Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу

«Операционные системы»

МОРСКОЙ БОЙ ПРИ ПОМОЩИ ИМЕНОВАННЫХ КАНАЛОВ

Студент: Филиппов Владимир Михайлович
Группа: М8О–210Б–22
Вариант: 1
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант: Морской бой. Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. 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

Архитектура программы: сервер-клиент. К серверу в любом порядке подключаются процессы, которые могут выполнять в своем сеансе консоли выполнять какие пользователю угодно команды. Список системных вызовов:

1. **mkfifo** – создает особый FIFO-файл с названием pathname. mode определяет уровни доступа для FIFO.
2. **open** – открывает файл с определенными правами доступа.
3. **read** – читает данные из файла.
4. **write —** записывает данные в файл.

Общий метод и алгоритм решения.

В основе алгоритма для клиента и сервера лежит машина с конечным числом состояний. Пользователь запускает клиент, после чего пытается подключиться к стандартному именованному каналу. Если у него получается, сервер и клиент инициализируют свои персональные каналы ввода-вывода. Затем пользователь может вводить в консоль клиентского процесса какие-то запросы, после чего процесс либо проводит первичную обработку этих данных, либо сразу отправляет их на сервер. В программе сервера реализована очередь сообщений, то есть сервер сначала их читает, а потом в порядке очереди обрабатывает. При

каждом цикле чтения сервер проверяет, не отключился ли клиент при помощи kill(pid, 0). Если не получает ответа, то отправляет сообщение серверу, чтобы тот почистил зависимости процесса и если надо завершил игры.

Для некоторых команд используется БД SQLite, к которой обращается сервер при запросах на вывод статистики (выводит поля win-lose таблицы users), авторизацию (проверяет существует ли запись с заданным полем username), создание аккаунта (добавляет пустую запись с заданным полем username).

Кроме того, процесс сервера сам по себе хранит данные в виде unordered map с ключом pid, и значением — структура user, которая хранит некоторые данные о подключенном процессе. Активные игры так же хранятся в виде unordered_map. В этом словаре ключ — идентификатор игры, а значение — структура Game, которая хранит некоторые данные об участниках игры, а так же экземпляры класса Battledfield (о котором ниже). Это используется для поиска игры, а также реализации подключения более двух процессов.

Процесс игры реализуется при помощи класса Battlefield. Внутри он хранит двумерный массив типа char, каждый символ которого означает некоторое состояние клетки. Кроме того, он реализует добавление корабля и попытку его подбития.

Для реализации поставленной задачи необходимо:

1. Вспомнить принципы работы именованных каналов. Изучить принципы работы с БД, а также некоторые запросы на языке SQLite.
2. Написать программы, реализующие сервер и клиент, а также заголовочный файл, который содержит структуру сообщения.
3. Написать программу, которая обслуживала бы процесс игры.
4. Протестировать.

Основные файлы программы

server.cpp:

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "message.h"
#include <iostream>
#include <sqlite3.h>
#include <list>
#include <algorithm>
#include <queue>
#include <vector>
#include <chrono>
#include <errno.h>
#include <battlefield.h>
#include <sstream>
#include <sys/types.h>
#include <signal.h>
int active_game_counter = 1;
std::string myfifo_read_default = "/tmp/myfifo_c-s_def";
void tokenize(std::string input, int &direction, std::string &column, int &row, int
&type) {
    std::vector<std::string> res;
    std::istringstream is(input);
    std::string part;
    while (is >> part) {
        res.push_back(part);
    }
    if (res.size() == 2) {
        direction = 0;
        column = res[0];
        row = std::stoi(res[1]);
        type = 0;
    } else {
        direction = std::stoi(res[0]);
        column = res[1];
        row = std::stoi(res[2]);
        type = std::stoi(res[3]);
    }
}
void updateResult(const std::string& username, bool isWin) {
    sqlite3 *db;
    if (sqlite3_open("../db/users.db", &db) != SQLITE_OK) {
        std::cerr << "Ошибка при открытии базы данных: " << sqlite3_errmsg(db)
<< std::endl;
        return;
    }

    const char *sql = isWin ? "UPDATE users SET win = win + 1 WHERE username
= ?;"
                            : "UPDATE users SET lose = lose + 1 WHERE username = ?;";
    sqlite3_stmt *stmt;
    if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
```

```
        std::cerr << "Ошибка при подготовке SQL-запроса: " << sqlite3_errmsg(db)
<< std::endl;
        sqlite3_close(db);
        return;
    }
    if (sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_STATIC) != SQLITE_OK)
{
        std::cerr << "Ошибка при привязке параметра к запросу: " <<
sqlite3_errmsg(db) << std::endl;
        sqlite3_finalize(stmt);
        sqlite3_close(db);
        return;
    }
    if (sqlite3_step(stmt) != SQLITE_DONE) {
        std::cerr << "Ошибка при выполнении SQL-запроса: " <<
sqlite3_errmsg(db) << std::endl;
    }
    sqlite3_finalize(stmt);
    sqlite3_close(db);
}
std::pair<int, int> user_stats(const std::string& username) {
    sqlite3* db;
    if (sqlite3_open("../db/users.db", &db) == SQLITE_OK) {
        std::string query = "SELECT win, lose FROM users WHERE username=?";
        sqlite3_stmt* stmt;
        if (sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, NULL) == SQLITE_OK) {
            sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_STATIC);
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                int win = sqlite3_column_int(stmt, 0);
                int lose = sqlite3_column_int(stmt, 1);
                sqlite3_finalize(stmt);
                sqlite3_close(db);
                return std::make_pair(win, lose);
            } else {
                std::cerr << "User not found" << std::endl;
            }
        } else {
            std::cerr << "Error preparing statement" << std::endl;
        }
        sqlite3_close(db);
    } else {
        std::cerr << "Error opening database" << std::endl;
    }
    return std::make_pair(0, 0); // Return default values if error occurs
}
bool autorize(const std::string &username) {
    sqlite3 *db;
    char *zErrMsg = 0;
    int rc;
    rc = sqlite3_open("../db/users.db", &db);
    if (rc) {
        throw std::logic_error("can't open db");
```

