Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования "Новгородский государственный университет имени Ярослава Мудрого"

Кафедра «Информационных технологий и систем»

Дисциплина «Операционные системы»

Курсовая работа

«Морской бой»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Федотов Дмитрий Борисович/

Подпись ФИО

Принял преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_/Ананьев Владислав Валерьевич/

Подпись ФИО

Великий Новгород

2021

ОГЛАВЛЕНИЕ

[Разработка проекта 2](#_Toc73710612)

[Введение 2](#_Toc73710613)

[Постановка задачи 2](#_Toc73710614)

[Выбор инструментальных средств 3](#_Toc73710615)

[Выбор модели 3](#_Toc73710616)

[Выбор протокола транспортной OSI-модели 3](#_Toc73710617)

[Результат работы 4](#_Toc73710618)

[Заключение 4](#_Toc73710619)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 4](#_Toc73710620)

[ПРИЛОЖЕНИЕ А 5](#_Toc73710621)

[ПРИЛОЖЕНИЕ Б 6](#_Toc73710622)

[ЛИСТИНГ SERVER.C 6](#_Toc73710623)

[ЛИСТИНГ CLIENT.C 17](#_Toc73710624)

[ЛИСТИНГ DEFINE.H 39](#_Toc73710625)

[ЛИСТИНГ FIELD.H 41](#_Toc73710626)

[ЛИСТИНГ FIELD.C 42](#_Toc73710627)

[ЛИСТИНГ UTILITY.H 45](#_Toc73710628)

[ЛИСТИНГ UTILITY.C 46](#_Toc73710629)

[ЛИСТИНГ RENDERER.H 47](#_Toc73710630)

[ЛИСТИНГ RENDERER.C 48](#_Toc73710631)

[ЛИСТИНГ COMPUTER.H 53](#_Toc73710632)

[ЛИСТИНГ COMPUTER.C 54](#_Toc73710633)

# Разработка проекта

## Введение

Целью данной работы является создания клона некогда популярной игры «Морской бой». Морской бой — это игра для двух участников, в которой игроки по очереди называют координаты на неизвестной им карте соперника. Если у соперника по этим координатам имеется корабль (координаты заняты кораблем), то корабль или его часть «топится», а попавший получает право сделать ещё один ход. Цель игрока — первым потопить все корабли противника.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux.

Основные правила и суть задачи. Количества игроков в данной игре — 2 игрока, причем должно быть 3 пары: игрок против игрока, игрок против компьютера, компьютер после компьютера.

В начале игры задается расположение кораблей каждого игрока – чтением координат кораблей из файла или случайным образом. В процессе расстановки кораблей программа должна проверять то, что их количество верное: 1 из 4 клеток (Линейный корабль), 2 из 3 клеток (Крейсера), 3 из 2х (Эсминцы), 4 корабля по 1 клетке (Торпедные катер). Корабли не могут соприкасаться краями и углами. Корабли не могут быть изогнуты или установлены по диагонали, т. е. все клетки корабля располагаются либо горизонтально, либо вертикально.

В начале каждого хода экран очищается и выводится 2 поля 10x10 клеток. На поле игрока, который сейчас должен ходить, отмечены пораженные клетки и корабли. На втором поле отмечены только пораженные клетки. Если в пораженной клетке был корабль, то это должно быть также указано (цветом или особым символом).

Ход игрока состоит в указании координат клетки поля соперника, которая далее будет считаться пораженной. В случае нахождении корабля в пораженной клетке, ход остается у того же игрока, в противном случае ход переходит противнику. За обработку хода каждого игрока отвечает отдельный поток.

Обязательное дополнение для реализации параллельных процессов: компьютерный игрок в процессе ожидания завершения хода другого игрока (неважно, компьютерного или нет) должен составить для себя список последовательности клеток, по которому он собирается поражать клетки другого игрока. Таким образом, к началу своего хода, компьютер уже должен быть готов ходить без дополнительных размышлений. В случае обнаружения корабля противника компьютерный игрок должен начать построение списка с нуля, с учетом выясненной информации. Но построение нового списка должно начаться только когда ход перейдет к противнику, текущий ход будет производиться без использования списка, непосредственным выбором координат стрельбы.

## Постановка задачи

Задачей данной курсовой работы является разработка игры «Морской бой», в которой должна быть возможность выбора того, кем будет управлять каждым игроком (Игрок — Игрок, Игрок — Компьютер, Компьютер — Компьютер). Игра должны работать под управлением операционной системы семейства Unix/Linux. Игра должна быть выполнена на языке Си, а сетевая часть с помощью сокетов, представленных системной библиотекой <sys/socket.h>

## Выбор инструментальных средств

Сетевая часть программы будет реализована с помощью библиотек сокетов Беркли на Си. В нашем случае мы разрабатываем игру, поэтому нам необходимо, чтобы каждый пакет точно дошел до своего получателя в целости и сохранности, поэтому был использован протокол передачи данных семейства TCP/IP

Графический интерфейс не используется, вместо него используется консольный интерфейс с применением библиотеки <ncurses.h>, предназначенной для управления ввода-выводом на терминал, в числе прочего, библиотека позволяет задавать экранные координаты и цвет выводимых символов. Помимо этого, <ncurses.h> предоставляет нам разные уровни абстракции, позволяющие не беспокоиться об аппаратных различиях терминалов и писать переносимый код.

Так как благодаря <ncurses.h> нам не нужно заботиться об переносимости, мы можем использовать любой дистрибутив линукса и не думать о том, что наша программа где-то не так заработает.

## Выбор модели

В сетевой игре могут принимать участие только два игрока. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Первая часть игры заключается в расположении игроками кораблей, причем корабли игрок может расставлять сам, задавая координаты вручную, либо игрок может случайно расставить корабли или считать их расположение с текстового файла. Сервер должен принимать координаты от клиентов, выполнять проверку координат, возможность размещение данного корабля на поле. В основной части игры сервер должен принимать координаты, проверять их правильность, координировать ходы клиентов (кто ходит, а кто ждет), отсылать состояния клиентов и результаты ходов.

## Выбор протокола транспортной OSI-модели

Перед тем как реализовывать сетевую часть игру, необходимо определиться, а какой протокол нужно использовать UDP или TCP/IP?

Начнем с TCP. Большим плюсом данного протокола — это его высокая надёжность, поскольку данный протокола позволяет не терять данные при передаче, запрашивая подтверждения о получении от принимающей стороны и в случае необходимости отправлять данные повторно. Но у этого протокола есть небольшой минус — это относительная низкая скорость передачи данных, за счет того, что выполнение надежной и упорядоченной передачи занимает больше времени.

Для протокола TCP есть альтернатива — протокол UDP. Данный уже протокол является противоположность по преимуществам и недостаткам TCP. Если TCP надежный, то UDP уже нет. TCP медленный, а UDP быстрый.

|  |  |  |  |
| --- | --- | --- | --- |
| Протокол TCP | | Протокол UDP | |
| Преимущества | Недостатки | Преимущества | Недостатки |
| Надежность. Подтверждения получения данных, повторной отправки в случае необходимости | Скорость. | Скорость. | Надежность. При получении отправленные данные могут приходить не полностью; |
| Упорядоченность. Гарантируется передача данных в том порядке, в котором они были отправлены | - | - | Упорядоченность. Порядок передачи данных не соблюдается |
| Метод передачи данных — потоковая. Границы фрагментов данных не имеют обозначения. | - | - | Метод передачи данных — датаграмма. Проверка пакетов на целостность осуществляется принимающей стороной только в случае получения сообщения |

Для своей курсовой работы я решил использовать TCP-протокол, потому что мне необходимо, чтобы все пакеты всегда доходили от сервера до клиентов и от клиентов до сервера. Если до одного из двух игроков не дойдет сообщение, то между одним из полей игрока и сервера будет иметься разница, которая в нашем случае не допустима.

## Результат работы

В ходе выполнения данной курсовой работы удалось разработать игру «Морской бой» с разными игровыми режимами. Консольный интерфейс удалось реализовать с помощью библиотеки <ncurses.h>. Управлять игроком может как компьютер игрок, так и живой игрок. Причем за обработку каждого игрока отвечает отдельный поток.

## Заключение

Таким образом, результат курсовой работы полностью соответствует требованиям к реализации, все поставленные задачи были выполнены. Программа работает на операционных системах семейства Linux и написана на языке Си (компилятор gcc). Серверная часть программы построена с помощью протокола TCP. Созданная программа логически завершена.

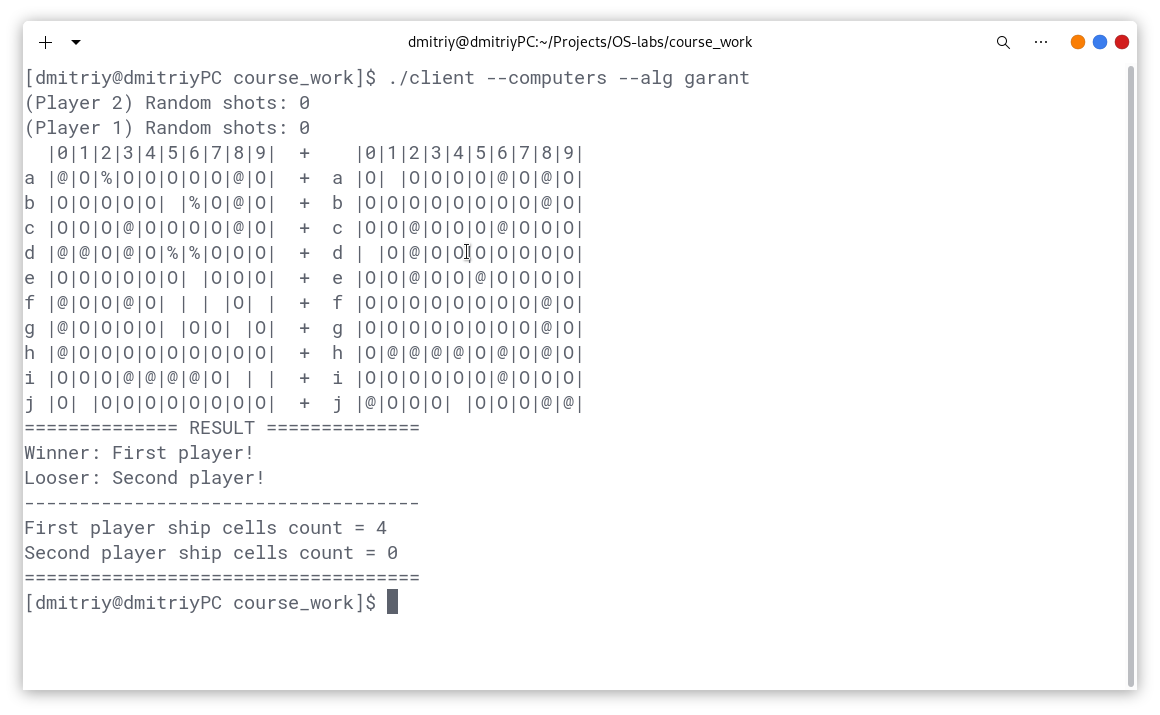
# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Карпов В. Е, Коньков К. А. «Основы операционных систем. Курс лекций» - Учебное пособие, 2005 — 536 с.

