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**

**“Проектирование консольной клиент-серверной игры”**

Студент: Прохоров Данила Михайлович

Группа: М8О-208Б-20

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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/dmprokhorov

**Постановка задачи**

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

Базовый функционал должен быть следующим:

• Клиент может присоединиться к серверу, введя логин (у меня это ID процесса).

• С сервером одновременно много играть несколько клиентов.

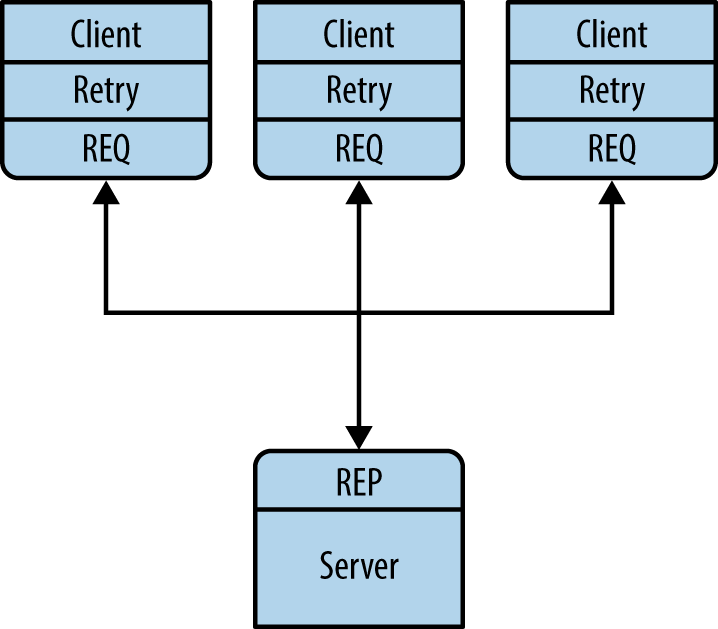
3. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику.

**Общие сведения о программе**

Программа состоит из двух файлов – server.cpp, client.cpp, в которых расположены код сервера, код клиента. Для удобства также был создан Makefile.

**Общий метод и алгоритм решения**

Общение между клиентом и сервером осуществляется как на схеме, изображённой ниже:



Сначала запускается сервер, потом запускаются клиенты, они сразу же посылают сигнал серверу, что готовы играть, там он их регистрирует по ID процессов. На сервере также хранятся словари с полями игроков, их статистикой, количествами уничтоженных кораблей и так далее. Когда сервер получают какую-то команду, то он получает также и ID игрока, достаёт из словарей всю связанную с игроком информацию и в зависимости от этого выбирает свою команду, посылаемую игроком.

