Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**«Консоль-серверная игра»**

Студент: Кирилин Иван Олегович

Группа: М8О-301Б-21

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Bonartze/OS

**Постановка задачи**

Консоль-серверная игра. Необходимо написать консоль-серверную игру. Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента игрок может выбрать одно из следующих действий (возможно больше, если предусмотрено вариантом):

- Присоединиться к одной из существующих игр по имени игры

**Вариант 3**:

3. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику

**Общие сведения о программе**

Программа состоит из двух файлов – client.cpp и server.cpp. В client.cpp лежит код клиента, а в server.cpp – код сервера.

**Общий метод и алгоритм решения**

Сначала происходит запуск сервера, ему дается свой ID. По этому ID к серверу можно подключить любое число клиентов и начать игру. Очередь сообщений была реализована с помощью библиотеки ZeroMQ.

**Исходный код**

**server.cpp**

#include "zmq.hpp"

#include "zmq.h"

#include <iostream>

#include <unistd.h>

#include <map>

#include <set>

#include <string>

#include <vector>

using namespace std;

void random(vector<vector<char> > & p)

{

int j=-1, k, v, l, x[2], y;

srand(time(0));

for(l=4; l>0; l--)

for(k=5;k-l;k--)

{

v = 1&rand();

//v = rand() % 2;

do for (x[v] = 1 + rand() % 10, x[1 - v] = 1 + rand() % 7, y = j = 0; j - l; y |= p[x[0]][x[1]] != '.', x[1 - v]++, j++); while(y);

x[1 - v] -= l + 1, p[x[0]][x[1]] = '/', x[v]--, p[x[0]][x[1]]='/', x [v]+=2, p[x[0]][x[1]]='/', x[v]--, x[1 - v]++;

for (j = -1; ++j - l; p[x[0]][x[1]] = 'X', x[v]--, p[x[0]][x[1]] = '/', x[v]+=2, p[x[0]][x[1]]='/', x[v]--, x[1 - v]++);

p[x[0]][x[1]] = '/', x[v]--, p[x[0]][x[1]] = '/', x[v]+=2, p[x[0]][x[1]] = '/';

}

for (int i = 0; i < 12; ++i)

{

replace(p[i].begin(), p[i].end(), '/', '.');

}

}

void send\_message(string message\_string, zmq::socket\_t& socket)

{

zmq::message\_t message\_back(message\_string.size());

memcpy(message\_back.data(), message\_string.c\_str(), message\_string.size());

if(!socket.send(message\_back))

{

cout << "Не удается отправить сообщение. " << getpid() << "\n";

}

}

void print(vector<vector<char> >& p)

{

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

{

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

{

cout << p[i][j];

}

cout << "\n";

}

}

int main()

{

zmq::context\_t context (1);

zmq::socket\_t socket (context, ZMQ\_REP); // устанавливает флаги на сокет

string port, reply;

cout << "Введите номер игры\n";

cin >> port;

socket.bind("tcp://\*:" + port); //Связывает объект Socket с локальной конечной точкой.

//TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю.

unsigned milliseconds;

cout << "Введите время, в течение которого сокет должен ждать ответа от клиента и отправлять сообщение клиенту \n";

cin >> milliseconds;

socket.setsockopt(ZMQ\_SNDTIMEO, (int)milliseconds); // устанавливает флаги и время ожидания от сервера

//Это ассоциативный контейнер, который работает по принципу — [ключ — значение]

map<int, pair<unsigned, unsigned> >statistics;

map<int, pair<unsigned, unsigned> > amount;

map<int, pair<vector<vector<char> >, vector<vector<char> > > > fields;

map<int, vector<pair<unsigned, unsigned> > > possible\_turns;

map<int, pair<unsigned, unsigned> > last\_commands;

map<int, bool> finishing;

map<int, vector<pair<unsigned, unsigned> > > variants; // после удачного попадания сервер пойдет на соседние клетки

while (true)

{

zmq::message\_t request;

socket.recv(&request); // Функция служит для чтения данных из сокета.