ПРИЛОЖЕНИЕ А



Аргументы командной строки



Результат режима “Компьютер против Компьютера”

# ПРИЛОЖЕНИЕ Б

# ЛИСТИНГ SERVER.C

#include "time.h"

#include "stdio.h"

#include "errno.h"

#include "unistd.h"

#include "stdlib.h"

#include "string.h"

#include "dirent.h"

#include "signal.h"

#include "sys/sem.h"

#include "sys/ipc.h"

#include "pthread.h"

#include "strings.h"

#include "sys/stat.h"

#include "arpa/inet.h"

#include "sys/types.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "field.h"

#include "define.h"

#include "utility.h"

#include "renderer.h"

typedef struct thread\_player\_info\_struct

{

char player\_num;

game\_field\* own\_field;

game\_field\* enemy\_field;

client\_states\* own\_state;

client\_states\* enemy\_state;

u\_int16\_t tcp\_socket\_fd;

} thread\_player\_info;

int sem\_id;

ship ship\_destroyed;

client\_shot global\_shot;

void print\_use\_help\_and\_exit()

{

printf("\*----------- Use this options ------------\*\n");

printf("| <nothing> == To start with any port |\n");

printf("| <port> == To start with selected port |\n");

printf("| <help> == To print help |\n");

printf("\*-----------------------------------------\*\n");

exit(0);

}

void signal\_handler(int nsig)

{

if (nsig == SIGINT)

{

semctl(sem\_id, 0, IPC\_RMID, 0);

exit(0);

}

}

void sem\_set\_state(unsigned short sem\_num, short state)

{

struct sembuf op;

op.sem\_op = state;

op.sem\_flg = 0;

op.sem\_num = sem\_num;

semop(sem\_id, &op, 1);

}

char is\_player\_input\_correct\_ship\_struct(ship\* shp)

{

if (shp->pos\_x < 0 || shp->pos\_x > 9)

return 0;

if (shp->pos\_y < 0 || shp->pos\_y > 9)

return 0;

if (shp->ship\_dir != SHIP\_DIRECTION\_VERTICAL && shp->ship\_dir != SHIP\_DIRECTION\_HORIZONTAL)

return 0;

return 1;

}

char is\_player\_input\_correct\_shot\_struct(client\_shot\* shot)

{

if (shot->pos\_x < 0 || shot->pos\_x > 9)

return 0;

if (shot->pos\_y < 0 || shot->pos\_y > 9)

return 0;

return 1;

}

/\* ========================================================================= \*/

/\* ========================================================================= \*/

/\* ========================================================================= \*/

void\* player\_pthread(void\* player\_struct)

{

signal(SIGINT, signal\_handler);

thread\_player\_info\* player\_info = (thread\_player\_info\*) player\_struct;

server\_answer serv\_answer;

char ship\_place\_mod = 0;

if (read(player\_info->tcp\_socket\_fd, &ship\_place\_mod, sizeof(ship\_place\_mod)) < 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (!ship\_place\_mod)

{

char placed\_ships\_count = 0;

while (placed\_ships\_count < SHIPS\_COUNT)

{

ship ship\_place;

serv\_answer = SERVER\_ERROR;

if (read(player\_info->tcp\_socket\_fd, &ship\_place, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (!is\_player\_input\_correct\_ship\_struct(&ship\_place))

{

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

continue;

}

if (!game\_field\_is\_can\_place\_ship(player\_info->own\_field, &ship\_place))

{

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

continue;

}

serv\_answer = SERVER\_OK;

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

game\_field\_place\_ship\_to\_field(player\_info->own\_field, &ship\_place);

ship\_struct\_assignment\_operation(&player\_info->own\_field->ships\_info[placed\_ships\_count], &ship\_place);

placed\_ships\_count++;

}

}

else

{

char placed\_ships\_count = 0;

while (placed\_ships\_count < SHIPS\_COUNT)

{

ship ship\_place;

serv\_answer = SERVER\_ERROR;

if (read(player\_info->tcp\_socket\_fd, &ship\_place, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (!is\_player\_input\_correct\_ship\_struct(&ship\_place))

{

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

continue;

}

if (!game\_field\_is\_can\_place\_ship(player\_info->own\_field, &ship\_place))

{

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

continue;

}

serv\_answer = SERVER\_OK;

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

game\_field\_place\_ship\_to\_field(player\_info->own\_field, &ship\_place);

ship\_struct\_assignment\_operation(&player\_info->own\_field->ships\_info[placed\_ships\_count], &ship\_place);

placed\_ships\_count++;

}

}

if (read(player\_info->tcp\_socket\_fd, player\_info->own\_state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

if (player\_info->player\_num == FIRST\_PLAYER\_NUM)

{

if (get\_random\_char\_from\_range(0, 1))

{

(\*player\_info->own\_state) = CLIENT\_STATE\_MAKE\_TURN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_WAIT\_TURN;

}

else

{

(\*player\_info->own\_state) = CLIENT\_STATE\_WAIT\_TURN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_MAKE\_TURN;

}

}

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

if (write(player\_info->tcp\_socket\_fd, player\_info->own\_state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

while (1)

{

client\_shot shot;

if (player\_info->player\_num == FIRST\_PLAYER\_NUM)

native\_renderer\_update(player\_info->own\_field, player\_info->enemy\_field);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_MAKE\_TURN)

{

serv\_answer = SERVER\_ERROR;

while (serv\_answer != SERVER\_OK)

{

if (read(player\_info->tcp\_socket\_fd, &shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (!is\_player\_input\_correct\_shot\_struct(&shot))

{

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

continue;

}

serv\_answer = SERVER\_OK;

if (write(player\_info->tcp\_socket\_fd, &serv\_answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

}

shot.state = game\_field\_get\_hit\_result(player\_info->enemy\_field, &shot);

shot\_struct\_assignment\_operation(&global\_shot, &shot);

if (shot.state == CELL\_STATE\_SHIP\_HIT)

{

char ship\_num = game\_field\_get\_hit\_ship\_num\_by\_coord(player\_info->enemy\_field->ships\_info, &shot);

if (game\_field\_check\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]))

{

global\_shot.state = CELL\_STATE\_SHIP\_DESTROY;

ship\_struct\_assignment\_operation(&ship\_destroyed, &player\_info->enemy\_field->ships\_info[ship\_num]);

game\_field\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]);

if (write(player\_info->tcp\_socket\_fd, &global\_shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (write(player\_info->tcp\_socket\_fd, &player\_info->enemy\_field->ships\_info[ship\_num], sizeof(ship)) < 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

}

else

{

if (write(player\_info->tcp\_socket\_fd, &global\_shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

}

}

else

{

if (write(player\_info->tcp\_socket\_fd, &global\_shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

}

if (shot.state == CELL\_STATE\_HIT)

{

\*(player\_info->own\_state) = CLIENT\_STATE\_WAIT\_TURN;

\*(player\_info->enemy\_state) = CLIENT\_STATE\_MAKE\_TURN;

}

else

{

player\_info->enemy\_field->ships\_cells\_count--;

if (player\_info->enemy\_field->ships\_cells\_count == 0)

{

\*(player\_info->own\_state) = CLIENT\_STATE\_GAME\_WIN;

\*(player\_info->enemy\_state) = CLIENT\_STATE\_GAME\_LOSE;

}

}

if (write(player\_info->tcp\_socket\_fd, player\_info->own\_state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_WIN)

break;

}

else

{

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

if (write(player\_info->tcp\_socket\_fd, &global\_shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (global\_shot.state == CELL\_STATE\_SHIP\_DESTROY)

if (write(player\_info->tcp\_socket\_fd, &ship\_destroyed, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

if (write(player\_info->tcp\_socket\_fd, player\_info->own\_state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(player\_info->tcp\_socket\_fd, player\_info->player\_num);

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_LOSE)

break;

}

}

return 0;

}

/\* ========================================================================= \*/

/\* ========================================================================= \*/

/\* ========================================================================= \*/

void print\_server\_port(const u\_int16\_t tcp\_socket\_fd)

{

struct sockaddr\_in socket\_address;

socklen\_t socket\_len = sizeof(socket\_address);

getsockname(tcp\_socket\_fd, (struct sockaddr\*) &socket\_address, &socket\_len);

printf("Server started in %i port\n", ntohs(socket\_address.sin\_port));

}

int main(int argc, char\* argv[])

{

signal(SIGINT, signal\_handler);

/\* ===================================== \*/

/\* ======== ARGUMENT PROCESSING ======== \*/

/\* ===================================== \*/

uint16\_t server\_port;

if (argc == 1)

server\_port = 0;

if (argc == 2)

{

if (strcmp(argv[1], "help") == 0)

print\_use\_help\_and\_exit();

server\_port = atoi(argv[1]);

}

else

print\_use\_help\_and\_exit();

/\* ===================================== \*/

/\* ======= SOCKET INITIALIZATION ======= \*/

/\* ===================================== \*/

uint32\_t received\_bytes = 0;

u\_int16\_t tcp\_socket\_fd;

u\_int16\_t tcp\_first\_player\_socket\_fd;

u\_int16\_t tcp\_second\_player\_socket\_fd;

socket\_address\_in server\_address;

socket\_address\_in client\_address;

client\_states first\_client\_state;

client\_states second\_client\_state;

/\* ============================= \*/

/\* === Trying to init socket === \*/

if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror(NULL);

return -1;

}

/\* ============================= \*/

/\* == Set up server variables == \*/

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(server\_port);

server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)

{

printf("Error! Cant get %i port! Trying to get another!\n", server\_port);

server\_address.sin\_port = htons(0);

if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

}

print\_server\_port(tcp\_socket\_fd);

/\* ============================= \*/

/\* === Set number of listens === \*/

if (listen(tcp\_socket\_fd, 1) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

/\* ===================================== \*/

/\* ====== CLIENTS INITIALIZATION ======= \*/

/\* ===================================== \*/

/\* ============================= \*/

/\* ======== FIRST PLAYER ======= \*/

tcp\_first\_player\_socket\_fd = accept(tcp\_socket\_fd, (socket\_address\*) &client\_address, &received\_bytes);

if (tcp\_first\_player\_socket\_fd < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

first\_client\_state = CLIENT\_STATE\_WAITING\_ANOTHER\_PLAYER;

if (write(tcp\_first\_player\_socket\_fd, &first\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

printf("First player is successfully connected!\n");

printf("Waiting second player...\n");

/\* ============================= \*/

/\* ======= SECOND PLAYER ======= \*/

tcp\_second\_player\_socket\_fd = accept(tcp\_socket\_fd, (socket\_address\*) &client\_address, &received\_bytes);

if (tcp\_second\_player\_socket\_fd < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

first\_client\_state = CLIENT\_STATE\_PLACING\_SHIPS;

second\_client\_state = CLIENT\_STATE\_PLACING\_SHIPS;

if (write(tcp\_first\_player\_socket\_fd, &first\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(tcp\_second\_player\_socket\_fd, &second\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

