

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

Хайруллов Д.Л.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Создать класс игры, состояния игры. Реализовать игровой цикл, сохранение и возможность загружать сохранения.

Задание

Создать класс игры, который реализует следующий игровой цикл:

Начало игры

Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

В случае проигрыша пользователь начинает новую игру

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.

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 самой игрой можно использовать обертки над командами

При работе с файлом используйте идиому RAII.

Выполнение работы

Класс GameState

- `Player* human_player`: Указатель на объект, представляющий человека-игрока.
- `AIPlayer* ai_player`: Указатель на объект, представляющий искусственного игрока (AI).
- `int current_round**`: Переменная, хранящая номер текущего раунда игры.
- `GameState(Player* human_player, AIPlayer* ai_player)`: Конструктор, который принимает указатели на объекты игрока и ИИ, и инициализирует поля класса.
- `friend FileWrapper& operator<<(FileWrapper& file, const GameState& state)`: Дружественная функция перегрузки оператора для записи состояния игры в объект `'FileWrapper'`.
- `friend FileWrapper& operator>>(FileWrapper& file, GameState& state)`: Дружественная функция перегрузки оператора для чтения состояния игры из объекта `'FileWrapper'`.

Класс Game

- `GameState* game_state`: Указатель на объект состояния игры, который хранит информацию о текущем прогрессе.
- `bool WIN_FLAG`: Флаг, указывающий на победу в игре (`true`, если игрок выиграл).
- `int round_number`: Номер текущего раунда игры, начиная с 1.
- `bool game_status`: Флаг, указывающий, продолжается ли игра (`true`, если игра активна).
- `GameField& player_game_field`: Ссылка на игровое поле человека-игрока.
- `GameField& ai_player_game_field`: Ссылка на игровое поле искусственного игрока (AI).

- `Player* human_player`: Указатель на объект человека-игрока.
- `AIPlayer* ai_player`: Указатель на объект искусственного игрока (AI).
- `Game(GameField& player_game_field, GameField& ai_player_game_field)`: Конструктор, который принимает ссылки на игровое поле для игрока и ИИ, и инициализирует поля класса.
- `~Game()`: Деструктор, который выполняет очистку ресурсов при завершении объекта.
- `void StartGame()`: Метод, который запускает игру и устанавливает начальные условия.
- `void Reset()`: Метод для сброса состояния игры к первоначальным значениям.
- `void PlacingShips()`: Метод, позволяющий человеку установить свои корабли на игровом поле.
- `void AIPlacingShips()`: Метод, который управляет процессом установки кораблей искусственным игроком.
- `void PlayerTurn()`: Метод, который обрабатывает ход человека-игрока.
- `void AIPlayerTurn()`: Метод, который управляет ходом искусственного игрока.

Класс FileWrapper

- `std::fstream file`: Объект типа `fstream`, который используется для работы с файловым потоком. Он позволяет как читать, так и записывать данные в файл.
- `FileWrapper(const std::string& filename, std::ios::openmode mode)`: Конструктор, который принимает название файла и режим открытия (например, чтение, запись и т.д.). Он инициализирует объект `file` с указанными параметрами.
- `~FileWrapper()`: Деструктор.

- `template <typename T> void write(const T& data):` Шаблонный метод для записи данных в файл. Он принимает данные любого типа T и записывает их в открытый файл.
- `template <typename T> void read(T& data):` Шаблонный метод для чтения данных из файла. Он принимает ссылку на переменную типа T и заполняет ее прочитанными из файла данными..