```
    } else {
        std::string query = "SELECT COUNT(*) FROM users WHERE username='" +
username + "';";
        int result;
        rc = sqlite3_exec(db, query.c_str(), [](void* data, int argc, char** argv, char**
azColName) {
            int* count = static_cast<int*>(data);
            *count = atoi(argv[0]);
            return 0;
        }, &result, &zErrMsg);
        if (rc != SQLITE_OK) {
            throw std::logic_error("problem with db");
        }
        sqlite3_close(db);
        return result > 0;
    }
}
bool add_user(const std::string &username) {
    sqlite3 *db;
    int rc = sqlite3_open("../db/users.db", &db);
    if (rc) {
        std::cerr << "Ошибка при открытии базы данных: " << sqlite3_errmsg(db)
<< std::endl;
        sqlite3_close(db);
        return false;
    }
    int status = 0;
    // Проверяем, существует ли запись с указанным username
    std::string selectQuery = "SELECT COUNT(*) FROM users WHERE username = ?";
    sqlite3_stmt *selectStmt;
    rc = sqlite3_prepare_v2(db, selectQuery.c_str(), -1, &selectStmt, nullptr);
    if (rc != SQLITE_OK) {
        std::cerr << "Ошибка при подготовке SQL-запроса: " << sqlite3_errmsg(db)
<< std::endl;
        sqlite3_close(db);
        return false;
    }
    sqlite3_bind_text(selectStmt, 1, username.c_str(), -1, SQLITE_STATIC);
    rc = sqlite3_step(selectStmt);
    if (rc == SQLITE_ROW) {
        int count = sqlite3_column_int(selectStmt, 0);
        if (count > 0) {
            // Запись существует
            status = -1;
        } else {
            // Запись не существует, выполняем вставку
            std::string insertQuery = "INSERT INTO users (username, win, lose) VALUES
(?, ?, ?)";
            sqlite3_stmt *insertStmt;
            rc = sqlite3_prepare_v2(db, insertQuery.c_str(), -1, &insertStmt, nullptr);
            if (rc != SQLITE_OK) {
```

```
            std::cerr << "Ошибка при подготовке SQL-запроса: " <<
sqlite3_errmsg(db) << std::endl;
            return false;
        }
        sqlite3_bind_text(insertStmt, 1, username.c_str(), -1, SQLITE_STATIC);
        sqlite3_bind_int(insertStmt, 2, 0); // Значение win
        sqlite3_bind_int(insertStmt, 3, 0); // Значение lose
        rc = sqlite3_step(insertStmt);
        if (rc != SQLITE_DONE) {
            std::cerr << "Ошибка при выполнении SQL-запроса: " <<
sqlite3_errmsg(db) << std::endl;
            sqlite3_close(db);
            return false;
        }
        // Запись успешно вставлена
        status = 1;
    }
  } else {
    std::cerr << "Ошибка при выполнении SQL-запроса: " <<
sqlite3_errmsg(db) << std::endl;
    sqlite3_close(db);
    return false;
  }
  sqlite3_finalize(selectStmt);
  sqlite3_close(db);
  return (status == 1);
}
struct Game {
  int pid1, pid2;
  std::string username1;
  std::string username2;
  Battlefield *btf1, *btf2;
  Game() = default;
  Game(const int &pid1_tmp, const int &pid2_tmp, const std::string
&username1_tmp, const std::string &username2_tmp) {
    pid1 = pid1_tmp; pid2 = pid2_tmp;
    username1 = username1_tmp;
    username2 = username2_tmp;
    btf1 = new Battlefield(); btf2 = new Battlefield();
  }
};
struct User {
  std::string username;
  int pid;
  int fdW;
  int fdR;
  int game_index;
  int game_status;
  User() = default;
  User(const std::string username_tmp, const int &pid_tmp, const int &fdW_tmp,
const int &game_index_tmp, const int &game_status_tmp, const int &fdR_tmp) {
    username = username_tmp;
```

```
        pid = pid_tmp;
        fdW = fdW_tmp; fdR = fdR_tmp;
        game_index = game_index_tmp;
        game_status = game_status_tmp;
    }
};
class Server {
    public:
        std::vector<int> _fdR;
        std::unordered_map<int, User> _users;
        std::queue<Message> _msgs;
        std::unordered_map<int, Game> _active_games;
        int fd_default_read;
        Server() {
            std::cout << "Initialize default channel" << std::endl;
            mkfifo(myfifo_read_default.c_str(), 0666);
            fd_default_read = open(myfifo_read_default.c_str(), O_RDONLY |
O_NONBLOCK);
            _fdR.push_back(fd_default_read);
            std::cout << "Default channel initialized" << std::endl;
        }
        ~Server() {
            for (auto fd: _fdR) {
                close(fd);
            }
            for (auto pid: _users) {
                close(pid.second.fdW);
            }

        }
        auto search_by_username(const std::string &username) {
            for (auto iter = _users.begin(); iter != _users.end(); iter++) {
                if (iter->second.username == username) {
                    return iter;
                }
            }
            return _users.end();
        }
        bool is_authorized_by_username(const std::string &username) {
            bool flag = false;
            for (auto user: _users) {
                if (user.second.username == username) {
                    flag = true;
                }
            }
            return flag;
        }
        bool is_authorized_by_pid(const int &pid) {
            auto search = _users.find(pid);
            return search != _users.end();
        }
        void try_recv() {
```

```
            auto start = std::chrono::system_clock::now();
            for (auto user: _users) {
                if (kill(user.second.pid, 0) != 0) {
                    Message msg_from_client(clear, "", user.second.pid);
                    _msgs.push(msg_from_client);
                }
            }
            while(true) {
                if
(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::n
ow() - start).count() > 100) {
                    break;
                }
                for (auto fd: _fdR) {
                    Message msg_from_client;
                    int num_of_bytes = recv(fd, msg_from_client);
                    if (num_of_bytes > 0) {
                        _msgs.push(msg_from_client);
                    }
                }
            }
        }
        void send_to(int pid, Message &msg) {
            send(_users[pid].fdW, msg);
        }
        void exec() {
            while (1) {
                this->try_recv();
                while (!_msgs.empty()) {
                    Message msg_to_make;
                    msg_to_make = _msgs.front();
                    _msgs.pop();
                    if (msg_to_make._cmd == create_user) {
                        bool success = add_user(msg_to_make._data);
                        if (success) {
                            Message msg_to_client(Commands::success, msg_to_make._data,
-1);
                            send_to(msg_to_make._pid, msg_to_client);
                        } else {
                            Message msg_to_client(Commands::fail, msg_to_make._data, -1);
                            send_to(msg_to_make._pid, msg_to_client);
                        }
                    } else if (msg_to_make._cmd == Commands::clear) {
                        std::cout << "proccess with pid = " << msg_to_make._pid << "
disconnect" << std::endl;
                        auto game_of_disconnected =
_active_games.find(_users[msg_to_make._pid].game_index);
                        if (game_of_disconnected != _active_games.end()) {
                            if (game_of_disconnected->second.pid1 == msg_to_make._pid) {
                                _users[game_of_disconnected->second.pid2].game_status = 0;
                                Message msg_to_client(Commands::disconnect, "", -1);
                                send_to(game_of_disconnected->second.pid2, msg_to_client);
```