/\* ===================================== \*/

/\* ======== GAME VARIABLES INIT ======== \*/

/\* ===================================== \*/

sem\_id = semget(IPC\_PRIVATE, 2, 0600 | IPC\_CREAT);

game\_field\* first\_player\_field = game\_field\_create(CREATE\_WITH\_SHIPS\_ARRAY);

game\_field\* second\_player\_field = game\_field\_create(CREATE\_WITH\_SHIPS\_ARRAY);

first\_player\_field->ships\_cells\_count = SHIPS\_CELLS\_COUNT;

second\_player\_field->ships\_cells\_count = SHIPS\_CELLS\_COUNT;

thread\_player\_info first\_player\_thread\_info;

thread\_player\_info second\_player\_thread\_info;

/\* ============================== \*/

/\* == FIRST PLAYER STRUCT INIT == \*/

first\_player\_thread\_info.player\_num = FIRST\_PLAYER\_NUM;

first\_player\_thread\_info.tcp\_socket\_fd = tcp\_first\_player\_socket\_fd;

first\_player\_thread\_info.own\_field = first\_player\_field;

first\_player\_thread\_info.enemy\_field = second\_player\_field;

first\_player\_thread\_info.own\_state = &first\_client\_state;

first\_player\_thread\_info.enemy\_state = &second\_client\_state;

/\* ============================= \*/

/\* = SECOND PLAYER STRUCT INIT = \*/

second\_player\_thread\_info.player\_num = SECOND\_PLAYER\_NUM;

second\_player\_thread\_info.tcp\_socket\_fd = tcp\_second\_player\_socket\_fd;

second\_player\_thread\_info.own\_field = second\_player\_field;

second\_player\_thread\_info.enemy\_field = first\_player\_field;

second\_player\_thread\_info.own\_state = &second\_client\_state;

second\_player\_thread\_info.enemy\_state = &first\_client\_state;

/\* ===================================== \*/

/\* ======= THREADS START ROUTINE ======= \*/

/\* ===================================== \*/

int first\_thread\_return\_value;

int second\_thread\_return\_value;

pthread\_t first\_player\_thread;

pthread\_t second\_player\_thread;

int thread\_result1 = pthread\_create(&first\_player\_thread, NULL, player\_pthread, &first\_player\_thread\_info);

int thread\_result2 = pthread\_create(&second\_player\_thread, NULL, player\_pthread, &second\_player\_thread\_info);

pthread\_join(first\_player\_thread, (void\*\*) &first\_thread\_return\_value);

pthread\_join(second\_player\_thread, (void\*\*) &second\_thread\_return\_value);

native\_renderer\_clear();

native\_renderer\_update(first\_player\_field, second\_player\_field);

printf("First thread exit with status: %i\n", first\_thread\_return\_value);

printf("Second thread exit with status: %i\n", second\_thread\_return\_value);

printf("Server shutting down...\n");

first\_player\_field = game\_field\_destroy(first\_player\_field);

second\_player\_field = game\_field\_destroy(second\_player\_field);

return 0;

}

# ЛИСТИНГ CLIENT.C

#include "time.h"

#include "stdio.h"

#include "errno.h"

#include "unistd.h"

#include "stdlib.h"

#include "string.h"

#include "signal.h"

#include "strings.h"

#include "sys/sem.h"

#include "sys/ipc.h"

#include "pthread.h"

#include "arpa/inet.h"

#include "sys/types.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "field.h"

#include "define.h"

#include "utility.h"

#include "computer.h"

#include "renderer.h"

#define SELF\_SHIPS 0

#define RANDOM\_SHIPS 1

#define FILE\_SHIPS 2

int sem\_id;

char ship\_place\_mod;

char computer\_array\_set;

ship ship\_destroyed;

client\_shot global\_shot;

void signal\_handler(int nsig)

{

if (nsig == SIGINT)

{

ncurses\_renderer\_destroy();

semctl(sem\_id, 0, IPC\_RMID, 0);

exit(0);

}

}

typedef struct thread\_player\_info\_struct

{

char player\_num;

game\_field\* own\_field;

game\_field\* enemy\_field;

client\_states\* own\_state;

client\_states\* enemy\_state;

} thread\_player\_info;

void sem\_set\_state(unsigned short sem\_num, short state)

{

struct sembuf op;

op.sem\_op = state;

op.sem\_flg = 0;

op.sem\_num = sem\_num;

semop(sem\_id, &op, 1);

}

void print\_programm\_use\_and\_exit()

{

printf("Use: ./client <args>\n");

printf("Gamemodes options:\n");

printf("\t<nothing> -- player versus computer gamemode\n");

printf("\t--computers -- computer versus computer gamemode\n");

printf("\t--connect <ip-address> <port> -- player versus player gamemode\n\n");

printf("Algorithms options:\n");

printf("\t--alg <algorithm name>\n");

printf("\t<nothing> -- set random algoritm by default\n");

printf("\trandom -- set by default\n");

printf("\tsteps -- to set colm steps algorithm\n");

printf("\tgarant -- to set garant algorithm\n");

printf("\tchess -- to set chess-board algorithm\n\n");

printf("Ship placed options:\n");

printf("\t<nothing> -- self input place\n");

printf("\t--random -- place random ships to the field\n");

printf("\t--file <file\_name> -- place ships from file\n");

exit(1);

}

char select\_next\_ship(int\* placed\_ships\_category\_count, ship\* shp)

{

\*placed\_ships\_category\_count = \*placed\_ships\_category\_count + 1;

switch (shp->ship\_ctg)

{

case BATTLESHIP\_CATEGORY:

shp->ship\_ctg = CRUISER\_CATEGORY;

\*placed\_ships\_category\_count = 0;

break;

case CRUISER\_CATEGORY:

if (\*placed\_ships\_category\_count == CRUISER\_SHIP\_COUNT)

{

shp->ship\_ctg = DESTROYER\_CATEGORY;

\*placed\_ships\_category\_count = 0;

}

break;

case DESTROYER\_CATEGORY:

if (\*placed\_ships\_category\_count == DESTROYER\_SHIP\_COUNT)

{

shp->ship\_ctg = TORPEDO\_BOAT\_CATEGORY;

\*placed\_ships\_category\_count = 0;

}

break;

case TORPEDO\_BOAT\_CATEGORY:

if (\*placed\_ships\_category\_count == TORPEDO\_SHIP\_COUNT)

{

\*placed\_ships\_category\_count = 0;

shp->ship\_ctg = BATTLESHIP\_CATEGORY;

return 1;

}

break;

default:

shp->ship\_ctg = BATTLESHIP\_CATEGORY;

break;

}

return 0;

}

void get\_random\_place\_ship(ship\* shp)

{

shp->pos\_x = get\_random\_char\_from\_range(0, 9);

shp->pos\_y = get\_random\_char\_from\_range(0, 9);

shp->ship\_dir = (get\_random\_char\_from\_range(0, 1)) ? SHIP\_DIRECTION\_VERTICAL : SHIP\_DIRECTION\_HORIZONTAL;

}

void place\_random\_ships\_to\_field(game\_field\* field)

{

ship ship\_to\_place;

char is\_all\_ship\_placed = 0;

int placed\_ships\_count = 0;

int placed\_ships\_category\_count = 0;

ship\_to\_place.ship\_ctg = BATTLESHIP\_CATEGORY;

while (!is\_all\_ship\_placed)

{

get\_random\_place\_ship(&ship\_to\_place);

if (!game\_field\_is\_can\_place\_ship(field, &ship\_to\_place))

continue;

game\_field\_place\_ship\_to\_field(field, &ship\_to\_place);

ship\_struct\_assignment\_operation(&field->ships\_info[placed\_ships\_count], &ship\_to\_place);

is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &ship\_to\_place);

placed\_ships\_count++;

}

}

char get\_next\_char\_from\_file(FILE\* file)

{

int ch;

while (ch != EOF)

{

ch = fgetc(file);

if (ch == '\r' || ch == '\n' || ch == ' ')

continue;

else

break;

}

if (ch == EOF)

{

printf("Error! Bad input from file! Fix the file and try again!\n");

exit(1);

}

return ch;

}

int read\_ships\_positions\_from\_file(game\_field\* field, const char\* file\_name)

{

FILE\* file = fopen(file\_name, "r");

if (!file)

return 0;

ship placed\_ship;

int placed\_ships\_count = 0;

int placed\_ships\_category\_count = 0;

char is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &placed\_ship);

while (!is\_all\_ship\_placed)

{

placed\_ship.pos\_x = get\_next\_char\_from\_file(file) - '0';

placed\_ship.pos\_y = get\_next\_char\_from\_file(file) - 'a';

placed\_ship.ship\_dir = get\_next\_char\_from\_file(file);

if (!is\_coordinates\_input\_correct(placed\_ship.pos\_x, placed\_ship.pos\_y))

{

printf("Incorrect coordinates in file! Line %i!\n", placed\_ships\_count + 1);

return 0;

}

if (!is\_ship\_direction\_input\_correct(placed\_ship.ship\_dir))

{

printf("Incorrect ship direction in file! Line %i!\n", placed\_ships\_count + 1);

return 0;

}

if (!game\_field\_is\_can\_place\_ship(field, &placed\_ship))

{

printf("Cant place this ship from file! Line %i!\n", placed\_ships\_count + 1);

return 0;

}

game\_field\_place\_ship\_to\_field(field, &placed\_ship);

ship\_struct\_assignment\_operation(&field->ships\_info[placed\_ships\_count], &placed\_ship);

is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &placed\_ship);

placed\_ships\_count++;

}

fclose(file);

return 1;

}

void clear\_bad\_ship\_from\_fields(game\_field\* field, ship\* shp)

{

for (int i = 0; i < shp->ship\_ctg; i++)

{

if (shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL)

field->field[shp->pos\_x][shp->pos\_y + i] = CELL\_STATE\_VOID;

else

field->field[shp->pos\_x + i][shp->pos\_y] = CELL\_STATE\_VOID;

}

}

void\* computer\_player\_pthread\_function(void\* thread\_arguments)

{

thread\_player\_info\* player\_info = (thread\_player\_info\*) thread\_arguments;

int random\_counter = 0;

char ship\_was\_hit = 0;

char need\_regen\_array = 0;

char direction\_is\_right = 0;

char current\_shot\_from\_array = 0;

computer\_shot first\_hit\_shot;

computer\_shot shots\_array[SHOT\_ARRAY\_SIZE];

hit\_direction hit\_dir = HIT\_DIRECTION\_UNDERFINED;

ship\_category higher\_tier\_ship\_stand = BATTLESHIP\_CATEGORY;

place\_random\_ships\_to\_field(player\_info->own\_field);

computer\_gen\_new\_shots(player\_info->enemy\_field, shots\_array, 0);

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

while (!is\_game\_end((\*player\_info->own\_state)))

{

client\_shot shot;

if ((\*player\_info->own\_state) == CLIENT\_STATE\_WAIT\_TURN)

{

if (need\_regen\_array || current\_shot\_from\_array >= SHOT\_ARRAY\_SIZE - 1)

{

char gen\_shots\_count = 0;

if (ship\_was\_hit)

{

shots\_array[gen\_shots\_count].pos\_x = first\_hit\_shot.pos\_x;

shots\_array[gen\_shots\_count].pos\_y = first\_hit\_shot.pos\_y;

if (hit\_dir == HIT\_DIRECTION\_UNDERFINED)

hit\_dir = computer\_get\_new\_hit\_direction(player\_info->enemy\_field, &first\_hit\_shot);

for (int i = 0; i < higher\_tier\_ship\_stand; i++)

