МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Реализация компьютерный игры «Морской бой» с компьютером

Студент гр. 7304 _____ Давыдов А.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2018

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Давыдов А.А.

Группа 7304

Тема работы (проекта): Реализовать интерактивную игру “Морской бой” с компьютером

Исходные данные:
Реализовать игру согласно правилам по ссылке
https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%80%D1%81%D0%BA%D0%BE%D0%B9_%D0%B1%D0%BE%D0%B9_(%D0%B8%D0%B3%D1%80%D0%B0)

Содержание пояснительной записки:
«Содержание», «Введение», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 00 страниц.

Дата выдачи задания: 13.04.2018

Дата сдачи реферата: 25.05.2018

Дата защиты реферата: 25.05.2018

Студент _____ Давыдов А.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

Были разработаны функции для случайной расстановки кораблей противника, проверки координат при расстановки на корректность(координанты вокруг корабля не должны содержать клеток других кораблей). Разработана функция для выбора компьютером клетки удара в зависимости от предыдущего хода(попадение/непопадение), уже ранее подбитых клеток и областей вокруг убитого корабля. В качестве задания к курсовому проекту добавлено оружие «Торпеда», которая подрывает клетки на поле противника, пока не встретит клетку корабля и не подорвет ее. Визуальная реализация игры приводится в консоли в виде матрицы 10x10 с наименованием координат латинскими буквами по строкам и цифрами по столбцам, которая выводится после каждого хода.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является реализация функций для корректного случайного заполнения поля противника координатами кораблей, проверки введенных координат кораблей союзника на корректность аналогичным образом, функций для изменения данных о состоянии клеток для структур кораблей противника и союзника, изменения клеток поля для форматного вывода в консоль, функций производящих удар компьютера по полю пользователя с учетом предыдущего хода и ранее подбитых клеток. Также необходимы дополнительные функции закрашивающих области вокруг убитых кораблей, для визуального удобства.

Для корректного заполнения карты противника и проверки координат союзника используется функция, подсчитывающая количество отмеченных координат на карте(проверка на пересечение), функция проверки на диагональность, метод закраски для подсчета площади кораблей для того, чтобы проверить, что вокруг корбля нет координат другого. Для того, чтобы компьютер не стрелял в клетки, которые подбил ранее, используется матрица, совпадающая по размерности с полем карты, в которой отмечаются ходы компьютера и помечаются области вокруг убитых кораблей, также есть специальный флаг, который отвечает за попадание в предыдущем ходе компьютера для того, чтобы компьютер стрелял в недиагональную клетку рядом с предыдущим выстрелом, если было попадание при предыдущем ходе.

1. СТРУКТУРЫ ДАННЫХ И ДИРЕКТИВЫ

1.1. Структуры данных для кораблей, поля боя, выбора ориентации для кораблей противника при рандоме.

```
#include "settings_ships.h"

typedef struct map
{
        char matrix_battle[FIELDSIZE][FIELDSIZE];
} Map;

typedef struct battle_ship
{
        int coordinate[4][2];
        int health[4];
} BattleShip;

typedef struct cruiser
{
        int coordinate[3][2];
        int health[3];
} Cruiser;

typedef struct destroyer
{
        int coordinate[2][2];
        int health[2];
} Destroyer;

typedef struct torpedo_boat
{
        int coordinate[1][2];
        int health[1];
} TorpedoBoat;


//struct for set orientation of enemy ships
typedef struct
{
        int horisontal;
        int right;
        int down;
} Orientation;
```

1.2. Директивы, описывающие основные настройки игры

1) Количество кораблей в зависимости от типа и общее количество занимаемых клеток.

```
#define FIELDSIZE 10

#define COUNTFOURDECK 1
#define COUNTTHREEDECK 2
#define COUNTTWODECK 3
#define COUNTONEDECK 4

#define COUNTCELLOFSHIPS 20
```

2) Флаги, опредляющие попадение/уничтожение, символы для клеток корабля, пустых и просмотренных клеток.

```
#pragma once
#define HIJACKING 1
#define DESTROYED 2

#define FIGURE '+'
#define EMPTYCAGE '0'
#define MARK 'x'
```

2. ОСНОВНЫЕ ФУНКЦИИ ПРОГРАММЫ

2.1. Функции проверки координат кораблей на корректность и инициализации вражеских кораблей случайным образом.

1) Функция для определения ориентации корабля и направления, в котором будут заполняться его координаты в зависимости от выбора текущей.

```
//Select orientation and next point of ship in map
void SetEnemyOrientation(Orientation *orientation, int x, int y, int deck)
{
        if(rand() % 2 == 1)
                orientation->horisontal = 1;
        if(y + deck - 1 < FIELDSIZE)
                orientation->right = 1;
        if(x + deck - 1 < FIELDSIZE)
                orientation->down = 1;
}
```

2) Функция для пометки всех координат вокруг выбранной, как занятных.

```
void MarkSpaceAroundCoordinate(char matrix_coordinate[FIELDSIZE][FIELDSIZE], int x, int y)
{
        //Mark upper coordintes
        if(x - 1 >= 0)
        {
                matrix_coordinate[x - 1][y] = 'x';
                if(y - 1 >= 0)
                        matrix_coordinate[x - 1][y - 1] = 'x';
                if(y + 1 < FIELDSIZE)
                        matrix_coordinate[x - 1][y + 1] = 'x';
        }
        //Mark lower coordinates
        if(x + 1 < FIELDSIZE)
        {
                matrix_coordinate[x + 1][y] = 'x';
                if(y - 1 >= 0)
                        matrix_coordinate[x + 1][y - 1] = 'x';
                if(y + 1 < FIELDSIZE)
                        matrix_coordinate[x + 1][y + 1] = 'x';
        }
        if(y - 1 >= 0)
                matrix_coordinate[x][y - 1] = 'x'; //Mark left coordinate
        if(y + 1 < FIELDSIZE)
                matrix_coordinate[x][y + 1] = 'x'; //Mark right coordinate
}
```

3) Функция случайной инициализации кораблей компьютера.

```
void InitializeEnemyShips(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat,
                                                                                        int first_call)
{
        //for different value which will return rand()
        if(first_call)
                srand(time(NULL));
        int x;
        int y;
        char matrix_coordinate[FIELDSIZE][FIELDSIZE];
        for(int i = 0; i < FIELDSIZE; ++i)
                for(int j = 0; j < FIELDSIZE; ++j)
                        matrix_coordinate[i][j] = '0';

        for(int i = 0; i < COUNTFOURDECK; ++i)
        {
                while(1)
                {
                        x = rand() % FIELDSIZE;
                        y = rand() % FIELDSIZE;
                        if(matrix_coordinate[x][y] == '0')
                                break;
                }
                Orientation orientation = {0, 0, 0};
                SetEnemyOrientation(&orientation, x, y, 4);

                for(int j = 0; j < 4; ++j)
                {
                        pbattle_ship[i].health[j] = 1;
                        pbattle_ship[i].coordinate[j][0] = x;
                        pbattle_ship[i].coordinate[j][1] = y;
                        matrix_coordinate[x][y] = 'x';
                        MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                        if(orientation.horisontal)
                        {
                                if(orientation.right)
                                        ++y;
                                else
                                        --y;
                        }
                        else
                                if(orientation.down)
                                        ++x;
                                else
                                        --x;
                        }
                }
        }
        for(int i = 0; i < COUNTTHREEDECK; ++i)
        {
                while(1)
                {
                        x = rand() % FIELDSIZE;
                        y = rand() % FIELDSIZE;
                        if(matrix_coordinate[x][y] == '0')
                                break;
                }
                Orientation orientation = {0, 0, 0};
                SetEnemyOrientation(&orientation, x, y, 4);

                for(int j = 0; j < 3; ++j)
                {
                        pcruiser[i].health[j] = 1;
                        pcruiser[i].coordinate[j][0] = x;
                        pcruiser[i].coordinate[j][1] = y;
                        matrix_coordinate[x][y] = 'x';
                        MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                        if(orientation.horisontal)
                        {
                                if(orientation.right)
                                        ++y;
                                else
                                        --y;
                        }
                        else
                        {
                                if(orientation.down)
                                        ++x;
                                else
                                        --x;
                        }
```

```
                }
        }
        for(int i = 0; i < COUNTTWODECK; ++i)
        {
                while(1)
                {
                        x = rand() % FIELDSIZE;
                        y = rand() % FIELDSIZE;
                        if(matrix_coordinate[x][y] == '0')
                                break;
                }
                Orientation orientation = {0, 0, 0};
                SetEnemyOrientation(&orientation, x, y, 2);

                for(int j = 0; j < 2; ++j)
                {
                        pdestroyer[i].health[j] = 1;
                        pdestroyer[i].coordinate[j][0] = x;
                        pdestroyer[i].coordinate[j][1] = y;
                        matrix_coordinate[x][y] = 'x';
                        MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                        if(orientation.horisontal)
                        {
                                if(orientation.right)
                                        ++y;
                                else
                                        --y;
                        }
                        else
                        {
                                if(orientation.down)
                                        ++x;
                                else
                                        --x;
                        }
                }
        }
        for(int i = 0; i < COUNTONEDECK; ++i)
        {
                while(1)
                {
                        x = rand() % FIELDSIZE;
                        y = rand() % FIELDSIZE;
                        if(matrix_coordinate[x][y] == '0')
                                break;
                }
                Orientation orientation = {0, 0, 0};
                SetEnemyOrientation(&orientation, x, y, 1);

                for(int j = 0; j < 1; ++j)
                {
                        ptorpedo_boat[i].health[j] = 1;
                        ptorpedo_boat[i].coordinate[j][0] = x;
                        ptorpedo_boat[i].coordinate[j][1] = y;
                        matrix_coordinate[x][y] = 'x';
                        MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                        if(orientation.horisontal)
                        {
                                if(orientation.right)
                                        ++y;
                                else
                                        --y;
                        }
                        else
                        {
                                if(orientation.down)
                                        ++x;
                                else
                                        --x;
                        }
                }
        }
}
```