Класс InputHandler

- `InputHandler():` Конструктор класса, отвечающий за инициализацию объекта `InputHandler`.
- `Coords CoordsInput(Player* human_player):` Метод, который отвечает за получение координат от игрока. Он может запрашивать у игрока ввод координат для атаки или размещения кораблей и возвращает объект типа `Coords`, представляющий введенные значения.
- `ShipOrientation OrientationInput(Player* human_player):` Метод, который получает ориентацию корабля от игрока. Он может запрашивать у игрока, как будет ориентирован корабль и возвращает значение типа `ShipOrientation`, определяющее направление расположения корабля.
- `std::string CommandInput(bool attack_flag, bool ability_flag):` Метод, предназначенный для получения команды от игрока. В зависимости от флагов `attack_flag` и `ability_flag`, метод может возвращать разные строки команд (например, действия для атаки или использование умений). Возвращает строку, содержащую команду от игрока.

Класс SavingsHandler

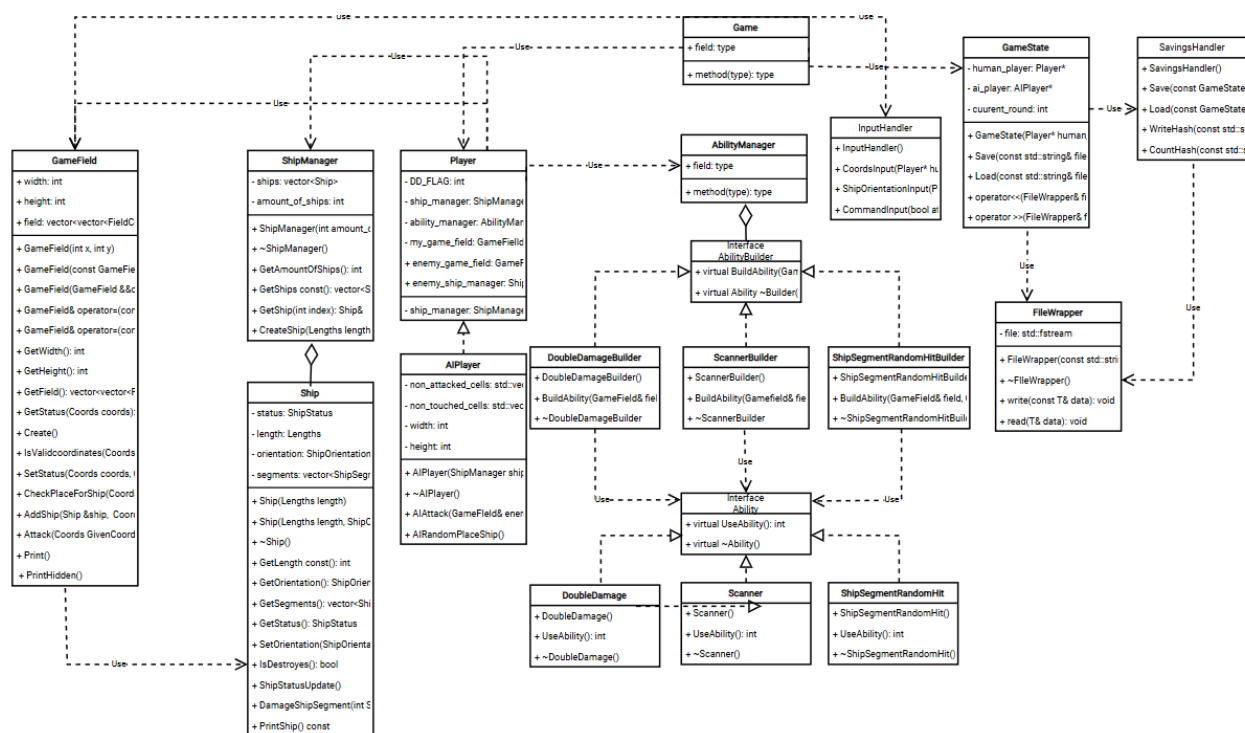
- `SavingsHandler():` Конструктор класса, который инициализирует объект `SavingsHandler`.
- `void Save(const GameState& game_state):` Метод, который сохраняет текущее состояние игры, представленное объектом типа `GameState`. Он

отвечает за сериализацию данных и запись их в файл или другое хранилище, чтобы сохранить прогресс игрока.

