
        return;
    } catch (HashMismatchException& e) {
        std::cerr << e.what() << std::endl;
        return;
    }
}

void Game::saveGame(const std::string& file) {
    this->gameState.saveGame(file);
}
```

Название файла: Game.hpp

```
#pragma once


#include "Ship.hpp"
#include "ShipManager.hpp"
#include "Field.hpp"
#include "Painter.hpp"
#include "Player.hpp"
#include "GameState.hpp"
#include "Abilities.hpp"
#include "AbilityManager.hpp"
#include "Exceptions/InvalidShipSizeException.hpp"
#include "Exceptions/InvalidCoordinateException.hpp"


class Game {
    private:
        Player player;
        Bot bot;
        GameState gameState;
        Painter painter;
        bool isPlayerWin;
        bool isBotWin;
        bool gameEnder;
    public:
        Game(Player player, Bot bot, GameState gameState, Painter
painter)
```

```
                :   player(player),   bot(bot),   gameState(gameState),
painter(painter), isPlayerWin(false), isBotWin(false), gameEnder(false) {}

            void usePlayerAbility();
            void doPlayerAttack();
            void doBotAttack();

            void startGame();
            void resetBot();
            void resetGame();
            void isGameEnded();

            void loadGame(const std::string& file);
            void saveGame(const std::string& file);
    };
```

Название файла: player.hpp

```
    #pragma once
    #include "shipmanager.h"
    #include "gamefield.h"
    #include "ability_manager.h"

    class Unit {
        protected:
            ShipManager& shipManager;
            GameField& field;
        public:
            Unit(ShipManager& shipManager, GameField& field)
                : shipManager(shipManager), field(field) {}

            virtual ShipManager& getShipManager() = 0;
            virtual GameField& getField() = 0;
            virtual ~Unit() {};
    };

    class Player : public Unit {
        private:
            AbilityManager& abilityManager;
```

```
        public:
            Player(ShipManager&    shipManager,    GameField&    field,
AbilityManager& abilityManager)
                :               Unit(shipManager,               field),
abilityManager(abilityManager) {}
            Player(const Player& other)
                :         Unit(other.shipManager,         other.field),
abilityManager(other.abilityManager) {}
            Player& operator=(const Player& other) {
                if (this != &other) {
                    this->shipManager = other.shipManager;
                    this->field = other.field;
                    this->abilityManager = other.abilityManager;
                }
                return *this;
            }

            ShipManager&    getShipManager()    override    {    return
shipManager; };
            GameField& getField() override { return field; };
            AbilityManager&      getAbilityManager()      {      return
abilityManager; };
    };

    class Enemy : public Unit {
        public:
            Enemy(ShipManager& shipManager, GameField& field)
                : Unit(shipManager, field) {}
            Enemy(const Enemy& other)
                : Unit(other.shipManager, other.field) {}

            Enemy& operator=(const Enemy& other) {
                if (this != &other) {
                    this->shipManager = other.shipManager;
                    this->field = other.field;
                }
                return *this;
            }
```

```
            ShipManager&    getShipManager()    override    {    return
shipManager; };
            GameField& getField() override { return field; };
    };
```

Название файла: Serialization.cpp

```
    #include "serialization.hpp"
    #include <fstream>

    void  Serialization::to_json(ShipManager&  shipManager,  std::string
key) {
        nlohmann::json jsm = nlohmann::json{};

        for (int i = 0; i < shipManager.getShipsNum(); i++) {
            Ship* temp = shipManager.getShip(i);
            std::string key = "ship" + std::to_string(i);
            jsm[key] = {
                {"length", temp->getLength()},
                {"horizontal", temp->isHorizontal()},
                {"index", temp->index},
                {"segments", nlohmann::json::array()}
            };

            for (int j = 0; j < temp->getLength(); j++) {
                Ship::Segment tempSegment = temp->getSegStatus(j);
                jsm[key]["segments"].push_back({
                    {"status", tempSegment}
                });
            }
        }

        j[key] = jsm;
    }

    void Serialization::to_json(GameField& field, std::string key) {
        nlohmann::json jf = nlohmann::json{};

        jf["width"] = field.getWidth();
```

```
        jf["height"] = field.getHeight();

        std::vector<std::vector<GameField::Cell>> temp = field.getField();
        for (int y = 0; y < field.getHeight(); y++) {
            for (int x = 0; x < field.getWidth(); x++) {
                std::string key = "cell" + std::to_string(y) + std::to_string(x);
                jf[key] = {
                    {"status", temp[y][x].getStatus()},
                    {"index", temp[y][x].getShipSegIndex()},
                    {"shipIndex", temp[y][x].getShipIndex()}
                };
            }
        }