4) Функция проверки координат кораблей на диагональность или горизонтальность, вызывается для всех кораблей в функции CorrectShips().

```
//Check coordinates of ship for diagonal and horisontal correct
int CorrectCoordinates(int coordinate[2][2], int deck)
{
        int horisontal_flag = 0;
        int vertical_flag = 0;
        if(coordinate[0][0] == coordinate[1][0])
                horisontal_flag = 1;
        if(coordinate[0][1] == coordinate[1][1])
                vertical_flag = 1;

        if(horisontal_flag && vertical_flag)
                return 0;

        //Check coordinates to vertical/diagonal coorect
        if(vertical_flag)
                for(int i = 0; i < deck - 1; ++i)
                {
                        int dx = coordinate[i][0] - coordinate[i + 1][0];
                        if(coordinate[i][1]!= coordinate[i + 1][1] || (dx!= 1 && dx!= -1))
                                return 0;
                }
        if(horisontal_flag)
                for(int i = 0; i < deck - 1; ++i)
                {
                        int dy = coordinate[i][1] - coordinate[i + 1][1];
                        if(coordinate[i][0]!= coordinate[i + 1][0] || (dy!= 1 && dy!= -1))
                                return 0;
                }


        return 1;
}
```

```
int CorrectShips(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer)
{
        for(int i = 0; i < COUNTFOURDECK; ++i)
                if(!CorrectCoordinates(pbattle_ship[i].coordinate, 4))
                        return 0;

        for(int i = 0; i < COUNTTHREEDECK; ++i)
                if(!CorrectCoordinates(pcruiser[i].coordinate, 3))
                        return 0;

        for(int i = 0; i < COUNTTWODECK; ++i)
                if(!CorrectCoordinates(pdestroyer[i].coordinate, 2))
                        return 0;

        return 1;
}
```

5) Функция проверки координат на пересечение

```
//Counting points + on map. Correct is 20
int CorrectInsert(Map *pmap)
{
        int count_cell = 0;

        for(int i = 0; i < FIELDSIZE; ++i)
                for(int j = 0; j < FIELDSIZE; ++j)
                        if(pmap->matrix_battle[i][j] == '+')
                                ++count_cell;

        if(count_cell == COUNTCELLOFSHIPS)
        {
                return 1;
        }
        else
                return 0;
}
```

6) Функция проверки координат кораблей на пустое пространство в соседних клеток с ними, вызывается с функцией поиска площадей кораблей FindCurScale().

```
//Check for empty space around ship with using function which find scale of ships from library find_scale_of_
ship.h
int ControlCheckShips(char **matrix_field)
{
        int coins_one = 0;
        int coins_two = 0;
        int coins_three = 0;
        int coins_four = 0;
        int cur_scale = 1;

        //10 is count of ships
        for(int i = 0; i < 10; ++i)
        {
                if(!CompleteTraversal(matrix_field))
                {
                        int start_point[2];
                        GetStartPoint(matrix_field, start_point);
                        matrix_field[start_point[0]][start_point[1]] = MARK;
                        FindCurScale(matrix_field, start_point, &cur_scale);
                        switch(cur_scale)
                        {
                                case 1:
                                        ++coins_one;
                                        break;
                                case 2:
                                        ++coins_two;
                                        break;
                                case 3:
                                        ++coins_three;
                                        break;
                                case 4:
                                        ++coins_four;
                                        break;
                        }
                        cur_scale = 1;
                }
        }

        if(coins_one == COUNTONEDECK && coins_two == COUNTTWODECK && coins_three == COUNTTHREEDECK && coins_f
our == COUNTFOURDECK)
                return 1;
        else
                return 0;
}
```

```
//Find scale of ship on battle field for check to correct arrangement
void FindCurScale(char **matrix_field, int cur_point[2], int *cur_scale)
{
        int x = cur_point[0];
        int y = cur_point[1];

        //Check right/left and up/down points
        if(x + 1 < FIELDSIZE  && matrix_field[x + 1][y] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x + 1][y] = MARK;
                cur_point[0] = x + 1;
                cur_point[1] = y;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x - 1 >= 0 && matrix_field[x - 1][y] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x - 1][y] = MARK;
                cur_point[0] = x - 1;
                cur_point[1] = y;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(y + 1 < FIELDSIZE && matrix_field[x][y + 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x][y + 1] = MARK;
                cur_point[0] = x;
                cur_point[1] = y + 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(y - 1 >= 0 && matrix_field[x][y - 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x][y - 1] = MARK;
                cur_point[0] = x;
                cur_point[1] = y - 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        //Check horisontal points
        if(x - 1 >= 0 && y + 1 < FIELDSIZE && matrix_field[x - 1][y + 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x - 1][y + 1] = MARK;
                cur_point[0] = x - 1;
                cur_point[1] = y + 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x - 1 >= 0 && y - 1 >= 0 && matrix_field[x - 1][y - 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x - 1][y - 1] = MARK;
                cur_point[0] = x - 1;
                cur_point[1] = y - 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x + 1 < FIELDSIZE && y + 1 < FIELDSIZE && matrix_field[x + 1][y + 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x + 1][y + 1] = MARK;
                cur_point[0] = x + 1;
                cur_point[1] = y + 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x + 1 < FIELDSIZE && y - 1 >= 0 && matrix_field[x + 1][y - 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x + 1][y - 1] = MARK;
                cur_point[0] = x + 1;
                cur_point[1] = y - 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
}
```

2.2. Функции для реализации игрового процесса.

1) Функция изменения карты после инициализации координат кораблей.

```
//Mark ships coordinate by + on map
void ChangeMap(Map *pmap, BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *pt
orpedo_boat)
{
	CleanMap(pmap); //because this function is called until coordinate of enemy ships incorrects
	for(int i = 0; i < COUNTFOURDECK; ++i)
		for(int j = 0; j < 4; ++j)
		{
			int x = pbattle_ship[i].coordinate[j][0];
			int y = pbattle_ship[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}

	for(int i = 0; i < COUNTTHREEDECK; ++i)
		for(int j = 0; j < 3; ++j)
		{
			int x = pcruiser[i].coordinate[j][0];
			int y = pcruiser[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}

	for(int i = 0; i < COUNTTWODECK; ++i)
		for(int j = 0; j < 2; ++j)
		{
			int x = pdestroyer[i].coordinate[j][0];
			int y = pdestroyer[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}

	for(int i = 0; i < COUNTONEDECK; ++i)
		for(int j = 0; j < 1; ++j)
		{
			int x = ptorpedo_boat[i].coordinate[j][0];
			int y = ptorpedo_boat[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}
}
```

2) Функции для отмечания удара на карте союзника и противника.

```
void MarkShotOnMap(Map *pmap, int shot[2])
{
	int x = shot[0];
	int y = shot[1];

	if(pmap->matrix_battle[x][y] == '+')
		pmap->matrix_battle[x][y] = 'X';
	else if(pmap->matrix_battle[x][y] == '0')
		pmap->matrix_battle[x][y] = 'x';
}


void MarkShotOnEnemyMap(Map *pmap, Map *pvisual_map, int shot[2])
{
	int x = shot[0];
	int y = shot[1];

	if(pmap->matrix_battle[x][y] == '+')
	{
		pvisual_map->matrix_battle[x][y] = 'X';
		pmap->matrix_battle[x][y] = 'X';
	}
	else if(pmap->matrix_battle[x][y] == '0')
	{
		pvisual_map->matrix_battle[x][y] = 'x';
		pmap->matrix_battle[x][y] = 'x';
	}
}
```

