**Введение**

В процессе выполнения работы были использованы знания по объектно-ориентированному программированию на языке С++. Это высокоуровневый и компилируемый язык программирования, который используется в различных областях.

C++ является расширением языка C и также обеспечивает поддержку объектно-ориентированного программирования. Основными принципами ООП являются:

* Абстракция — выделение наиболее важных характеристик и информации об объекте;
* Полиморфизм — способность использовать объекты с одинаковым интерфейсом без информации о внутреннем устройстве объекта;
* Наследование — концепция, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения;
* Инкапсуляция — это свойство объекта/класса регулировать доступ к определенным своим компонентам извне самого объекта/класса.

В ходе выполнения курсовой работы была реализована программа «Морской бой-1», которая создана для игры против компьютера в односторонний морской бой.

**Техническое задание**

1. Название программы

Название программы: «Морской бой-1»

1. Цель создания и назначения программы

Целью данного проекта является приобретение практических навыков по проектированию в поэтапной разработке объектно-ориентированных программ.

«Морской бой-1» - это программа, которая позволяет поиграть в односторонний морской бой против компьютера.

1. Требования к программе
2. Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* Осуществлять выбор координат пользователем;
* Программа должна уметь проверять здоровье кораблей и отображать потопленные;
* Программа должна хранить месторасположение кораблей на поле;

1. Требования к программному и техническому обеспечению

* Оперативная память объемом не менее 512 МБ;
* ОС Windows следующих версий: Windows 7, Windows 8, Windows 8.1, Windows 10;
* Стандартный пакет C++.

1. Требования к программной документации

Состав программной документации должен включать в себя:

1. Техническое задание;
2. Код программы и ее описание;
3. Программу и методики испытаний;
4. Руководство пользователя.
5. Стадии и этапы разработки
6. Стадии разработки

Разработка должна быть проведена в три стадии:

1. Разработка технического задания;
2. Рабочее проектирование;
3. Релиз.
4. Этапы разработки

Разработка программы состоит из нижеперечисленных этапов:

1. Создание концепта программы;
2. Написание программы;
3. Испытание программы.
4. Порядок контроля и приема.
5. Виды испытаний

Проверка ввода данных для корректной работы программы.

1. Общие требования к приемке работы

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. Постановка задачи;
2. Определение и уточнение требований к техническим средствам;
3. Определение требований к программе;
4. Определение стадий и сроков разработки программы и документации;
5. Согласование и утверждение технического задания.

**Проектирование программы**

Для реализации проекта будет использован язык программирования С++.

Целью работы является написание приложения для игры в морской бой. В начале работы программы пользователю будет представлен выбор: открыть файл «Инструкция», открыть файл «Автор» или начать пользоваться программой. После начнется игра против компьютера в односторонний морской бой.

**Разработка программы**

1. Класс Ships, в котором описаны основные характеристики кораблей (направление, здоровье, длина, начальные и конечные координаты). На рисунке 1.1 представлена его UML диаграмма.
   1. Поля
      1. direc – направление корабля;
      2. health – здоровье корабля;
      3. l – длина корабля;
      4. x0, x1 - начальная и конечная координата по x;
      5. y0, y1 - начальная и конечная координата по y;

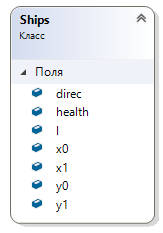


Рис. 1.1 UML диаграмма класса Ships

1. Класс Pole, в котором описаны методы для работы с полем кораблей и выводом на экран. На рисунке 1.2 представлена его UML диаграмма.
2. Методы
3. menu - выводит меню на экран;
4. SetCursorPosition - устанавливает курсор в необходимое место;
5. zap - заполняет поле кораблей нулями;
6. zapoln - заполняет поле для игры;
7. vivod - выводит игровое поле;
8. clear - очищает поле кораблей после их расстановки;

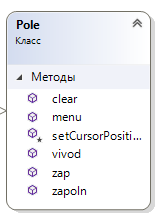


Рис. 1.2 UML диаграмма класса Pole

1. Класс Play является основным классом программы «Морской бой-1», в котором описана и реализована вся логика программы. На рисунке 3.3 представлена его UML диаграмма.
2. Поля
3. a, k - счетчик при создание кораблей;
4. ships – данные кораблей;
5. Методы
6. Sozd - создание корблей;
7. Chetire - создание четырехпалубного корабля;
8. Tri - создание трехпалубного корабля;
9. Dva - создание двупалубного корабля;
10. Odin - создание однопалубного корабля;
11. Xod - осуществляет игровой процесс;
12. check - проверяет здоровье кораблей;
13. death - выполняет обводку потопленного корабля;

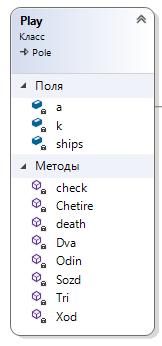


Рис. 1.3 UML диаграмма класса Play

**Руководство пользователя**

1. Запуск программы

Запуск программы осуществляется через файл SB.exe. После запуска отобразится меню программы, где пользователь может ознакомиться с инструкцией, информацией об авторе или начать игру.