- `void Load(GameState& game_state)`: Метод, предназначенный для загрузки состояния игры из файла. Он считывает сохраненные данные и обновляет объект `GameState` переданного в качестве ссылки. Это позволяет восстановить игру до последнего сохраненного состояния.
- `void WriteHash(const std::string& filename, FileWrapper& file_hash)`: Метод записывает хеш-сумму (криптографический код для проверки целостности) в файл с указанным именем. Использует объект `FileWrapper` для работы с хеш-суммой, чтобы обеспечить безопасность и подлинность данных.
- `CountHash(const std::string& filename)`: Метод, который подсчитывает количество хеш-значений, записанных в файл с указанным именем. Возвращает целое число, показывающее, сколько хешей было найдено, что может быть полезно для проверки состояния сохранений или их целостности.

Разработанный программный код см. в Приложении А.

UML-диаграмма классов:



Тестирование

Создание поля, создание менеджера кораблей, их расстановке на игровом поле, атака клеток поля, вывод поля в раскрытом и скрытом режимах:

```
int main()
{
    GameField game_field(10, 10);
    ShipManager manager(2, std::vector<Lengths>{Three, Two});
    std::vector<Ship> ships = manager.GetShips();

    manager.PushShip(Four);

    ships = manager.GetShips();

    game_field.AddShip(ships[0], {1, 1}, Horizontal);
    game_field.AddShip(ships[1], {2, 9}, Vertical);
    game_field.AddShip(ships[2], {4, 4}, Horizontal);

    game_field.Attack({1, 1});
    game_field.Attack({1, 1});
    game_field.Attack({1, 2});
    game_field.Attack({2, 1});

    game_field.Print();
    game_field.PrintHidden();

    return 0;
}
```

Рис 1. Пример программы

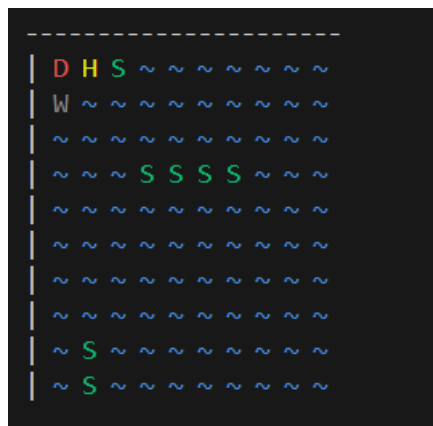


Рис 2. Вывод доски в раскрытом режиме

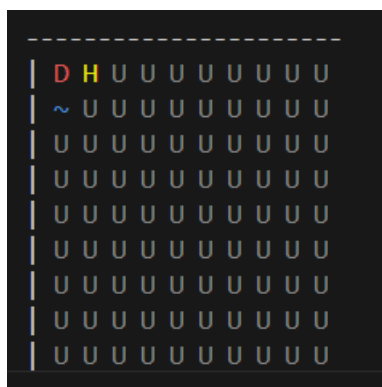


Рис 3. Вывод доски в скрытом режиме

Выводы

Были реализованы необходимые классы игры, состояния игры. Были реализованы игровой цикл и возможности сохранять и загружать состояние игры. Была построена UML диаграмма.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <vector>

#include "headers/Ship.h"
#include "headers/ShipManager.h"
#include "headers/GameField.h"
#include "headers/Structures.h"

int main()
{
    GameField game_field(10, 10);
    ShipManager manager(2, std::vector<Lengths>{Three, Two});
    std::vector<Ship> ships = manager.GetShips();

    manager.PushShip(Four);

    ships = manager.GetShips();

    game_field.AddShip(ships[0], {1, 1}, Horizontal);
    game_field.AddShip(ships[1], {2, 9}, Vertical);
    game_field.AddShip(ships[2], {4, 4}, Horizontal);

    game_field.Attack({1, 1});
    game_field.Attack({1, 1});
    game_field.Attack({1, 2});
    game_field.Attack({2, 1});