```
                    } else if (game_of_disconnected->second.pid2 ==
msg_to_make._pid) {
                        _users[game_of_disconnected->second.pid1].game_status = 0;
                        Message msg_to_client(Commands::disconnect, "", -1);
                        send_to(game_of_disconnected->second.pid1, msg_to_client);
                    }
                    _active_games.erase(_users[msg_to_make._pid].game_index);
                }
                close(_users[msg_to_make._pid].fdW);
                close(_users[msg_to_make._pid].fdR);
                _fdR.erase(std::find(_fdR.begin(), _fdR.end(),
_users[msg_to_make._pid].fdR));
                _users.erase(_users.find(msg_to_make._pid));
            } else if (msg_to_make._cmd == Commands::login) {
                bool not_available_login =
is_authorized_by_username(msg_to_make._data);
                if (not_available_login) {
                    Message msg_to_client(Commands::fail, "This username is already
logged", -1);
                    send_to(msg_to_make._pid, msg_to_client);
                    continue;
                }
                bool success = autorize(msg_to_make._data);
                if (success) {
                    _users[msg_to_make._pid].username =
std::string(msg_to_make._data);
                    Message msg_to_client(Commands::success, msg_to_make._data,
-1);
                    send_to(msg_to_make._pid, msg_to_client);
                } else {
                    Message msg_to_client(Commands::fail, "No account with this
username", -1);
                    send_to(msg_to_make._pid, msg_to_client);
                }
                std::cout << "Login:\n";
                for (auto iter: _users) {
                    std::cout << iter.second.username << ' ' << iter.second.pid <<
std::endl;
                }
            } else if (msg_to_make._cmd == Commands::stats) {
                if (is_authorized_by_pid(msg_to_make._pid)) {
                    auto [win, lose] = user_stats(msg_to_make._data);
                    Message msg_to_client(Commands::success, std::to_string(win) +
' ' + std::to_string(lose), -1);
                    send_to(msg_to_make._pid, msg_to_client);
                } else {
                    Message msg_to_client(Commands::fail, "not_autorized", -1);
                    send_to(msg_to_make._pid, msg_to_client);
                }
            } else if (msg_to_make._cmd == Commands::connect) {
                std::cout << "Making connection pipe for proc " <<
msg_to_make._pid << std::endl;
```

```
                    std::string name_fifo_read = "/tmp/myfifo_c-s_" +
std::to_string(msg_to_make._pid);
                    std::string name_fifo_write = "/tmp/myfifo_s-c_" +
std::to_string(msg_to_make._pid);
                    mkfifo(name_fifo_read.c_str(), 0666);
                    mkfifo(name_fifo_write.c_str(), 0666);
                    int fd_write = open(name_fifo_write.c_str(), O_WRONLY);
                    int fd_read = open(name_fifo_read.c_str(), O_RDONLY |
O_NONBLOCK);
                    _fdR.push_back(fd_read);
                    User tmp_user{"", msg_to_make._pid, fd_write, 0, 0, fd_read};
                    _users.insert({msg_to_make._pid, tmp_user});
                    auto elem = _users.find(msg_to_make._pid);
                    Message msg_to_client(Commands::success, "Succesfully
connected", -1);
                    if (elem != _users.end()) {
                        send_to(elem->second.pid, msg_to_client);
                    }
                } else if (msg_to_make._cmd == Commands::find) {
                    std::cout << "Find opponent for " <<
_users[msg_to_make._pid].username << std::endl;
                    if (is_authorized_by_username(msg_to_make._data)) {
                        if (search_by_username(std::string(msg_to_make._data))-
>second.game_status == 1) {
                            Message msg_to_client1(Commands::success, "Succesfully
connected1", -1);
                            Message msg_to_client2(Commands::success, "Succesfully
connected2", -1);
                            auto *user1 = &_users[msg_to_make._pid];
                            auto *user2 =
&search_by_username(std::string(msg_to_make._data))->second;
                            user1->game_status = 2;
                            user2->game_status = 2;
                            user1->game_index = active_game_counter;
                            user2->game_index = active_game_counter;
                            Game game(user1->pid, user2->pid, user1->username, user2-
>username);
                            _active_games.insert({active_game_counter, game});
                            active_game_counter++;
                            send_to(user2->pid, msg_to_client2);
                            send_to(user1->pid, msg_to_client1);
                        } else {
                            _users[msg_to_make._pid].game_status = 1;
                        }
                    } else {
                        Message msg_to_client(Commands::fail, "Opponent not online", -
1);
                        send_to(msg_to_make._pid, msg_to_client);
                    }
                } else if (msg_to_make._cmd == Commands::place_ship) {
                    std::cout << msg_to_make._data << std::endl;
                    std::string column; int row; int type; int direction;
```

```
                tokenize(std::string(msg_to_make._data), direction, column, row,
type);
                auto *deduction_proccess =
&_active_games[_users[msg_to_make._pid].game_index];
                if (deduction_proccess->pid1 == msg_to_make._pid) {
                    std::cout << "Place on first" << std::endl;
                    deduction_proccess->btf1->place_ship(column.c_str()[0], row,
Direction(direction), ShipType(type));
                    Message msg_to_client1(Commands::success, "Succesfully
placed", -1);
                    send_to(msg_to_make._pid, msg_to_client1);
                } else if (deduction_proccess->pid2 == msg_to_make._pid) {
                    std::cout << "Place on second" << std::endl;
                    deduction_proccess->btf2->place_ship(column.c_str()[0], row,
Direction(direction), ShipType(type));
                    Message msg_to_client1(Commands::success, "Succesfully
placed", -1);
                    send_to(msg_to_make._pid, msg_to_client1);
                } else {
                    std::cout << "Proccess was disconnected and it was last attempt
to do smt" << std::endl;
                    continue;
                }
                deduction_proccess->btf1->print();
                deduction_proccess->btf2->print();

