

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И  
ИНФОРМАТИКИ»

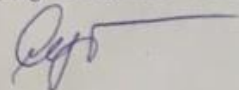
КУРСОВАЯ РАБОТА

по дисциплине «Объектно Ориентированное Программирование»  
Вариант 1 «Морской бой»

Выполнил:  
студент группы ИС-741  
Герасимов С.Д.

Проверил:  
ассистент кафедры ПМиК  
Суходоева Наталья Николаевна

28.11.18 *ошмичко*



Новосибирск 2018

## Содержание:

Задание:.....	3
Описание иерархии объектов и методов объектов:.....	3
Алгоритм основной программы: .....	6
Работа программы: .....	7
Заключение:.....	10
Код программы: .....	11
Game.h.....	11
Game.cpp .....	12
board.cpp .....	14
play.cpp.....	17
bot.cpp .....	19
main.cpp.....	22

## Задание:

Реализовать игру «Морской бой»(графический режим).

## Описание иерархии объектов и методов объектов:

Для технической реализации проекта были использованы следующие объекты и сопутствующие им алгоритмы:

### 1. Константы:

#### 1) Определяющие положение корабля

```
#define HOR 0//горизонтальное положение корабля  
#define VER 1//вертикальное
```

#### 2) Для окраски

```
enum ConsoleColor  
{  
    Black = 0,  
    Blue = 1, //пустая или не прострелянная клетка  
    Green = 2,  
    Cyan = 3, //окрашивание корабля  
    Red = 4, //для окрашивания попадания  
    Magenta = 5,  
    Brown = 6,  
    LightGray = 7,  
    DarkGray = 8,  
    LightBlue = 9,  
    LightGreen = 10,  
    LightCyan = 11,  
    LightRed = 12,  
    LightMagenta = 13,  
    Yellow = 14, //окрашивание промаха  
    White = 15  
};
```

#### 3) Определяющие результат выстрела

```
enum Status {  
    HITTED = -1, //прострелянная  
    MISS = 0, //промах  
    HIT = 1, //попадание  
    KILL = 2 //потопление  
};
```

#### 4) Для считывания клавиатуры

```
enum Keys {  
    UP = 72,  
    DOWN = 80,  
    RIGHT = 77,  
    LEFT = 75,  
    ENTER = 13,  
    SPACE = 32  
};
```

## 2. Класс Point- для хранения координат поля

```
class Point {
protected:
    int x;//координата по вертикали
    int y;//координата по горизонтали
public:
    Point(int x = 0, int y = 0) : x(x), y(y) {};//конструктор с параметрами по умолчанию
    friend Board;//дружественные классы для доступа к Point
    friend Player;
    friend Bot;
};
```

## 3. Класс Ship-наследуемый класс от Point, для хранения информации о корабле, используется при расстановке кораблей

```
class Ship : public Point {
private:
    int size;//размер корабля(1-4)
    int dir;//положение корабля(вертикальное или горизонтальное)
public:
    Ship(int size, int dir, int x = 0, int y = 0) : Point(x, y), size(size), dir(dir) {};//
    friend Board;//дружественные классы для доступа к Ship
    friend Player;
    friend Bot;
};
```

## 4. Класс Board – абстрактный класс, задающий интерфейс

```
class Board {
protected:
    ConsoleColor deck[10][10];//содержит информацию о поле
    int hp;//количество здоровья(сумма не подбитых ячеек всех кораблей)
public:
    Board(int hp = 20); //конструктор с параметром по умолчанию, задающий колво здоровья
    //перезузка функции board_print
    void board_print(); //вывод доски
    void board_print(Point a); //вывод доски с выделением точки(для выбора поля для стрельбы)
    void board_print(Ship a); //вывод доски с выделением корабля(для расстановки кораблей)
    int set_ship(Ship a); //функция проверяющая допустимость расстановки корабля
    int check_status(Point a, int dir); //функция проверяющая "убит" или "ранен" корабль
    int shot(Point a); //функция возвращающая результат стрельбы
    int check_win() { //функция проверяющая статус игры, если чье то здоровье опускается до 0
        if (this->hp == 0) return 0; //...то он считается проигравшим
        else return 1;
    };
    //чистые виртуальные методы, переопределяемые при наследовании
    virtual void prepare() = 0; //чистый виртуальный метод реализуется для подготовки к игре
    virtual int play(Board *) = 0; //чистый виртуальный метод задающий геймплей
    friend Player; //для уменьшения здоровья
    friend Bot;
};
```

## 5. Класс Player-наследуемый от абстрактного класса Board, задает функции для взаимодействия с игроком

```
class Player: public Board {
public:
    Player(int hp = 20): Board(hp) {};//
    void prepare(); //подготовка к игре(расстановка кораблей)
    int play(Board *); //сам геймплей, принимает указатель на объект содержащий информацию о противнике
};
```

6. Класс Bot-наследуемый от абстрактного класса Board, служит реализацией игры компьютера.

```
class Bot: public Board {
private:
    Point shoot; //точка для стрельбы
    //переменные, показывающие прострелянные стороны, при попадании в неопалубный корабль
    int right;
    int left;
    int up;
    int down;
public:
    Bot(int hp = 20) : Board(hp) { //конструктор, задающий начальные параметры
        right = left = up = down = 0;
        shoot.x = -1;
        shoot.y = -1;
    };
    void prepare(); //подготовка к игре(расстановка кораблей)
    int play(Board *); //реализация игры бота, принимает указатель на объект содержащий информацию о игроке
};
```

7. Класс Game-класс, определяющий меню, саму игру

```
class Game {
private:
    string rules; //Хранит правила, считываемые из файла rules.txt
public:
    Game();
    void menu(); //функция для работы(перемещение, выбор) с меню
    void gameplay(int mode); //функция, задающая игру и очередность ходов, mode - определяет колво игроков
    void print(int i); //вывод меню или правил, i - определяет состояние меню для вывода на экран
};
```

### Дополнительные функции:

```
void SetColor(ConsoleColor text, ConsoleColor background); //функция для покраски
void check_ready(); //проверка готовности игрока
```

SetColor() – используется в классе Board, для покраски доски; используется в классе Game, для покраски меню.