string message(static\_cast<char\*>(request.data()), request.size()), reply;

string command = message.substr(0, message.find(" "));

int ID = stoi(message.substr(message.find(" ") + 1));

cout << message << "\n";

if (command == "ID")

{

statistics[ID] = make\_pair(0, 0);

amount[ID] = make\_pair(0, 0);

send\_message("OK", socket);

}

else if (command == "Statistics")

{

if (statistics.find(ID) != statistics.end()) // чтобы не совпадали ID у пользователей

{

pair<unsigned, unsigned> numbers = statistics[ID];

reply = to\_string(numbers.first) + " " + to\_string(numbers.second);

}

else

{

reply = "Игрок с таким ID уже сужествует";

}

send\_message(reply, socket);

}

else if (command == "Get") // вывод поля сервера

{

print(fields[ID].first);

send\_message("OK", socket);

}

else if (command == "Exit") // все чистим

{

if (statistics.find(ID) != statistics.end())

{

statistics.erase(ID);

amount.erase(ID);

fields.erase(ID);

possible\_turns.erase(ID);

last\_commands.erase(ID);

finishing.erase(ID);

variants.erase(ID);

}

send\_message("Спасибо за игру!", socket);

}

else if (command == "Begin")

{

amount[ID] = make\_pair(0, 0);

vector<vector<char> > server\_field (12, vector<char>(12, '.'));

vector<vector<char> > player\_field (12, vector<char>(12, '.'));

random(server\_field);

fields[ID] = make\_pair(server\_field, player\_field);

server\_field.clear();

player\_field.clear();

vector<pair<unsigned, unsigned> > turns (100);

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

{

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

{

turns[i \* 10 + j] = make\_pair(i, j);

}

}

//cout << "Created turns\n";

possible\_turns[ID] = turns;

finishing[ID] = false;

last\_commands[ID] = make\_pair(-1, -1);

variants[ID].clear();

variants[ID].push\_back(make\_pair(1, 0));

variants[ID].push\_back(make\_pair(-1, 0));

variants[ID].push\_back(make\_pair(0, 1));

variants[ID].push\_back(make\_pair(0, -1));

turns.clear();

//reply = "Start";

send\_message("Start", socket);

}

else if (command.substr(0, 3) == "Try") // substr возвращает с 0 символа длинной 3

{

string coordinates = message.substr(0, message.find(" ")); // считывается что-то в духе Try21 2522

cout << coordinates << "\n";

unsigned horizontal = unsigned(coordinates[3]) - unsigned('0') + 1; // переход из букв в цифры

unsigned vertical = unsigned(coordinates[4]) - unsigned('0') + 1;

cout << horizontal << " " << vertical << "\n";

pair<vector<vector<char> >, vector<vector<char> > > squares = fields[ID];

if (squares.first[vertical][horizontal] == 'X') // если попали в сервекский корабль меняем X на К

{

squares.first[vertical][horizontal] = 'K';

reply = "Killed";

int v = vertical, h = horizontal; // начинаем проверку по вертикали и горизонтали убили мы корабль или подбили

while ((v > 1) && (squares.first[v][h] == 'K'))

{

--v;

}

if (squares.first[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = vertical; h = horizontal;

while ((v < 10) && (squares.first[v][h] == 'K'))

{

++v;

}

if (squares.first[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = vertical; h = horizontal;

while ((h > 1) && (squares.first[v][h] == 'K'))

{

--h;

}

if (squares.first[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = vertical; h = horizontal;

while ((h < 10) && (squares.first[v][h] == 'K'))

{

++h;

}

if (squares.first[v][h] == 'X')

{

reply = "Wounded";

}

}

}

}

if (reply == "Killed")

{

amount[ID] = make\_pair(++amount[ID].first, amount[ID].second); // увеличиваем число кораблей убитых сервером

if (amount[ID].first == 10)

{

reply = "Won";

statistics[ID] = make\_pair(++statistics[ID].first, statistics[ID].second);

}

}

}

else if ((squares.first[vertical][horizontal] == 'K') || (squares.first[vertical][horizontal] == 'w'))

{

reply = "Another"; // уже стреляли сюда

}

else if (squares.first[vertical][horizontal] == '.')