            } else if (msg_to_make._cmd == Commands::ready_to_play) {
                auto *deduction_proccess =
&_active_games[_users[msg_to_make._pid].game_index];
                Message msg_to_client1(Commands::success, "Ready to play", 1);
                Message msg_to_client2(Commands::success, "Ready to play", 2);
                if (deduction_proccess->pid1 == msg_to_make._pid &&
_users[deduction_proccess->pid2].game_status == 3) {
                    send_to(deduction_proccess->pid1, msg_to_client1);
                    send_to(deduction_proccess->pid2, msg_to_client2);
                }
                if (deduction_proccess->pid2 == msg_to_make._pid &&
_users[deduction_proccess->pid1].game_status == 3) {
                    send_to(deduction_proccess->pid1, msg_to_client1);
                    send_to(deduction_proccess->pid2, msg_to_client2);
                }
                _users[msg_to_make._pid].game_status = 3;
            } else if (msg_to_make._cmd == Commands::kill_ship) {
                auto *deduction_proccess =
&_active_games[_users[msg_to_make._pid].game_index];
                std::string column; int row; int type; int direction;
                tokenize(std::string(msg_to_make._data), direction, column, row,
type);
                bool success;
                if (deduction_proccess->pid1 == msg_to_make._pid) {
                    success = deduction_proccess->btf2->try_kill(column.c_str()[0],
row);
```

```
                    if (success) {
                        if (deduction_proccess->btf2->end_game_check()) {
                            Message msg_to_client1(Commands::end_game, "You win",
1);
                            Message msg_to_client2(Commands::end_game, "You lose",
2);
                            updateResult(deduction_proccess->username1, 1);
                            updateResult(deduction_proccess->username2, 0);

_active_games.erase(_active_games.find(_users[msg_to_make._pid].game_index));
                            _users[deduction_proccess->pid2].game_status = 0;
                            _users[deduction_proccess->pid1].game_status = 0;
                            _users[deduction_proccess->pid2].game_index = 0;
                            _users[deduction_proccess->pid1].game_index = 0;
                            send_to(deduction_proccess->pid1, msg_to_client1);
                            send_to(deduction_proccess->pid2, msg_to_client2);
                        } else {
                            Message msg_to_client1(Commands::success, "Catch", 1);
                            Message msg_to_client2(Commands::success, column, row);
                            send_to(deduction_proccess->pid1, msg_to_client1);
                            send_to(deduction_proccess->pid2, msg_to_client2);
                        }
                    } else {
                        Message msg_to_client1(Commands::fail, "Failed attempt", 1);
                        Message msg_to_client2(Commands::fail, column, row);
                        send_to(deduction_proccess->pid1, msg_to_client1);
                        send_to(deduction_proccess->pid2, msg_to_client2);
                    }
                } else if (deduction_proccess->pid2 == msg_to_make._pid) {
                    success = deduction_proccess->btf1->try_kill(column.c_str()[0],
row);
                    if (success) {
                        if (deduction_proccess->btf1->end_game_check()) {
                            Message msg_to_client1(Commands::end_game, "You lose",
1);
                            Message msg_to_client2(Commands::end_game, "You win",
2);

_active_games.erase(_active_games.find(_users[msg_to_make._pid].game_index));
                            _users[deduction_proccess->pid2].game_status = 0;
                            _users[deduction_proccess->pid1].game_status = 0;
                            _users[deduction_proccess->pid2].game_index = 0;
                            _users[deduction_proccess->pid1].game_index = 0;
                            updateResult(deduction_proccess->username2, 1);
                            updateResult(deduction_proccess->username1, 0);
                            send_to(deduction_proccess->pid1, msg_to_client1);
                            send_to(deduction_proccess->pid2, msg_to_client2);
                        } else {
                            Message msg_to_client2(Commands::success, "Catch", 1);
                            Message msg_to_client1(Commands::success, column, row);
                            send_to(deduction_proccess->pid1, msg_to_client1);
                            send_to(deduction_proccess->pid2, msg_to_client2);
```

```
                    }
                } else {
                    Message msg_to_client2(Commands::fail, "Failed attempt", 1);
                    Message msg_to_client1(Commands::fail, column, row);
                    send_to(deduction_proccess->pid1, msg_to_client1);
                    send_to(deduction_proccess->pid2, msg_to_client2);
                }
            } else {
                std::cout << "Proccess was disconnected and it was last attempt
to do smt" << std::endl;
            }
            deduction_proccess->btf1->print();
            deduction_proccess->btf2->print();
        }
    }
  }
};