3) Функция изменения состояния координат корабля в зависимости от удара противника.

```
//Change health if shot and point of ship is equal
int ChangeHealthShip(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorped
o_boat, Map *pmap, int shot[2])
{
        int x = shot[0];
        int y = shot[1];

        for(int i = 0; i < COUNTFOURDECK; ++i)
                for(int j = 0; j < 4; ++j)
                        if(pbattle_ship[i].coordinate[j][0] == x && pbattle_ship[i].coordinate[j][1] == y &&
pmap->matrix_battle[x][y] == '+')
                        {
                                pbattle_ship[i].health[j] = 0;
                                if(DestroyedShip(pbattle_ship[i].health, 4))
                                        return DESTROYED;
                                else
                                        return HIJACKING;
                        }

        for(int i = 0; i < COUNTTHREEDECK; ++i)
                for(int j = 0; j < 3; ++j)
                        if(pcruiser[i].coordinate[j][0] == x && pcruiser[i].coordinate[j][1] == y && pmap->ma
trix_battle[x][y] == '+')
                        {
                                pcruiser[i].health[j] = 0;
                                if(DestroyedShip(pcruiser[i].health, 3))
                                        return DESTROYED;
                                else
                                        return HIJACKING;
                        }

        for(int i = 0; i < COUNTTWODECK; ++i)
                for(int j = 0; j < 2; ++j)
                        if(pdestroyer[i].coordinate[j][0] == x && pdestroyer[i].coordinate[j][1] == y && pmap
->matrix_battle[x][y] == '+')
                        {
                                pdestroyer[i].health[j] = 0;
                                if(DestroyedShip(pdestroyer[i].health, 2))
                                        return DESTROYED;
                                else
                                        return HIJACKING;
                        }
_
        for(int i = 0; i < COUNTONEDECK; ++i)
                for(int j = 0; j < 1; ++j)
                        if(ptorpedo_boat[i].coordinate[j][0] == x && ptorpedo_boat[i].coordinate[j][1] == y &
& pmap->matrix_battle[x][y] == '+')
                        {
                                ptorpedo_boat[i].health[j] = 0;
                                if(DestroyedShip(ptorpedo_boat[i].health, 1))
                                        return DESTROYED;
                                else
                                        return HIJACKING;
                        }
        return 0;
}
```

4) Функция проверки корабля на полное разрушение

```
//Check ship for destroy
int DestroyedShip(int health[1], int deck)
{
        int destroy_flag = 0;

        for(int i = 0; i < deck; ++i)
                destroy_flag += health[i];
        if(!destroy_flag)
                return 1;
        else
                return 0;
}
```

5) Функция, отметки координат, как подбитых вокруг убитого корабля пользователя.

```
void MarkSpaceAroundDestroyedShip(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, Torpedo
Boat *ptorpedo_boat, Map *pmap)
{
        if(DestroyedShip(pbattle_ship[0].health, 4))
                for(int i = 0; i < 4; ++i)
                {
                        int x = pbattle_ship[0].coordinate[i][0];
                        int y = pbattle_ship[0].coordinate[i][1];
                        MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                }

        for(int i = 0; i < COUNTTWODECK; ++i)
                if(DestroyedShip(pcruiser[i].health, 3))
                        for(int j = 0; j < 3; ++j)
                        {
                                int x = pcruiser[i].coordinate[j][0];
                                int y = pcruiser[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                        }
        for(int i = 0; i < COUNTTHREEDECK; ++i)
                if(DestroyedShip(pdestroyer[i].health, 2))
                        for(int j = 0; j < 2; ++j)
                        {
                                int x = pdestroyer[i].coordinate[j][0];
                                int y = pdestroyer[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                         }
        for(int i = 0; i < COUNTONEDECK; ++i)
                if(DestroyedShip(ptorpedo_boat[i].health, 1))
                        for(int j = 0; j < 1; ++j)
                        {
                                int x = ptorpedo_boat[i].coordinate[j][0];
                                int y = ptorpedo_boat[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                         }
}
```

6) Функция отметки координат вокруг убитого корабля, как просмотренных в матрице предыдущих ударов противника

```
//mark for exclude points in next hits of enemy
void MarkSpaceAroundCoordinateInMatrixHits(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer
, TorpedoBoat *ptorpedo_boat, char matrix_hits[FIELDSIZE][FIELDSIZE])
{
        if(DestroyedShip(pbattle_ship[0].health, 4))
                for(int i = 0; i < 4; ++i)
                {
                        int x = pbattle_ship[0].coordinate[i][0];
                        int y = pbattle_ship[0].coordinate[i][1];
                        MarkSpaceAroundCoordinate(matrix_hits, x, y);
                }
        for(int i = 0; i < COUNTTWODECK; ++i)
                if(DestroyedShip(pcruiser[i].health, 3))
                        for(int j = 0; j < 3; ++j)
                        {
                                int x = pcruiser[i].coordinate[j][0];
                                int y = pcruiser[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(matrix_hits, x, y);
                        }
        for(int i = 0; i < COUNTTHREEDECK; ++i)
                if(DestroyedShip(pdestroyer[i].health, 2))
                        for(int j = 0; j < 2; ++j)
                        {
                                int x = pdestroyer[i].coordinate[j][0];
                                int y = pdestroyer[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(matrix_hits, x, y);
                         }
        for(int i = 0; i < COUNTONEDECK; ++i)
                if(DestroyedShip(ptorpedo_boat[i].health, 1))
                        for(int j = 0; j < 1; ++j)
                        {
                                int x = ptorpedo_boat[i].coordinate[j][0];
                                int y = ptorpedo_boat[i].coordinate[j][1];
                                MarkSpaceAroundCoordinate(matrix_hits, x, y);
                        }
}
```

7) Функция выбора клетки для выстрела компьютером в зависимости от предыдущего хода

```
void EnemyHit(int shot[2], char matrix_hits[FIELDSIZE][FIELDSIZE], int hijacking_flag)
{
        int x = shot[0];
        int y = shot[1];
        int new_hit_flag = 0;

        if(!hijacking_flag)
        {
                while(!new_hit_flag)
                {
                        x = ((unsigned int)rand()) % 10;
                        y = ((unsigned int)rand()) % 10;

                        if(matrix_hits[x][y] == '0')
                        {
                                matrix_hits[x][y] = 'x';
                                new_hit_flag = 1;
                        }
                }
        }
        else
        {
                int time_x = x;
                int time_y = y;
                int count_try = 40; //except, when around shot point look points yet

                while(!new_hit_flag && count_try)
                {
                        time_x = x + RandomOffset();
                        time_y = y + RandomOffset();

                        if(!OutOfRange(time_x, time_y) && matrix_hits[time_x][time_y] == '0' &&(abs(x-time_x)
!= abs(y-time_y)))
                        {
                                matrix_hits[time_x][time_y] = 'x';
                                new_hit_flag = 1;
                                x = time_x;
                                y = time_y;
                                break;
                        }
                        --count_try;
                }
        ▉       if(!count_try)
                {
                        x = ((unsigned int)rand()) % 10;
                        y = ((unsigned int)rand()) % 10;
                        matrix_hits[x][y] = 'x';
                }
        }

        shot[0] = x;
        shot[1] = y;
}
```

8) Функция, реализующая удар торпедой по случайной строчке для компьютера

```
//Additionan weapons
int EnemyTorpedo(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_bo
at, Map *pmap, char matrix_hits[FIELDSIZE][FIELDSIZE], int row)
{
        for(int j = 0; j < FIELDSIZE; ++j)
                if(pmap->matrix_battle[row][j] == '+')
                {
                        int shot[2] = {row, j};
                        int res =  ChangeHealthShip(pbattle_ship, pcruiser, pdestroyer, ptorpedo_boat, pmap,
shot);
                        pmap->matrix_battle[row][j] = 'X';
                        matrix_hits[row][j] = 'x';
                        return res;
                }
                else if(pmap->matrix_battle[row][j] == '0')
                {
                        pmap->matrix_battle[row][j] = 'x';
                        matrix_hits[row][j] = 'x';
                }
        return 0;
}
```

9) Функция, реализующая удар торпедой по выбранной строчке для пользователя

```
int UserTorpedo(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boa
t, Map *pmap, Map *pvisual_map, int row)
{
        if(row < 0 || row > 9)
        {
                printf("Line out of range (0-9)!\n");
                return 0;
        }
        for(int j = 0; j < FIELDSIZE; ++j)
                if(pmap->matrix_battle[row][j] == '+')
                {
                        int shot[2] = {row, j};
                        int res =  ChangeHealthShip(pbattle_ship, pcruiser, pdestroyer, ptorpedo_boat, pmap,
shot);
                        pmap->matrix_battle[row][j] = 'X';
                        pvisual_map->matrix_battle[row][j] = 'x';
                        return res;
                }
                else if(pmap->matrix_battle[row][j] == '0')
                {
                        pvisual_map->matrix_battle[row][j] = 'x';
                        pmap->matrix_battle[row][j] = 'x';
                }
        return 0;
}
```

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

3.1. Тесты ходов компьютера.