{

if (computer\_get\_next\_shot\_by\_direction(&shots\_array[gen\_shots\_count], hit\_dir))

continue;

shots\_array[gen\_shots\_count + 1].pos\_x = shots\_array[gen\_shots\_count].pos\_x;

shots\_array[gen\_shots\_count + 1].pos\_y = shots\_array[gen\_shots\_count].pos\_y;

gen\_shots\_count++;

}

}

computer\_gen\_new\_shots(player\_info->enemy\_field, shots\_array, gen\_shots\_count);

need\_regen\_array = 0;

current\_shot\_from\_array = 0;

}

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

game\_field\_set\_shot(player\_info->own\_field, &global\_shot);

if (global\_shot.state == CELL\_STATE\_SHIP\_DESTROY)

game\_field\_ship\_destroy(player\_info->own\_field, &ship\_destroyed);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_LOSE)

break;

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

}

else

{

if (need\_regen\_array && ship\_was\_hit)

{

if (hit\_dir == HIT\_DIRECTION\_UNDERFINED)

hit\_dir = computer\_get\_new\_hit\_direction(player\_info->enemy\_field, &first\_hit\_shot);

computer\_shot temp;

temp.pos\_x = shot.pos\_x;

temp.pos\_y = shot.pos\_y;

if (computer\_get\_next\_shot\_by\_direction(&temp, hit\_dir))

{

shot.pos\_x = first\_hit\_shot.pos\_x;

shot.pos\_y = first\_hit\_shot.pos\_y;

hit\_dir = computer\_get\_opposite\_hit\_direction(hit\_dir);

computer\_get\_next\_shot\_by\_direction(&temp, hit\_dir);

}

shot.pos\_x = temp.pos\_x;

shot.pos\_y = temp.pos\_y;

}

else

{

if (current\_shot\_from\_array >= SHOT\_ARRAY\_SIZE - 1)

{

shot.pos\_x = get\_random\_char\_from\_range(0, 9);

shot.pos\_y = get\_random\_char\_from\_range(0, 9);

need\_regen\_array = 1;

if (!game\_field\_check\_coordinates\_to\_hit(player\_info->enemy\_field, shot.pos\_x, shot.pos\_y))

random\_counter++;

}

else

{

shot.pos\_x = shots\_array[current\_shot\_from\_array].pos\_x;

shot.pos\_y = shots\_array[current\_shot\_from\_array].pos\_y;

current\_shot\_from\_array++;

}

}

if (game\_field\_check\_coordinates\_to\_hit(player\_info->enemy\_field, shot.pos\_x, shot.pos\_y))

continue;

shot.state = game\_field\_get\_hit\_result(player\_info->enemy\_field, &shot);

game\_field\_set\_shot(player\_info->enemy\_field, &shot);

if (shot.state == CELL\_STATE\_SHIP\_HIT)

{

if (!ship\_was\_hit)

{

ship\_was\_hit = 1;

first\_hit\_shot.pos\_x = shot.pos\_x;

first\_hit\_shot.pos\_y = shot.pos\_y;

need\_regen\_array = 1;

hit\_dir = HIT\_DIRECTION\_UNDERFINED;

}

else

direction\_is\_right = 1;

char ship\_num = game\_field\_get\_hit\_ship\_num\_by\_coord(player\_info->enemy\_field->ships\_info, &shot);

if (game\_field\_check\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]))

{

if (ship\_was\_hit)

current\_shot\_from\_array = (int) higher\_tier\_ship\_stand + 1;

ship\_was\_hit = 0;

need\_regen\_array = 1;

direction\_is\_right = 0;

hit\_dir = HIT\_DIRECTION\_UNDERFINED;

shot.state = CELL\_STATE\_SHIP\_DESTROY;

higher\_tier\_ship\_stand = computer\_update\_higher\_ship\_tier\_stand(higher\_tier\_ship\_stand, player\_info->enemy\_field->ships\_info[ship\_num].ship\_ctg);

ship\_struct\_assignment\_operation(&ship\_destroyed, &player\_info->enemy\_field->ships\_info[ship\_num]);

game\_field\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]);

}

}

shot\_struct\_assignment\_operation(&global\_shot, &shot);

if (shot.state == CELL\_STATE\_HIT)

{

if (ship\_was\_hit)

{

if (!direction\_is\_right)

hit\_dir = HIT\_DIRECTION\_UNDERFINED;

else

hit\_dir = computer\_get\_opposite\_hit\_direction(hit\_dir);

need\_regen\_array = 1;

shot.pos\_x = first\_hit\_shot.pos\_x;

shot.pos\_y = first\_hit\_shot.pos\_y;

}

(\*player\_info->own\_state) = CLIENT\_STATE\_WAIT\_TURN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_MAKE\_TURN;

}

else

{

player\_info->enemy\_field->ships\_cells\_count--;

if (player\_info->enemy\_field->ships\_cells\_count == 0)

{

(\*player\_info->own\_state) = CLIENT\_STATE\_GAME\_WIN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_GAME\_LOSE;

}

}

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_WIN)

break;

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

}

}

printf("(Player %i) Random shots: %i\n", (player\_info->player\_num + 1), random\_counter);

return 0;

}

void\* player\_pthread\_function(void\* thread\_arguments)

{

thread\_player\_info\* player\_info = (thread\_player\_info\*) thread\_arguments;

game\_field\* computer\_field = game\_field\_create(CREATE\_WITHOUT\_SHIPS\_ARRAY);

ncurses\_renderer\_init();

if (!ship\_place\_mod)

{

ship place\_msg;

int placed\_ships\_count = 0;

int placed\_ships\_category\_count = 0;

char is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &place\_msg);

while (!is\_all\_ship\_placed)

{

ncurses\_render\_update(player\_info->own\_field, computer\_field, CLIENT\_STATE\_PLACING\_SHIPS);

place\_msg.pos\_x = ncurses\_get\_user\_input(0) - '0';

place\_msg.pos\_y = ncurses\_get\_user\_input(1) - 'a';

if (!is\_coordinates\_input\_correct(place\_msg.pos\_x, place\_msg.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

place\_msg.ship\_dir = (ship\_direction) ncurses\_get\_user\_input(2);

if (!is\_ship\_direction\_input\_correct(place\_msg.ship\_dir))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

if (!game\_field\_is\_can\_place\_ship(player\_info->own\_field, &place\_msg))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

game\_field\_place\_ship\_to\_field(player\_info->own\_field, &place\_msg);

ship\_struct\_assignment\_operation(&player\_info->own\_field->ships\_info[placed\_ships\_count], &place\_msg);

is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &place\_msg);

placed\_ships\_count++;

}

}

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

while (!is\_game\_end((\*player\_info->own\_state)))

{

client\_shot shot;

ncurses\_render\_update(player\_info->own\_field, computer\_field, -1);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_WAIT\_TURN)

{

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

game\_field\_set\_shot(player\_info->own\_field, &global\_shot);

if (global\_shot.state == CELL\_STATE\_SHIP\_DESTROY)

game\_field\_ship\_destroy(player\_info->own\_field, &ship\_destroyed);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_LOSE)

break;

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

}

else

{

if (player\_info->own\_field->ships\_cells\_count == 0)

break;

shot.pos\_x = ncurses\_get\_user\_input(0) - '0';

shot.pos\_y = ncurses\_get\_user\_input(1) - 'a';

if (!is\_coordinates\_input\_correct(shot.pos\_x, shot.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

if (game\_field\_check\_coordinates\_to\_hit(player\_info->enemy\_field, shot.pos\_x, shot.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

shot.state = game\_field\_get\_hit\_result(player\_info->enemy\_field, &shot);

game\_field\_set\_shot(computer\_field, &shot);

if (shot.state == CELL\_STATE\_SHIP\_HIT)

{

char ship\_num = game\_field\_get\_hit\_ship\_num\_by\_coord(player\_info->enemy\_field->ships\_info, &shot);

if (game\_field\_check\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]))

{

shot.state = CELL\_STATE\_SHIP\_DESTROY;

ship\_struct\_assignment\_operation(&ship\_destroyed, &player\_info->enemy\_field->ships\_info[ship\_num]);

game\_field\_ship\_destroy(computer\_field, &player\_info->enemy\_field->ships\_info[ship\_num]);

game\_field\_ship\_destroy(player\_info->enemy\_field, &player\_info->enemy\_field->ships\_info[ship\_num]);

}

}

if (shot.state == CELL\_STATE\_HIT)

{

(\*player\_info->own\_state) = CLIENT\_STATE\_WAIT\_TURN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_MAKE\_TURN;

}

else

{

player\_info->enemy\_field->ships\_cells\_count--;

if (player\_info->enemy\_field->ships\_cells\_count == 0)

{

(\*player\_info->own\_state) = CLIENT\_STATE\_GAME\_WIN;

(\*player\_info->enemy\_state) = CLIENT\_STATE\_GAME\_LOSE;

}

}

shot\_struct\_assignment\_operation(&global\_shot, &shot);

sem\_set\_state(get\_enemy\_num(player\_info->player\_num), SEMAPHORE\_GIVE\_ONE);

if ((\*player\_info->own\_state) == CLIENT\_STATE\_GAME\_WIN)

break;

sem\_set\_state(player\_info->player\_num, SEMAPHORE\_REDUCE\_ONE);

}

}

ncurses\_renderer\_destroy();

return 0;

}

void player\_versus\_player(char\* server\_ip, int server\_port, char\* file\_name)

{

game\_field\* own\_field = game\_field\_create(CREATE\_WITH\_SHIPS\_ARRAY);

game\_field\* enemy\_field = game\_field\_create(CREATE\_WITHOUT\_SHIPS\_ARRAY);

if (ship\_place\_mod == RANDOM\_SHIPS)

place\_random\_ships\_to\_field(own\_field);

else if (ship\_place\_mod == FILE\_SHIPS)

{

if (!read\_ships\_positions\_from\_file(own\_field, file\_name))

{

printf("Error! Input file with ships coord is incorrect! Fix the file and try again!\n");

exit(1);

}

}

u\_int16\_t tcp\_socket\_fd = 0;

client\_states state;

server\_answer answer;

socket\_address\_in server\_address;

socket\_address\_in client\_address;

bzero(&server\_address, sizeof(server\_address));

bzero(&client\_address, sizeof(client\_address));

if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror(NULL);

exit(1);

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(server\_port);

if (inet\_aton(server\_ip, &server\_address.sin\_addr) == 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (connect(tcp\_socket\_fd, (socket\_address\*) &server\_address, sizeof(server\_address)) < 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

// Get client state from server

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

// Check is all two players ready to game

if (state == CLIENT\_STATE\_WAITING\_ANOTHER\_PLAYER)

{

printf("Waiting another player...\n");

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

}

if (state == CLIENT\_STATE\_PLACING\_SHIPS)

{

printf("Ready to play! Press any button!\n");

getchar();

fflush(stdin);

}

if (write(tcp\_socket\_fd, &ship\_place\_mod, sizeof(ship\_place\_mod)) < 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

// Game variables

ncurses\_renderer\_init();

ncurses\_render\_update(own\_field, enemy\_field, state);

if (!ship\_place\_mod)