int main() {
    Server server;
    server.exec();
}
```

client.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>
#include <string>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "message.h"
#include <iostream>
#include "battlefield.h"
void text_art() {
    std::cout << '\n' << '\n';
    std::cout << "\t\t                ########################\n";
    std::cout << "\t\t              #####             ######\n";
    std::cout << "\t\t            ####                  ###\n";
    std::cout << "\t\t           ###                    ###\n";
    std::cout << "\t\t          ##                      ##\n";
    std::cout << "\t\t         ##                       ##\n";
    std::cout << "\t\t        ##                        ##\n";
    std::cout << "\t\t       ##                         ##\n";
    std::cout << "\t\t       ##  ##                     ##  ##\n";
    std::cout << "\t\t       ##  ##                     ##  ##\n";
```

```
   std::cout << "\t\t         ##  ##                      ##  ##\n";
   std::cout << "\t\t         ##  ##                     ##   ##\n";
   std::cout << "\t\t          ##  ##                     ##  ##\n";
   std::cout << "\t\t          ##  ##      ###      ###      ##  ##\n";
   std::cout << "\t\t           ## ##  ##########     ##########  ## ## \n";
   std::cout << "\t\t            #### ############    ############ ####\n";
   std::cout << "\t\t             ###  ##########     ##########  ##\n";
   std::cout << "\t\t    ####      #   #########      #########   #      ####\n";
   std::cout << "\t\t   ## ##     ##   ######    # #   #######   ##    ### ##\n";
   std::cout << "\t\t   ##   ##    ##          ## ##          ##   ###   ##\n";
   std::cout << "\t\t  ##    ####  ##         ### ###         ## #####    ##\n";
   std::cout << "\t\t##       ########       #### ###       ########       ###\n";
   std::cout << "\t\t##   ###      ########    #### ###    ########     #### ###\
n";
   std::cout << "\t\t ###########     #######   ##  ##   ######
###########\n";
   std::cout << "\t\t        #####   ##  ##         ## # ##   #####\n";
   std::cout << "\t\t          #####  # # ###        ### ###  #####\n";
   std::cout << "\t\t            ####  #  # # # # # # # #  # #####\n";
   std::cout << "\t\t              ##  ## # # # # # # # ####  #\n";
   std::cout << "\t\t             ####  # # # # # # # # ###  ######\n";
   std::cout << "\t\t           ##### ##   ##### # #####   ##  #####\n";
   std::cout << "\t\t    ##########     ###             ##     ##########\n";
   std::cout << "\t\t   ##         ######           #######         ##\n";
   std::cout << "\t\t    ##      ####   #####      #####   ####      ##\n";
   std::cout << "\t\t     ###   ####       #############       ####   ###\n";
   std::cout << "\t\t      ##  ##                         ##  ##\n";
   std::cout << "\t\t      ## ##                           ## ##\n";
   std::cout << "\t\t       ####                            ####\n" << std::endl << '\n'
<< '\n';
   std::cout << "  /$$$$$$  /$$$$$$$$  /$$$$$$        /$$$$$$$  /$$$$$$  /$$$$$$$
$/$$$$$$$$/$$       /$$$$$$$$\n";
   std::cout << " /$$__  $$| $$_____/ /$$__  $$       | $$__  $$ /$$__  $$|__
$$__/__  $$__/ $$     | $$_____/\n";
   std::cout << "| $$  \\__/| $$     | $$  \\ $$       | $$  \\ $$| $$  \\ $$  | $$    | $$  | $$
| $$\n";
   std::cout << "|  $$$$$$ | $$$$$   | $$$$$$$$ /$$$$$$| $$$$$$$ | $$$$$$$$   | $$
| $$  | $$     | $$$$$\n";
   std::cout << " \\____  $$| $$__/   | $$__  $$|_____/| $$__  $$| $$__  $$  | $$    |
$$  | $$     | $$__/\n";
   std::cout << " /$$  \\ $$| $$      | $$  | $$       | $$  \\ $$| $$  | $$  | $$    | $$  | $$
| $$\n";
```

```
    std::cout << "|  $$$$$$/| $$$$$$$$| $$  | $$        | $$$$$$$/| $$  | $$   | $$     | $$
| $$$$$$$$| $$$$$$$$\n";
    std::cout << " \\_______/ |________/|__/  |__/        |_______/ |__/  |__/   |__/     |__/
|________/|________/\n" << std::endl << '\n' << '\n';
}
std::string myfifo_write_default = "/tmp/myfifo_c-s_def";
std::string myfifo_read = "/tmp/myfifo_s-c_" + std::to_string(getpid());
std::string myfifo_write = "/tmp/myfifo_c-s_" + std::to_string(getpid());
int fdR, fdW;
std::string username = "";
bool default_connection() {
    mkfifo(myfifo_read.c_str(), 0666);
    mkfifo(myfifo_write.c_str(), 0666);
    mkfifo(myfifo_write_default.c_str(), 0666);
    int fd_write_default = open(myfifo_write_default.c_str(), O_WRONLY);
    std::string reply;
    Message msg_to_server(Commands::connect, "", getpid());
    Message reply_from_server(Commands::fail, reply, 0);
    send(fd_write_default, msg_to_server);
    if (close(fd_write_default) == -1) {
        throw std::logic_error("bad with close");
    }
    fdR = open(myfifo_read.c_str(), O_RDONLY);
    fdW = open(myfifo_write.c_str(), O_WRONLY);
    recv(fdR, reply_from_server);
    if (reply_from_server._cmd == success) {
        return true;
    } else {
        close(fdR);
        close(fdW);
        std::cout << "Problem with connection" << std::endl;
        return false;
    }
}

bool start_game(Battlefield &btf) {
    while (btf.one_amount() < 1 && btf.two_amount() < 1 && btf.three_amount() <
1 && btf.four_amount() < 1) {
        btf.print();
        int a,b,c,d;
        std::cout << "Please, enter direction (0 for horizontal, 1 for " \
```

```
            "vertical), coordinates of ship and his type.\nExample: 0 A 1 4" <<
std::endl;
        char column; int row; int type; int direction;
        std::cin >> direction >> column >> row >> type;
        std::string msg = std::to_string(direction) + ' ' + column + ' ' +
std::to_string(row) + ' ' + std::to_string(type);
        if (btf.place_ship(column, row, Direction(direction), ShipType(type)) == 0) {
            Message msg_to_server(Commands::place_ship, msg, getpid());
            send(fdW, msg_to_server);
            Message reply(Commands::fail, "", -1);
            recv(fdR, reply);
            if (reply._cmd != success) {
                if (reply._cmd == disconnect) {
                    return 0;
                }
                throw std::logic_error("Not placed");
            }
        } else {
            std::cout << "\nYou have done something wrong" << std::endl;
        }
    }
    btf.print();
    Message msg_to_server(Commands::ready_to_play, "", getpid());
    send(fdW, msg_to_server);
    return 1;
}
bool operate_game(Battlefield &own, Battlefield &opponent, int number) {
    if (number == 1) {
        while (1) {
            own.print(); opponent.print();
            std::string column; int row;
            std::cout << "Please, enter coordinates of ship.\nExample: B 7 " <<
std::endl;
            std::cin >> column >> row;
            std::string msg = column + ' ' + std::to_string(row);
            Message msg_to_server(Commands::kill_ship, msg, getpid());
            Message is_killed(Commands::fail, "", -1);
            send(fdW, msg_to_server);
            recv(fdR, is_killed);
            if (is_killed._cmd == success) {
                std::cout << is_killed._data << std::endl;
                opponent.set(is_killed._data[0], is_killed._pid, '+');
```

```
            } else if (is_killed._cmd == end_game) {
                std::cout << is_killed._data << std::endl;
                return 1;