Check\_ready() – используется в классе Game, для проверки готовности игроков и перехода очереди к следующему игроку.

## Алгоритм основной программы:

При запуске программы пользователю, в главном меню, предлагается выбрать режим игры (1 или 2 игрока) или прочтение правил игры.

При выборе игры вызывается соответствующая функция класса Game, принимающая в качестве параметра кол-во игроков, далее в функции создается массив указателей базового класса на объекты производного класса и им присваиваются производные классы в соответствии с количеством игроков

```
void Game::gameplay(int mode)
{
    Board **player;
    int i, j;
    player = new (Board *[2]);
    player[0] = new Player;
    if (mode == 1)
        player[1] = new Bot;
    else if (mode == 2)
        player[1] = new Player;
```

Далее управление передается функциям подготовки (расстановка кораблей), где игрок и бот/второй игрок подготавливают свое поле.

```
for (i = 0; i < 2; i++) {
    player[i]->prepare();
    if (mode == 2)
        check_ready();
}
```

После того, как соперники подготовили свое поле, вызывается бесконечный цикл, который осуществляет последовательное выполнение ходов, выходом из цикла служит удачная проверка на поражение одного из игроков

```
while (1) {
    win = player[i]->play(player[j]);
    system("pause");
    if (win == 0) {
        win = i;
        break;
    }
    if (mode == 2)
        check_ready();

    win = player[j]->play(player[i]);
    if (mode == 2)
        system("pause");
    if (win == 0) {
        win = j;
        break;
    }
    if (mode == 2)
        check_ready();
}
```

## Работа программы:

При запуске программы выводиться главное меню, изображенное на рис.1

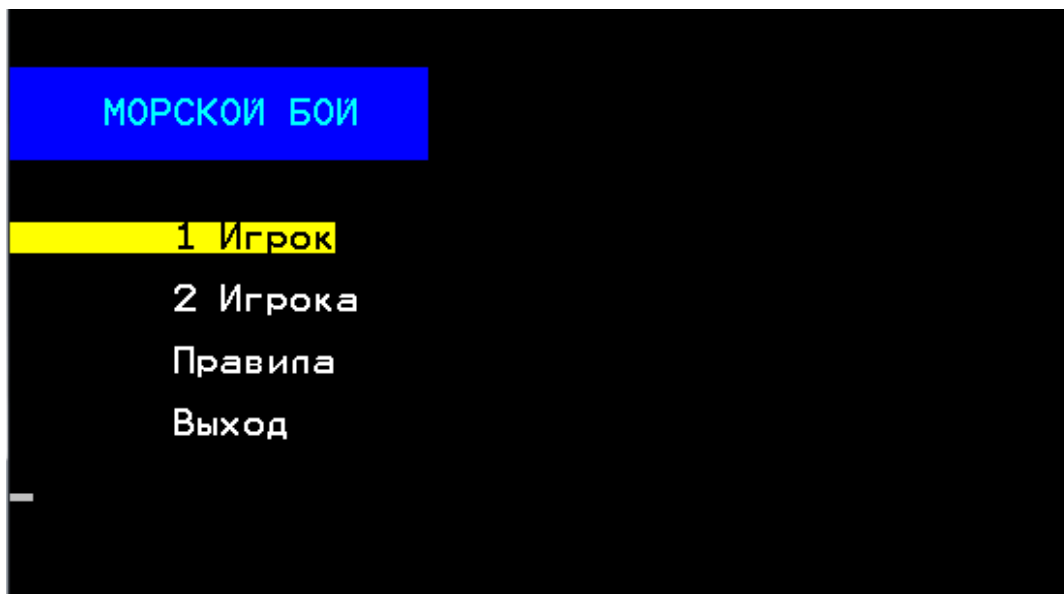


Рис.1 Главное меню

При выборе пункта правила происходит переход в раздел правила рис.2

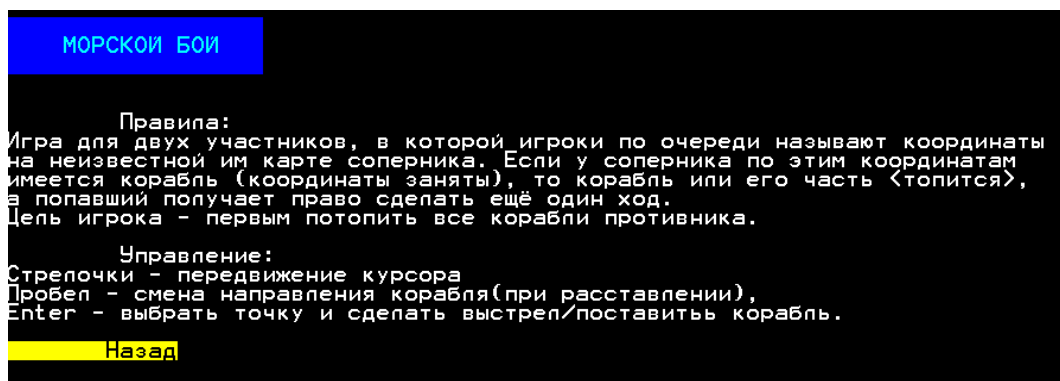


Рис.2 Раздел правил игры

При выборе игры (1 игрок или 2 игрока), происходит переход в игру, и предлагается расставить корабли рис.3, расстановка повторяется, если выбрана игра за 2 игроков