    game_field.Print();
    game_field.PrintHidden();

    return 0;
}
```

Название файла: structures.h

```

#ifndef STRUCTURES_HPP
#define STRUCTURES_HPP

#include <memory>
#include "Ship.h"

class Ship;

enum Lengths
{
    One = 1,
    Two = 2,
    Three = 3,
    Four = 4
};

enum ShipOrientation{Vertical,Horizontal};

enum class ShipSegmentStatus { Intact, Damaged, Destroyed };

enum class ShipStatus { Intact, Damaged, Destroyed };

enum class CellStatus
{
    Empty,
    Ship,
    DamagedShip,
    DestroyedShip,
    Unknown
};

struct Coords
{
    int x;
    int y;

    bool operator==(const Coords &other) const{
        return x == other.x && y == other.y;
    }
}

```

```
};

struct ShipSegment
{
    Coords ShipSegmentCoords;
    ShipSegmentStatus status;
};
```

```
struct FieldCell
{
    Ship* pShip;
    Coords coords;
    CellStatus status;
};
```

```
#endif
```

Название файла: ship.h

```
#ifndef SHIP_HPP
#define SHIP_HPP

#include "Structures.h"
#include <vector>
#include <algorithm>
#include <stdexcept>
#include <iostream>

class Ship
{
private:
    ShipStatus status;
    Lengths length;
    ShipOrientation orientation;
    std::vector<ShipSegment> segments;

public:
    Ship(Lengths length);
    Ship(Lengths length, ShipOrientation orientation);
```

```

    ~Ship() {}

    const int GetLength() const;
    const ShipOrientation GetOrientation() const;
    const std::vector<ShipSegment> GetSegments() const;

    void SetOrientation(ShipOrientation orientation);
    bool IsDestroyed() const;
    void ShipStatusUpdate();
    void FillSegments(Coords coords);
    void DamageShipSegment(Coords coords);
    void PrintShip() const;

};

#endif

```

Название файла: Ship.h

```

#ifndef SHIP_H
#define SHIP_H

#include <vector>
#include <algorithm>
#include <stdexcept>

enum ShipOrientation
{
    Horizontal,
    Vertical
};

enum Length
{
    ONE = 1,
    TWO = 2,
    THREE = 3,
    FOUR = 4
};

```

```

enum ShipSegmentState
{
    Intact,
    Damaged,
    Destroyed
};

class Ship
{
private:
    Length length_;
    ShipOrientation orientation_;
    std::vector<ShipSegmentState> segments_;

public:
    Ship();
    Ship(Length length);
    Ship(Length length, ShipOrientation orientation);
    Ship(const Ship &other);
    Ship(Ship &&other);
    ~Ship() {};

    Ship &operator=(const Ship &other);
    Ship &operator=(Ship &&other);

    const Length &GetLength() const;
    const ShipOrientation &GetOrientation() const;
    const std::vector<ShipSegmentState> &GetSegments() const;

    void SetOrientation(ShipOrientation orientation);
    void HitSegment(int index);
    bool IsDestroyed() const;
};

namespace std
{
    bool operator<(const std::reference_wrapper<Ship> &a, const
std::reference_wrapper<Ship> &b);
}

```



```
#endif
```

Название файла: Ship.cpp

```
#include "../headers/Ship.h"
```

```
//-----
```

```
//CONSTRUCTORS
```

```
//-----
```

```
Ship::Ship(Lengths length) : Ship(length, Horizontal) {}
```

```
Ship::Ship(Lengths length, ShipOrientation orientation) : length(length),  
orientation(orientation)
```