        j[key] = jf;
    }

    void Serialization::to_json(AbilityManager& abilityManager, std::string key) {
        nlohmann::json jam = nlohmann::json{};

        if (abilityManager.queueIsEmpty()) {
            jam["abilities"].push_back("");
        } else {
            for (int i = 0; i < abilityManager.getQueueSize(); i++) {
                jam["abilities"].push_back(
                    abilityManager.getAbility(i).getName()
                );
            }
        }

        j[key] = jam;
    }
```

Название файла: Serialization.hpp

```
#pragma once
```

```
#include <string.h>
#include ".vscode/nlohmann/json.hpp"
#include "shipmanager.h"
#include "gamefield.h"
#include "ability_manager.h"

class Serialization {
    private:
        nlohmann::json& j;
    public:
        Serialization(nlohmann::json& j) : j(j) {};

        void to_json(ShipManager& shipManager, std::string key);
        void to_json(GameField& field, std::string key);
        void  to_json(AbilityManager&  abilityManager,  std::string
key);
};
```

Название файла: Deserialization.cpp

```
#include "deserialization.hpp"

void      Deserialization::from_json(ShipManager&      shipManager,
std::string key) {
    const auto& jsm = j.at(key);
    std::vector<unsigned> shipSizes;

    for (const auto& jship : jsm) {
        shipSizes.push_back(jship.at("length"));
    }

    ShipManager tempShipManager(shipSizes);
    shipManager = tempShipManager;

    for (size_t i = 0; i < shipSizes.size(); i++) {
        std::string key = "ship" + std::to_string(i);
        Ship* ship = shipManager.getShip(i);
        if (jsm.at(key).at("horizontal") == false) {
            ship->changePosition();
        }
```

```
            ship->index = jsm.at(key).at("index");

            for (int j = 0; j < shipSizes[i]; j++) {
                if   (jsm.at(key).at("segments").at(j).at("status")   !=
Ship::Segment::Intact) {
                    ship->getSegDamaged(j);
                    if (jsm.at(key).at("segments").at(j).at("status") ==
Ship::Segment::Destroyed) {
                        ship->getSegDamaged(j);
                    }
                }
            }
        }

        unsigned destroyedShipsNum = 0;
        for (size_t i = 0; i < shipSizes.size(); i++) {
            Ship* ship = shipManager.getShip(i);
            if (ship->isDestroyed()) {
                destroyedShipsNum++;
            }
        }
        shipManager.setDestroyedShips(destroyedShipsNum);
    }

    void Deserialization::from_json(GameField& field, std::string key) {
        const auto& jf = j.at(key);
        field = GameField(jf.at("width"), jf.at("height"));

        for (int y = 0; y < field.getHeight(); y++) {
            for (int x = 0; x < field.getWidth(); x++) {
                std::string   key   =   "cell"   +   std::to_string(y)   +
std::to_string(x);
                GameField::Cell& cell = field.getCell(x, y);
                cell.setStatus(jf.at(key).at("status"));
                cell.setShipSegIndex(jf.at(key).at("index"));
                cell.setShipIndex(jf.at(key).at("shipIndex"));
            }
        }
```

```
    }

    void Deserialization::from_json(AbilityManager& abilityManager,
std::string key) {
        const auto& jam = j.at(key);
        abilityManager = AbilityManager();
        abilityManager.clearQueue();

        for (const auto& jability : jam.at("abilities")) {
           if (jability == "Double Damage") {
               abilityManager.addAbility(new DoubleDamage());
           }
           else if(jability == "Scanner"){
               abilityManager.addAbility(new Scanner());
           }
           else if (jability == "Random Attack") {
               abilityManager.addAbility(new RandomAttack());
           }
        }
    }
```

Название файла: Deserialization.hpp

```
#pragma once
#include "shipmanager.h"
#include "gamefield.h"
#include "ability_manager.h"
#include <string.h>
#include ".vscode/nlohmann/json.hpp"

class Deserialization {
    private:
        nlohmann::json& j;
    public:
        Deserialization(nlohmann::json& j) : j(j) {};

        void from_json(ShipManager& shipManager, std::string key);
        void from_json(GameField& field, std::string key);
```

```
        void from_json(AbilityManager& abilityManager, std::string
key);
};
```

Название файла: Wrapper.cpp

```
#include "wrapper.hpp"

Wrapper::Wrapper(const std::string& filename) : file(filename) {};

Wrapper::~Wrapper() {
    if (file.is_open()) file.close();
}

void Wrapper::read(nlohmann::json& j) {
    if (!file.is_open() || !file.good()) {
        throw UnableToOpenFileException();
    }
    file >> j;
}

void Wrapper::write(nlohmann::json& j) {
    if (!file.is_open() || !file.good()) {
        throw UnableToOpenFileException();
    }
    file << j.dump(4);
}
```

Название файла: Wrapper.hpp