{

reply = "Missed";

squares.first[vertical][horizontal] = 'w';

}

fields[ID] = make\_pair(squares.first, squares.second);

send\_message(reply, socket);

}

else if (command == "Amount") // количество убитых кораблей у сервера

{

cout << "Amount: " << amount[ID].first << "\n";

send\_message("OK", socket);

}

else if (command == "Turns") // оставшиеся шаги

{

vector<pair<unsigned, unsigned> > turns = possible\_turns[ID];

for (int i = 0; i < turns.size(); i++)

{

cout << turns[i].first << " " << turns[i].second << "\n";

}

cout << "Length is " << turns.size() << "\n";

send\_message("Ok", socket);

}

else if ((command == "Do") || (command == "Killed"))

{

if (command == "Killed") // попадание сервера и убийство корабля

{

pair<vector<vector<char> >, vector<vector<char> > > squares = fields[ID];

vector<pair<unsigned, unsigned> > turns = possible\_turns[ID];

unsigned vertical = last\_commands[ID].first, horizontal = last\_commands[ID].second; // последняя введенная команда по горизонтали и вертикали

squares.second[vertical][horizontal] = 'K'; // ставим в поле пользователя

if ((vertical >= 0) && (vertical < 10) && (horizontal >= 0) && (horizontal < 10)) // удаление из turns с помощью итераторов

{

vector<pair<unsigned, unsigned> >::iterator it;

if ((it = find(turns.begin(), turns.end(), make\_pair(vertical, horizontal))) != turns.end())

{

turns.erase(turns.begin() + distance(turns.begin(), it));

//Вышеупомянутая функция erase () используется для удаления нескольких элементов из вектора на основе позиции, указанной в первом и втором аргументах этой функции.

}

if (squares.second[vertical][horizontal] == '.')

{

squares.second[vertical][horizontal] = 'w';

}

}

fields[ID] = make\_pair(squares.first, squares.second);

possible\_turns[ID] = turns;

finishing[ID] = false;

variants[ID].clear();

variants[ID].push\_back(make\_pair(1, 0));

variants[ID].push\_back(make\_pair(-1, 0));

variants[ID].push\_back(make\_pair(0, 1));

variants[ID].push\_back(make\_pair(0, -1));

amount[ID] = make\_pair(amount[ID].first, ++amount[ID].second); // +1 убитый корабль сервером

if (amount[ID].second == 10)

{

reply = "Lost";

amount[ID] = make\_pair(0, 0);

}

}

if (!finishing[ID]) // компуктер рандомно стреляет если у пользователчя предыдущий ход был неудачным

{

int length = possible\_turns[ID].size();

srand(time(0));

int number = rand() % length;

vector<pair<unsigned, unsigned> > coordinates = possible\_turns[ID];

pair<unsigned, unsigned> turn = coordinates[number];

cout << "Turn is " << turn.first << " " << turn.second << "\n";

coordinates.erase(coordinates.begin() + number);

possible\_turns[ID] = coordinates;

last\_commands[ID] = turn;

reply = "Try" + to\_string(turn.first) + to\_string(turn.second);

cout << "Tried coordinates " << turn.first << " " << turn.second << "\n";

}

else //если ход был удачный

{

int length, number, k = 1;

vector<pair<unsigned, unsigned> > positions;

pair<unsigned, unsigned> turn;

do

{

length = variants[ID].size();

number = rand() % length;

positions = variants[ID];

turn = positions[number];

positions.erase(positions.begin() + number);

}

while ((length > 0) && (!((last\_commands[ID].first + turn.first > 0) && (last\_commands[ID].first + turn.first < 11) && (last\_commands[ID].second + turn.second > 0)

&& (last\_commands[ID].second + turn.second < 11))));

pair<vector<vector<char> >, vector<vector<char> > > squares = fields[ID];

vector<pair<unsigned, unsigned> > turns = possible\_turns[ID];

unsigned vertical, horizontal;

do

{

send\_message("Try" + to\_string(last\_commands[ID].first + turn.first \* k) + to\_string(last\_commands[ID].second + turn.second \* k), socket);