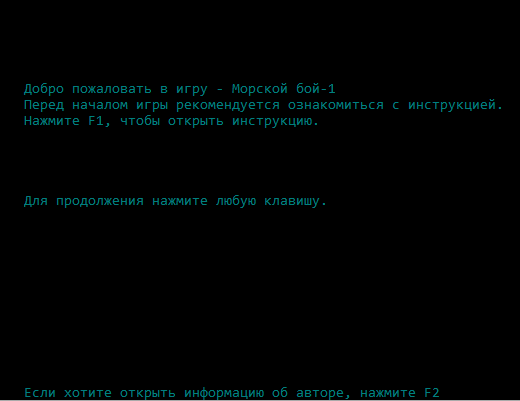


Рис. 2.1 Меню

1. Инструкция (рис. 2.2);

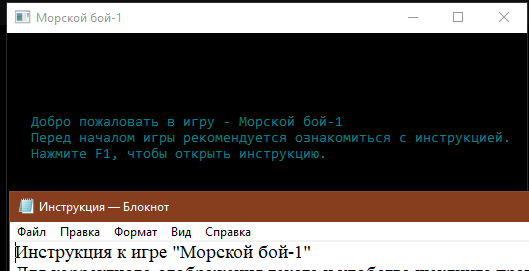


Рис. 2.2 Инструкция

1. Об авторе (рис. 2.3);

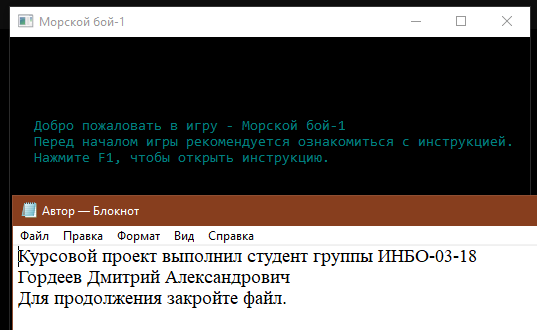


Рис. 2.3 Об авторе

1. Начало игры (рис. 2.4).

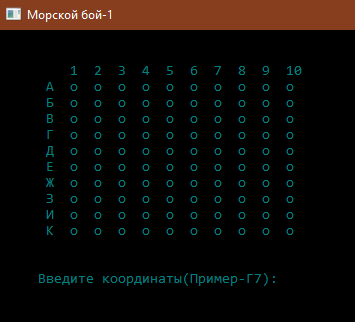


Рис. 2.4 Игра

Координаты вводятся заглавными буквами кириллицы с клавиатуры (рис. 2.5):

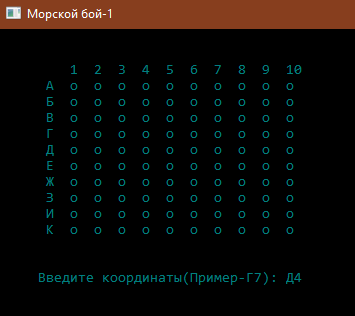


Рис. 2.5 Ввод координат

Далее происходит выстрел по заданным координатам и пользователю выводится результат (рис. 2.6):

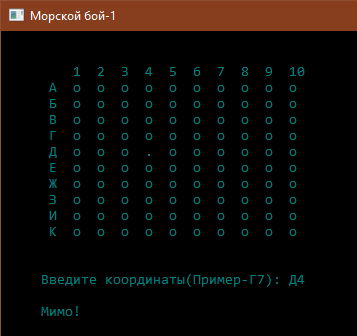


Рис. 2.6 Результат

В случае уничтожения корабля, данный корабль обводится точками (рис. 2.7):