1) Тест создания случайной расстоновки кораблей для компьютера корректным образом

```
  0 1 2 3 4 5 6 7 8 9
A + 0 + 0 0 0 0 0 0 0
B 0 0 + 0 0 0 0 0 0 +
C 0 0 + 0 0 + 0 0 0 0
D 0 0 0 0 0 0 + 0 0 0 0
E 0 0 0 0 0 0 + 0 0 0 0
F 0 0 0 0 0 0 + 0 0 + +
G 0 0 0 0 0 0 0 0 0 0 0
H 0 0 0 + + + 0 0 0 0
I + 0 0 0 0 0 0 + 0 +
J + 0 0 0 0 + 0 + 0 0
student@3402-10:~/BattleSea/CursWorkSem2$ ./a.out
  0 1 2 3 4 5 6 7 8 9
A 0 0 0 0 0 0 0 0 0 0
B 0 0 0 + 0 + + + + 0
C + 0 0 0 0 0 0 0 0 0
D + 0 0 0 0 0 + 0 0 0
E + 0 + 0 0 0 + 0 0 0
F 0 0 0 0 0 0 + 0 0 0
G + + 0 0 0 0 0 0 0 0
H 0 0 0 + + 0 0 + 0 +
I 0 0 0 0 0 0 0 0 0 +
J 0 0 0 0 + 0 0 0 0 0
student@3402-10:~/BattleSea/CursWorkSem2$ ./a.out
  0 1 2 3 4 5 6 7 8 9
A 0 0 + 0 0 0 0 0 0 0
B 0 0 0 0 + + + + 0 0
C 0 0 0 0 0 0 0 0 0 0
D 0 0 0 0 0 0 0 + + + 0
E + 0 0 + + 0 0 0 0 0
F 0 0 0 0 0 0 0 0 0 +
G 0 0 0 0 0 0 0 0 0 0
H + 0 + + 0 0 0 0 0 0
I + 0 0 0 0 + + + 0 0
J 0 0 0 + 0 0 0 0 0 0
```

2) Тест ходов использования торпеды компьютером.

```
Enemy used torpedo!
User map:
  0 1 2 3 4 5 6 7 8 9
A + + + + 0 + + + 0 0
B x 0 0 0 0 0 0 0 0 0
C + + + 0 + + 0 + + 0
D x 0 0 x 0 0 x x x 0
E + + 0 + 0 + x X x +
F 0 0 x 0 x 0 x x x x
G x x x x x x x x x x
H 0 0 0 0 0 0 0 0 0 0
I 0 0 0 0 x 0 0 0 0 0
J 0 0 0 0 0 0 0 0 0 0
```

3) Тест следующего хода компьютера при при попадании

```
Enemy shot:C0
User map:
  0 1 2 3 4 5 6 7 8 9
A + + + + 0 + X X x 0
B x 0 0 0 x 0 x x x 0
C X + + 0 + + 0 + X x
D x 0 x x x 0 x x x x
E + + x X x + x X x +
F x 0 x x x 0 x x x x
G x x x x x x x x x x
H 0 0 0 0 x 0 0 0 0 0
I 0 0 0 0 x 0 0 x x 0
J 0 0 0 0 x 0 x 0 0 0
Enemy hijacking in your ship!
Enemy shot:C1
User map:
  0 1 2 3 4 5 6 7 8 9
A + + + + 0 + X X x 0
B x 0 0 0 x 0 x x x 0
C X X + 0 + + 0 + X x
D x 0 x x x 0 x x x x
E + + x X x + x X x +
F x 0 x x x 0 x x x x
G x x x x x x x x x x
H 0 0 0 0 x 0 0 0 0 0
I 0 0 0 0 x 0 0 x x 0
J 0 0 0 0 x 0 x 0 0 0
```

3.2 Тесты ходов пользователя.

1) Проверка на корректность введенных пользователем координат

```
Each coorinates line enter with new line
Please enter four coordinates for one battle ship to formate A1A2A3A4:
A0B0C0D0
Please enter three coordinates for two cruisers to formate A1A2A3:
D2E2F2
F4F5F6
Please enter three coordinates for three destroyers to formate A1A2:
J0J1
J3J4
J6J7
Please enter one coordinates for four destroyers to formate A1:
A9
C9
E9
F9
Start user map:
  0 1 2 3 4 5 6 7 8 9
A + 0 0 0 0 0 0 0 0 +
B + 0 0 0 0 0 0 0 0 0
C + 0 0 0 0 0 0 0 0 +
D + 0 + 0 0 0 0 0 0 0
E 0 0 + 0 0 0 0 0 0 +
F 0 0 + 0 + + + 0 0 +
G 0 0 0 0 0 0 0 0 0 0
H 0 0 0 0 0 0 0 0 0 0
I 0 0 0 0 0 0 0 0 0 0
J + + 0 + + 0 + + 0 0
Ships no insert between each other
Incorrect coordinates of ships!
```

```
Each coorinates line enter with new line
Please enter four coordinates for one battle ship to formate A1A2A3A4:
A0A1A2A3
Please enter three coordinates for two cruisers to formate A1A2A3:
A5A6A7
C0C1C2
Please enter three coordinates for three destroyers to formate A1A2:
C4C5
C7C8
E0E1
Please enter one coordinates for four destroyers to formate A1:
E3
E5
E7
E9
Start user map:
  0 1 2 3 4 5 6 7 8 9
A + + + + 0 + + + 0 0
B 0 0 0 0 0 0 0 0 0 0
C + + + 0 + + 0 + + 0
D 0 0 0 0 0 0 0 0 0 0
E + + 0 + 0 + 0 + 0 +
F 0 0 0 0 0 0 0 0 0 0
G 0 0 0 0 0 0 0 0 0 0
H 0 0 0 0 0 0 0 0 0 0
I 0 0 0 0 0 0 0 0 0 0
J 0 0 0 0 0 0 0 0 0 0
Ships no insert between each other
Correct coordinates of ships
```

2) Тест использования торпеды пользователем

```
Manuals for using weapons
For using torpedo choise row and enter T<value>
Enter coordinate of hit in format A1
T1
You used torpedo!
Enemy map:
  0 1 2 3 4 5 6 7 8 9
A 0 0 0 0 0 0 0 0 0 0
B x x x X 0 0 0 0 0 0
C 0 0 0 0 0 0 0 0 0 0
D 0 0 0 0 0 0 0 0 0 0
E 0 0 0 0 0 0 0 0 0 0
F 0 0 0 0 0 0 0 0 0 0
G 0 0 0 0 0 0 0 0 0 0
H 0 0 0 0 0 0 0 0 0 0
I 0 0 0 0 0 0 0 0 0 0
J 0 0 0 0 0 0 0 0 0 0
--------
Destroyed!
---------
Count of enemy ships = 9
Enter coordinate of hit in format A1
```

3) Тест закраски координат вокруг убитого корабля

```
Enter coordinate of hit in format A1
I4
Enemy map:
  0 1 2 3 4 5 6 7 8 9
A 0 0 0 0 0 0 0 0 0 0
B 0 0 0 0 0 0 0 0 0 0
C 0 0 0 0 0 0 0 0 0 0
D 0 0 0 0 0 0 0 0 0 0
E 0 0 0 0 0 0 0 0 0 0
F 0 0 0 0 X 0 0 0 0 0
G 0 0 0 0 X 0 0 0 0 0
H 0 0 0 0 X 0 0 0 0 0
I 0 0 0 0 X 0 0 0 0 0
J 0 0 0 0 0 0 0 0 0 0
--------
Destroyed!
---------
Count of enemy ships = 9
Enter coordinate of hit in format A1
A0
Enemy map:
  0 1 2 3 4 5 6 7 8 9
A X 0 0 0 0 0 0 0 0 0
B 0 0 0 0 0 0 0 0 0 0
C 0 0 0 0 0 0 0 0 0 0
D 0 0 0 0 0 0 0 0 0 0
E 0 0 0 x x x 0 0 0 0
F 0 0 0 x x x 0 0 0 0
G 0 0 0 x x x 0 0 0 0
H 0 0 0 x x x 0 0 0 0
I 0 0 0 x x x 0 0 0 0
J 0 0 0 x x x 0 0 0 0
```

ЗАКЛЮЧЕНИЕ

Были написаны структуры данных, для хранения координат и ячеек здоровья для основных по правилам игры кораблей. Были реализованы функции для проверки координат кораблей на корректность, функции для случайного заполнения карты противника кораблями корректным образом, функции изменяющие данные о здоровье кораблей в структурах данных и на поле противника, функция для выбора клетки ударом компьютера в зависимости от предыдущего удара и клеток, в которые он раннее уже стрелял. По заданию преподавателя были реализованы и добавлены функции реализующие дополнительное оружие — торпеда. Реализованные модули функций были протестированы по отдельности, в разделе 3 приводятся основные тесты программы при полной передаче хода компьютеру и пользователю.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://cppstudio.com
2. Википедия

Приложение А

Описание заголовочных файлов приводится после файла с кодом основной программы.

sea_battle.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#include "struct_ships.h"
#include "settings_ships.h"
#include "game_matrix.h"
#include "ships_flags.h"
#include "find_scale_of_ship.h"




void PrintMap(const Map * pmap)
{
	printf("  "); //two space for first index in top line
	for(int i = 0; i < FIELDSIZE; ++i)
		printf("%d ", index_rows[i]);
	printf("\n");
```