{

ship place\_msg;

int placed\_ships\_category\_count = 0;

char is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &place\_msg);

while (!is\_all\_ship\_placed)

{

ncurses\_render\_update(own\_field, enemy\_field, state);

place\_msg.pos\_x = ncurses\_get\_user\_input(0) - '0';

place\_msg.pos\_y = ncurses\_get\_user\_input(1) - 'a';

if (!is\_coordinates\_input\_correct(place\_msg.pos\_x, place\_msg.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

place\_msg.ship\_dir = (ship\_direction) ncurses\_get\_user\_input(2);

if (!is\_ship\_direction\_input\_correct(place\_msg.ship\_dir))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

if (!game\_field\_is\_can\_place\_ship(own\_field, &place\_msg))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

if (write(tcp\_socket\_fd, &place\_msg, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (read(tcp\_socket\_fd, &answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (answer == SERVER\_ERROR)

continue;

game\_field\_place\_ship\_to\_field(own\_field, &place\_msg);

is\_all\_ship\_placed = select\_next\_ship(&placed\_ships\_category\_count, &place\_msg);

}

}

else

{

char placed\_ships\_count = 0;

while (placed\_ships\_count < SHIPS\_COUNT)

{

if (write(tcp\_socket\_fd, &own\_field->ships\_info[placed\_ships\_count], sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (read(tcp\_socket\_fd, &answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (answer == SERVER\_ERROR)

{

ship new\_ship;

new\_ship.ship\_ctg = own\_field->ships\_info[placed\_ships\_count].ship\_ctg;

clear\_bad\_ship\_from\_fields(own\_field, &own\_field->ships\_info[placed\_ships\_count]);

do

{

get\_random\_place\_ship(&new\_ship);

} while (!game\_field\_is\_can\_place\_ship(own\_field, &new\_ship));

game\_field\_place\_ship\_to\_field(own\_field, &new\_ship);

continue;

}

placed\_ships\_count++;

}

}

// Send that you are placed all ships and ready to battle

state = CLIENT\_STATE\_WAITING\_ANOTHER\_PLAYER;

ncurses\_render\_update(own\_field, enemy\_field, state);

if (write(tcp\_socket\_fd, &state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

// Receive from server a player turn

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

// Main cycle

while (1)

{

client\_shot shot;

ncurses\_render\_update(own\_field, enemy\_field, state);

if (state == CLIENT\_STATE\_WAIT\_TURN)

{

// Get enemy shot;

if (read(tcp\_socket\_fd, &shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

game\_field\_set\_shot(own\_field, &shot);

if (shot.state == CELL\_STATE\_SHIP\_DESTROY)

{

ship destroyed\_ship\_info;

if (read(tcp\_socket\_fd, &destroyed\_ship\_info, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

game\_field\_ship\_destroy(own\_field, &destroyed\_ship\_info);

}

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (state == CLIENT\_STATE\_GAME\_LOSE)

break;

}

else

{

shot.pos\_x = ncurses\_get\_user\_input(0) - '0';

shot.pos\_y = ncurses\_get\_user\_input(1) - 'a';

if (!is\_coordinates\_input\_correct(shot.pos\_x, shot.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

if (game\_field\_check\_coordinates\_to\_hit(enemy\_field, shot.pos\_x, shot.pos\_y))

{

ncurses\_incorrect\_coord\_input\_msg();

continue;

}

// There you send information to the server about your turn

if (write(tcp\_socket\_fd, &shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (read(tcp\_socket\_fd, &answer, sizeof(server\_answer)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (answer == SERVER\_ERROR)

continue;

// Read server info about your hit

if (read(tcp\_socket\_fd, &shot, sizeof(client\_shot)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

game\_field\_set\_shot(enemy\_field, &shot);

if (shot.state == CELL\_STATE\_SHIP\_DESTROY)

{

ship destroyed\_ship\_info;

if (read(tcp\_socket\_fd, &destroyed\_ship\_info, sizeof(ship)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

game\_field\_ship\_destroy(enemy\_field, &destroyed\_ship\_info);

}

// Read server info about your state

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) <= 0)

throw\_exeption\_and\_exit(tcp\_socket\_fd, 0);

if (state == CLIENT\_STATE\_GAME\_WIN)

break;

}

}

ncurses\_renderer\_destroy();

if (state == CLIENT\_STATE\_GAME\_WIN)

printf("You are win the game! :)\n");

else

printf("You are lose :(\n");

own\_field = game\_field\_destroy(own\_field);

enemy\_field = game\_field\_destroy(enemy\_field);

close(tcp\_socket\_fd);

exit(0);

}

int main(int argc, char\* argv[])

{

srand(time(NULL));

signal(SIGINT, signal\_handler);

char ip\_index\_from\_argv = 0;

char port\_index\_from\_argv = 0;

char file\_name\_index\_from\_argv = 0;

void\* (\*first\_player\_function\_ptr)(void\* thread\_arguments);

void\* (\*second\_player\_function\_ptr)(void\* thread\_arguments);

first\_player\_function\_ptr = &player\_pthread\_function;

second\_player\_function\_ptr = &computer\_player\_pthread\_function;

for (int i = 1; i < argc; i++)

{

if (strcmp(argv[i], "--connect") == 0)

{

if (i + 2 > argc)

{

printf("Not enough connect arguments! Required: --connect <ip-address> <port>\n");

return 0;

}

if (first\_player\_function\_ptr != computer\_player\_pthread\_function)

{

ip\_index\_from\_argv = i + 1;

port\_index\_from\_argv = i + 2;

}

i += 2;

}

else if (strcmp(argv[i], "--computers") == 0)

{

if (ip\_index\_from\_argv == 0)

first\_player\_function\_ptr = &computer\_player\_pthread\_function;

}

else if (strcmp(argv[i], "--alg") == 0)

{

if (i + 1 > argc)

{

printf("Not enough algorithm arguments! Required: --alg <algorithm>\n");

return 0;

}

i++;

if (strcmp(argv[i], "random") == 0)

computer\_array\_set = RANDOM\_ARRAY\_SET;

else if (strcmp(argv[i], "steps") == 0)

computer\_array\_set = SHOTS\_STEPS\_SET;

else if (strcmp(argv[i], "garant") == 0)

computer\_array\_set = GARANT\_NON\_RANDOM\_SET;

else if (strcmp(argv[i], "chess") == 0)

computer\_array\_set = CHESS\_ARRAY\_SET;

else

{

printf("Unknown algorithm! User --help to see avaible algoritms\n");

return 0;

}

}

else if (strcmp(argv[i], "--random") == 0)

{

if (ship\_place\_mod != FILE\_SHIPS)

ship\_place\_mod = RANDOM\_SHIPS;

}

else if (strcmp(argv[i], "--file") == 0)

{

if (i + 1 > argc)

{

printf("Not enough arguments! Required: --file <file\_name>\n");

return 0;

}

i++;

if (ship\_place\_mod != RANDOM\_SHIPS)

{

ship\_place\_mod = FILE\_SHIPS;

file\_name\_index\_from\_argv = i;

}

}

else if (strcmp(argv[i], "--help") == 0)

print\_programm\_use\_and\_exit();

else

{

printf("Unknown command! Use --help to see arguments list!\n");

return 0;

}

}

/\* \*/

if (ip\_index\_from\_argv != 0)

player\_versus\_player(argv[ip\_index\_from\_argv], atoi(argv[port\_index\_from\_argv]), argv[file\_name\_index\_from\_argv]);

/\* \*/

/\* \*/

game\_field\* first\_field = game\_field\_create(CREATE\_WITH\_SHIPS\_ARRAY);

game\_field\* second\_field = game\_field\_create(CREATE\_WITH\_SHIPS\_ARRAY);

client\_states first\_player\_state;

client\_states second\_player\_state;

thread\_player\_info first\_player\_thread\_info;

thread\_player\_info second\_player\_thread\_info;

/\* \*/

/\* \*/

if (first\_player\_function\_ptr == player\_pthread\_function)

{

if (ship\_place\_mod == RANDOM\_SHIPS)

place\_random\_ships\_to\_field(first\_field);

else if (ship\_place\_mod == FILE\_SHIPS)

{

if (!read\_ships\_positions\_from\_file(first\_field, argv[file\_name\_index\_from\_argv]))

{

printf("Error! Input file with ships coord is incorrect! Fix the file and try again!\n");

return 0;

}

}

}

/\* \*/

/\* \*/

first\_player\_thread\_info.player\_num = FIRST\_PLAYER\_NUM;

first\_player\_thread\_info.own\_field = first\_field;

first\_player\_thread\_info.enemy\_field = second\_field;

first\_player\_thread\_info.own\_state = &first\_player\_state;

first\_player\_thread\_info.enemy\_state = &second\_player\_state;

/\* \*/

/\* \*/

second\_player\_thread\_info.player\_num = SECOND\_PLAYER\_NUM;

second\_player\_thread\_info.own\_field = second\_field;

second\_player\_thread\_info.enemy\_field = first\_field;

second\_player\_thread\_info.own\_state = &second\_player\_state;

second\_player\_thread\_info.enemy\_state = &first\_player\_state;

/\* \*/

/\* \*/

if (get\_random\_char\_from\_range(0, 1))

{

first\_player\_state = CLIENT\_STATE\_MAKE\_TURN;

second\_player\_state = CLIENT\_STATE\_WAIT\_TURN;

}

else

{

first\_player\_state = CLIENT\_STATE\_WAIT\_TURN;

second\_player\_state = CLIENT\_STATE\_MAKE\_TURN;

}

/\* \*/

/\* ================================================= \*/

sem\_id = semget(IPC\_PRIVATE, 2, 0600 | IPC\_CREAT);

first\_field->ships\_cells\_count = SHIPS\_CELLS\_COUNT;

second\_field->ships\_cells\_count = SHIPS\_CELLS\_COUNT;

/\* ================================================= \*/

int first\_thread\_return\_value;

int second\_thread\_return\_value;

pthread\_t first\_player\_thread;

pthread\_t second\_player\_thread;

int thread\_result1 = pthread\_create(&first\_player\_thread, NULL, first\_player\_function\_ptr, &first\_player\_thread\_info);

int thread\_result2 = pthread\_create(&second\_player\_thread, NULL, second\_player\_function\_ptr, &second\_player\_thread\_info);

pthread\_join(first\_player\_thread, (void\*\*) &first\_thread\_return\_value);

pthread\_join(second\_player\_thread, (void\*\*) &second\_thread\_return\_value);

native\_renderer\_update(first\_field, second\_field);

printf("============== RESULT ==============\n");

if (first\_player\_state == CLIENT\_STATE\_GAME\_WIN)

{

printf("Winner: First player!\n");

printf("Looser: Second player!\n");

}

else

{

printf("Looser: First player!\n");

printf("Winner: Second player!\n");

}

printf("------------------------------------\n");

printf("First player ship cells count = %i\n", first\_field->ships\_cells\_count);

printf("Second player ship cells count = %i\n", second\_field->ships\_cells\_count);

printf("====================================\n");

semctl(sem\_id, 0, IPC\_RMID, 0);

first\_field = game\_field\_destroy(first\_field);

second\_field = game\_field\_destroy(second\_field);

return 0;