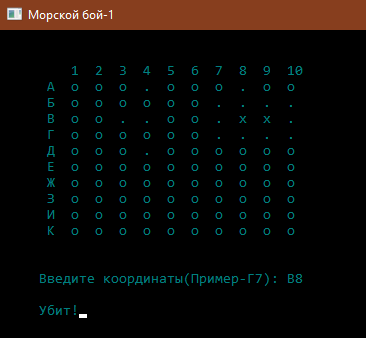


Рис. 2.7 Отображение потопленного корабля

В случае ввода некорректных данных, программа предупредит пользователя об этом (рис. 2.8):

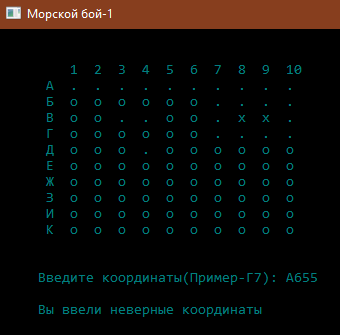


Рис. 2.8 Уведомление о неверно введенных данных

**Заключение**

В результате выполнения курсовой работы была реализована программа “Морской бой-1”, используя принципы объектно-ориентированного программирования. Выполнение курсовой работы способствовало изучению дополнительного материала и закреплению теоретического материала на практике.

**SB.h**

#pragma once

#ifndef SB\_H

#define SB\_H

#include <iostream>

#include <ctime>

#include <windows.h>

#include <string>

#include <conio.h>

using namespace std;

const int NotUsed = system("color 3");

class Ships {

public:

int health;

int x0, y0, x1, y1;

int direc;

int l;

};

class Pole {

protected:

void setCursorPosition(int x, int y);

public:

void vivod(string pole[11][11]);

int clear(int(&pole)[10][10]);

int zap(int(&pole)[10][10]);

string zapoln(string(&pole)[11][11]);

void menu();

};

class Play :public Pole{

private:

Ships ships[10];

int k = 1;

int a = 0;

public:

int Sozd(int(&pole)[10][10], int z, int x, int y, int s, Ships \*ships);

int Chetire(int(&pole)[10][10]);

int Tri(int(&pole)[10][10]);

int Dva(int(&pole)[10][10]);

int Odin(int(&pole)[10][10]);

void death(int i);

void check(int(&pole)[10][10]);

void Xod(int(&pole)[10][10]);

};

#endif

**SB.cpp**

#include "pch.h"

#include "SB.h"

void Pole::setCursorPosition(int x, int y)

{

static const HANDLE hOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

cout.flush();

COORD coord = { (SHORT)x, (SHORT)y };

SetConsoleCursorPosition(hOut, coord);

}

void Pole::menu() {

setCursorPosition(3,5);

cout << "Добро пожаловать в игру - Морской бой-1";

setCursorPosition(3, 6);

cout << "Перед началом игры рекомендуется ознакомиться с инструкцией.";

setCursorPosition(3, 7);

cout << "Нажмите F1, чтобы открыть инструкцию.";

setCursorPosition(3, 24);

cout << "Если хотите открыть информацию об авторе, нажмите F2";

setCursorPosition(3, 12);

cout << "Для продолжения нажмите любую клавишу.";

int c;

while (true) {

c = \_getch();

if (c == 0) {

c = \_getch();

if (c == 59) {

system("Инструкция.txt");

}

else if (c == 60) {

system("Автор.txt");

}

}

else break;

}

system("cls");

}

void Pole::zap(int(&pole)[10][10]) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

pole[i][j] = 0;

}

}

}

void Pole::zapoln(string(&pole)[11][11]) {

pole[0][0] = " ";

for (int i = 1; i < 11; i++) {

for (int j = 1; j < 11; j++) {

pole[i][j] = "o";

}

}

for (int i = 1; i < 11; i++) {

if (49 + i > 58) {

pole[0][i] = "10";

}

else {

pole[0][i] = 48 + i;

}

}

for (int j = 1; j < 11; j++) {

if (191 + j == 201) { pole[j][0] = 202; }

else { pole[j][0] = 191 + j; }

}

}

void Pole::vivod(string pole[11][11]) {

setCursorPosition(5, 2);

for (int i = 0; i < 11; i++) {

setCursorPosition(5, 2 + i);

for (int j = 0; j < 11; j++) {

cout << " " << pole[i][j] << " ";

}

}

}

void Pole::clear(int(&pole)[10][10]) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (pole[i][j] == 9)pole[i][j] = 0;