```
{
```

```
    status = ShipStatus::Intact;
```

```
    for (int i = 0; i < this->length; i++)
```

```
    {
```

```
        segments.push_back(ShipSegment{Coords{0, 0},
```

```
ShipSegmentStatus::Intact});
```

```
    }
```

```
}
```

```
//-----
```

```
//GET_METHODS
```

```
//-----
```

```
const std::vector<ShipSegment> Ship::GetSegments() const
```

```
{
```

```
    return segments;
```

```
}
```

```
const int Ship::GetLength() const
```

```
{
```

```
    return length;
```

```
}
```

```
const ShipOrientation Ship::GetOrientation() const
```

```

{
    return orientation;
}

//-----
//SHIP_METHODS
//-----

void Ship::SetOrientation(ShipOrientation orientation)
{
    this->orientation = orientation;
}

bool Ship::IsDestroyed() const
{
    for (auto seg : segments){
        if (seg.status != ShipSegmentStatus::Destroyed) {
            return false;
        }
    }
    return true;
}

void Ship::ShipStatusUpdate()
{
    if (this->IsDestroyed()){
        this->status = ShipStatus::Destroyed;
        return;
    }

    for (auto seg : segments){
        if (seg.status != ShipSegmentStatus::Intact){
            this->status = ShipStatus::Damaged;
            return;
        }
    }
}

```

```

void Ship::FillSegments(Coords coords)
{
    for (int i = 0; i < segments.size(); i++){
        segments[i].ShipSegmentCoords = (orientation ==
ShipOrientation::Vertical)
        ? Coords{coords.x, coords.y + i}
        : Coords{coords.x + i, coords.y};
    }
}

void Ship::DamageShipSegment(Coords coords)
{
    int index = -1;
    for(int i = 0; i < segments.size(); i++)
    {
        if(segments[i].ShipSegmentCoords == coords)
        {
            index = i;
            break;
        }
    }

    if (index < 0)
    {
        throw std::invalid_argument("Invalid index");
    }
    if (segments[index].status == ShipSegmentStatus::Intact)
    {
        segments[index].status = ShipSegmentStatus::Damaged;
    }
    else if (segments[index].status == ShipSegmentStatus::Damaged)
    {
        segments[index].status = ShipSegmentStatus::Destroyed;
    }
    else if (segments[index].status == ShipSegmentStatus::Destroyed)
    {
        std::cout << "This ship segment is already destroyed";
    }
}

```

```

        this->ShipStatusUpdate();
    }

void Ship::PrintShip() const
{
    std::cout << "Ship status is " << (this->IsDestroyed() ? "Destroyed" :
    "Intact") << '\n';
    std::cout << "Length of ship is " << segments.size() << '\n';
    std::cout << "Coordinates of ship are " <<
    segments[0].ShipSegmentCoords.x << ' ' << segments[0].ShipSegmentCoords.y
    << '\n';
    std::cout << "Orientation of ship is " << ((orientation ==
    ShipOrientation::Vertical) ? "Vertical" : "Horizontal") << "\n\n";

    for(auto seg : segments)
    {
        std::string word;
        switch (seg.status)
        {
            case ShipSegmentStatus::Intact:
                word = "Intact";
                break;
            case ShipSegmentStatus::Damaged:
                word = "Damaged";
                break;
            case ShipSegmentStatus::Destroyed:
                word = "Destroyed";
                break;
        }

        std::cout << "Coordinates of segment are " <<
        seg.ShipSegmentCoords.x << ' ' << seg.ShipSegmentCoords.y << '\n';
        std::cout << "Status of segment is " << word << '\n';
    }
    std::cout << '\n';
}

```

Название файла: ShipManager.h

```
#ifndef SHIP_MANAGER_H
#define SHIP_MANAGER_H

#include <vector>
#include <iostream>
#include "GameField.h"

class ShipManager
{
private:

    std::vector<Ship> ships;

public:
    ShipManager(int shipsAmount, std::vector<Lengths> sizes);
    ~ShipManager() {};

    const std::vector<Ship> GetShips() const;

    void CreateShip(Lengths length, ShipOrientation orientation =
ShipOrientation::Horizontal);
    void PushShip(Lengths length);

};

#endif
```

Название файла: ShipManager.cpp

```
#include "../headers/ShipManager.h"