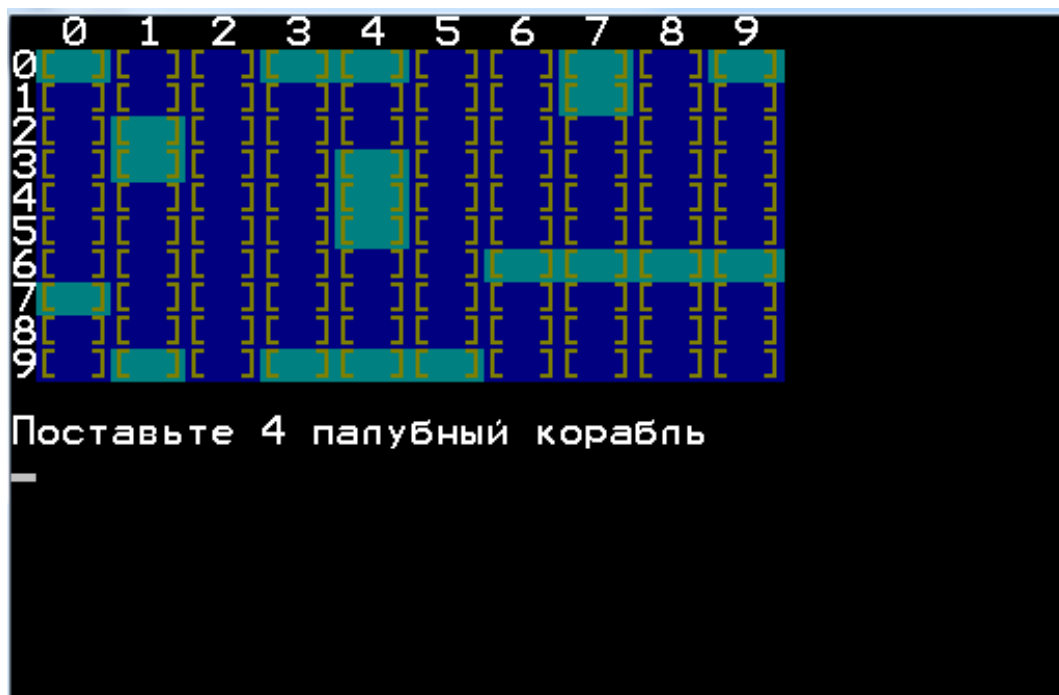


Рис.3 Расстановка кораблей

После расстановки происходит переход в игру, где игроку (или игрокам по очереди) предлагается совершить ход рис.4. Если игра против компьютера, то после хода игрока выводится сообщение о ходе компьютера рис.5.