}

# ЛИСТИНГ DEFINE.H

#ifndef DEFINE\_H

#define DEFINE\_H

#define RANDOM\_ARRAY\_SET 0

#define SHOTS\_STEPS\_SET 1

#define GARANT\_NON\_RANDOM\_SET 2

#define CHESS\_ARRAY\_SET 3

#define FIRST\_PLAYER\_NUM 0

#define SECOND\_PLAYER\_NUM 1

#define CREATE\_WITHOUT\_SHIPS\_ARRAY 0

#define CREATE\_WITH\_SHIPS\_ARRAY 1

#define SEMAPHORE\_GIVE\_ONE 1

#define SEMAPHORE\_REDUCE\_ONE -1

#define FIELD\_SIZE 10

#define SHIPS\_COUNT 10

#define SHOT\_ARRAY\_SIZE 30

#define SHIPS\_CELLS\_COUNT 20

#define TORPEDO\_SHIP\_COUNT 4

#define DESTROYER\_SHIP\_COUNT 3

#define CRUISER\_SHIP\_COUNT 2

#define BATTLESHIP\_SHIP\_COUNT 1

typedef enum client\_states\_enum

{

CLIENT\_STATE\_WAITING\_ANOTHER\_PLAYER,

CLIENT\_STATE\_PLACING\_SHIPS,

CLIENT\_STATE\_BATTLE,

CLIENT\_STATE\_GAME\_WIN,

CLIENT\_STATE\_GAME\_LOSE,

CLIENT\_STATE\_WAIT\_TURN,

CLIENT\_STATE\_MAKE\_TURN

} client\_states;

typedef enum cell\_state\_enum

{

CELL\_STATE\_VOID = ' ',

CELL\_STATE\_HIT = 'O',

CELL\_STATE\_SHIP = '%',

CELL\_STATE\_SHIP\_HIT = 'X',

CELL\_STATE\_SHIP\_DESTROY = '@'

} cell\_state;

typedef enum ship\_category\_enum

{

TORPEDO\_BOAT\_CATEGORY = 1,

DESTROYER\_CATEGORY,

CRUISER\_CATEGORY,

BATTLESHIP\_CATEGORY

} ship\_category;

typedef enum ship\_direction\_enum

{

SHIP\_DIRECTION\_VERTICAL = 'v',

SHIP\_DIRECTION\_HORIZONTAL = 'h'

} ship\_direction;

typedef enum server\_answer\_enum

{

SERVER\_OK,

SERVER\_ERROR

} server\_answer;

typedef struct sockaddr socket\_address;

typedef struct sockaddr\_in socket\_address\_in;

typedef struct client\_shot\_struct

{

char pos\_x;

char pos\_y;

cell\_state state;

} client\_shot;

typedef struct computer\_shot\_struct

{

char pos\_x;

char pos\_y;

} computer\_shot;

typedef struct ship\_struct

{

char pos\_x;

char pos\_y;

ship\_category ship\_ctg;

ship\_direction ship\_dir;

} ship;

typedef struct game\_field\_struct

{

ship\* ships\_info;

char ships\_cells\_count;

char field[FIELD\_SIZE][FIELD\_SIZE];

} game\_field;

#endif

# ЛИСТИНГ FIELD.H

#ifndef FIELD\_H

#define FIELD\_H

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "define.h"

#include "utility.h"

/\* create \ destroy functions \*/

game\_field\* game\_field\_create(char create\_ships\_array);

game\_field\* game\_field\_destroy(game\_field\* field);

/\* game functions \*/

void game\_field\_set\_shot(game\_field\* field, client\_shot\* shot);

void game\_field\_ship\_destroy(game\_field\* field, ship\* shp);

void game\_field\_place\_ship\_to\_field(game\_field\* field, ship\* ship\_to\_place);

char game\_field\_check\_coordinates\_to\_hit(game\_field\* field, char x, char y);

char game\_field\_is\_can\_place\_ship(game\_field\* field, ship\* ship\_to\_place);

cell\_state game\_field\_get\_hit\_result(game\_field\* field, client\_shot\* shot);

char game\_field\_get\_hit\_ship\_num\_by\_coord(ship\* ships, client\_shot\* shot);

char game\_field\_check\_ship\_destroy(game\_field\* field, ship\* check\_ship);

#endif

# ЛИСТИНГ FIELD.C

#include "field.h"

#include "define.h"

#include "stdlib.h"

/\* create \ destroy functions \*/

game\_field\* game\_field\_create(char create\_ships\_array)

{

game\_field\* field = malloc(sizeof(game\_field));

memset(field->field, CELL\_STATE\_VOID, sizeof(field->field));

field->ships\_cells\_count = 0;

if (create\_ships\_array)

field->ships\_info = malloc(SHIPS\_COUNT \* sizeof(ship));

else

field->ships\_info = NULL;

return field;

}

game\_field\* game\_field\_destroy(game\_field\* field)

{

if (field->ships\_info)

free(field->ships\_info);

free(field);

return NULL;

}

/\* game functions \*/

void game\_field\_set\_shot(game\_field\* field, client\_shot\* shot)

{

switch (shot->state)

{

case CELL\_STATE\_HIT: field->field[shot->pos\_x][shot->pos\_y] = CELL\_STATE\_HIT; break;

case CELL\_STATE\_SHIP\_DESTROY:

case CELL\_STATE\_SHIP\_HIT: field->field[shot->pos\_x][shot->pos\_y] = CELL\_STATE\_SHIP\_HIT; break;

default: break;

}

}

void game\_field\_ship\_destroy(game\_field\* field, ship\* shp)

{

int min\_x = get\_max(0, shp->pos\_x - 1);

int min\_y = get\_max(0, shp->pos\_y - 1);

int max\_x = get\_min(FIELD\_SIZE - 1, shp->pos\_x + ((shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL) ? 1 : shp->ship\_ctg));

int max\_y = get\_min(FIELD\_SIZE - 1, shp->pos\_y + ((shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL) ? shp->ship\_ctg : 1));

for (int i = min\_x; i <= max\_x; i++)

for (int j = min\_y; j <= max\_y; j++)

field->field[i][j] = CELL\_STATE\_HIT;

for (int i = 0; i < shp->ship\_ctg; i++)

{

if (shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL)

field->field[shp->pos\_x][shp->pos\_y + i] = CELL\_STATE\_SHIP\_DESTROY;

else

field->field[shp->pos\_x + i][shp->pos\_y] = CELL\_STATE\_SHIP\_DESTROY;

}

}

void game\_field\_place\_ship\_to\_field(game\_field\* field, ship\* shp)

{

for (int i = 0; i < shp->ship\_ctg; i++)

{

if (shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL)

field->field[shp->pos\_x][shp->pos\_y + i] = CELL\_STATE\_SHIP;

else

field->field[shp->pos\_x + i][shp->pos\_y] = CELL\_STATE\_SHIP;

}

}

char game\_field\_check\_coordinates\_to\_hit(game\_field\* field, char x, char y)

{

if ((x >= 0 && x <= 9) && (y >= 0 && y <= 9))

if (field->field[x][y] != CELL\_STATE\_HIT &&

field->field[x][y] != CELL\_STATE\_SHIP\_HIT &&

field->field[x][y] != CELL\_STATE\_SHIP\_DESTROY)

return 0;

return 1;

}

char game\_field\_is\_can\_place\_ship(game\_field\* field, ship\* shp)

{

if (field->field[shp->pos\_x][shp->pos\_y] == CELL\_STATE\_SHIP)

return 0;

if (shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL && shp->pos\_y + shp->ship\_ctg >= FIELD\_SIZE + 1)

return 0;

if (shp->ship\_dir == SHIP\_DIRECTION\_HORIZONTAL && shp->pos\_x + shp->ship\_ctg >= FIELD\_SIZE + 1)

return 0;

int min\_x = get\_max(0, shp->pos\_x - 1);

int min\_y = get\_max(0, shp->pos\_y - 1);

int max\_x = get\_min(FIELD\_SIZE - 1, shp->pos\_x + ((shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL) ? 1 : shp->ship\_ctg));

int max\_y = get\_min(FIELD\_SIZE - 1, shp->pos\_y + ((shp->ship\_dir == SHIP\_DIRECTION\_VERTICAL) ? shp->ship\_ctg : 1));

for (int i = min\_x; i <= max\_x; i++)

for (int j = min\_y; j <= max\_y; j++)

if (field->field[i][j] == CELL\_STATE\_SHIP)

return 0;

return 1;

}

cell\_state game\_field\_get\_hit\_result(game\_field\* field, client\_shot\* shot)

{

cell\_state result = (field->field[shot->pos\_x][shot->pos\_y] == CELL\_STATE\_SHIP) ? CELL\_STATE\_SHIP\_HIT : CELL\_STATE\_HIT;

field->field[shot->pos\_x][shot->pos\_y] = result;

return result;

}

char game\_field\_get\_hit\_ship\_num\_by\_coord(ship\* ships, client\_shot\* shot)

{

for (int i = 0; i < SHIPS\_COUNT; i++)

{

if (ships[i].ship\_dir == SHIP\_DIRECTION\_VERTICAL)

{

if (shot->pos\_x == ships[i].pos\_x && shot->pos\_y >= ships[i].pos\_y && shot->pos\_y <= ships[i].pos\_y + ships[i].ship\_ctg)

return i;

}

else

{

if (shot->pos\_y == ships[i].pos\_y && shot->pos\_x >= ships[i].pos\_x && shot->pos\_x <= ships[i].pos\_x + ships[i].ship\_ctg)

return i;

}

}

return -1;

}

char game\_field\_check\_ship\_destroy(game\_field\* field, ship\* check\_ship)

{

for (int i = 0; i < check\_ship->ship\_ctg; i++)

{

if (check\_ship->ship\_dir == SHIP\_DIRECTION\_VERTICAL)

{

if (field->field[check\_ship->pos\_x][check\_ship->pos\_y + i] == CELL\_STATE\_SHIP)

return 0;

}

else

{

if (field->field[check\_ship->pos\_x + i][check\_ship->pos\_y] == CELL\_STATE\_SHIP)

return 0;

}

}

return 1;

}

# ЛИСТИНГ UTILITY.H

#ifndef UTILITY\_H

#define UTILITY\_H

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "define.h"

#include "renderer.h"

int get\_enemy\_num(char player\_num);

int get\_max(int first, int second);

int get\_min(int first, int second);

char is\_coordinates\_input\_correct(char x, char y);

char is\_ship\_direction\_input\_correct(ship\_direction pos);

char is\_game\_end(client\_states state);

void throw\_exeption\_and\_exit(u\_int16\_t tcp\_socket\_fd, char thread\_num);

void ship\_struct\_assignment\_operation(ship\* first\_ship, ship\* second\_ship);

void shot\_struct\_assignment\_operation(client\_shot\* first\_shot, client\_shot\* second\_shot);

char get\_random\_char\_from\_range(char min, char max);

#endif

# ЛИСТИНГ UTILITY.C

#include "utility.h"

inline int get\_enemy\_num(char player\_num)

{

return (!player\_num) ? 1 : 0;

}

inline int get\_max(int first, int second)

{

return (first > second) ? first : second;

}

inline int get\_min(int first, int second)

{

return (first < second) ? first : second;

}

// return true if bad coordinates input

inline char is\_coordinates\_input\_correct(char x, char y)