}

}

}

void Play::Sozd(int(&pole)[10][10], int z, int x, int y, int s, Ships \*ships) {

int n = 1, c = 0;

ships[10 - k].health = z;

a = 0;

s = 0;

int m[4] = { 0,0,0,0 };

while (s == 0 && a != 4) {

c = rand() % 4;

if (m[c] == -1) {

while (m[c] == -1)c = rand() % 4;

}

switch (c)

{

case 0:

if (x + z <= 9) {

if (pole[x + z][y] == 0 || pole[x + z][y] == 9) {

for (int i = 1; i < z; i++) {

if (pole[x + i][y] == 0)n++;

}

if (n == z) {

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

pole[x + i][y] = z;

if (y + 1 <= 9)pole[x + i][y + 1] = 9;

if (y - 1 >= 0)pole[x + i][y - 1] = 9;

}

if (x - 1 >= 0) {

pole[x - 1][y] = 9;

if (y - 1 >= 0)pole[x - 1][y - 1] = 9;

if (y + 1 <= 9)pole[x - 1][y + 1] = 9;

}

if (x + z <= 9) {

pole[x + z][y] = 9;

if (y - 1 >= 0)pole[x + z][y - 1] = 9;

if (y + 1 <= 9)pole[x + z][y + 1] = 9;

}

s = 1;

ships[10 - k].l = z;

ships[10 - k].x0 = x;

ships[10 - k].y0 = y;

ships[10 - k].x1 = x + z - 1;

ships[10 - k].y1 = y;

ships[10 - k].direc = 1;

}

else {

m[0] = -1;

a++;

}

}

else {

m[0] = -1;

a++;

}

}

else {

m[0] = -1;

a++;

}

break;

case 1:

if (x - z >= 0) {

if (pole[x - z][y] == 0 || pole[x - z][y] == 9) {

for (int i = 1; i < z; i++) {

if (pole[x - i][y] == 0)n++;

}

if (n == z) {

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

pole[x - i][y] = z;

if (y + 1 <= 9)pole[x - i][y + 1] = 9;

if (y - 1 >= 0)pole[x - i][y - 1] = 9;

}

if (x + 1 <= 9) {

pole[x + 1][y] = 9;

if (y - 1 >= 0)pole[x + 1][y - 1] = 9;

if (y + 1 <= 9)pole[x + 1][y + 1] = 9;

}

if (x - z >= 0) {

pole[x - z][y] = 9;

if (y - 1 >= 0)pole[x - z][y - 1] = 9;

if (y + 1 <= 9)pole[x - z][y + 1] = 9;

}

s = 1;

ships[10 - k].l = z;

ships[10 - k].x0 = x;

ships[10 - k].y0 = y;

ships[10 - k].x1 = x - z + 1;

ships[10 - k].y1 = y;

ships[10 - k].direc = 2;

}

else {

m[1] = -1;

a++;

}

}

else {

m[1] = -1;

a++;

}

}

else {

m[1] = -1;

a++;

}

break;

case 2:

if (y + z <= 9) {

if (pole[x][y + z] == 0 || pole[x][y + z] == 9) {

for (int i = 1; i < z; i++) {

if (pole[x][y + i] == 0)n++;

}

if (n == z) {

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

pole[x][y + i] = z;

if (x + 1 <= 9)pole[x + 1][y + i] = 9;

if (x - 1 >= 0)pole[x - 1][y + i] = 9;

}

if (y - 1 >= 0) {

pole[x][y - 1] = 9;

if (x - 1 >= 0)pole[x - 1][y - 1] = 9;

if (x + 1 <= 9)pole[x + 1][y - 1] = 9;

}

if (y + z <= 9) {

pole[x][y + z] = 9;

if (x - 1 >= 0)pole[x - 1][y + z] = 9;

if (x + 1 <= 9)pole[x + 1][y + z] = 9;

}

s = 1;

ships[10 - k].l = z;

ships[10 - k].x0 = x;

ships[10 - k].y0 = y;

ships[10 - k].x1 = x;

ships[10 - k].y1 = y + z - 1;

ships[10 - k].direc = 3;

}

else {

m[2] = -1;

a++;

}

}

else {

m[2] = -1;

a++;

}

}

else {

m[2] = -1;

a++;

}

break;

case 3:

if (y - z >= 0) {

if (pole[x][y - z] == 0 || pole[x][y - z] == 9) {

for (int i = 1; i < z; i++) {

if (pole[x][y - i] == 0)n++;

}

if (n == z) {

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

pole[x][y - i] = z;

if (x + 1 <= 9)pole[x + 1][y - i] = 9;

if (x - 1 >= 0)pole[x - 1][y - i] = 9;