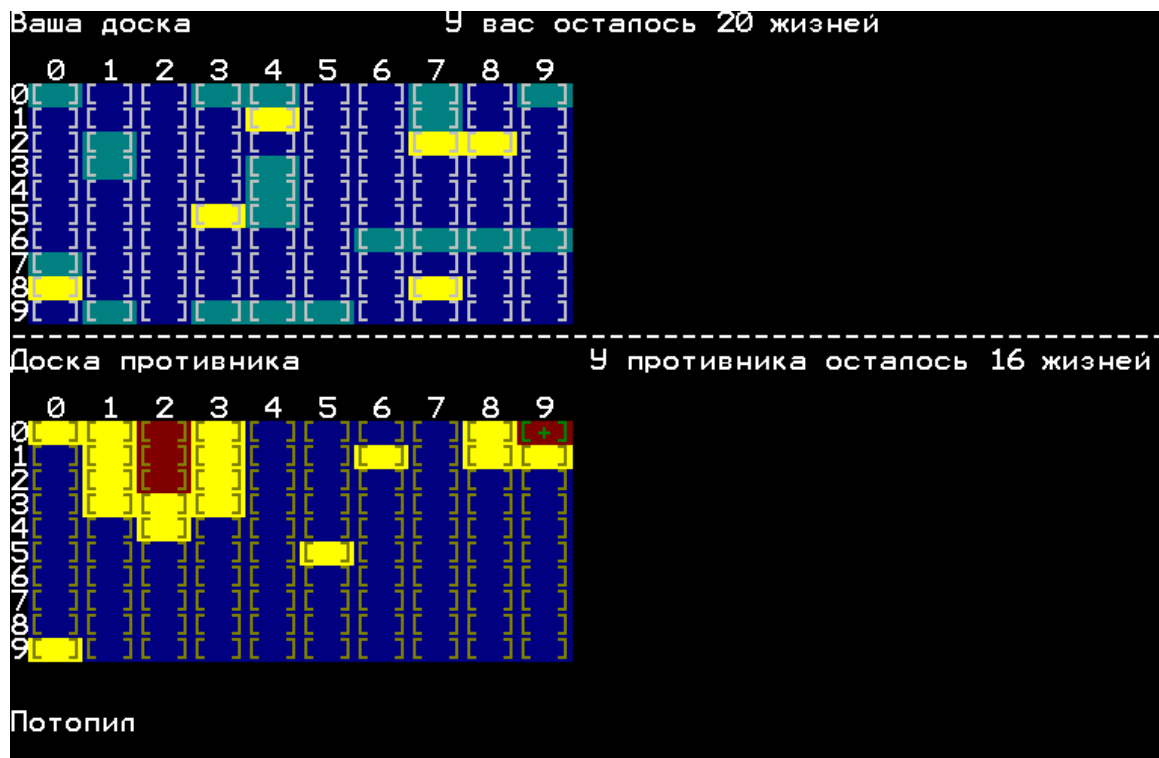


Рис.4 Игровой процесс, ход игрока



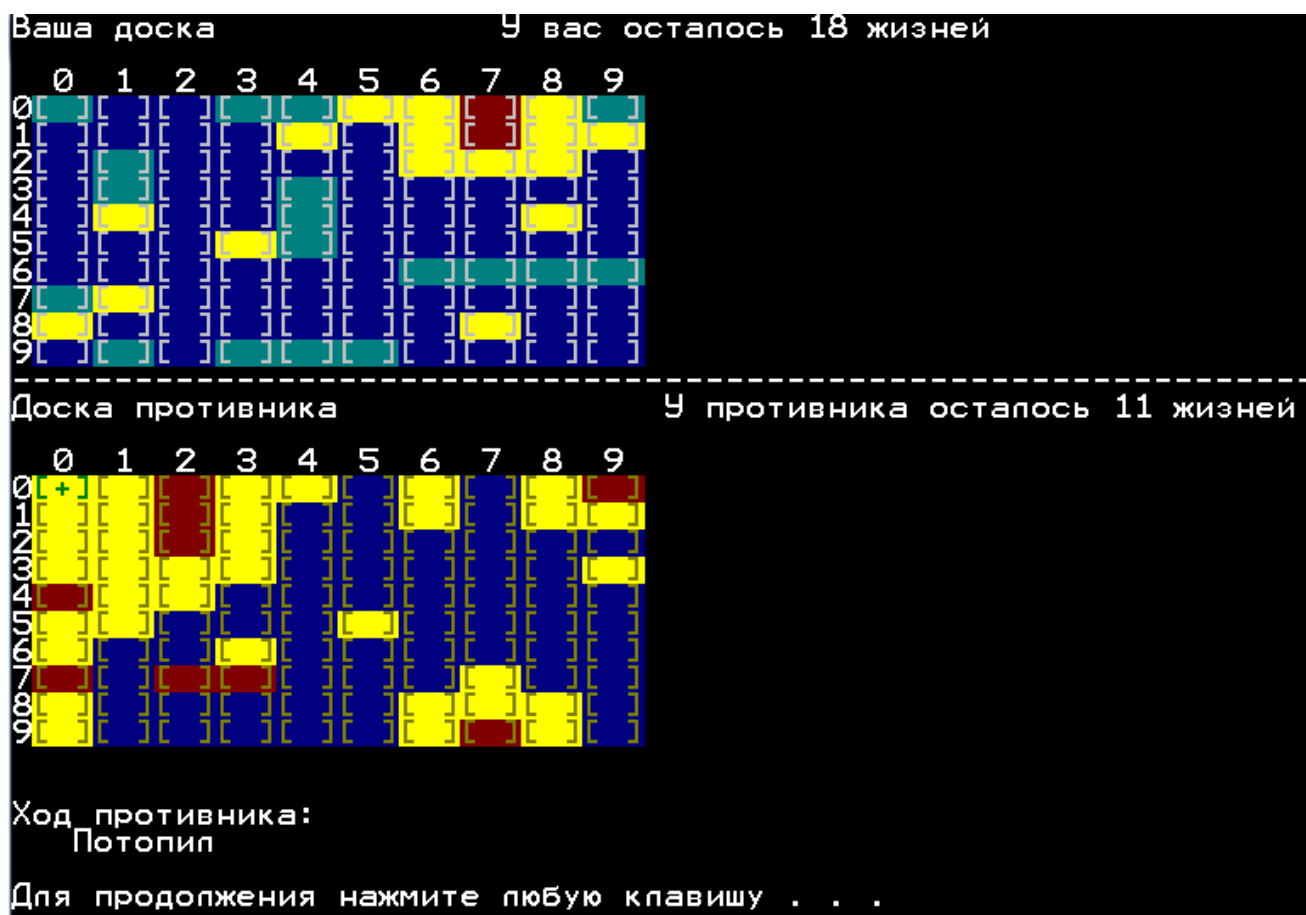


Рис.5 Сообщение о ходе компьютера

## Заключение:

В данной работе мною были использованы следующие принципы объектно-ориентированного программирования:

*Инкапсуляция* – принцип ООП, используя который, имеется возможность объединить данные и методы, работающие с ними, в одном общем типе – классе, что делает код более логичным и понятным, а также защитить данные внутри класса от вмешательства из сторонних частей программы.

*Наследование* позволяет использовать методы и поля базового класса в классах-потомках. Благодаря данному принципу нет необходимости создавать несколько одинаковых методов и полей данных в других классах, что значительно уменьшает содержимое кода и упрощает сам код.

Благодаря *полиморфизму* можно использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Это позволяет работать с одним и тем же объектом, но при этом выполнять внешне похожие, но различно реализуемые задачи.

*Абстрактный класс* – класс, в котором есть хотя бы один чистый (обнуленный) виртуальный метод. Объекты таких классов создавать запрещено. Он служит основой для производных классов. В данной программе таким классом является Board, задающий функции работы с доской и задающий интерфейс для реализации этапов игры (чистые виртуальные функции).

*Чистая виртуальная функция* — это функция-член, которую предполагается переопределить в производных классах. В данной программе это методы класса Board, переопределяемые в наследуемых классах Player и Bot.

*Перегрузка функций* – это один из типов полиморфизма, обеспечиваемого C++. В C++ несколько функций могут иметь одно и то же имя. В этом случае функция, идентифицируемая этим именем, называется перегруженной.

*Дружественный класс* позволяет получить доступ к защищенным данным.

*Параметры по умолчанию* позволяют задать начальные свойства данных, для последующей работы с ними.

В данной курсовой работе демонстрируются преимущества объектно-ориентированного программирования.

## Код программы:

### Game.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>
#include <Windows.h>

#define HOR 0//горизонтальное положение корабля
#define VER 1//вертикальное

using namespace std;

enum ConsoleColor
{
    Black = 0,
    Blue = 1, //пустая или не прострелянная клетка
    Green = 2,
    Cyan = 3, //окрашивание корабля
    Red = 4, //для окрашивания попадания
    Magenta = 5,
    Brown = 6,
    LightGray = 7,
    DarkGray = 8,
    LightBlue = 9,
    LightGreen = 10,
    LightCyan = 11,
    LightRed = 12,
    LightMagenta = 13,
    Yellow = 14, //окрашивание промаха
    White = 15
};

enum Status {
    HITTED = -1, //прострелянная
    MISS = 0, //промах
    HIT = 1, //попадание
    KILL = 2 //потопление
};

enum Keys {
    UP = 72,
    DOWN = 80,
    RIGHT = 77,
    LEFT = 75,
    ENTER = 13,
    SPACE = 32
};

class Board;
class Gamer;
class Player;
class Bot;

class Point {
protected:
    int x; //координата по вертикали
    int y; //координата по горизонтали
public:
    Point(int x = 0, int y = 0) : x(x), y(y) {} //конструктор с параметрами по умолчанию
    friend Board; //дружественные классы для доступа к Point
    friend Player;
    friend Bot;
};

class Ship : public Point {
private:
    int size; //размер корабля (1-4)
    int dir; //положение корабля (вертикальное или горизонтальное)
public:
    Ship(int size, int dir, int x = 0, int y = 0) : Point(x, y), size(size), dir(dir) {}
    friend Board; //дружественные классы для доступа к Ship
    friend Player;
    friend Bot;
};
```