{

return !((x < 0 || x > 9) || (y < 0 || y > 9));

}

// return true if bad direction input

inline char is\_ship\_direction\_input\_correct(ship\_direction pos)

{

return !(pos != SHIP\_DIRECTION\_VERTICAL && pos != SHIP\_DIRECTION\_HORIZONTAL);

}

char is\_game\_end(client\_states state)

{

return (state == CLIENT\_STATE\_GAME\_WIN || state == CLIENT\_STATE\_GAME\_LOSE) ? 1 : 0;

}

void throw\_exeption\_and\_exit(u\_int16\_t tcp\_socket\_fd, char thread\_num)

{

printf("Cant read or send data. Thread number: %i. Shutting down...\n", thread\_num);

perror(NULL);

close(tcp\_socket\_fd);

endwin();

exit(1);

}

void ship\_struct\_assignment\_operation(ship\* first\_ship, ship\* second\_ship)

{

first\_ship->pos\_x = second\_ship->pos\_x;

first\_ship->pos\_y = second\_ship->pos\_y;

first\_ship->ship\_dir = second\_ship->ship\_dir;

first\_ship->ship\_ctg = second\_ship->ship\_ctg;

}

void shot\_struct\_assignment\_operation(client\_shot\* first\_shot, client\_shot\* second\_shot)

{

first\_shot->pos\_x = second\_shot->pos\_x;

first\_shot->pos\_y = second\_shot->pos\_y;

first\_shot->state = second\_shot->state;

}

char get\_random\_char\_from\_range(char min, char max)

{

if (!min)

return rand() % (max + 1);

return min + rand() + (max - min + 1);

}

# ЛИСТИНГ RENDERER.H

#ifndef RENDERER\_H

#define RENDERER\_H

#include "define.h"

#include "stdlib.h"

#include "ncurses.h"

typedef WINDOW window;

void ncurses\_renderer\_init();

void ncurses\_renderer\_destroy();

void ncurses\_update\_window\_positions\_and\_colors();

void ncurses\_render\_field(game\_field\* field, window\* field\_window);

void ncurses\_render\_bottom\_bar(client\_states state);

void ncurses\_render\_update(game\_field\* first\_field, game\_field\* second\_field, client\_states state);

void ncurses\_resize\_windows();

int ncurses\_get\_user\_input(char input\_stage);

void ncurses\_update\_player\_state(client\_states state);

void ncurses\_incorrect\_coord\_input\_msg();

/\* =========================================================================== \*/

void native\_renderer\_update(game\_field\* first\_field, game\_field\* second\_field);

void native\_renderer\_print\_field\_row(game\_field\* field, int row);

void native\_renderer\_clear();

void native\_renderer\_print\_row\_nums();

char native\_renderer\_get\_user\_input();

#endif

# ЛИСТИНГ RENDERER.C

#include "sys/types.h"

#include "renderer.h"

#include "define.h"

#include "signal.h"

#include <curses.h>

#define WINDOW\_GAP\_HORIZONTAL 2

#define WINDOW\_GAP\_VERTICAL 2

#define FIELD\_WINDOW\_WIDTH 27

#define FIELD\_WINDOW\_HEIGHT 13

#define BOTTOM\_WINDOW\_WIDTH 56

#define BOTTOM\_WINDOW\_HEIGHT 6

/\* NCURSES RENDERER \*/

typedef struct ncurses\_window\_info\_struct

{

int window\_pos\_x;

int window\_pos\_y;

window\* window\_ptr;

} ncurses\_window\_info;

ncurses\_window\_info first\_field\_window;

ncurses\_window\_info second\_field\_window;

ncurses\_window\_info bottom\_window\_bar;

void ncurses\_update\_window\_positions\_and\_colors()

{

first\_field\_window.window\_pos\_x = COLS / 2 - FIELD\_WINDOW\_WIDTH;

first\_field\_window.window\_pos\_y = LINES / 2 - WINDOW\_GAP\_VERTICAL - 1 - FIELD\_WINDOW\_HEIGHT / 2;

bottom\_window\_bar.window\_pos\_x = first\_field\_window.window\_pos\_x;

second\_field\_window.window\_pos\_x = COLS / 2 + WINDOW\_GAP\_HORIZONTAL;

second\_field\_window.window\_pos\_y = first\_field\_window.window\_pos\_y;

bottom\_window\_bar.window\_pos\_y = first\_field\_window.window\_pos\_y + FIELD\_WINDOW\_HEIGHT + 1;

first\_field\_window.window\_ptr = newwin(FIELD\_WINDOW\_HEIGHT, FIELD\_WINDOW\_WIDTH, first\_field\_window.window\_pos\_y, first\_field\_window.window\_pos\_x);

second\_field\_window.window\_ptr = newwin(FIELD\_WINDOW\_HEIGHT, FIELD\_WINDOW\_WIDTH, second\_field\_window.window\_pos\_y, second\_field\_window.window\_pos\_x);

bottom\_window\_bar.window\_ptr = newwin(BOTTOM\_WINDOW\_HEIGHT, BOTTOM\_WINDOW\_WIDTH, bottom\_window\_bar.window\_pos\_y, bottom\_window\_bar.window\_pos\_x);

wbkgd(first\_field\_window.window\_ptr, COLOR\_PAIR(1));

wbkgd(second\_field\_window.window\_ptr, COLOR\_PAIR(1));

wbkgd(bottom\_window\_bar.window\_ptr, COLOR\_PAIR(1));

wclear(first\_field\_window.window\_ptr);

wclear(second\_field\_window.window\_ptr);

wclear(bottom\_window\_bar.window\_ptr);

bkgd(COLOR\_PAIR(2));

}

void ncurses\_renderer\_init()

{

initscr();

refresh();

start\_color();

init\_pair(1, COLOR\_BLACK, COLOR\_WHITE);

init\_pair(2, COLOR\_WHITE, COLOR\_BLUE);

ncurses\_update\_window\_positions\_and\_colors();

wrefresh(first\_field\_window.window\_ptr);

wrefresh(first\_field\_window.window\_ptr);

wrefresh(bottom\_window\_bar.window\_ptr);

}

void ncurses\_render\_field(game\_field\* field, window\* field\_window)

{

attron(COLOR\_PAIR(1));

wmove(field\_window, 1, 4);

for (int i = 0; i < FIELD\_SIZE; i++)

wprintw(field\_window, "|%i", i);

wprintw(field\_window, "|");

for (int i = 0; i < FIELD\_SIZE; i++)

{

wmove(field\_window, i + 2, 2);

wprintw(field\_window, "%c ", (i + 'a'));

for (int j = 0; j < FIELD\_SIZE; j++)

wprintw(field\_window, "|%c", field->field[j][i]);

wprintw(field\_window, "|\n");

}

box(field\_window, 0, 0);

attroff(COLOR\_PAIR(1));

}

void ncurses\_render\_bottom\_bar(client\_states state)

{

box(bottom\_window\_bar.window\_ptr, 0, 0);

wmove(bottom\_window\_bar.window\_ptr, 1, 1);

wprintw(bottom\_window\_bar.window\_ptr, "Input bars block");

wmove(bottom\_window\_bar.window\_ptr, 2, 1);

wprintw(bottom\_window\_bar.window\_ptr, "Position x: ");

wmove(bottom\_window\_bar.window\_ptr, 3, 1);

wprintw(bottom\_window\_bar.window\_ptr, "Position y: ");

if (state == CLIENT\_STATE\_PLACING\_SHIPS)

{

wmove(bottom\_window\_bar.window\_ptr, 4, 1);

wprintw(bottom\_window\_bar.window\_ptr, "Ship direc: ");

}

wmove(bottom\_window\_bar.window\_ptr, 0, 17);

waddch(bottom\_window\_bar.window\_ptr, ACS\_TTEE);

wmove(bottom\_window\_bar.window\_ptr, BOTTOM\_WINDOW\_HEIGHT - 1, 17);

waddch(bottom\_window\_bar.window\_ptr, ACS\_BTEE);

for (int i = 1; i < BOTTOM\_WINDOW\_HEIGHT - 1; i++)

{

wmove(bottom\_window\_bar.window\_ptr, i, 17);

waddch(bottom\_window\_bar.window\_ptr, ACS\_VLINE);

}

wmove(bottom\_window\_bar.window\_ptr, 1, (BOTTOM\_WINDOW\_WIDTH) / 2 + 1);

wprintw(bottom\_window\_bar.window\_ptr, "Game information");

}

void ncurses\_incorrect\_coord\_input\_msg()

{

wmove(bottom\_window\_bar.window\_ptr, 3, 20);

wprintw(bottom\_window\_bar.window\_ptr, "Incorrect coordinates! Try again!");

wrefresh(bottom\_window\_bar.window\_ptr);

getch();

}

void ncurses\_render\_update(game\_field\* first\_field, game\_field\* second\_field, client\_states state)

{

clear();

refresh();

wclear(bottom\_window\_bar.window\_ptr);

ncurses\_render\_field(first\_field, first\_field\_window.window\_ptr);

ncurses\_render\_field(second\_field, second\_field\_window.window\_ptr);

ncurses\_render\_bottom\_bar(state);

ncurses\_update\_player\_state(state);

wrefresh(first\_field\_window.window\_ptr);

wrefresh(second\_field\_window.window\_ptr);

wrefresh(bottom\_window\_bar.window\_ptr);

}

void ncurses\_resize\_windows()

{

delwin(first\_field\_window.window\_ptr);

delwin(second\_field\_window.window\_ptr);

ncurses\_update\_window\_positions\_and\_colors();

}

int ncurses\_get\_user\_input(char input\_stage)

{

if (input\_stage == 0)

wmove(bottom\_window\_bar.window\_ptr, 2, 14);

else if (input\_stage == 1)

wmove(bottom\_window\_bar.window\_ptr, 3, 14);

else if (input\_stage == 2)

wmove(bottom\_window\_bar.window\_ptr, 4, 14);

int key\_input = wgetch(bottom\_window\_bar.window\_ptr);

if (key\_input == KEY\_RESIZE)

ncurses\_resize\_windows();

else if (key\_input == 27)

kill(0, SIGINT);

return key\_input;

}

void ncurses\_update\_player\_state(client\_states state)

{

wmove(bottom\_window\_bar.window\_ptr, 2, 20);

if (state == CLIENT\_STATE\_MAKE\_TURN)

wprintw(bottom\_window\_bar.window\_ptr, "Your turn");

else if (state == CLIENT\_STATE\_WAIT\_TURN)

wprintw(bottom\_window\_bar.window\_ptr, "Wait enemy turn");

else if (state == CLIENT\_STATE\_PLACING\_SHIPS)

wprintw(bottom\_window\_bar.window\_ptr, "Placing ships");

else if (state == CLIENT\_STATE\_WAITING\_ANOTHER\_PLAYER)

wprintw(bottom\_window\_bar.window\_ptr, "Waiting another player");

}

void ncurses\_renderer\_destroy()

{

delwin(first\_field\_window.window\_ptr);

delwin(second\_field\_window.window\_ptr);

endwin();

}

/\* NCURSES RENDERER END \*/

/\* render functions \*/

void native\_renderer\_update(game\_field\* first\_field, game\_field\* second\_field)

{

native\_renderer\_print\_row\_nums();

for (int i = 0; i < FIELD\_SIZE; i++)