```
	for(int i = 0; i < FIELDSIZE; ++i)
	{
		printf("%c ", rows[i]);
		for(int j = 0; j < FIELDSIZE; ++j)
			printf("%c ", pmap->matrix_battle[i][j]);
		printf("\n");
	}
}




void InitializeMap(Map *pmap)
{
	for(int i = 0; i < FIELDSIZE; ++i)
		for(int j = 0; j < FIELDSIZE; ++j)
			pmap->matrix_battle[i][j] = '0';
}


void InitializeMatrixHits(char matrix_hits[FIELDSIZE][FIELDSIZE])
{
	for(int i = 0; i < FIELDSIZE; ++i)
		for(int j = 0; j < FIELDSIZE; ++j)
			matrix_hits[i][j] = '0';
}


void CopyMap(Map *pmap, char **matrix_field)
{
```

```
	for(int i = 0; i < FIELDSIZE; ++i)
		for(int j = 0; j < FIELDSIZE; ++j)
			matrix_field[i][j] = pmap->matrix_battle[i][j];
}



void MarkSpaceAroundCoordinate(char matrix_coordinate[FIELDSIZE][FIELDSIZE], int x, int y)
{
	//Mark upper coordintes
	if(x - 1 >= 0)
	{
		matrix_coordinate[x - 1][y] = 'x';
		if(y - 1 >= 0)
			matrix_coordinate[x - 1][y - 1] = 'x';
		if(y + 1 < FIELDSIZE)
			matrix_coordinate[x - 1][y + 1] = 'x';
	}
	//Mark lower coordinates
	if(x + 1 < FIELDSIZE)
	{
		matrix_coordinate[x + 1][y] = 'x';
		if(y - 1 >= 0)
			matrix_coordinate[x + 1][y - 1] = 'x';
		if(y + 1 < FIELDSIZE)
			matrix_coordinate[x + 1][y + 1] = 'x';
	}
	if(y - 1 >= 0)
```

```
            matrix_coordinate[x][y - 1] = 'x'; //Mark left coordinate
        if(y + 1 < FIELDSIZE)
            matrix_coordinate[x][y + 1] = 'x'; //Mark right coordinate
}

//Select orientation and next point of ship in map
void SetEnemyOrientation(Orientation *orientation, int x, int y, int deck)
{
        if(rand() % 2 == 1)
            orientation->horisontal = 1;
        if(y + deck - 1 < FIELDSIZE)
            orientation->right = 1;
        if(x + deck - 1 < FIELDSIZE)
            orientation->down = 1;
}


void InitializeEnemyShips(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer
*pdestroyer, TorpedoBoat *ptorpedo_boat,                int first_call)
{
        //for different value which will return rand()
        if(first_call)
            srand(time(NULL));
        int x;
        int y;
        char matrix_coordinate[FIELDSIZE][FIELDSIZE];
        for(int i = 0; i < FIELDSIZE; ++i)
            for(int j = 0; j < FIELDSIZE; ++j)
                matrix_coordinate[i][j] = '0';
```

```
for(int i = 0; i < COUNTFOURDECK; ++i)
{
	while(1)
	{
		x = rand() % FIELDSIZE;
		y = rand() % FIELDSIZE;
		if(matrix_coordinate[x][y] == '0')
			break;
	}
	Orientation orientation = {0, 0, 0};
	SetEnemyOrientation(&orientation, x, y, 4);

	for(int j = 0; j < 4; ++j)
	{
		pbattle_ship[i].health[j] = 1;
		pbattle_ship[i].coordinate[j][0] = x;
		pbattle_ship[i].coordinate[j][1] = y;
		matrix_coordinate[x][y] = 'x';
		MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
		if(orientation.horisontal)
		{
			if(orientation.right)
				++y;
			else
				--y;
		}
		else
		{
			if(orientation.down)
				++x;
```

```
                else
                        --x;
                }
        }
}
for(int i = 0; i < COUNTTHREEDECK; ++i)
{
        while(1)
        {
                x = rand() % FIELDSIZE;
                y = rand() % FIELDSIZE;
                if(matrix_coordinate[x][y] == '0')
                        break;
        }
        Orientation orientation = {0, 0, 0};
        SetEnemyOrientation(&orientation, x, y, 4);

        for(int j = 0; j < 3; ++j)
        {
                pcruiser[i].health[j] = 1;
                pcruiser[i].coordinate[j][0] = x;
                pcruiser[i].coordinate[j][1] = y;
                matrix_coordinate[x][y] = 'x';
                MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                if(orientation.horisontal)
                {
                        if(orientation.right)
                                ++y;
                        else
                                --y;
```

```
                }
                else
                {
                        if(orientation.down)
                                ++x;
                        else
                                --x;
                }
        }
}
for(int i = 0; i < COUNTTWODECK; ++i)
{
        while(1)
        {
                x = rand() % FIELDSIZE;
                y = rand() % FIELDSIZE;
                if(matrix_coordinate[x][y] == '0')
                        break;
        }
        Orientation orientation = {0, 0, 0};
        SetEnemyOrientation(&orientation, x, y, 2);

        for(int j = 0; j < 2; ++j)
        {
                pdestroyer[i].health[j] = 1;
                pdestroyer[i].coordinate[j][0] = x;
                pdestroyer[i].coordinate[j][1] = y;
                matrix_coordinate[x][y] = 'x';
                MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                if(orientation.horisontal)
```

```
                {
                        if(orientation.right)
                                ++y;
                        else
                                --y;
                }
                else
                {
                        if(orientation.down)
                                ++x;
                        else
                                --x;
                }
        }
}
for(int i = 0; i < COUNTONEDECK; ++i)
{
        while(1)
        {
                x = rand() % FIELDSIZE;
                y = rand() % FIELDSIZE;
                if(matrix_coordinate[x][y] == '0')
                        break;
        }
        Orientation orientation = {0, 0, 0};
        SetEnemyOrientation(&orientation, x, y, 1);

        for(int j = 0; j < 1; ++j)
        {
                ptorpedo_boat[i].health[j] = 1;
```

```
                    ptorpedo_boat[i].coordinate[j][0] = x;
                    ptorpedo_boat[i].coordinate[j][1] = y;
                    matrix_coordinate[x][y] = 'x';
                    MarkSpaceAroundCoordinate(matrix_coordinate, x, y);
                    if(orientation.horisontal)
                    {
                        if(orientation.right)
                            ++y;
                        else
                            --y;
                    }
                    else
                    {
                        if(orientation.down)
                            ++x;
                        else
                            --x;
                    }
                }
            }
}




void CleanMap(Map *pmap)
{
    for(int i = 0; i < FIELDSIZE; ++i)
        for(int j = 0; j < FIELDSIZE; ++j)
            pmap->matrix_battle[i][j] = '0';
```

```
}

//Mark ships coordinate by + on map
void ChangeMap(Map *pmap, BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat)
{
	CleanMap(pmap); //because this function is called until coordinate of enemy ships incorrects
	for(int i = 0; i < COUNTFOURDECK; ++i)
		for(int j = 0; j < 4; ++j)
		{
			int x = pbattle_ship[i].coordinate[j][0];
			int y = pbattle_ship[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}

	for(int i = 0; i < COUNTTHREEDECK; ++i)
		for(int j = 0; j < 3; ++j)
		{
			int x = pcruiser[i].coordinate[j][0];
			int y = pcruiser[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
		}

	for(int i = 0; i < COUNTTWODECK; ++i)
		for(int j = 0; j < 2; ++j)
		{
			int x = pdestroyer[i].coordinate[j][0];
			int y = pdestroyer[i].coordinate[j][1];
			pmap->matrix_battle[x][y] = '+';
```

```
            }

        for(int i = 0; i < COUNTONEDECK; ++i)
            for(int j = 0; j < 1; ++j)
            {
                int x = ptorpedo_boat[i].coordinate[j][0];
                int y = ptorpedo_boat[i].coordinate[j][1];
                pmap->matrix_battle[x][y] = '+';
            }
}



//Check coordinates of ship for diagonal and horisontal correct
int CorrectCoordinates(int coordinate[2][2], int deck)
{
        int horisontal_flag = 0;
        int vertical_flag = 0;
        if(coordinate[0][0] == coordinate[1][0])
            horisontal_flag = 1;
        if(coordinate[0][1] == coordinate[1][1])
            vertical_flag = 1;

        if(horisontal_flag && vertical_flag)
            return 0;

        //Check coordinates to vertical/diagonal coorect
        if(vertical_flag)
            for(int i = 0; i < deck - 1; ++i)
            {
```

```
                int dx = coordinate[i][0] - coordinate[i + 1][0];
                if(coordinate[i][1]!= coordinate[i + 1][1] || (dx!= 1 && dx!= -1))
                    return 0;
            }
        if(horisontal_flag)
            for(int i = 0; i < deck - 1; ++i)
            {
                int dy = coordinate[i][1] - coordinate[i + 1][1];
                if(coordinate[i][0]!= coordinate[i + 1][0] || (dy!= 1 && dy!= -1))
                    return 0;
            }


        return 1;
}


int CorrectShips(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer)
{
        for(int i = 0; i < COUNTFOURDECK; ++i)
            if(!CorrectCoordinates(pbattle_ship[i].coordinate, 4))
                return 0;

        for(int i = 0; i < COUNTTHREEDECK; ++i)
            if(!CorrectCoordinates(pcruiser[i].coordinate, 3))
                return 0;

        for(int i = 0; i < COUNTTWODECK; ++i)
            if(!CorrectCoordinates(pdestroyer[i].coordinate, 2))
                return 0;
```