++k;

zmq::message\_t answer;

socket.recv(&answer);

string string(static\_cast<char\*>(answer.data()), answer.size());

reply = string.substr(0, string.find(" "));

vertical = last\_commands[ID].first + turn.first \* k, horizontal = last\_commands[ID].second + turn.second \* k;

turns.erase(turns.begin() + distance(turns.begin(), find(turns.begin(), turns.end(), make\_pair(vertical, horizontal))));

if ((reply == "Wounded") || (reply == "Killed"))

{

if ((vertical >= 0) && (vertical < 10) && (horizontal >= 0) && (horizontal < 10))

{

squares.second[vertical][horizontal] = 'K';

vector<pair<unsigned, unsigned> >::iterator it;

if ((it = find(turns.begin(), turns.end(), make\_pair(vertical, horizontal))) != turns.end())

{

turns.erase(turns.begin() + distance(turns.begin(), it));

}

if (squares.second[vertical][horizontal] == '.')

{

squares.second[vertical][horizontal] = 'w';

}

}

}

else if (reply == "Missed")

{

squares.second[vertical][horizontal] = 'w';

}

//fields[ID] = {squares.first, squares.second};

}

while (reply == "Wounded");

fields[ID] = make\_pair(squares.first, squares.second);

possible\_turns[ID] = turns;

variants[ID] = positions;

if (reply == "Missed")

{

reply = "Do";

}

else if (reply == "Killed")

{

finishing[ID] = false;

variants[ID].clear();

variants[ID].push\_back(make\_pair(1, 0));

variants[ID].push\_back(make\_pair(-1, 0));

variants[ID].push\_back(make\_pair(0, 1));

variants[ID].push\_back(make\_pair(0, -1));

amount[ID] = make\_pair(amount[ID].first, ++amount[ID].second);

if (amount[ID].second == 10)

{

reply = "Lost";

amount[ID] = make\_pair(0, 0);

}

else

{

int length = possible\_turns[ID].size();

srand(time(0));

int number = rand() % length;

vector<pair<unsigned, unsigned> > coordinates = possible\_turns[ID];

pair<unsigned, unsigned> turn = coordinates[number];

coordinates.erase(coordinates.begin() + number);

possible\_turns[ID] = coordinates;

last\_commands[ID] = turn;

reply = "Try" + to\_string(turn.first) + to\_string(turn.second);

cout << "Tried coordinates " << turn.first << " " << turn.second << "\n";

}

}

}

send\_message(reply, socket);

}

else if (command == "Missed") // сервер промахнулся

{

pair<vector<vector<char> >, vector<vector<char> > > squares = fields[ID];

squares.second[last\_commands[ID].first][last\_commands[ID].second] = 'w';

fields[ID] = make\_pair(squares.first, squares.second);

send\_message("Do", socket);

}

else if (command == "Wounded") // сервер после подбития корабля

{

finishing[ID] = true;

int length, number, k = 1;

vector<pair<unsigned, unsigned> > positions;

pair<unsigned, unsigned> turn;

do

{

length = variants[ID].size();

number = rand() % length;

positions = variants[ID];

turn = positions[number];

positions.erase(positions.begin() + number);

}

while ((length > 0) && (!((last\_commands[ID].first + turn.first > 0) && (last\_commands[ID].first + turn.first < 11) && (last\_commands[ID].second + turn.second > 0)

&& (last\_commands[ID].second + turn.second < 11))));

pair<vector<vector<char> >, vector<vector<char> > > squares = fields[ID];

vector<pair<unsigned, unsigned> > turns = possible\_turns[ID];

unsigned vertical, horizontal;

do

{

send\_message("Try" + to\_string(last\_commands[ID].first + turn.first \* k) + to\_string(last\_commands[ID].second + turn.second \* k), socket);