}

if (y + 1 <= 9) {

pole[x][y + 1] = 9;

if (x - 1 >= 0)pole[x - 1][y + 1] = 9;

if (x + 1 <= 9)pole[x + 1][y + 1] = 9;

}

if (y - z >= 0) {

pole[x][y - z] = 9;

if (x - 1 >= 0)pole[x - 1][y - z] = 9;

if (x + 1 <= 9)pole[x + 1][y - z] = 9;

}

s = 1;

ships[10 - k].l = z;

ships[10 - k].x0 = x;

ships[10 - k].y0 = y;

ships[10 - k].x1 = x;

ships[10 - k].y1 = y - z + 1;

ships[10 - k].direc = 4;

}

else {

m[3] = -1;

a++;

}

}

else {

m[3] = -1;

a++;

}

}

else {

m[3] = -1;

a++;

}

break;

}

}

}

void Play::Chetire(int(&pole)[10][10]) {

int s = 0;

int x;

int y;

while (s != 1) {

x = rand() % 10;

y = rand() % 10;

if (pole[x][y] == 0) {

s = 1;

}

}

Sozd(pole, 4, x, y, s, ships);

if (a == 4) {

Chetire(pole);

}

k++;

};

void Play::Tri(int(&pole)[10][10]) {

int s = 0, x, y;

while (s != 1) {

x = rand() % 10;

y = rand() % 10;

if (pole[x][y] == 0) {

s = 1;

}

}

Sozd(pole, 3, x, y, s, ships);

if (a == 4) {

Tri(pole);

}

k++;

};

void Play::Dva(int(&pole)[10][10]) {

int s = 0, x, y;

while (s != 1) {

x = rand() % 10;

y = rand() % 10;

if (pole[x][y] == 0) {

s = 1;

}

}

Sozd(pole, 2, x, y, s, ships);

if (a == 4) {

Dva(pole);

}

k++;

};

void Play::Odin(int(&pole)[10][10]) {

int s = 0, x, y;

while (s != 1) {

x = rand() % 10;

y = rand() % 10;

if (pole[x][y] == 0) {

s = 1;

}

}

Sozd(pole, 1, x, y, s, ships);

if (a == 4) {

Odin(pole);

}

k++;

};

void Play::death(int i) {

switch (ships[i].direc) {

case 1:

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + ships[i].y0 \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

};

if (ships[i].x1 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0) \* 3, 3 + (ships[i].x1 + 1));

cout << ".";

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x1 + 1));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x1 + 1));

cout << ".";

}

}

for (int p = 0; p < ships[i].l; p++) {

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 + p));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 + p));

cout << ".";

}

}

setCursorPosition(5, 17);

cout << " ";

setCursorPosition(5, 17);

cout << "Убит!";

break;

case 2:

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + ships[i].y0 \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

};

if (ships[i].x1 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0) \* 3, 3 + (ships[i].x1 - 1));

cout << ".";

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x1 - 1));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x1 - 1));

cout << ".";

}

}

for (int p = 0; p < ships[i].l; p++) {

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 - p));

cout << ".";

}

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 - p));

cout << ".";

}

}

setCursorPosition(5, 17);

cout << " ";

setCursorPosition(5, 17);

cout << "Убит!";

break;

case 3:

if (ships[i].y0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0));

cout << ".";

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 - 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

};

if (ships[i].y1 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y1 + 1) \* 3, 3 + (ships[i].x1));

cout << ".";

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y1 + 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y1 + 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

}

for (int p = 0; p < ships[i].l; p++) {

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 + p) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + p) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

}

setCursorPosition(5, 17);

cout << " ";

setCursorPosition(5, 17);

cout << "Убит!";

break;

case 4:

if (ships[i].y0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0));

cout << ".";

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 + 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

};

if (ships[i].y1 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y1 - 1) \* 3, 3 + (ships[i].x1));

cout << ".";

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y1 - 1) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y1 - 1) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

}

for (int p = 0; p < ships[i].l; p++) {

if (ships[i].x0 - 1 >= 0) {

setCursorPosition(9 + (ships[i].y0 - p) \* 3, 3 + (ships[i].x0 - 1));

cout << ".";

}

if (ships[i].x0 + 1 <= 9) {

setCursorPosition(9 + (ships[i].y0 - p) \* 3, 3 + (ships[i].x0 + 1));

cout << ".";