```

};

class Board {
protected:
    ConsoleColor deck[10][10]; // содержит информацию о поле
    int hp; // количество здоровья (сумма не подбитых ячеек всех кораблей)
public:
    Board(int hp = 20); // конструктор с параметром по умолчанию, задающий кол-во здоровья
    // перепуска функции board_print
    void board_print(); // вывод доски
    void board_print(Point a); // вывод доски с выделением точки (для выбора поля для стрельбы)
    void board_print(Ship a); // вывод доски с выделением корабля (для расстановки кораблей)
    int set_ship(Ship a); // функция проверяющая допустимость расстановки корабля
    int check_status(Point a, int dir); // функция проверяющая "убит" или "ранен" корабль
    int shot(Point a); // функция возвращающая результат стрельбы
    int check_win() { // функция проверяющая статус игры, если чье-то здоровье опускается до 0
        if (this->hp == 0) return 0; // ...то он считается проигравшим
        else return 1;
    };
    // чистые виртуальные методы, переопределяемые при наследовании
    virtual void prepare() = 0; // чистый виртуальный метод реализуется для подготовки к игре
    virtual int play(Board *) = 0; // чистый виртуальный метод задающий геймплей
    friend Player; // для уменьшения здоровья
    friend Bot;
};

class Player: public Board {
public:
    Player(int hp = 20): Board(hp) {};
    void prepare(); // подготовка к игре (расстановка кораблей)
    int play(Board *); // сам геймплей, принимает указатель на объект содержащий информацию о противнике
};

class Bot: public Board {
private:
    Point shoot; // точка для стрельбы
    // переменные, показывающие простреленные стороны, при попадании в неопалубный корабль
    int right;
    int left;
    int up;
    int down;
public:
    Bot(int hp = 20) : Board(hp) { // конструктор, задающий начальные параметры
        right = left = up = down = 0;
        shoot.x = -1;
        shoot.y = -1;
    };
    void prepare(); // подготовка к игре (расстановка кораблей)
    int play(Board *); // реализация игры бота, принимает указатель на объект содержащий информацию о игроке
};

class Game {
private:
    string rules; // хранит правила, считываемые из файла rules.txt
public:
    Game();
    void menu(); // функция для работы (перемещение, выбор) с меню
    void game_play(int mode); // функция, задающая игру и очередность ходов, mode - определяет кол-во игроков
    void print(int i); // вывод меню или правил, i - определяет состояние меню для вывода на экран
};

void set_color(ConsoleColor text, ConsoleColor background); // функция для покраски

void check_ready(); // проверка готовности игрока

```

## Game.cpp

```

#include "game.h"

void set_color(ConsoleColor text, ConsoleColor background)
{
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));
}

void check_ready()
{

```

```

        system("cls");
        while (_getch() != ENTER) {
            SetColor(Black, Yellow);
            cout<< "\n\n\t\tКогда будете готовы, нажмите Enter\n\n";
            SetColor(White, Black);
        }
    }

Game::Game()
{
    ifstream f("rules.txt");
    getline(f, this->rules, '\0');
    f.close();
}

void Game::menu()
{
    int c = 1;
    this->print(c);
    int s;

    while (1) {
        this->print(c);
        s = _getch();
        if (s == DOWN) {
            c++;
            if (c > 4)
                c = 1;
            this->print(c);
        }
        else if (s == UP) {
            c--;
            if (c < 1)
                c = 4;
            this->print(c);
        }
        else if (s == ENTER) {
            if (c == 1)
                this->gameplay(c);
            else if (c == 2)
                this->gameplay(c);
            else if (c == 3)
                this->print(5);
            else if (c == 4)
                break;
        }
    }
}

void Game::print(int i)
{
    system("cls");
    SetColor(LightCyan, LightBlue);
    cout<< "\n\n\n\n";
    cout<< "    МОРСКОЙБОЙ    \n";
    cout<< "    \n\n\n";
    SetColor(White, Black);
    if (i == 1) {
        SetColor(Black, Yellow);
        cout<< "    1 Игрок\n\n";
        SetColor(White, Black);
        cout<< "    2 Игрока\n\n";
        cout<< "    Правила\n\n";
        cout<< "    Выход\n\n";
    }
    else if (i == 2) {
        cout<< "    1 Игрок\n\n";
        SetColor(Black, Yellow);
        cout<< "    2 Игрока\n\n";
        SetColor(White, Black);
        cout<< "    Правила\n\n";
        cout<< "    Выход\n\n";
    }
    else if (i == 3) {
        cout<< "    1 Игрок\n\n";
        cout<< "    2 Игрока\n\n";
        SetColor(Black, Yellow);
        cout<< "    Правила\n\n";
        SetColor(White, Black);
        cout<< "    Выход\n\n";
    }
    else if (i == 4) {
        cout<< "    1 Игрок\n\n";
    }
}

```

```

        cout<< "          2 Игрока\n\n";
        cout<< "          Правила\n\n";
        SetColor(Black, Yellow);
        cout<< "          Выход\n\n";
        SetColor(White, Black);
    }
    else if (i == 5) {
        cout<< "\tПравила: \n" << rules;
        cout<< "\n\n\tУправление: \n";
        cout<< "Стрелочки - передвижение курсора\nПробел - смена направления корабля (при
расставлении), \n";
        cout<< "Enter - выбрать точку и сделать выстрел/поставить корабль.";
        SetColor(Black, Yellow);
        cout<< "\n\n          Назад\n\n";
        SetColor(White, Black);
        while (_getch() != ENTER)
            continue;
    }
}

void Game::gameplay(int mode)
{
    Board **player;
    inti, j;
    player = new (Board *[2]);
    player[0] = new Player;
    if (mode == 1)
        player[1] = new Bot;
    elseif(mode == 2)
        player[1] = new Player;

    check_ready();

    for (i = 0; i< 2; i++) {
        player[i]->prepare();
        if(mode == 2)
            check_ready();
    }
    i = 0;
    j = 1;
    int win;
    while (1) {
        win = player[i]->play(player[j]);
        system("pause");
        if (win == 0) {
            win = i;
            break;
        }
        if(mode == 2)
            check_ready();

        win = player[j]->play(player[i]);
        if (mode == 2)
            system("pause");
        if (win == 0) {
            win = j;
            break;
        }
        if (mode == 2)
            check_ready();
    }
    system("cls");
    if (mode == 1) {
        if(win == 0)
            cout<< "\n\n\n\t\t!!!!ПОЗДРАВЛЯЕМ ВЫ ПОВЕДИЛИ!!!!\n";
        else
            cout<< "\n\n\n\t\tК сожалению вы проиграли :-(\n";
    }
    if(mode == 2)
        cout<< "\n\n\n\t\t!!!!ПОЗДРАВЛЯЕМ ПОВЕДИЛ ИГРОК " <<win + 1 << "!!!!\n";
    system("pause");
}

```

## board.cpp