++k;

zmq::message\_t answer;

socket.recv(&answer);

string string(static\_cast<char\*>(answer.data()), answer.size());

reply = string.substr(0, string.find(" "));

vertical = last\_commands[ID].first + turn.first \* k, horizontal = last\_commands[ID].second + turn.second \* k;

turns.erase(turns.begin() + distance(turns.begin(), find(turns.begin(), turns.end(), make\_pair(vertical, horizontal))));

if ((reply == "Wounded") || (reply == "Killed"))

{

if ((vertical >= 0) && (vertical < 10) && (horizontal >= 0) && (horizontal < 10))

{

squares.second[vertical][horizontal] = 'K';

vector<pair<unsigned, unsigned> >::iterator it;

if ((it = find(turns.begin(), turns.end(), make\_pair(vertical, horizontal))) != turns.end())

{

turns.erase(turns.begin() + distance(turns.begin(), it));

}

if (squares.second[vertical][horizontal] == '.')

{

squares.second[vertical][horizontal] = 'w';

vector<pair<unsigned, unsigned> >::iterator it;

}

}

}

else if (reply == "Missed")

{

squares.second[vertical][horizontal] = 'w';

}

}

while (reply == "Wounded");

fields[ID] = make\_pair(squares.first, squares.second);

variants[ID] = positions;

if (reply == "Missed")

{

reply = "Do";

}

else if (reply == "Killed")

{

finishing[ID] = false;

variants[ID].clear();

variants[ID].push\_back(make\_pair(1, 0));

variants[ID].push\_back(make\_pair(-1, 0));

variants[ID].push\_back(make\_pair(0, 1));

variants[ID].push\_back(make\_pair(0, -1));

amount[ID] = make\_pair(amount[ID].first, ++amount[ID].second);

if (amount[ID].second == 10)

{

reply = "Lost";

amount[ID] = make\_pair(0, 0);

statistics[ID] = make\_pair(statistics[ID].first, ++statistics[ID].second);

}

else // ход сервера после убийства коробля пользователя в случае если игра еще не закончилась

{

int length = possible\_turns[ID].size();

srand(time(0));

int number = rand() % length;

vector<pair<unsigned, unsigned> > coordinates = possible\_turns[ID];

pair<unsigned, unsigned> turn = coordinates[number];

coordinates.erase(coordinates.begin() + number);

possible\_turns[ID] = coordinates;

last\_commands[ID] = turn;

reply = "Try" + to\_string(turn.first) + to\_string(turn.second);

cout << "Tried coordinates " << turn.first << " " << turn.second << "\n";

}

}

send\_message(reply, socket);

}

}

return 0;

}

**client.cpp**

#include <zmq.hpp>

#include <zmq.h> // библиотека для создания многопоточных приложений

#include <iostream>

#include <unistd.h>

#include <vector>

#include <algorithm>

using namespace std;

void send\_message(string message\_string, zmq::socket\_t& socket) // отправка сообщений на сервер

{

zmq::message\_t message\_back(message\_string.size());

memcpy(message\_back.data(), message\_string.c\_str(), message\_string.size()); // копирующая содержимое одной области памяти в другую.

if(!socket.send(message\_back))

{

cout << "Не удается отправить сообщение. " << getpid() << "\n";

}

}

string recieve\_message(zmq::socket\_t& socket) // получение сообщений с сервера

{

zmq::message\_t reply;

if (!socket.recv(&reply)) // чтение данных из сокета

{

cout << "Не удается получить сообщение от сервера\n";

exit(1);

}

string message(static\_cast<char\*>(reply.data()), reply.size());

return message;

}

void random(vector<vector<char> >& p)

{

int j=-1, k, v, l, x[2], y;

srand(time(0)); // srand() используется для установки начала последовательности, генерируемой функцией rand()

for(l=4; l>0; l--)

for(k=5;k-l;k--)