ShipManager::ShipManager(int AmountOfShips, std::vector<Lengths> lengths)
{
    for (int i = 0; i < AmountOfShips; i++){
        this->CreateShip(lengths[i], ShipOrientation::Horizontal);
    }
}

const std::vector<Ship> ShipManager::GetShips() const
```

```

{
    return ships;
}

void ShipManager::CreateShip(Lengths length, ShipOrientation orientation)
{
    ships.emplace_back(Ship(length, orientation));
}

void ShipManager::PushShip(Lengths length)
{
    this->CreateShip(length, ShipOrientation::Horizontal);
}

```

Название файла: GameField.h

```

#ifndef GAME_FIELD_H
#define GAME_FIELD_H

#include <map>
#include <iostream>
#include <vector>
#include "Ship.h"

class GameField
{
private:
    int height;
    int width;

    std::vector<std::vector<FieldCell>> field;

public:
    GameField(int x, int y);
    GameField(const GameField &other);
    GameField(GameField &&other);
    GameField& operator=(const GameField &other);
    ~GameField() {};
}

```

```

const int GetWidth() const;
const int GetHeight() const;
const std::vector<std::vector<FieldCell>> GetField() const;

void Create();

bool IsValidCoordinate(Coords coords) const;
void SetStatus(Coords cords, CellStatus status);
bool CheckPlaceForShip(Coords coords);
    void AddShip(Ship &ship, Coords GivenCoords, ShipOrientation
orientation);
    void Attack(Coords GivenCoords);
    void Print();
    void PrintHidden() const;
};

```

```

#endif

```

Название файла: GameField.cpp

```

#include "../headers/GameField.h"

```

```

//-----
//CONSTRUCTORS
//-----

```

```

GameField::GameField(int x, int y) : width(x), height(y)
{
    if (x <= 0 || y <= 0)
    {
        throw std::invalid_argument("Invalid game field size values");
    }
    this->Create();
}

```

```

GameField::GameField(const GameField &other)
{
    width = other.width;
    height = other.height;
    field = other.field;
}

GameField::GameField(GameField &&other)
{
    width = other.width;
    height = other.height;
    field = std::move(other.field);
    other.height = 0;
    other.width = 0;
}

GameField &GameField::operator=(const GameField &other)
{
    if (this != &other)
    {
        width = other.width;
        height = other.height;
        field = other.field;
    }
    return *this;
}

//-----
//GET_METHODS
//-----

const int GameField::GetWidth() const
{
    return width;
}

const int GameField::GetHeight() const
{
    return height;
}

```



```

}

const std::vector<std::vector<FieldCell>> GameField::GetField() const
{
    return field;
}

//-----
//FIELD_METHODS
//-----

void GameField::Create()
{
    field.resize(height);
    for (int y = 0; y < height; y++) {
        field[y].reserve(width);
        for (int x = 0; x < width; x++) {
            field[y][x] = FieldCell{NULL, Coords{x, y}, CellStatus::Unknown};
        }
    }
}

bool GameField::IsValidCoordinate(Coords coords) const
{
    return coords.y < height && coords.x < width && coords.x >= 0 &&
coords.y >= 0;
}

void GameField::SetStatus(Coords coords, CellStatus status)
{
    field[coords.y][coords.x].status = status;

    if(status == CellStatus::DamagedShip || status ==
CellStatus::DestroyedShip)
    {
        field[coords.y][coords.x].pShip->DamageShipSegment(coords);
    }
}

```