```
#pragma once

#include "file_exceptions.h"
#include ".vscode/nlohmann/json.hpp"

#include <iostream>
#include <fstream>

class Wrapper {
private:
    std::fstream file;
```

```
    public:
        Wrapper(const std::string& filename);

        ~Wrapper();

        void read(nlohmann::json& j);
        void write(nlohmann::json& j);
    };
```

Название файла: GameState.cpp

```
    #include "game_state.hpp"

    std::string GameState::getHash(const std::string& data) {
        std::hash<std::string> hash_fn;
        size_t hash = hash_fn(data);

        std::stringstream ss;
        ss << std::hex << hash;
        return ss.str();
    }

    Wrapper& operator<<(Wrapper& fileWrapper, GameState& state) {
        nlohmann::json j;
        nlohmann::json data;
        Serialization seri(data);

        seri.to_json(state.getPlayer().getShipManager(),
"playerShipManager");
        seri.to_json(state.getPlayer().getField(), "playerField");
        seri.to_json(state.getPlayer().getAbilityManager(),
"playerAbilityManager");
        seri.to_json(state.getEnemy().getShipManager(),
"enemyShipManager");
        seri.to_json(state.getEnemy().getField(), "enemyField");

        std::string jsonString = data.dump();

        j["data"] = data;
```

```
    j["hashValue"] = state.getHash(jsonString);

    try {
        fileWrapper.write(j);
    }
    catch (UnableToOpenFileException& e){
        std::cerr << e.what() << std::endl;
    }

    return fileWrapper;
}

Wrapper& operator>>(Wrapper& fileWrapper, GameState& state) {
    nlohmann::json j;

    try {
        fileWrapper.read(j);
    }
    catch (UnableToOpenFileException& e) {
        std::cerr << e.what() << std::endl;
        return fileWrapper;
    }


    nlohmann::json data = j.at("data");
    std::string savedHashValue = j.at("hashValue");

    std::string jsonString = data.dump();

    if (savedHashValue != state.getHash(jsonString)) {
        throw HashMismatchException();
    }

    Deserialization deseri(data);
    ShipManager shipManager;
    GameField field;
    AbilityManager abilityManager;

    ShipManager enemyShipManager;
```

```
        GameField enemyField;

        deseri.from_json(abilityManager, "playerAbilityManager");
        deseri.from_json(shipManager, "playerShipManager");
        deseri.from_json(field, "playerField");

        deseri.from_json(enemyShipManager, "enemyShipManager");
        deseri.from_json(enemyField, "enemyField");

        state.getPlayer().getShipManager() = shipManager;
        state.getPlayer().getField() = field;
        state.getPlayer().getAbilityManager() = abilityManager;

        state.getEnemy().getShipManager() = enemyShipManager;
        state.getEnemy().getField() = enemyField;

        state.placeShips(state.getPlayer().getShipManager(),
state.getPlayer().getField());
        state.placeShips(state.getEnemy().getShipManager(),
state.getEnemy().getField());

        return fileWrapper;
    }

    void  GameState::placeShips(ShipManager&  shipManager,  GameField&
field) {
        for (int y = 0; y < field.getHeight(); y++) {
            for (int x = 0; x < field.getWidth(); x++) {
                auto& cell = field.getCell(x, y);
                if                 (cell.getStatus()                 ==
GameField::Cell::CellStatus::occupied) {

cell.setShip(shipManager.getShip(cell.getShipIndex()));
                }
            }
        }
    }

    void GameState::loadGame(const std::string& file) {
```

```
        Wrapper fileWrapper(file);
        fileWrapper >> *this;
    }

    void GameState::saveGame(const std::string& file) {
        std::ofstream ofs(file, std::ofstream::out | std::ofstream::trunc);
        Wrapper fileWrapper(file);
        fileWrapper << *this;
    }
```

Название файла: GameState.hpp

```
    #pragma once

    #include "serialization.hpp"
    #include "deserialization.hpp"
    #include "player.hpp"
    #include "wrapper.hpp"
    #include "file_exceptions.h"

    #include <fstream>

    class GameState {
        private:
            Player& player;
            Enemy& enemy;
        public:
            GameState(Player& player, Enemy& enemy) : player(player), enemy(enemy) {};

            std::string getHash(const std::string& data);
            void placeShips(ShipManager& shipManager, GameField& field);

            void loadGame(const std::string& file);
            void saveGame(const std::string& file);

            Player& getPlayer() { return this->player; };
            Enemy& getEnemy() { return this->enemy; }
```

```
};
```