{

v = 1&rand();

do for (x[v] = 1 + rand() % 10, x[1 - v] = 1 + rand() % 7, y = j = 0; j - l; y |= p[x[0]][x[1]] != '.', x[1 - v]++, j++); while(y);

x[1 - v] -= l + 1, p[x[0]][x[1]] = '/', x[v]--, p[x[0]][x[1]] = '/', x [v] += 2, p[x[0]][x[1]] = '/', x[v]--, x[1 - v]++;

for (j = -1; ++j - l; p[x[0]][x[1]] = 'X', x[v]--, p[x[0]][x[1]] = '/', x[v] += 2, p[x[0]][x[1]] = '/', x[v]--, x[1 - v]++);

p[x[0]][x[1]] = '/', x[v]--, p[x[0]][x[1]] = '/', x[v]+=2, p[x[0]][x[1]] = '/';

}

for (int i = 0; i < 12; ++i)

{

replace(p[i].begin(), p[i].end(), '/', '.');

}

}

void flood(vector<vector<char> >& p)

{

for (int i = 0; i < 12; i++)

{

p[i].clear();

p[i] = vector<char>(12, '.');

}

}

void print(vector<vector<char> >& p)

{

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

{

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

{

cout << p[i][j];

}

cout << "\n";

}

}

int main()

{

zmq::context\_t context (1); // Класс context\_t инкапсулирует функции, связанные с инициализацией и завершением контекста

zmq::socket\_t socket (context, ZMQ\_REQ); // обмен данными между клиентом и сервером

//Клиент использует ZMQ\_REQ для отправки сообщений и получения ответов от сервера.

string port;

cout << "Введите номер игры, к которой хотите присоединиться\n";

cin >> port;

cout << "Подключение к серверу…" << endl;

unsigned milliseconds;

cout << "Введите время, в течение которого сокет может ожидать ответа от сервера.\n"; //Со́кет — название программного интерфейса для обеспечения обмена данными между процессами.

cin >> milliseconds;

socket.setsockopt(ZMQ\_SNDTIMEO, (int)milliseconds); //Устанавливает тайм-аут для операции отправки на сокете.

socket.setsockopt(ZMQ\_RCVTIMEO, (int)milliseconds); //Устанавливает тайм-аут для операции получения на сокете.

socket.connect ("tcp://localhost:" + port);

send\_message("ID " + to\_string(getpid()), socket); //

zmq::message\_t reply;

recieve\_message(socket); // Ждем чтобы сервер начлал игру

cout << "Если вы готовы начинать игру, введите 'Begin' " << endl;

vector<vector<char> > my\_field(12, vector<char> (12, '.'));

vector<vector<char> > server\_field(12, vector<char> (12, '.'));

string command;

bool playing = false;

while (cin >> command)

{

if (command == "Begin")

{

playing = true;

int number;

flood(my\_field); //функция заливки

flood(server\_field);

cout << "Для начала давайте построим вашу площадку для игры.\n";

cout << "Вы хотите расставить корабли самостоятельно или сгенерировать случайную комбинацию? Если первое, введите 1, иначе - 2.\n";

do

{

cin >> number;

if ((number < 1) || (number > 2))

{

cout << "Ошибка, вы можете ввести только 1 или 2\n";

}

}

while ((number < 1) || (number > 2));

if (number == 1)

{

int amount = 0, amounts[4], v1, v2;

for (int i = 0; i < 4; ++i)

{

amounts[i] = 0;

}

char h1, h2;

cout << "Всего вам нужно ввести кораблей, 1 четырехпалубный, 2 трехпалубных, 3 двухпалубных и 4 однопалубных.\n";

cout << "Для создание корабля введите комбинацию из четырех символов, например, А 4 А 5 или С 7 D 7\n";

while (amount < 10)

{

cin >> h1 >> v1 >> h2 >> v2;

if ((v1 < 1) || (v2 < 1) || (v1 > 10) || (v2 > 10))

{

cout << "Вы вышли за границу поля. Число должно быть больше 0 и меньше 11.\n";

continue;

}

if (!((h1 >= 'A') && (h1 <= 'J') && (h2 >= 'A') && (h2 <= 'J')))

{

cout << "Вы вышли за границу поля. Буквы должны быть не меньше А и не больше J.\n";

continue;

}

if ((v1 != v2) && (h1 != h2))

{

cout << "Корабль должен быть параллелен одной из осей координат\n";

continue;