```

#include "game.h"

Board::Board(int hp)
{

```

```

        this->hp = hp;
        for (inti = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                this->deck[i][j] = Blue;
            }
        }
    }

void Board::board_print()
{
    cout<< "Ваша доска \t\tу вас осталось "<< this->hp<<" жизней\n\n";
    for (inti = 0; i < 10; i++)
        cout<< " " <<i;
    cout<<endl;
    for (inti = 0; i < 10; i++) {
        cout<<i;
        for (int j = 0; j < 10; j++) {
            SetColor(LightGray, deck[i][j]);
            cout<< "[ ]";
            SetColor(White, Black);
        }
        cout<<endl;
    }
}

void Board::board_print(Point a)
{
    cout<< "Доска противника \t\tу противника осталось " <<this->hp<< " жизней\n\n";
    for (inti = 0; i < 10; i++)
        cout<< " " <<i;
    cout<<endl;
    for (inti = 0; i < 10; i++) {
        cout<<i;
        for (int j = 0; j < 10; j++) {
            if (i == a.x&& j == a.y) {
                if (deck[i][j] == Cyan)
                    SetColor(Green, Blue);
                else
                    SetColor(Green, deck[i][j]);
                cout<< "[+]";
                SetColor(White, Black);
            }
            else {
                if(deck[i][j] == Cyan)
                    SetColor(Brown, Blue);
                else
                    SetColor(Brown, deck[i][j]);
                cout<< "[ ]";
                SetColor(White, Black);
            }
        }
        cout<<endl;
    }
}

void Board::board_print(Ship a)
{
    system("cls");
    for (inti = 0; i < 10; i++)
        cout<< " " <<i;
    cout<<endl;
    for (inti = 0; i < 10; i++) {
        cout<<i;
        for (int j = 0; j < 10; j++) {
            if (a.dir == VER &&i>= a.x&&i<= a.x + a.size - 1 && j == a.y
                || a.dir == HOR &&i == a.x&& j >= a.y&& j <= a.y + a.size - 1) {
                if (deck[i][j] == Cyan) {
                    SetColor(Black, Red);
                }
                else {
                    SetColor(Brown, Cyan);
                }
                cout<< "[ ]";
                SetColor(White, Black);
            }
            else {
                SetColor(Brown, deck[i][j]);
                cout<< "[ ]";
                SetColor(White, Black);
            }
        }
        cout<<endl;
    }
}

```

```

}

int Board::set_ship(Ship a)
{
    for (inti = 0; i<a.size + 2; i++) {
        if (a.dir == HOR) {
            if (a.y - 1 + i>= 0 &&a.y - 1 + i<= 9) {
                if (deck[a.x][a.y - 1 + i] == Cyan)
                    return 0;
                else if (a.x - 1 >= 0 && deck[a.x - 1][a.y - 1 + i] == Cyan)
                    return 0;
                else if (a.x + 1 <= 9 && deck[a.x + 1][a.y - 1 + i] == Cyan)
                    return 0;
            }
        }
        else if (a.dir == VER) {
            if (a.x - 1 + i>= 0 &&a.x - 1 + i<= 9) {
                if (deck[a.x - 1 + i][a.y] == Cyan)
                    return 0;
                else if (a.y - 1 >= 0 && deck[a.x - 1 + i][a.y - 1] == Cyan)
                    return 0;
                else if (a.y + 1 <= 9 && deck[a.x - 1 + i][a.y + 1] == Cyan)
                    return 0;
            }
        }
    }

    for (inti = 0; i<a.size; i++) {
        if (a.dir == HOR) {
            deck[a.x][a.y + i] = Cyan;
        }
        else if (a.dir == VER) {
            deck[a.x + i][a.y] = Cyan;
        }
    }
    return 1;
}

int Board::check_status(Point a, intdir)
{
    inti = 0, hp = 0;
    int x, y;

    for (int j = 0, x = a.x, y = a.y; j < 4; j++, x--) {
        if (x < 0 || this->deck[x][y] == Yellow || this->deck[x][y] == Blue)
            break;
        if (this->deck[x][y] == Cyan)
            hp++;
    }

    for (int j = 0, x = a.x, y = a.y; j < 4; j++, x++) {
        if (x > 9 || this->deck[x][y] == Yellow || this->deck[x][y] == Blue)
            break;
        if (this->deck[x][y] == Cyan)
            hp++;
    }

    for (int j = 0, x = a.x, y = a.y; j < 4; j++, y--) {
        if (y < 0 || this->deck[x][y] == Yellow || this->deck[x][y] == Blue)
            break;
        if (this->deck[x][y] == Cyan)
            hp++;
    }

    for (int j = 0, x = a.x, y = a.y; j < 4; j++, y++) {
        if (y > 9 || this->deck[x][y] == Yellow || this->deck[x][y] == Blue)
            break;
        if (this->deck[x][y] == Cyan)
            hp++;
    }

    if (hp> 0)
        return HIT;
    else {
        for (int j = 0, x = a.x, y = a.y; j <= 4; j++, x--) {
            if (x < 0)
                break;
            else if (this->deck[x][y] == Blue || this->deck[x][y] == Yellow) {
                this->deck[x][y] = Yellow;
                if (y > 0)
                    this->deck[x][y - 1] = Yellow;
            }
        }
    }
}

```