            } else if (is_killed._cmd == fail) {
                opponent.set(column[0], row, '*');
                std::cout << is_killed._data << std::endl;
            } else if (is_killed._cmd == disconnect ){
                return 0;
            }
            own.print(); opponent.print();
            std::cout << "Waiting opponent" << std::endl;
            Message opponent_move(Commands::fail, "", -1);
            recv(fdR, opponent_move);
            std::cout << opponent_move._cmd << ' ' << opponent_move._data << ' ' <<
opponent_move._pid << std::endl;
            if (opponent_move._cmd == success) {
                std::cout << opponent_move._data << std::endl;
                own.set(opponent_move._data[0], opponent_move._pid, '+');
            } else if (opponent_move._cmd == end_game) {
                std::cout << opponent_move._data << std::endl;
                return 1;
            } else if (opponent_move._cmd == fail) {
                own.set(opponent_move._data[0], opponent_move._pid, '*');
                std::cout << opponent_move._data << std::endl;
            } else if (opponent_move._cmd == disconnect) {
                return 0;
            }
        }
    } else {
        while (1) {
            own.print(); opponent.print();
            std::cout << "Waiting opponent" << std::endl;
            Message opponent_move(Commands::fail, "", -1);
            recv(fdR, opponent_move);
            if (opponent_move._cmd == success) {
                own.set(opponent_move._data[0], opponent_move._pid, '+');
                std::cout << opponent_move._data << std::endl;
            } else if (opponent_move._cmd == end_game) {
                std::cout << opponent_move._data << std::endl;
                return 1;
            } else if (opponent_move._cmd == fail) {
                own.set(opponent_move._data[0], opponent_move._pid, '*');
```

```
            std::cout << opponent_move._data << std::endl;
        } else if (opponent_move._cmd == disconnect) {
            return 0;
        } else {
            throw std::logic_error("unknown command");
        }
        own.print(); opponent.print();
        std::string column; int row;
        std::cout << "Please, enter coordinates of ship.\nExample: B 7 " <<
std::endl;
        std::cin >> column >> row;
        std::string msg = column + ' ' + std::to_string(row);
        Message msg_to_server(Commands::kill_ship, msg, getpid());
        Message is_killed(Commands::fail, "", -1);
        send(fdW, msg_to_server);
        recv(fdR, is_killed);
        if (is_killed._cmd == success) {
            std::cout << is_killed._data << std::endl;
            opponent.set(is_killed._data[0], is_killed._pid, '+');
        } else if (is_killed._cmd == end_game) {
            std::cout << is_killed._data << std::endl;
            return 1;
        } else if (is_killed._cmd == fail) {
            opponent.set(column[0], row, '*');
            std::cout << is_killed._data << std::endl;
        } else if (is_killed._cmd == disconnect) {
            return 0;
        } else {
            throw std::logic_error("unknown cmd");
        }
    }
  }
}

int main(int argc, char* argv[]) {
  text_art();
  default_connection();
  std::cout << "Welcome to menu! Follow instruction:\n\n";
  std::cout << "| [login *username*] to login in account |    | [create *username*]
to create new account |    |";
  std::cout << " [find *opponent*] to find opponent |    | [stats 1] for print your
account stats |\n" << std::endl;
```

```
    std::string input, login;
    while (1) {
        std::cout << "Place for your command: ";
        std::cin >> input >> login;
        if (input == "login") {
            if (username != "") {
                std::cout << "\nYou are already loged in " + username << ' ' << '\n' <<
std::endl;
                continue;
            }
            std::string reply;
            Message msg_to_server(Commands::login, login, getpid());
            Message reply_from_server(Commands::login, reply, 0);
            send(fdW, msg_to_server);
            recv(fdR, reply_from_server);
            if (reply_from_server._cmd == success) {
                std::cout << "\nAccout succesfully login in " << reply_from_server._data
<< std::endl;
                username = login;
            } else if (reply_from_server._cmd == fail) {
                std::cout << '\n' << reply_from_server._data << std::endl;
            }
        } else if (input == "create") {
            std::string reply;
            Message msg_to_server(Commands::create_user, login, getpid());
            Message reply_from_server(create_user, reply, -1);
            send(fdW, msg_to_server);
            recv(fdR, reply_from_server);
            if (reply_from_server._cmd == success) {
                std::cout << "\nAccount succesfuly created " << reply_from_server._data
<< std::endl;
            } else if (reply_from_server._cmd == fail) {
                std::cout << "\nAccount is exist already " << reply_from_server._data
<< std::endl;
            }
        } else if (input == "stats") {
            if (username == "") {
                std::cout << "\nYou are not authorized" << std::endl;
                continue;
            }
            Message msg_to_server(Commands::stats, username, getpid());
            send(fdW, msg_to_server);
```

```
            Message reply_from_server(create_user, "", -1);
            recv(fdR, reply_from_server);
            if (reply_from_server._cmd == success) {
                std::cout << "\nStats of: " << username << std::endl;
                std::cout << "Win-Lose: " << reply_from_server._data << std::endl;
            } else if (reply_from_server._cmd == fail) {
                std::cout << "Shit happened " << reply_from_server._data << std::endl;
            }
        } else if (input == "find") {
            if (username == "") {
                std::cout << "You are not authorized" << std::endl;
                continue;
            }
            if (username == login) {
                std::cout << "You can't play with yourself" << std::endl;
                continue;
            }
            Message msg_to_server(Commands::find, login, getpid());
            send(fdW, msg_to_server);
            Message reply_from_server(fail, "", -1);
            recv(fdR, reply_from_server);
            if (reply_from_server._cmd == success) {
                Battlefield own_battlefield;
                Battlefield opponent_battlefield;
                if (!start_game(own_battlefield)) {
                    std::cout << "Opponent disconnected" << std::endl;
                    fsync(fdR);
                    continue;
                }
                recv(fdR, reply_from_server);
                if (reply_from_server._cmd == success) {
                    if (!operate_game(own_battlefield, opponent_battlefield,
reply_from_server._pid)) {
                        std::cout << "Opponent disconnected" << std::endl;
                        fsync(fdR);
                        continue;
                    }
                }
            } else {
                std::cout << '\n' << reply_from_server._data << std::endl;
            }
        }
```

```
        std::cout << std::endl;
    }
}
```