}

if ((v1 - v2 > 4) || (h1 - h2 > 4))

{

cout << "Корабль слишком большой, он не может быть больше 4\n";

continue;

}

int ih1 = int(h1) - int('A') + 1, ih2 = int(h2) - int('A') + 1;

if (v2 < v1)

{

swap(v1, v2);

}

if (ih2 < ih1)

{

swap(ih1, ih2);

}

if (v1 == v2)

{

bool possible = true;

for (int i = ih1 - 1; i < ih2 + 2; ++i)

{

for (int j = -1; j < 2; ++j)

{

if ((v1 + j > 0) && (v1 + j < 11) && (i > 0) && (i < 11))

{

if (my\_field[v1 + j][i] == 'X')

{

possible = false;

break;

}

}

}

if (!possible)

{

break;

}

}

if (!possible)

{

cout << "Невозможно так ставить корабль, так как он будет соприкасаться с другим.\n";

continue;

}

else

{

int number = ih2 - ih1;

if (amounts[number] == 4 - number)

{

cout << "У вас уже достаточно кораблей этого типа.\n";

continue;

}

++amounts[number];

for (int i = ih1; i < ih2 + 1; ++i)

{

my\_field[v1][i] = 'X';

}

++amount;

cout << "Отлично, мы построили поле с вашими кораблями!\n";

}

}

else if (ih1 == ih2)

{

bool possible = true;

for (int i = v1 - 1; i < v2 + 2; ++i)

{

for (int j = -1; j < 2; ++j)

{

if ((ih1 + j > 0) && (ih1 + j < 11) && (i > 0) && (i < 11))

{

if (my\_field[i][ih1 + j] == 'X')

{

possible = false;

break;

}

}

}

if (!possible)

{

break;

}

}

if (!possible)

{

cout << "Невозможно так ставить корабль, так как он будет соприкасаться с другим.\n";

continue;

}

else

{

int number = v2 - v1;

if (amounts[number] == 4 - number)

{

cout << "У вас уже достаточно кораблей этого типа.\n";

continue;

}

++amounts[number];

for (int i = v1; i < v2 + 1; ++i)

{

my\_field[i][ih1] = 'X';

}

++amount;

cout << "Отлично, мы построили поле с вашими кораблями!\n";

}

}

}

}

else if (number == 2)

{

cout << "Если вы хотите сохранить данную генерацию, введите 1. Введите 2, если хотите сгенерировать новое расположение кораблей.\n";

int indicator;

do

{

flood(my\_field);

random(my\_field);

print(my\_field);

do

{

cin >> indicator;

if ((indicator < 1) || (indicator > 2))

{

cout << "Вы можете ввести только 1 или 2\n";

}

}

while ((indicator < 1) || (indicator > 2));

}

while (indicator != 1);

}

send\_message("Begin " + to\_string(getpid()), socket);

recieve\_message(socket);

cout << "Выберите действие, которое хотите сделать.\n";

cout << "Чтобы сделать ход введите 'Try'.\n";

cout << "Чтобы закончить игру и выйти введите 'Exit'.\n";

cout << "Чтобы посмотреть свою статистику введите 'Statistics'.\n";

cout << "Чтобы посмотреть на свое поле введите 'My'.\n";

cout << "Чтобы посмотреть поле сервера введите 'Server'.\n";

continue;

}

else if (command == "Get") // показывает на сервере какие там корабли

{

send\_message("Get " + to\_string(getpid()), socket);

recieve\_message(socket);

}

if (command == "Exit")

{

send\_message("Exit " + to\_string(getpid()), socket);

string reply = recieve\_message(socket);

cout << reply;

return 0;

}

if (command == "Statistics")

{

send\_message("Statistics " + to\_string(getpid()), socket);

string reply = recieve\_message(socket);

cout << "У вас " + reply.substr(0, reply.find(" ")) + " побед и " + reply.substr(reply.find(" ") + 1) + " проигрышей\n";

cout << "Введите ход\n";

continue;

}

if (command == "My")

{

cout << "Ваше поле\n";

print(my\_field);

cout << "Введите ход\n";

continue;