```

bool GameField::CheckPlaceForShip(Coords coords)
{
    if (!IsValidCoordinate(coords))
    {
        return false;
    }

    int count = 0;
    for (int y = coords.y - 1; y <= coords.y + 1; y++)
    {
        for (int x = coords.x - 1; x <= coords.x + 1; x++)
        {
            if (x < 0 || y < 0 || x > width - 1 || y > height - 1)
            {
                count++;
            }
            else if (field[y][x].status != CellStatus::Ship)
            {
                count++;
            }
        }
    }
    return count == 9;
}

void GameField::AddShip(Ship &ship, Coords GivenCoords, ShipOrientation
orientation)
{
    Coords coords{GivenCoords.x - 1, GivenCoords.y - 1};
    std::vector<Coords> cells;
    bool flag = true;

    ship.SetOrientation(orientation);

    if (ship.GetOrientation() == Horizontal)
    {
        for (int i = 0; i < ship.GetLength(); i++)
        {

```

```

        if (CheckPlaceForShip({coords.x + i, coords.y}))
        {
            cells.push_back({coords.x + i, coords.y});
        }
        else
        {
            flag = false;
            break;
        }
    }
}
else if (ship.GetOrientation() == Vertical)
{
    for (int i = 0; i < ship.GetLength(); i++)
    {
        if (CheckPlaceForShip({coords.x, coords.y + i}))
        {
            cells.push_back({coords.x, coords.y + i});
        }
        else
        {
            flag = false;
            break;
        }
    }
}
if (flag)
{
    ship.FillSegments(coords);

    Ship* ptr_ship = &ship;

    for (Coords coordinate : cells)
    {
        field[coordinate.y][coordinate.x].pShip = ptr_ship;
        this->SetStatus(coordinate, CellStatus::Ship);
    }
}
else
{

```

```

        std::cout << "Can't place the ship at these coordinates\n";
    }
}

void GameField::Attack(Coords GivenCoords)
{
    Coords coords{GivenCoords.x - 1, GivenCoords.y - 1};

    if(field[coords.y][coords.x].status == CellStatus::Unknown)
    {
        this->SetStatus(coords, CellStatus::Empty);
    }
    else if(field[coords.y][coords.x].status == CellStatus::Ship)
    {
        this->SetStatus(coords, CellStatus::DamagedShip);
    }
    else if(field[coords.y][coords.x].status == CellStatus::DamagedShip)
    {
        this->SetStatus(coords, CellStatus::DestroyedShip);
    }
    else
    {
        std::cout << "This cell can't be attacked\n";
    }
}

void GameField::Print()
{
    std::cout<<"\n";
    for (int i = 0; i <= width; i++){
        std::cout << "--";
    }

    std::cout <<'\n';
    for (int y = 0; y < height; y++){
        std::cout << "| ";

        for (int x = 0; x < width; x++){
            switch (field[y][x].status)
            {

```

```

        case CellStatus::Ship:
            std::cout<< "\033[32m" <<"S " << "\033[0m";
            break;
        case CellStatus::DamagedShip:
            std::cout << "\033[33m" << "H " << "\033[0m";
            break;

        case CellStatus::DestroyedShip:
            std::cout << "\033[31m" << "D " << "\033[0m";
            break;
        case CellStatus::Empty:
            std::cout << "\033[30m" << "W " << "\033[0m";
            break;
        default:
            std::cout << "\033[34m" << "~ " << "\033[0m";
            break;
    }
}

std::cout << "\n";
}

std::cout << std::endl;
}

void GameField::PrintHidden() const
{
    std::cout<<"\n";
    for (int i = 0; i <= width; i++){
        std::cout << "--";
    }

    std::cout <<'\n';
    for (int y = 0; y < height; y++){
        std::cout << "| ";

        for (int x = 0; x < width; x++){
            switch (field[y][x].status)
            {
                case CellStatus::DamagedShip:
                    std::cout << "\033[33m" << "H " << "\033[0m";

```

```

        break;
    case CellStatus::DestroyedShip:
        std::cout << "\033[31m" << "D " << "\033[0m";
        break;
    case CellStatus::Empty:
        std::cout << "\033[34m" << "~ " << "\033[0m";
        break;
    default:
        std::cout << "\033[30m" << "U " << "\033[0m";
        break;
    }
}
std::cout << "\n";
}
std::cout << std::endl;
}

```