message.h

```
#pragma once
#include <fcntl.h>
#include <unistd.h>
#include <string>
#include <cstring>
#include <vector>
enum Commands {
    login = 0,
    create_user = 1,
    success = 3,
    fail = 4,
    stats = 5,
    connect = 6,
    find,
    place_ship,
    ready_to_play,
    kill_ship,
    end_game,
    clear,
    disconnect
};
struct Message {
    Message(Commands cmd, std::string msg, int pid) {
        _cmd = cmd;
        strcpy(_data, msg.c_str());
        _pid = pid;
    }
    Message() = default;
    Commands _cmd;
    char _data[64];
    int _pid;
};

void send(int fd, Message &msg) {
    write(fd, &msg, sizeof(msg));
}
size_t recv(int fd, Message &msg) {
```

```
    return read(fd, &msg, sizeof(msg));
}
```

battlefield.h:

```
#pragma once
#include <array>
#include <iostream>
#include <unordered_map>
#include <vector>
#define BTF_SIZE 10
#define SIZE_ERROR 2
#define PLACEMENT_ERROR 3
#define AMOUNT_OF_SHIPS_ERROR 4
enum Direction {
    Horisontal = 0,
    Vertical = 1,
};
enum DirectionOfShip {
    left, right, up, down, left_right, up_down
};
enum ShipType {
    four_square = 4,
    three_square = 3,
    two_square = 2,
    one_square = 1
};
class Battlefield {
    private:
        std::array<std::array<char, BTF_SIZE>, BTF_SIZE> _btf;
        std::unordered_map<ShipType, int> ships_amount;
    public:
        Battlefield();
        void print();
        int place_ship(char x, int y, Direction dir, ShipType type);
        bool try_kill(char x, int y);
        int four_amount() { return ships_amount.find(four_square)->second;}
        int three_amount() { return ships_amount.find(three_square)->second;}
        int two_amount() { return ships_amount.find(two_square)->second;}
        int one_amount() { return ships_amount.find(one_square)->second;}
        bool end_game_check();
        void set(char column, int x, char mark);
};
```

battlefield.cpp

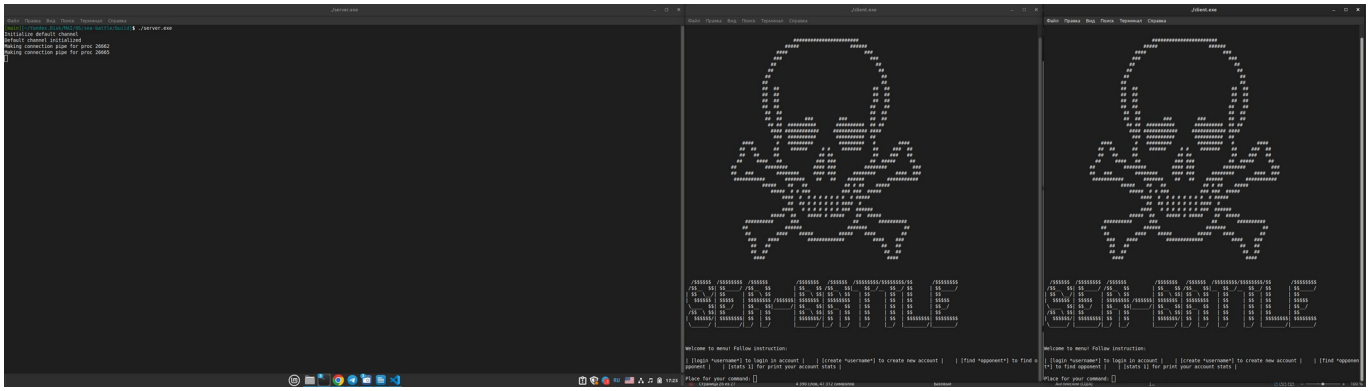
```
#include "battlefield.h"
Battlefield::Battlefield() {
    for (auto &row : _btf) {
        for (auto &element : row) {
            element = '0';
        }
    }
    ships_amount = {
        {four_square, 0},
        {three_square, 0},
        {two_square, 0},
        {one_square, 0}
    };
}
void Battlefield::print() {
    char column = 'A';
    std::cout << "   ";
    for (auto i: _btf) {
        std::cout << column << ' ';
        column++;
    }
    std::cout << std::endl;
    int row = 1;
    for (auto i: _btf) {
        if (row >= 10) {
            std::cout << row << " ";
        } else {
            std::cout << row << "  ";
        }
        for (auto j: i) {
            std::cout << j << ' ';
        }
        row++;
        std::cout << std::endl;
    }
    std::cout << std::endl;
}
char intToChar(int number) {
    return '0' + number;
}
bool Battlefield::end_game_check() {
```