}

if (command == "Amount")

{

send\_message("Amount " + to\_string(getpid()), socket);

recieve\_message(socket);

continue;

}

if (command == "Server")

{

cout << "Поле сервера\n";

print(server\_field);

cout << "Введите ход\n";

continue;

}

if (command == "Turns") // Cколько ходов осталось

{

send\_message("Turns " + to\_string(getpid()), socket);

recieve\_message(socket);

}

if (command == "Try")

{

if (!playing)

{

cout << "На данный момент вы не начали игру\n";

continue;

}

else

{

int v;

char h;

while (true)

{

cin >> h >> v;

if (!((h >= 'A') && (h <= 'J')))

{

cout << "Буквы должны находится в интервале между A и J\n";

continue;

}

else if ((v < 1) || (v > 10))

{

cout << "Цифры должны находится в интервале от 1 до 10\n";

continue;

}

break;

}

send\_message("Try" + to\_string(int(h) - int('A')) + to\_string(v - 1) + " " + to\_string(getpid()), socket);

string reply = recieve\_message(socket);

cout << "Ваш ход пользователь: " << reply << "\n";

if ((reply == "Killed") || (reply == "Wounded"))

{

server\_field[v][int(h) - int('A') + 1] = 'K';

if (reply == "Killed")

{

cout << "Вы убили корабль сервера!\n";

}

else

{

cout << "Вы подбили корабль сервера!\n";

}

cout << "Введите ход\n";

continue;

}

if (reply == "Another")

{

cout << "Вы уже вводили эти координаты, введите новые\n";

continue;

}

if (reply == "Won")

{

cout << "Вы победили, поздравляю!\n";

playing = false;

continue;

}

if (reply == "Missed") //при промахе пользователя серверу отправляем Do с возможностью сделать ход

{

server\_field[v][int(h) - int('A') + 1] = 'w';

send\_message("Do " + to\_string(getpid()), socket);

while (true)

{

reply = recieve\_message(socket);

if (reply.substr(0, 3) == "Try")

{

cout << "Очередь сервера: " << char(int(reply[4] - int('0') + 'A')) << " " << int(reply[3]) - int('0') + 1 << "\n";

//reply = to\_string(int(reply[4]) - int('0')) + " " + reply[3];

}

else

{

cout << "Ход сервера: " << reply << "\n";

}

if ((reply == "Lost") || (reply == "Do"))

{

break;

}

int hor = int(reply[4]) - int('0') + 1, ver = int(reply[3]) - int('0') + 1;

if (my\_field[ver][hor] == 'X')

{

reply = "Killed"; // проверка убил или попал счервер

int v = ver, h = hor;

my\_field[v][h] = 'K';

while ((v > 1) && (my\_field[v][h] == 'K'))

{

--v;

}

if (my\_field[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = ver; h = hor;

while ((v < 10) && (my\_field[v][h] == 'K'))

{

++v;

}

if (my\_field[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = ver; h = hor;

while ((h > 1) && (my\_field[v][h] == 'K'))

{

--h;

}

if (my\_field[v][h] == 'X')

{

reply = "Wounded";

}

if (reply == "Killed")

{

v = ver; h = hor;

while ((h < 10) && (my\_field[v][h] == 'K'))

{

++h;

}

if (my\_field[v][h] == 'X')

{

reply = "Wounded";

}

}

}

}

}

else

{

reply = "Missed";

my\_field[ver][hor] = 'w';

}

cout << "Ваш ход сервер " << reply << "\n";

send\_message(reply + " " + to\_string(getpid()), socket); // здесь можно получить Kill если сервер попал

}

if (reply == "Lost")

{

cout << "Вы проиграли эту игру\n";

playing = false;

continue;

}

}

}

}

}

return 0;

}

**Демонстрация работы программы**

Игра долгая, поэтому демонстрацию прикладывать не буду.

**Выводы**

Благодаря данной лабораторной работе я создал свою консоль-серверную игру. Также я научился работать с очередями сообщений, которые помогали мне настроить общение между клиентом и сервером.