```

        if (y < 9)
            this->deck[x][y + 1] = Yellow;
        break;
    }
    if (y > 0 && this->deck[x][y - 1] != Red && this->deck[x][y - 1] != Cyan)
        this->deck[x][y - 1] = Yellow;
    if (y < 9 && this->deck[x][y + 1] != Red && this->deck[x][y + 1] != Cyan)
        this->deck[x][y + 1] = Yellow;
}

for (int j = 0, x = a.x, y = a.y; j <= 4; j++, x++) {
    if (x > 9)
        break;
    else if (this->deck[x][y] == Blue || this->deck[x][y] == Yellow) {
        this->deck[x][y] = Yellow;
        if (y > 0)
            this->deck[x][y - 1] = Yellow;
        if (y < 9)
            this->deck[x][y + 1] = Yellow;
        break;
    }
    if (y > 0 && this->deck[x][y - 1] != Red && this->deck[x][y - 1] != Cyan)
        this->deck[x][y - 1] = Yellow;
    if (y < 9 && this->deck[x][y + 1] != Red && this->deck[x][y + 1] != Cyan)
        this->deck[x][y + 1] = Yellow;
}

for (int j = 0, x = a.x, y = a.y; j <= 4; j++, y--) {
    if (y < 0)
        break;
    else if (this->deck[x][y] == Blue || this->deck[x][y] == Yellow) {
        this->deck[x][y] = Yellow;
        if (x > 0)
            this->deck[x - 1][y] = Yellow;
        if (x < 9)
            this->deck[x + 1][y] = Yellow;
        break;
    }
    if (x > 0 && this->deck[x - 1][y] != Red && this->deck[x - 1][y] != Cyan)
        this->deck[x - 1][y] = Yellow;
    if (x < 9 && this->deck[x + 1][y] != Red && this->deck[x + 1][y] != Cyan)
        this->deck[x + 1][y] = Yellow;
}

for (int j = 0, x = a.x, y = a.y; j <= 4; j++, y++) {
    if (y > 9)
        break;
    else if (this->deck[x][y] == Blue || this->deck[x][y] == Yellow) {
        this->deck[x][y] = Yellow;
        if (x > 0)
            this->deck[x - 1][y] = Yellow;
        if (x < 9)
            this->deck[x + 1][y] = Yellow;
        break;
    }
    if (x > 0 && this->deck[x - 1][y] != Red && this->deck[x - 1][y] != Cyan)
        this->deck[x - 1][y] = Yellow;
    if (x < 9 && this->deck[x + 1][y] != Red && this->deck[x + 1][y] != Cyan)
        this->deck[x + 1][y] = Yellow;
}

    return KILL;
}

}

int Board::shot(Point a)
{
    if (this->deck[a.x][a.y] == Yellow || this->deck[a.x][a.y] == Red)
        return HITTED;
    else if (this->deck[a.x][a.y] == Blue) {
        this->deck[a.x][a.y] = Yellow;
        return MISS;
    }
    else if (this->deck[a.x][a.y] == Cyan) {
        this->deck[a.x][a.y] = Red;
        return check_status(a, 0);
    }
}

```

**play.cpp**

```

#include "game.h"

void Player::prepare()
{
    Point c;
    int s;
    int check = 1;
    for(int i = 0, count = 4; i < 4; i++, count--) {
        for (int j = 0; j < count; j++) {
            Ship s4(i+1, HOR);
            while (1) {
                system("cls");
                this->board_print(s4);
                cout<< "\nПоставьте " << i + 1 << " палубный корабль\n";
                if (check != 1)
                    cout<< "!!!!ОШИБКА!!!!\nНедопустимое место (расстояние между кораблями
не меньше одной клетки).\n";
                s = _getch();
                if (s == SPACE) {
                    if (s4.dir == HOR) {
                        if (s4.x + s4.size > 10)
                            s4.x = 0;
                        s4.dir = VER;
                    }
                    else {
                        if (s4.y + s4.size > 10)
                            s4.y = 0;
                        s4.dir = HOR;
                    }
                    continue;
                }
                if (s == ENTER) {
                    check = this->set_ship(s4);
                    if (check != 1)
                        continue;
                    else
                        break;
                }
                if (s4.dir == HOR) {
                    if (s == DOWN) {
                        s4.x++;
                        if (s4.x > 9)
                            s4.x = 0;
                    }
                    else if (s == UP) {
                        s4.x--;
                        if (s4.x < 0)
                            s4.x = 9;
                    }
                    else if (s == LEFT) {
                        s4.y--;
                        if (s4.y < 0)
                            s4.y = 10 - s4.size;
                    }
                    else if (s == RIGHT) {
                        s4.y++;
                        if (s4.y + s4.size > 10)
                            s4.y = 0;
                    }
                }
                else {
                    if (s == DOWN) {
                        s4.x++;
                        if (s4.x + s4.size > 10)
                            s4.x = 0;
                    }
                    elseif (s == UP) {
                        s4.x--;
                        if (s4.x < 0)
                            s4.x = 10 - s4.size;
                    }
                    else if (s == LEFT) {
                        s4.y--;
                        if (s4.y < 0)
                            s4.y = 9;
                    }
                    else if (s == RIGHT) {
                        s4.y++;
                        if (s4.y > 9)
                            s4.y = 0;
                    }
                }
            }
        }
    }
}

```

```

    }
}

int Player::play(Board *p2)
{
    system("cls");
    Point a;
    int s;//считываниеклавиш
    int check = 10;//проверка выстрела
    while (1) {
        system("cls");
        this->board_print();
        cout<< "-----\n";
        p2->board_print(a);
        if (check == HITTED)
            cout<< "\n\nВы сюда уже стреляли, попробуйте снова\n\n";
        else if (check == HIT)
            cout<< "\n\nПопадание\n\n";
        else if (check == KILL)
            cout<< "\n\nПотопил\n\n";
        s = _getch();
        if (s == DOWN) {
            a.x++;
            if (a.x > 9)
                a.x = 0;
        }
        else if (s == UP) {
            a.x--;
            if (a.x < 0)
                a.x = 9;
        }
        else if (s == RIGHT) {
            a.y++;
            if (a.y > 9)
                a.y = 0;
        }
        else if (s == LEFT) {
            a.y--;
            if (a.y < 0)
                a.y = 9;
        }
        else if (s == ENTER) {
            check = p2->shot(a);
            if (check == MISS) {
                system("cls");
                this->board_print();
                cout<< "-----\n";
                p2->board_print(a);
                cout<< "\n\nПромаш\n\n";
                return 1;
            }
            else if (check == HIT || check == KILL) {
                p2->hp--;
                if (p2->check_win() == 0) {
                    cout<< "\n!!!!ПОВЕДА!!!!";
                    return 0;
                }
            }
        }
    }
}
}

```

## bot.cpp