}

}

setCursorPosition(5, 17);

cout << " ";

setCursorPosition(5, 17);

cout << "Убит!";

break;

}

}

void Play::check(int(&pole)[10][10]) {

for (int i = 0; i < 10; i++) {

switch (ships[i].direc) {

case 1:

for (int j = 0; j < ships[i].l; j++) {

if (pole[ships[i].x0 + j][ships[i].y0] == 7) {

ships[i].health--;

pole[ships[i].x0 + j][ships[i].y0] = 8;

if (ships[i].health == 0)death(i);

}

};

break;

case 2:

for (int j = 0; j < ships[i].l; j++) {

if (pole[ships[i].x0 - j][ships[i].y0] == 7) {

ships[i].health--;

pole[ships[i].x0 - j][ships[i].y0] = 8;

if (ships[i].health == 0)death(i);

}

};

break;

case 3:

for (int j = 0; j < ships[i].l; j++) {

if (pole[ships[i].x0][ships[i].y0 + j] == 7) {

ships[i].health--;

pole[ships[i].x0][ships[i].y0 + j] = 8;

if (ships[i].health == 0)death(i);

}

};

break;

case 4:

for (int j = 0; j < ships[i].l; j++) {

if (pole[ships[i].x0][ships[i].y0 - j] == 7) {

ships[i].health--;

pole[ships[i].x0][ships[i].y0 - j] = 8;

if (ships[i].health == 0)death(i);

}

};

break;

}

}

}

void Play::Xod(int(&pole)[10][10]) {

int z = 0;

string cord;

while (z != 20) {

setCursorPosition(5, 15);

cout << "Введите координаты(Пример-Г7): ";

cin >> cord;

if ((int)cord[0] <= -54 && (int)cord[0] >= -64 && (int)cord[0] != -53 && cord.length() <= 3 && (48 < cord[1] && cord[1] < 58)) {

if ((cord.length() == 3 && cord[1] == 49 && cord[2] == 48) || (cord.length() == 2)) {

if (cord[1] == 49 && cord[2] == 48)cord[1] = 58;

if ((int)cord[0] == -54)cord[0] = 201;

if (pole[(int)cord[0] + 64][cord[1] - 49] != 7 && pole[(int)cord[0] + 64][cord[1] - 49] != 8) {

if (pole[(int)cord[0] + 64][cord[1] - 49] != 0) {

pole[(int)cord[0] + 64][cord[1] - 49] = 7;

setCursorPosition(5, 17);

cout << "Ранил!";

setCursorPosition(9 + (cord[1] - 49) \* 3, 3 + (int)cord[0] + 64);

cout << "x";

z++;

check(pole);

}

else {

pole[(int)cord[0] + 64][cord[1] - 49] = 7;

setCursorPosition(9 + (cord[1] - 49) \* 3, 3 + (int)cord[0] + 64);

cout << ".";

setCursorPosition(5, 17);

cout << "Мимо!";

}

}

else {

setCursorPosition(5, 17);

cout << "Вы уже производили атаку по этим координатам";

}

Sleep(2000);

setCursorPosition(5, 15);

cout << " ";

setCursorPosition(5, 17);

cout << " ";

}

else {

setCursorPosition(5, 17);

cout << "Вы ввели неверные координаты";

Sleep(1000);

setCursorPosition(5, 15);

cout << " ";

setCursorPosition(5, 17);

cout << " ";

}

}

else {

setCursorPosition(5, 17);

cout << "Вы ввели неверные координаты";

Sleep(1000);

setCursorPosition(5, 15);

cout << " ";

setCursorPosition(5, 17);

cout << " ";

}

}

setCursorPosition(5, 17);

cout << "Все вражеские корабли уничтожены";

Sleep(2000);

system("cls");

setCursorPosition(10, 10);

cout << "Вы победили!!!";

}

**SB\_main.cpp**

#include "pch.h"

#include "SB.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Russian");

system("mode con cols=65 lines=25");

system("title Морской бой-1");

srand(time(NULL));

Pole Pole;

Play Play;

Pole.menu();

string pole[11][11];

int polecom[10][10];

Pole.zapoln(pole);

Pole.vivod(pole);

Pole.zap(polecom);

Play.Chetire(polecom);

for (int i = 0; i < 2; i++)Play.Tri(polecom);

for (int i = 0; i < 3; i++)Play.Dva(polecom);

for (int i = 0; i < 4; i++)Play.Odin(polecom);

Pole.clear(polecom);

Play.Xod(polecom);

system("pause>>NULL");

}