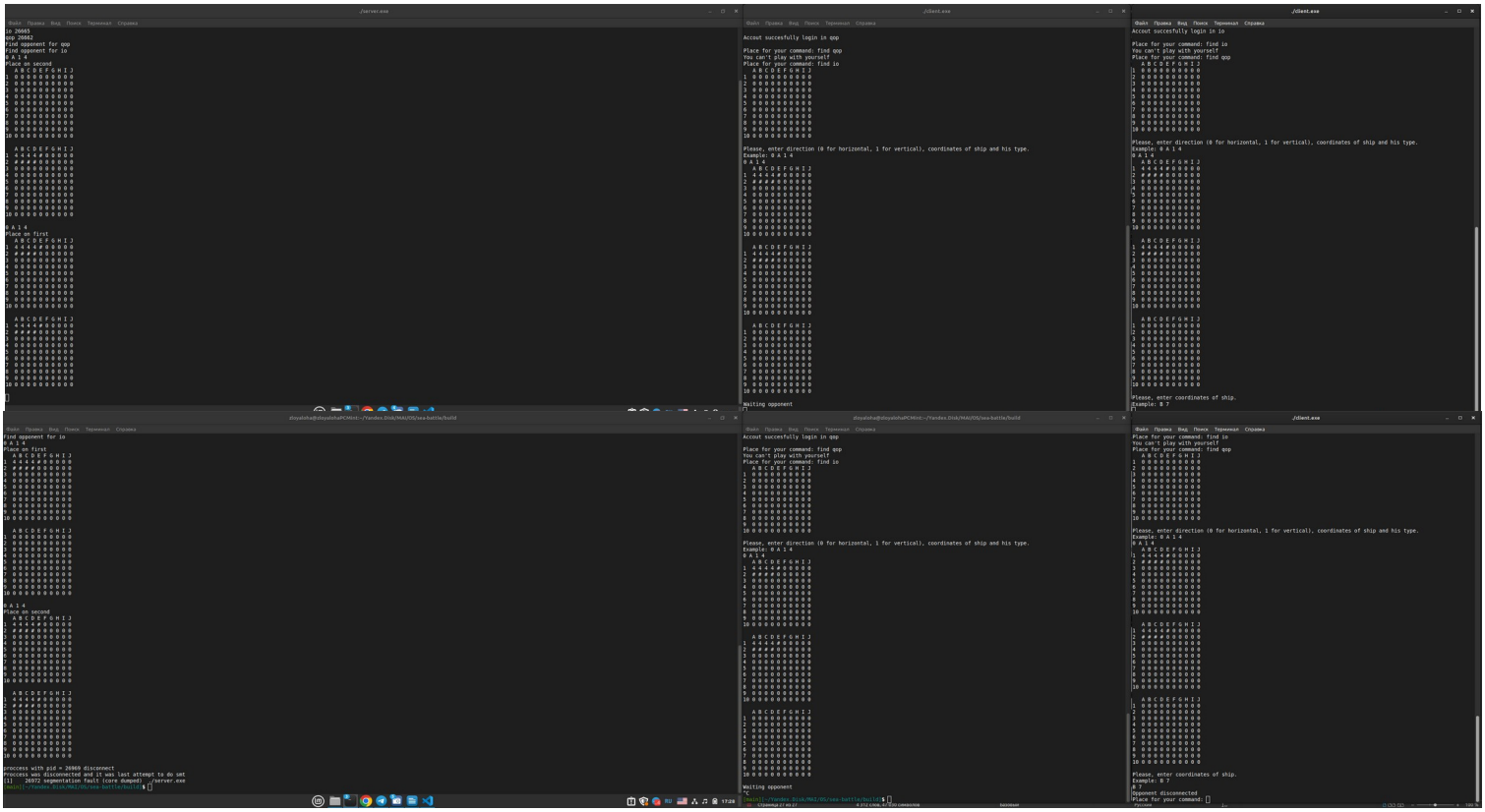
```
    for (auto i: _btf) {
        for (auto j: i) {
            if (j >= '1' && j <= '4') {
                return false;
            }
        }
    }
    return true;
}
int Battlefield::place_ship(char column_char, int row, Direction dir, ShipType size)
{
    int column = column_char - 'A';
    row--;
    char char_ship = intToChar(int(size));
    if (size == four_square && ships_amount.find(four_square)->second == 1) {
        return AMOUNT_OF_SHIPS_ERROR;
    }
    if (size == three_square && ships_amount.find(three_square)->second == 2) {
        return AMOUNT_OF_SHIPS_ERROR;
    }
    if (size == two_square && ships_amount.find(two_square)->second == 3) {
        return AMOUNT_OF_SHIPS_ERROR;
    }
    if (size == one_square && ships_amount.find(one_square)->second == 4) {
        return AMOUNT_OF_SHIPS_ERROR;
    }
    if (column < 0 || row < 0 || row > BTF_SIZE || column > BTF_SIZE) {
        return SIZE_ERROR;
    }
    if (dir == Direction::Horisontal) {
        for (size_t i = column; i < column + size; ++i) {
            if (_btf[row][i] != '0') {
                return PLACEMENT_ERROR;
            }
        }
        if (column + size > BTF_SIZE) {
            return SIZE_ERROR;
        }
        ships_amount[size] = ships_amount[size] + 1;
        for (int i = column; i < column + size; i++) {
            _btf[row][i] = char_ship;
            if (i == column) {
```

```
            if (i - 1 >= 0) {
                _btf[row][i - 1] = '#';
            }
        }
        if (i == column + size - 1) {
            if (i + 1 < BTF_SIZE) {
                _btf[row][i + 1]  = '#';
            }
        }
        if (row + 1 < BTF_SIZE) {
            _btf[row + 1][i] = '#';
        }
        if (row - 1 >= 0) {
            _btf[row - 1][i] = '#';
        }
    }
    return 0;
} else {
    for (int i = row; i < row + size; ++i) {
        if (_btf[i][column] != '0') {
            return PLACEMENT_ERROR;
        }
    }
    if (row + size > BTF_SIZE) {
        return SIZE_ERROR;
    }
    ships_amount[size]++;
    for (int i = row; i < row + size; ++i) {
        _btf[i][column] = char_ship;
        if (i == row) {
            if (i - 1 >= 0) {
                _btf[i - 1][column] = '#';
            }
        }
        if (i == row + size - 1) {
            if (i + 1 < BTF_SIZE) {
                _btf[i + 1][column]  = '#';
            }
        }
        if (column + 1 < BTF_SIZE) {
            _btf[i][column + 1] = '#';
        }
```

```
            if (column - 1 >= 0) {
                _btf[i][column - 1]  = '#';
            }
        }
        return 0;
    }
}
void Battlefield::set(char column, int tmp, char mark) {
    int y = column - 'A';
    int x = tmp - 1;
    _btf[x][y] = mark;
}
bool Battlefield::try_kill(char column, int y) {
    int x = y - 1;
    y = column - 'A';
    if (_btf[x][y] >= '1' && _btf[x][y] <= '4') {
        _btf[x][y] = '+';
        return true;
    } else {
        _btf[x][y] = '*';
        return false;
    }
}
```

Пример работы





Вывод

Для меня курсовой проект был хорошим способом попрактиковаться в навыках, полученных втечение семестра. В особенности при выполнении Л.Р. 1 и 5-7. Улучшений для данной реализации игры морской бой может быть много. Например улучшить авторизацию, поиск игры, сам процесс игры может быть немного оптимизирован. Можно оптимизировать сервер, для того, чтобы для обработки каждого запроса создавался отдельный поток. Кроме того, сам код требует рефакторинга для более удобного чтения и поддерживания.

Это мой первый опыт создания чего-то большого и не совсем учебного. Поэтому я получил большое удовольствие. Кроме того, я столкнулся со многими проблемами, которые помогли мне лучше разобраться с работой с файлами, работой сервера и т.д.