```

#include "game.h"
#include <ctime>

void Bot::prepare()
{
    srand(time(NULL));
    int check = 1;
    int x, y, dir;
    for (inti = 0, count = 4; i < 4; i++, count--) {
        for (int j = 0; j < count; j++) {
            while (1) {
                dir = rand() % 2;
                x = rand() % 10;

```

```

        y = rand() % 10;
        if (dir == HOR && y + i > 9)
            continue;
        else if (dir == VER && x + i > 9)
            continue;
        Ship s4(i + 1, dir, x, y);
        check = this->set_ship(s4);
        if (check != 1)
            continue;
        else
            break;
    }
}

}

int Bot::play(Board *p2)
{
    srand(time(NULL));
    Point a, b;
    intcheck = 10; // проверка выстрела

    while (1) {
        system("cls");
        p2->board_print();
        cout<< "-----\n";
        this->board_print(b);
        cout<< "\n\nХод противника:\n ";
        if (shoot.x == -1 && shoot.y == -1) {
            shoot.x = rand() % 10;
            shoot.y = rand() % 10;
            check = p2->shot(shoot);
        }
        else {
            check = HIT;
            p2->hp++;
        }
        if (check == MISS) {
            shoot.x = -1;
            shoot.y = -1;
            system("cls");
            p2->board_print();
            cout<< "-----\n";
            this->board_print(b);
            cout<< "\n\nХод противника:\n ";
            cout<< "Промак\n\n";
            system("pause");
            return 1;
        }
        else if (check == HITTED) {
            shoot.x = -1;
            shoot.y = -1;
            continue;
        }
        else if (check == HIT) {
            p2->hp--;
            if (this->right == 0) {
                system("cls");
                p2->board_print();
                cout<< "-----\n";
                this->board_print(b);
                cout<< "\n\nХод противника:\n ";
                cout<< "Попадание\n\n";
                system("pause");
            }
            if (this->right == 0) { // проверка направо
                a = shoot;
                if (a.y < 9) {
                    a.y++;
                    check = p2->shot(a);
                }
                else
                    check = HITTED;

                this->right = 1;

                while (check == HIT) {
                    p2->hp--;

```

```

system("cls");
p2->board_print();
cout<< "-----\n";

this->board_print(b);
cout<< "\n\nХод противника:\n ";
cout<< "Попадание\n\n";
system("pause");

if (a.y< 9) {
    a.y++;
    check = p2->shot(a);
}
else
    break;
}
if (check == MISS) {
    system("cls");
    p2->board_print();
    cout<< "-----\n";

    this->board_print(b);
    cout<< "\n\nХод противника:\n ";
    cout<< "Промех\n\n";
    system("pause");
    return 1;
}
else if (this->right != 0 &&this->left == 0) { //проверка налево (не запускается если
весь корабль был справа)
    a = shoot;
    if (a.y> 0) {
        a.y--;
        check = p2->shot(a);
    }
    else
        check = HITTED;

    this->left = 1;

    if (check == MISS) {
        system("cls");
        p2->board_print();
        cout<< "-----\n";

        this->board_print(b);
        cout<< "\n\nХод противника:\n ";
        cout<< "Промех\n\n";
        system("pause");
        return 1;
    }

    while (check == HIT) {
        p2->hp--;
        system("cls");
        p2->board_print();
        cout<< "-----\n";

        this->board_print(b);
        cout<< "\n\nХод противника:\n ";
        cout<< "Попадание\n\n";
        system("pause");
        if (a.y> 0) {
            a.y--;
            check = p2->shot(a);
        }
        else
            break;
    }
}
elseif (this->down == 0) { //проверка вниз (не запускается, если корабль стоит
горизонтально)
    a = shoot;
    if (a.x< 9) {
        a.x++;
        check = p2->shot(a);
    }
    else
        check = HITTED;

    this->down = 1;

    while (check == HIT) {

```

```

        p2->hp--;
        system("cls");
        p2->board_print();
        cout<< "-----\n";

        this->board_print(b);
        cout<< "\n\nХод противника:\n    ";
        cout<< "Попадание\n\n";
        system("pause");

        if (a.x< 9) {
            a.x++;
            check = p2->shot(a);
        }
        else
            break;
    }
    if (check == MISS) {
        system("cls");
        p2->board_print();
        cout<< "-----\n";

        this->board_print(b);
        cout<< "\n\nХод противника:\n    ";
        cout<< "Промач\n\n";
        system("pause");
        return 1;
    }
}
else if (this->down != 0 &&this->up == 0) { //проверка вверх(не запускается если весь
корабль был снизу)
    a = shoot;
    a.x--;
    check = p2->shot(a);
    this->up = 1;

    while (check == HIT) {
        p2->hp--;
        system("cls");
        p2->board_print();
        cout<< "-----\n";

        this->board_print(b);
        cout<< "\n\nХод противника:\n    ";
        cout<< "Попадание\n\n";
        system("pause");

        a.x--;
        check = p2->shot(a);
    }
}
}
if (check == KILL) {
    shoot.x = -1;
    shoot.y = -1;
    this->right = 0;
    this->left = 0;
    this->up = 0;
    this->down = 0;
    p2->hp--;
    system("cls");
    p2->board_print();
    cout<< "-----\n";

    this->board_print(b);
    cout<< "\n\nХод противника:\n    ";
    cout<< "Потопил\n\n";
    system("pause");
    if (p2->check_win() == 0) {
        return 0;
    }
}
}
}

```

## main.cpp

```
#include "game.h"
```

```
int main() {  
    setlocale( 0, "rus" );  
  
    Game a;  
    a.menu();  
  
    system("pause");  
    return 0;  
}
```