```
        return 1;
}

//Counting points + on map. Correct is 20
int CorrectInsert(Map *pmap)
{
        int count_cell = 0;

        for(int i = 0; i < FIELDSIZE; ++i)
                for(int j = 0; j < FIELDSIZE; ++j)
                        if(pmap->matrix_battle[i][j] == '+')
                                ++count_cell;

        if(count_cell == COUNTCELLOFSHIPS)
        {
                return 1;
        }
        else
                return 0;
}

//Check for empty space around ship with using function which find scale of ships from library find_scale_of_ship.h
int ControlCheckShips(char **matrix_field)
{
        int coins_one = 0;
        int coins_two = 0;
        int coins_three = 0;
        int coins_four = 0;
```

```
int cur_scale = 1;

//10 is count of ships
for(int i = 0; i < 10; ++i)
{
	if(!CompleteTraversal(matrix_field))
	{
		int start_point[2];
		GetStartPoint(matrix_field, start_point);
		matrix_field[start_point[0]][start_point[1]] = MARK;
		FindCurScale(matrix_field, start_point, &cur_scale);
		switch(cur_scale)
		{
			case 1:
				++coins_one;
				break;
			case 2:
				++coins_two;
				break;
			case 3:
				++coins_three;
				break;
			case 4:
				++coins_four;
				break;
		}
		cur_scale = 1;
	}
}
```

```
	if(coins_one == COUNTONEDECK && coins_two == COUNTTWODECK
&& coins_three == COUNTTHREEDECK && coins_four == COUNTFOURDECK)
		return 1;
	else
		return 0;
}



//Check ship for destroy
int DestroyedShip(int health[1], int deck)
{
	int destroy_flag = 0;

	for(int i = 0; i < deck; ++i)
		destroy_flag += health[i];
	if(!destroy_flag)
		return 1;
	else
		return 0;
}

//Change health if shot and point of ship is equal
int ChangeHealthShip(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer
*pdestroyer, TorpedoBoat *ptorpedo_boat, Map *pmap, int shot[2])
{
	int x = shot[0];
	int y = shot[1];

	for(int i = 0; i < COUNTFOURDECK; ++i)
```

```
            for(int j = 0; j < 4; ++j)
                if(pbattle_ship[i].coordinate[j][0] == x &&
pbattle_ship[i].coordinate[j][1] == y && pmap->matrix_battle[x][y] == '+')
                {
                    pbattle_ship[i].health[j] = 0;
                    if(DestroyedShip(pbattle_ship[i].health, 4))
                        return DESTROYED;
                    else
                        return HIJACKING;
                }

        for(int i = 0; i < COUNTTHREEDECK; ++i)
            for(int j = 0; j < 3; ++j)
                if(pcruiser[i].coordinate[j][0] == x && pcruiser[i].coordinate[j][1]
== y && pmap->matrix_battle[x][y] == '+')
                {
                    pcruiser[i].health[j] = 0;
                    if(DestroyedShip(pcruiser[i].health, 3))
                        return DESTROYED;
                    else
                        return HIJACKING;
                }

        for(int i = 0; i < COUNTTWODECK; ++i)
            for(int j = 0; j < 2; ++j)
                if(pdestroyer[i].coordinate[j][0] == x &&
pdestroyer[i].coordinate[j][1] == y && pmap->matrix_battle[x][y] == '+')
                {
                    pdestroyer[i].health[j] = 0;
                    if(DestroyedShip(pdestroyer[i].health, 2))
```

```
                return DESTROYED;
            else
                return HIJACKING;
        }

    for(int i = 0; i < COUNTONEDECK; ++i)
        for(int j = 0; j < 1; ++j)
            if(ptorpedo_boat[i].coordinate[j][0] == x &&
ptorpedo_boat[i].coordinate[j][1] == y && pmap->matrix_battle[x][y] == '+')
        {
            ptorpedo_boat[i].health[j] = 0;
            if(DestroyedShip(ptorpedo_boat[i].health, 1))
                return DESTROYED;
            else
                return HIJACKING;
        }
    return 0;
}


void MarkShotOnMap(Map *pmap, int shot[2])
{
    int x = shot[0];
    int y = shot[1];

    if(pmap->matrix_battle[x][y] == '+')
        pmap->matrix_battle[x][y] = 'X';
    else if(pmap->matrix_battle[x][y] == '0')
        pmap->matrix_battle[x][y] = 'x';
}
```

```
void MarkShotOnEnemyMap(Map *pmap, Map *pvisual_map, int shot[2])
{
	int x = shot[0];
	int y = shot[1];

	if(pmap->matrix_battle[x][y] == '+')
	{
		pvisual_map->matrix_battle[x][y] = 'X';
		pmap->matrix_battle[x][y] = 'X';
	}
	else if(pmap->matrix_battle[x][y] == '0')
	{
		pvisual_map->matrix_battle[x][y] = 'x';
		pmap->matrix_battle[x][y] = 'x';
	}
}


void MarkSpaceAroundDestroyedShip(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat, Map *pmap)
{
	if(DestroyedShip(pbattle_ship[0].health, 4))
		for(int i = 0; i < 4; ++i)
		{
			int x = pbattle_ship[0].coordinate[i][0];
			int y = pbattle_ship[0].coordinate[i][1];
			MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
		}
```

```
        for(int i = 0; i < COUNTTWODECK; ++i)
                if(DestroyedShip(pcruiser[i].health, 3))
                    for(int j = 0; j < 3; ++j)
                    {
                            int x = pcruiser[i].coordinate[j][0];
                            int y = pcruiser[i].coordinate[j][1];
                            MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                    }
        for(int i = 0; i < COUNTTHREEDECK; ++i)
                if(DestroyedShip(pdestroyer[i].health, 2))
                    for(int j = 0; j < 2; ++j)
                    {
                            int x = pdestroyer[i].coordinate[j][0];
                            int y = pdestroyer[i].coordinate[j][1];
                            MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                      }
        for(int i = 0; i < COUNTONEDECK; ++i)
                if(DestroyedShip(ptorpedo_boat[i].health, 1))
                    for(int j = 0; j < 1; ++j)
                    {
                            int x = ptorpedo_boat[i].coordinate[j][0];
                            int y = ptorpedo_boat[i].coordinate[j][1];
                            MarkSpaceAroundCoordinate(pmap->matrix_battle, x, y);
                      }
}


//mark for exclude points in next hits of enemy
```

```
void MarkSpaceAroundCoordinateInMatrixHits(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat, char matrix_hits[FIELDSIZE][FIELDSIZE])
{
	if(DestroyedShip(pbattle_ship[0].health, 4))
		for(int i = 0; i < 4; ++i)
		{
			int x = pbattle_ship[0].coordinate[i][0];
			int y = pbattle_ship[0].coordinate[i][1];
			MarkSpaceAroundCoordinate(matrix_hits, x, y);
		}
	for(int i = 0; i < COUNTTWODECK; ++i)
		if(DestroyedShip(pcruiser[i].health, 3))
		    for(int j = 0; j < 3; ++j)
		    {
			int x = pcruiser[i].coordinate[j][0];
			int y = pcruiser[i].coordinate[j][1];
			MarkSpaceAroundCoordinate(matrix_hits, x, y);
		    }
	for(int i = 0; i < COUNTTHREEDECK; ++i)
		if(DestroyedShip(pdestroyer[i].health, 2))
		    for(int j = 0; j < 2; ++j)
		    {
			int x = pdestroyer[i].coordinate[j][0];
			int y = pdestroyer[i].coordinate[j][1];
			MarkSpaceAroundCoordinate(matrix_hits, x, y);
		    }
	for(int i = 0; i < COUNTONEDECK; ++i)
		if(DestroyedShip(ptorpedo_boat[i].health, 1))
		    for(int j = 0; j < 1; ++j)
```

```
                {
                        int x = ptorpedo_boat[i].coordinate[j][0];
                        int y = ptorpedo_boat[i].coordinate[j][1];
                        MarkSpaceAroundCoordinate(matrix_hits, x, y);
                }
}



//Trancfer char to index for matrix of battle
int CharToIndex(char c)
{
        for(int i = 0; i < FIELDSIZE; ++i)
                if(c == rows[i])
                        return i;

        printf("Coordinate out of range A-J!\n");
        exit(1);
}


int OutOfRange(int x, int y)
{
        if(x < 0 || x > 9 || y < 0 | y > 9)
                return 1;
        else
                return 0;
}
```

```
//Random offset for new hit for computer
int RandomOffset()
{
	int r_num = (unsigned int)rand() % 10;

	if(r_num < 3 )
		return 1;
	else if(r_num >= 3 && r_num < 6 )
		return 0;
	else
		return 1;
}


void EnemyHit(int shot[2], char matrix_hits[FIELDSIZE][FIELDSIZE], int hijacking_flag)
{
	int x = shot[0];
	int y = shot[1];
	int new_hit_flag = 0;

	if(!hijacking_flag)
	{
		while(!new_hit_flag)
		{
			x = ((unsigned int)rand()) % 10;
			y = ((unsigned int)rand()) % 10;

			if(matrix_hits[x][y] == '0')
```

```
                {
                        matrix_hits[x][y] = 'x';
                        new_hit_flag = 1;
                }
        }
}
else
{
        int time_x = x;
        int time_y = y;
        int count_try = 40; //except, when around shot point look points yet

        while(!new_hit_flag && count_try)
        {
                time_x = x + RandomOffset();
                time_y = y + RandomOffset();

                if(!OutOfRange(time_x, time_y) && matrix_hits[time_x][time_y]
== '0' &&(abs(x-time_x)!= abs(y-time_y)))
                {
                        matrix_hits[time_x][time_y] = 'x';
                        new_hit_flag = 1;
                        x = time_x;
                        y = time_y;
                        break;
                }
                --count_try;
        }
        if(!count_try)
        {
```