{

native\_renderer\_print\_field\_row(first\_field, i);

printf(" + ");

native\_renderer\_print\_field\_row(second\_field, i);

printf("\n");

}

}

void native\_renderer\_print\_field\_row(game\_field\* field, int row)

{

printf("%c ", (row + 'a'));

for (int i = 0; i < FIELD\_SIZE; i++)

printf("|%c", field->field[i][row]);

printf("|");

}

void native\_renderer\_print\_row\_nums()

{

printf(" ");

for (int i = 0; i < FIELD\_SIZE; i++)

printf("|%i", i);

printf("| + ");

for (int i = 0; i < FIELD\_SIZE; i++)

printf("|%i", i);

printf("|\n");

}

char native\_renderer\_get\_user\_input()

{

char pressed\_key;

scanf(" %c", &pressed\_key);

return pressed\_key;

}

void native\_renderer\_clear()

{

system("clear");

}

# ЛИСТИНГ COMPUTER.H

#ifndef COMPUTER\_H

#define COMPUTER\_H

#include "field.h"

#include "define.h"

#include "utility.h"

extern char computer\_array\_set;

typedef enum hit\_direction\_enum

{

HIT\_DIRECTION\_UNDERFINED = 1,

HIT\_DIRECTION\_UP,

HIT\_DIRECTION\_DOWN,

HIT\_DIRECTION\_LEFT,

HIT\_DIRECTION\_RIGHT

} hit\_direction;

hit\_direction computer\_get\_opposite\_hit\_direction(hit\_direction dir);

hit\_direction computer\_get\_new\_hit\_direction(game\_field\* field, computer\_shot\* first\_shot);

ship\_category computer\_update\_higher\_ship\_tier\_stand(ship\_category current, ship\_category ctg);

char computer\_get\_next\_shot\_by\_direction(computer\_shot\* current, hit\_direction hit\_dir);

void computer\_gen\_new\_shots(game\_field\* field, computer\_shot\* shots, char gen\_shots\_count);

#endif

# ЛИСТИНГ COMPUTER.C

#include "computer.h"

#include "define.h"

#include "field.h"

#include "utility.h"

#include <stdlib.h>

char alg\_step = 3;

char alg\_pos\_x = 0;

char alg\_pos\_y = 0;

hit\_direction computer\_get\_new\_hit\_direction(game\_field\* field, computer\_shot\* first\_shot)

{

if (first\_shot->pos\_x - 1 >= 0 && field->field[first\_shot->pos\_x - 1][first\_shot->pos\_y] != CELL\_STATE\_HIT)

return HIT\_DIRECTION\_LEFT;

if (first\_shot->pos\_y - 1 >= 0 && field->field[first\_shot->pos\_x][first\_shot->pos\_y - 1] != CELL\_STATE\_HIT)

return HIT\_DIRECTION\_UP;

if (first\_shot->pos\_x + 1 <= FIELD\_SIZE - 1 && field->field[first\_shot->pos\_x + 1][first\_shot->pos\_y] != CELL\_STATE\_HIT)

return HIT\_DIRECTION\_RIGHT;

if (first\_shot->pos\_y + 1 <= FIELD\_SIZE - 1 && field->field[first\_shot->pos\_x][first\_shot->pos\_y + 1] != CELL\_STATE\_HIT)

return HIT\_DIRECTION\_DOWN;

return HIT\_DIRECTION\_UNDERFINED;

}

char computer\_get\_next\_shot\_by\_direction(computer\_shot\* current, hit\_direction hit\_dir)

{

if (hit\_dir == HIT\_DIRECTION\_LEFT && current->pos\_x - 1 >= 0)

{

current->pos\_x = current->pos\_x - 1;

return 0;

}

if (hit\_dir == HIT\_DIRECTION\_UP && current->pos\_y - 1 >= 0)

{

current->pos\_y = current->pos\_y - 1;

return 0;

}

if (hit\_dir == HIT\_DIRECTION\_RIGHT && current->pos\_x + 1 <= FIELD\_SIZE - 1)

{

current->pos\_x = current->pos\_x + 1;

return 0;

}

if (hit\_dir == HIT\_DIRECTION\_DOWN && current->pos\_y + 1 <= FIELD\_SIZE - 1)

{

current->pos\_y = current->pos\_y + 1;

return 0;

}

return 1;

}

hit\_direction computer\_get\_opposite\_hit\_direction(hit\_direction dir)

{

if (dir == HIT\_DIRECTION\_UP)

return HIT\_DIRECTION\_DOWN;

if (dir == HIT\_DIRECTION\_LEFT)

return HIT\_DIRECTION\_RIGHT;

if (dir == HIT\_DIRECTION\_DOWN)

return HIT\_DIRECTION\_UP;

if (dir == HIT\_DIRECTION\_RIGHT)

return HIT\_DIRECTION\_LEFT;

return HIT\_DIRECTION\_UNDERFINED;

}

ship\_category computer\_update\_higher\_ship\_tier\_stand(ship\_category current, ship\_category ctg)

{

if (current == ctg)

{

if (current == BATTLESHIP\_CATEGORY)

return CRUISER\_CATEGORY;

if (current == CRUISER\_CATEGORY)

return DESTROYER\_CATEGORY;

if (current == DESTROYER\_CATEGORY)

return TORPEDO\_BOAT\_CATEGORY;

}

return BATTLESHIP\_CATEGORY;

}

char computer\_get\_free\_cells(game\_field\* field)

{

char free\_cells\_count = 0;

for (int i = 0; i < FIELD\_SIZE; i++)

for (int j = 0; j < FIELD\_SIZE; j++)

if (!game\_field\_check\_coordinates\_to\_hit(field, i, j))

free\_cells\_count++;

return free\_cells\_count;

}

void computer\_fill\_array\_linear\_coords(game\_field\* field, computer\_shot\* shots, char gen\_shots\_count)

{

for (int i = 0; i < FIELD\_SIZE; i++)

{

for (int j = 0; j < FIELD\_SIZE; j++)

{

if (game\_field\_check\_coordinates\_to\_hit(field, j, i))

continue;

shots[gen\_shots\_count].pos\_x = j;

shots[gen\_shots\_count].pos\_y = i;

if (gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

gen\_shots\_count++;

else

break;

}

}

}

void computer\_gen\_new\_shots(game\_field\* field, computer\_shot\* shots, char gen\_shots\_count)

{

if (computer\_array\_set == RANDOM\_ARRAY\_SET)

{

if (field->ships\_cells\_count < 4)

computer\_fill\_array\_linear\_coords(field, shots, gen\_shots\_count);

else

{

while (gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

{

shots[gen\_shots\_count].pos\_x = get\_random\_char\_from\_range(0, 9);

shots[gen\_shots\_count].pos\_y = get\_random\_char\_from\_range(0, 9);

if (game\_field\_check\_coordinates\_to\_hit(field, shots[gen\_shots\_count].pos\_x, shots[gen\_shots\_count].pos\_y))

continue;

gen\_shots\_count++;

}

}

}

else if (computer\_array\_set == SHOTS\_STEPS\_SET)

{

char free\_cells\_count = computer\_get\_free\_cells(field);

char pos\_x = alg\_pos\_x;

char pos\_y = alg\_pos\_y;

char iteration\_count = 0;

char iteration\_total = (free\_cells\_count < FIELD\_SIZE) ? free\_cells\_count : FIELD\_SIZE;

char can\_change\_alg\_pos = 0;

if (field->field[alg\_pos\_x][alg\_pos\_y] != CELL\_STATE\_SHIP\_HIT)

can\_change\_alg\_pos = 1;

while (iteration\_count < iteration\_total && gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

{

if (pos\_y + alg\_step > FIELD\_SIZE - 1)

{

pos\_x++;

if (pos\_x > FIELD\_SIZE - 1)

{

pos\_x = 0;

alg\_step--;

}

pos\_y = abs(pos\_y + alg\_step - FIELD\_SIZE);

}

else

pos\_y += alg\_step;

if (alg\_step <= 0)

{

computer\_fill\_array\_linear\_coords(field, shots, gen\_shots\_count);

break;

}

if (game\_field\_check\_coordinates\_to\_hit(field, pos\_x, pos\_y))

continue;

if (field->field[pos\_x][pos\_y] == CELL\_STATE\_SHIP && can\_change\_alg\_pos)

{

alg\_pos\_x = pos\_x;

alg\_pos\_y = pos\_y;

can\_change\_alg\_pos = 0;

}

shots[gen\_shots\_count].pos\_x = pos\_x;

shots[gen\_shots\_count].pos\_y = pos\_y;

iteration\_count++;

gen\_shots\_count++;

}

}

else if (computer\_array\_set == CHESS\_ARRAY\_SET)

{

char pos\_x = 0;

char pos\_y = 0;

char can\_change\_alg\_pos = 0;

if (field->field[alg\_pos\_x][alg\_pos\_y] != CELL\_STATE\_SHIP\_HIT)

can\_change\_alg\_pos = 1;

for (int i = 0; i < FIELD\_SIZE; i++)

{

for (int j = 0; j < FIELD\_SIZE; j++)

{

if ((i % 2 == 0 && j % 2 == 0) ||

(i % 2 != 0 && j % 2 != 0))

{

pos\_x = j;

pos\_y = i;

}

else

continue;

if (game\_field\_check\_coordinates\_to\_hit(field, pos\_x, pos\_y))

continue;

shots[gen\_shots\_count].pos\_x = pos\_x;

shots[gen\_shots\_count].pos\_y = pos\_y;

if (field->field[i][j] == CELL\_STATE\_SHIP && can\_change\_alg\_pos)

{

alg\_pos\_x = pos\_x;

alg\_pos\_y = pos\_y;

can\_change\_alg\_pos = 0;

}

if (gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

gen\_shots\_count++;

else

break;

}

}

if (gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

computer\_fill\_array\_linear\_coords(field, shots, gen\_shots\_count);

}

else

{

char free\_cells\_count = computer\_get\_free\_cells(field);

char temp\_field[FIELD\_SIZE][FIELD\_SIZE];

for (int i = 0; i < FIELD\_SIZE; i++)

for (int j = 0; j < FIELD\_SIZE; j++)

temp\_field[i][j] = field->field[i][j];

while (free\_cells\_count && gen\_shots\_count < SHOT\_ARRAY\_SIZE - 1)

{

shots[gen\_shots\_count].pos\_x = get\_random\_char\_from\_range(0, 9);

shots[gen\_shots\_count].pos\_y = get\_random\_char\_from\_range(0, 9);

if (temp\_field[shots[gen\_shots\_count].pos\_x][shots[gen\_shots\_count].pos\_y] == CELL\_STATE\_HIT ||

temp\_field[shots[gen\_shots\_count].pos\_x][shots[gen\_shots\_count].pos\_y] == CELL\_STATE\_SHIP\_DESTROY ||

temp\_field[shots[gen\_shots\_count].pos\_x][shots[gen\_shots\_count].pos\_y] == CELL\_STATE\_SHIP\_HIT)

continue;

temp\_field[shots[gen\_shots\_count].pos\_x][shots[gen\_shots\_count].pos\_y] = CELL\_STATE\_HIT;

gen\_shots\_count++;

free\_cells\_count--;

}

}

}