```
                x = ((unsigned int)rand()) % 10;
                y = ((unsigned int)rand()) % 10;
                matrix_hits[x][y] = 'x';
            }
        }

        shot[0] = x;
        shot[1] = y;
}



//Additionan weapons
int EnemyTorpedo(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat, Map *pmap, char matrix_hits[FIELDSIZE][FIELDSIZE], int row)
{
        for(int j = 0; j < FIELDSIZE; ++j)
            if(pmap->matrix_battle[row][j] == '+')
            {
                int shot[2] = {row, j};
                int res =  ChangeHealthShip(pbattle_ship, pcruiser, pdestroyer, ptorpedo_boat, pmap, shot);
                pmap->matrix_battle[row][j] = 'X';
                matrix_hits[row][j] = 'x';
                return res;
            }
            else if(pmap->matrix_battle[row][j] == '0')
            {
                pmap->matrix_battle[row][j] = 'x';
```

```
                matrix_hits[row][j] = 'x';
            }
        return 0;
}


int UserTorpedo(BattleShip *pbattle_ship, Cruiser *pcruiser, Destroyer *pdestroyer, TorpedoBoat *ptorpedo_boat, Map *pmap, Map *pvisual_map, int row)
{
        if(row < 0 || row > 9)
        {
            printf("Line out of range (0-9)!\n");
            return 0;
        }
        for(int j = 0; j < FIELDSIZE; ++j)
            if(pmap->matrix_battle[row][j] == '+')
            {
                int shot[2] = {row, j};
                int res =  ChangeHealthShip(pbattle_ship, pcruiser, pdestroyer, ptorpedo_boat, pmap, shot);
                pmap->matrix_battle[row][j] = 'X';
                pvisual_map->matrix_battle[row][j] = 'x';
                return res;
            }
            else if(pmap->matrix_battle[row][j] == '0')
            {
                pvisual_map->matrix_battle[row][j] = 'x';
                pmap->matrix_battle[row][j] = 'x';
            }
        return 0;
```

```
}



int main()
{
	Map user_map;
	InitializeMap(&user_map);

	BattleShip user_battle_ship[1];
	Cruiser user_cruiser[2];
	Destroyer user_destroyer[3];
	TorpedoBoat user_torpedo_boat[4];

	Map enemy_map;
	Map enemy_visual_map; //for incapsulation data of position the enemy ships
	InitializeMap(&enemy_map);
	InitializeMap(&enemy_visual_map);

	BattleShip enemy_battle_ship[1];
	Cruiser enemy_cruiser[2];
	Destroyer enemy_destroyer[3];
	TorpedoBoat enemy_torpedo_boat[4];

	InitializeEnemyShips(&enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0], 1);
	ChangeMap(&enemy_map, &enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0]);
```

```
char **matrix_field = (char **)malloc(sizeof(char *) * FIELDSIZE); //matrix for check space around ship to empty
for(int i = 0; i < FIELDSIZE; ++i)
    matrix_field[i] = (char *)malloc(sizeof(char) * FIELDSIZE);
CopyMap(&enemy_map, matrix_field);
//because function of random dot't check ships for insert(only horisonatal and vertical corrects)
while(1)
{
    if(!CorrectInsert(&enemy_map) || !ControlCheckShips(matrix_field))
    {
    InitializeEnemyShips(&enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0], 0);
    ChangeMap(&enemy_map, &enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0]);
    CopyMap(&enemy_map, matrix_field);
    }
    else
        break;
}
PrintMap(&enemy_map);


//Alignment ships by the user
printf("Each coorinates line enter with new line\n");
printf("Please enter four coordinates for one battle ship to formate A1A2A3A4:\n");

for(int i = 0; i < COUNTFOURDECK; ++i)
{
```

```
            char row;
            int col;

            for(int j = 0; j < 4; ++j)
            {
                user_battle_ship[i].health[j] = 1;

                if(scanf("%c%d", &row, &col)!= 2)
                {
                    printf("Error input!\n");
                    return 0;
                }
                if(OutOfRange(CharToIndex(toupper(row)), col))
                {
                    printf("Coordinate out of range!\n");
                    return 0;
                }
                else
                {
                    //printf("[%c, %d]\n", col, row);
                    user_battle_ship[i].coordinate[j][0]                    =
CharToIndex(toupper(row));
                    user_battle_ship[i].coordinate[j][1] = col;
                }
            }
            getchar(); //for \n
        }

        printf("Please enter three coordinates for two cruisers to formate A1A2A3:\n");
```

```
	for(int i = 0; i < COUNTTHREEDECK; ++i)
	{
		char row;
		int col;

		for(int j = 0; j < 3; ++j)
		{
			user_cruiser[i].health[j] = 1;

			if(scanf("%c%d", &row, &col)!= 2)
			{
				printf("Error input!\n");
				return 0;
			}
			if(OutOfRange(CharToIndex(toupper(row)), col))
			{
				printf("Coordiante out of range!\n");
				return 0;
			}
			else
			{
				//printf("[%c, %d]\n", col, row);
				user_cruiser[i].coordinate[j][0] =
CharToIndex(toupper(row));
				user_cruiser[i].coordinate[j][1] = col;
			}
		}
		getchar();
	}
```

```
printf("Please enter three coordinates for three destroyers to formate A1A2:\n");

for(int i = 0; i < COUNTTWODECK; ++i)
{
	char row;
	int col;

	for(int j = 0; j < 2; ++j)
	{
		user_destroyer[i].health[j] = 1;

		if(scanf("%c%d", &row, &col)!= 2)
		{
			printf("Error input!\n");
			return 0;
		}
		if(OutOfRange(CharToIndex(toupper(row)), col))
		{
			printf("Coordinate out of range!\n");
			return 0;
		}
		else
		{
			//printf("[%c, %d]\n", col, row);
			user_destroyer[i].coordinate[j][0] = CharToIndex(toupper(row));
			user_destroyer[i].coordinate[j][1] = col;
		}
	}
	getchar();
```

```
    }

    printf("Please enter one coordinates for four destroyers to formate A1:\n");

    for(int i = 0; i < COUNTONEDECK; ++i)
    {
        char row;
        int col;

        for(int j = 0; j < 1; ++j)
        {
            user_torpedo_boat[i].health[j] = 1;

            if(scanf("%c%d", &row, &col)!= 2)
            {
                printf("Error input!\n");
                return 0;
            }
            if(OutOfRange(CharToIndex(toupper(row)), col))
            {
                printf("Coordinate out of range!\n");
                return 0;
            }
            else
            {
                //printf("[%c, %d]\n", col, row);
                user_torpedo_boat[i].coordinate[j][0]                    =
CharToIndex(toupper(row));
                user_torpedo_boat[i].coordinate[j][1] = col;
            }
```

```
        }
        getchar();
    }


    ChangeMap(&user_map, &user_battle_ship[0], &user_cruiser[0],
&user_destroyer[0], &user_torpedo_boat[0]);
    CopyMap(&user_map, matrix_field); //check to empty space around user ships
    printf("Start user map:\n");
    PrintMap(&user_map);

    //Check for correct coordinate of ships
    if(CorrectInsert(&user_map))
        printf("Ships no insert between each other\n");
    else
    {
        printf("Ships insert between each other!\n");
        return 0;
    }
    if(CorrectShips(&user_battle_ship[0], &user_cruiser[0], &user_destroyer[0])
&& ControlCheckShips(matrix_field))
        printf("Correct coordinates of ships\n");
    else
    {
        printf("Incorrect coordinates of ships!\n");
        return 0;
    }

    //Battle progress
    int count_user_ships = 10;
```

```
int count_enemy_ships = 10;
int count_user_torpedo = 1; //additional weapons x1
int count_enemy_torpedo = 1; //additional weapons x1
int user_hit = 1;
int enemy_hit = 0;
char matrix_hits[FIELDSIZE][FIELDSIZE]; //matrix for coordinates, which computer already select
int hijacking_flag = 0; //flag for next enemy hit(random point or around with some point)

InitializeMatrixHits(matrix_hits);
srand(time(NULL));

printf("Manuals for using weapons\n");
printf("For using torpedo choise row and enter T<value>\n");

while(count_user_ships > 0 && count_enemy_ships > 0)
{
    int col;
    char row;
    int shot[2];

    if(enemy_hit)
    {
        int course_res; //for save result which return UseTorpedo() or ChangeHealthShips()
        //Use torpedo
        if(rand() % 4 == 1 && count_enemy_torpedo == 1)
        {
            printf("Enemy used torpedo!\n");
```

```
                course_res = EnemyTorpedo(&user_battle_ship[0], &user_cruiser[0], &user_destroyer[0], &user_torpedo_boat[0], &user_map, matrix_hits, rand() % 10);
                printf("User map:\n");
                PrintMap(&user_map);
                --count_enemy_torpedo;
            }
            else
            {
                EnemyHit(shot, &matrix_hits[0], hijacking_flag);
                printf("Enemy shot:%c%u\n", rows[shot[0]], shot[1]);
                course_res = ChangeHealthShip(&user_battle_ship[0], &user_cruiser[0], &user_destroyer[0], &user_torpedo_boat[0], &user_map, shot);
                MarkShotOnMap(&user_map, shot);
                printf("User map:\n");
                PrintMap(&user_map);
            }
            if(course_res == DESTROYED)
            {
                printf("Enemy destroyed your ship!\n");
                --count_user_ships;
                MarkSpaceAroundDestroyedShip(&user_battle_ship[0], &user_cruiser[0], &user_destroyer[0], &user_torpedo_boat[0], &user_map);

        MarkSpaceAroundCoordinateInMatrixHits(&user_battle_ship[0], &user_cruiser[0], &user_destroyer[0], &user_torpedo_boat[0], matrix_hits);
            }
            else if(course_res == HIJACKING)
            {
                printf("Enemy hijacking in your ship!\n");
```

```
                hijacking_flag = 1;
            }
            else
            {
                hijacking_flag = 0;
                enemy_hit = 0;
                user_hit = 1;
            }
        }
        if(user_hit)
        {
            int course_res; //for save return of UseTorpedo() or ChangeHealthShip()
            printf("Enter coordinate of hit in format A1\n");
            if(scanf("%c%d", &row, &col)!= 2 && toupper(row)!= 'T')
            {
                printf("%c%d\n", row, col);
                printf("Incorrect coordinate!\n");
                return 0;
            }
            getchar();

            //Use torpedo
            if(count_user_torpedo == 1 && toupper(row) == 'T' && col < 10 && col >= 0)
            {
                printf("You used torpedo!\n");
                course_res = UserTorpedo(&enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0], &enemy_map, &enemy_visual_map, col);
```

```
                printf("Enemy map:\n");
                PrintMap(&enemy_visual_map);
                --count_user_torpedo;
        }
        else
        {
                shot[0] = CharToIndex(toupper(row));
                shot[1] = col;
                if(OutOfRange(shot[0], shot[1]))
                {
                        printf("(%d, %d)\n", shot[0], shot[1]);
                        return 0;
                }

        //variable for check shot to hikacking to ship or destroyed to ship
                course_res = ChangeHealthShip(&enemy_battle_ship[0], &enemy_cruiser[0], &enemy_destroyer[0], &enemy_torpedo_boat[0], &enemy_map, shot);
                MarkShotOnEnemyMap(&enemy_map, &enemy_visual_map, shot);
                printf("Enemy map:\n");
                PrintMap(&enemy_visual_map);

        }
        if(course_res == DESTROYED)
        {
                --count_enemy_ships;
                printf("--------\nDestroyed!\n---------\n");
                printf("Count of enemy ships = %d\n", count_enemy_ships);
```

```
                        MarkSpaceAroundDestroyedShip(&enemy_battle_ship[0],
&enemy_cruiser[0],          &enemy_destroyer[0],          &enemy_torpedo_boat[0],
&enemy_visual_map);
                    }
                    else if(course_res == HIJACKING)
                        printf("--------\nHijackig!\n---------\n");
                    else
                    {
                        user_hit = 0;
                        enemy_hit = 1;
                    }
                }
            }
            if(!count_enemy_ships)
                printf("--------\nYou win!\n---------");
            else if(!count_user_ships)
                printf("--------\nYou lose\n---------");


            return 0;
}
```

find_scale_of_ship.c

```
#include "settings_ships.h"
#include "ships_flags.h"

int CompleteTraversal(char **matrix_field)
{
    for(int i = 0; i < FIELDSIZE; ++i)
```

```
            for(int j = 0; j < FIELDSIZE; ++j)
                if(matrix_field[i][j] == FIGURE)
                    return 0;

    return 1;
}


void GetStartPoint(char **matrix_field, int start_point[2])
{
    for(int i = 0; i < FIELDSIZE; ++i)
        for(int j = 0; j < FIELDSIZE; ++j)
            if(matrix_field[i][j] == FIGURE)
            {
                start_point[0] = i;
                start_point[1] = j;
                return;
            }
}


//Find scale of ship on battle field for check to correct arrangement
void FindCurScale(char **matrix_field, int cur_point[2], int *cur_scale)
{
    int x = cur_point[0];
    int y = cur_point[1];

    //Check right/left and up/down points
    if(x + 1 < FIELDSIZE  && matrix_field[x + 1][y] == FIGURE)
    {
```

```
        ++*cur_scale;
        matrix_field[x + 1][y] = MARK;
        cur_point[0] = x + 1;
        cur_point[1] = y;
        FindCurScale(matrix_field, cur_point, cur_scale);
}
if(x - 1 >= 0 && matrix_field[x - 1][y] == FIGURE)
{
        ++*cur_scale;
        matrix_field[x - 1][y] = MARK;
        cur_point[0] = x - 1;
        cur_point[1] = y;
        FindCurScale(matrix_field, cur_point, cur_scale);
}
if(y + 1 < FIELDSIZE && matrix_field[x][y + 1] == FIGURE)
{
        ++*cur_scale;
        matrix_field[x][y + 1] = MARK;
        cur_point[0] = x;
        cur_point[1] = y + 1;
        FindCurScale(matrix_field, cur_point, cur_scale);
}
if(y - 1 >= 0 && matrix_field[x][y - 1] == FIGURE)
{
        ++*cur_scale;
        matrix_field[x][y - 1] = MARK;
        cur_point[0] = x;
        cur_point[1] = y - 1;
        FindCurScale(matrix_field, cur_point, cur_scale);
}
```

```
        //Check horisontal points
        if(x - 1 >= 0 && y + 1 < FIELDSIZE && matrix_field[x - 1][y + 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x - 1][y + 1] = MARK;
                cur_point[0] = x - 1;
                cur_point[1] = y + 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x - 1 >= 0 && y - 1 >= 0 && matrix_field[x - 1][y - 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x - 1][y - 1] = MARK;
                cur_point[0] = x - 1;
                cur_point[1] = y - 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x + 1 < FIELDSIZE && y + 1 < FIELDSIZE && matrix_field[x + 1][y + 1]
== FIGURE)
        {
                ++*cur_scale;
                matrix_field[x + 1][y + 1] = MARK;
                cur_point[0] = x + 1;
                cur_point[1] = y + 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
        if(x + 1 < FIELDSIZE && y - 1 >= 0 && matrix_field[x + 1][y - 1] == FIGURE)
        {
                ++*cur_scale;
                matrix_field[x + 1][y - 1] = MARK;
```

```
                cur_point[0] = x + 1;
                cur_point[1] = y - 1;
                FindCurScale(matrix_field, cur_point, cur_scale);
        }
}
```

struct_ships.h

```
#pragma once

#include "settings_ships.h"

typedef struct map
{
        char matrix_battle[FIELDSIZE][FIELDSIZE];
} Map;

typedef struct battle_ship
{
        int coordinate[4][2];
        int health[4];
} BattleShip;

typedef struct cruiser
{
        int coordinate[3][2];
        int health[3];
} Cruiser;

typedef struct destroyer
```

```
{
	int coordinate[2][2];
	int health[2];
} Destroyer;

typedef struct torpedo_boat
{
	int coordinate[1][2];
	int health[1];
} TorpedoBoat;


//struct for set orientation of enemy ships
typedef struct
{
	int horisontal;
	int right;
	int down;
} Orientation;
```

settings_ships.h

```
#pragma once
#define FIELDSIZE 10

#define COUNTFOURDECK 1
#define COUNTTHREEDECK 2
#define COUNTTWODECK 3
#define COUNTONEDECK 4

#define COUNTCELLOFSHIPS 20
```

ships_flags.h

```
#pragma once
#define HIJACKING 1
#define DESTROYED 2

#define FIGURE '+'
#define EMPTYCAGE '0'
#define MARK 'x'

#pragma once
```

game_matrix.h

```
char const rows[FIELDSIZE] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
int const index_rows[FIELDSIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

find_scale_of_ship.h

```
#pragma once
int CompleteTraversal(char **matrix_field);

void GetStartPoint(char **matrix_field, int start_point[2]);

void FindCurScale(char **matrix_field, int cur_point[2], int *cur_scale);
```

Makefile

```
play: sea_battle.o find_scale_of_ship.o
		gcc -o play sea_battle.o find_scale_of_ship.o && ./play
sea_battle.o: sea_battle.c game_matrix.h ships_flags.h settings_ships.h
coordinate_enemy_ships.h struct_ships.h
	gcc -c sea_battle.c
```

```
find_scale_of_ship.o: find_scale_of_ship.c find_scale_of_ship.h ships_flags.h
	gcc -c find_scale_of_ship.c
clean:
	rm -rf *.o play
```