**Исходный код**

**server.cpp**

|  |
| --- |
| #include <zmq.hpp> |
|  | #include <iostream> |
|  | #include <unistd.h> |
|  | #include <map> |
|  | #include <set> |
|  | #include <string> |
|  | #include <vector> |
|  |  |
|  | void random(std::vector<std::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) |
|  | { |
|  | std::replace(p[i].begin(), p[i].end(), '/', '.'); |
|  | } |
|  | } |
|  |  |
|  | void send\_message(std::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)) |
|  | { |
|  | std::cout << "Error: can't send message from node with pid " << getpid() << "\n"; |
|  | } |
|  | } |
|  |  |
|  | void print(std::vector<std::vector<char>>& p) |
|  | { |
|  | for (int i = 1; i < 11; ++i) |
|  | { |
|  | for (int j = 1; j < 11; ++j) |
|  | { |
|  | std::cout << p[i][j]; |
|  | } |
|  | std::cout << "\n"; |
|  | } |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | zmq::context\_t context (1); |
|  | zmq::socket\_t socket (context, ZMQ\_REP); |
|  | std::string port, reply; |
|  | std::cout << "Enter the port\n"; |
|  | std::cin >> port; |
|  | socket.bind("tcp://\*:" + port); |
|  | unsigned milliseconds; |
|  | std::cout << "Enter the time that socket should wait for answer from client and send message to client (it is a single value)\n"; |
|  | std::cin >> milliseconds; |
|  | socket.setsockopt(ZMQ\_SNDTIMEO, (int)milliseconds); |
|  | std::map<int, std::pair<unsigned, unsigned>> statistics; |
|  | std::map<int, std::pair<unsigned, unsigned>> amount; |
|  | std::map<int, std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>>> fields; |
|  | std::map<int, std::vector<std::pair<unsigned, unsigned>>> possible\_turns; |
|  | std::map<int, std::pair<unsigned, unsigned>> last\_commands; |
|  | std::map<int, bool> finishing; |
|  | std::map<int, std::vector<std::pair<unsigned, unsigned>>> variants; |
|  | while (true) |
|  | { |
|  | zmq::message\_t request; |
|  | socket.recv(&request); |
|  | std::string message(static\_cast<char\*>(request.data()), request.size()), reply; |
|  | std::string command = message.substr(0, message.find(" ")); |
|  | int ID = std::stoi(message.substr(message.find(" ") + 1)); |
|  | std::cout << message << "\n"; |
|  | if (command == "ID") |
|  | { |
|  | statistics[ID] = {0, 0}; |
|  | amount[ID] = {0, 0}; |
|  | //reply = "OK"; |
|  | send\_message("OK", socket); |
|  | } |
|  | else if (command == "Statistics") |
|  | { |
|  | if (statistics.find(ID) != statistics.end()) |
|  | { |
|  | std::pair<unsigned, unsigned> numbers = statistics[ID]; |
|  | reply = std::to\_string(numbers.first) + " " + std::to\_string(numbers.second); |
|  | } |
|  | else |
|  | { |
|  | reply = "Error: player with such ID already exists"; |
|  | } |
|  | send\_message(reply, socket); |
|  | } |
|  | else if (command == "Get") |
|  | { |
|  | print(fields[ID].first); |
|  | send\_message("OK", socket); |
|  | } |
|  | else if (command == "Exit") |
|  | { |
|  | 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); |
|  | } |
|  | //reply = "It was nice to play with you, bye!"; |
|  | send\_message("It was nice to play with you, bye!", socket); |
|  | } |
|  | else if (command == "Begin") |
|  | { |
|  | //std::cout << "Recieved Begin\n"; |
|  | amount[ID] = {0, 0}; |
|  | std::vector<std::vector<char>> server\_field (12, std::vector<char>(12, '.')); |
|  | std::vector<std::vector<char>> player\_field (12, std::vector<char>(12, '.')); |
|  | //std::cout << "Created vectors\n"; |
|  | random(server\_field); |
|  | fields[ID] = {server\_field, player\_field}; |
|  | //std::cout << "Created fields\n"; |
|  | server\_field.clear(); |
|  | player\_field.clear(); |
|  | //std::cout << "Cleared vectors\n"; |
|  | std::vector<std::pair<unsigned, unsigned>> turns (100); |
|  | for (int i = 0; i < 10; i++) |
|  | { |
|  | for (int j = 0; j < 10; ++j) |
|  | { |
|  | turns[i \* 10 + j] = {i, j}; |
|  | } |
|  | } |
|  | //std::cout << "Created turns\n"; |
|  | possible\_turns[ID] = turns; |
|  | finishing[ID] = false; |
|  | last\_commands[ID] = {-1, -1}; |
|  | variants[ID] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; |
|  | turns.clear(); |
|  | //reply = "Start"; |
|  | send\_message("Start", socket); |
|  | } |
|  | else if (command.substr(0, 3) == "Try") |
|  | { |
|  | std::string coordinates = message.substr(0, message.find(" ")); |
|  | std::cout << coordinates << "\n"; |
|  | unsigned horizontal = unsigned(coordinates[3]) - unsigned('0') + 1; |
|  | unsigned vertical = unsigned(coordinates[4]) - unsigned('0') + 1; |
|  | std::cout << horizontal << " " << vertical << "\n"; |
|  | std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>> squares = fields[ID]; |
|  | if (squares.first[vertical][horizontal] == 'X') |
|  | { |
|  | squares.first[vertical][horizontal] = 'K'; |
|  | //std::vector<std::pair<unsigned, unsigned>> turns = possible\_turns[ID]; |
|  | for (int i = -1; i < 2; ++i) |
|  | { |
|  | for (int j = -1; j < 2; ++j) |
|  | { |
|  | if ((vertical + i > 0) && (vertical + i < 11) && (horizontal + j > 0) && (horizontal + j < 11)) |
|  | { |
|  | if (squares.first[vertical + i][horizontal + j] == '.') |
|  | { |
|  | squares.first[vertical + i][horizontal + j] = 'w'; |
|  | /\*std::vector<std::pair<int, int>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | arr.erase(arr.begin() + std::distance(turns.begin(), it); |
|  | }\*/ |
|  |  |
|  | } |
|  | } |
|  | } |
|  | } |
|  | //possible\_turns[ID] = turns; |
|  | reply = "Killed"; |
|  | int v = vertical, h = horizontal; |
|  | 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] = {++amount[ID].first, amount[ID].second}; |
|  | if (amount[ID].first == 10) |
|  | { |
|  | reply = "Won"; |
|  | statistics[ID] = {++statistics[ID].first, statistics[ID].second}; |
|  | } |
|  | } |
|  | } |
|  | else if ((squares.first[vertical][horizontal] == 'K') || (squares.first[vertical][horizontal] == 'w')) |
|  | { |
|  | reply = "Another"; |
|  | } |
|  | else if (squares.first[vertical][horizontal] == '.') |
|  | { |
|  | reply = "Missed"; |
|  | squares.first[vertical][horizontal] = 'w'; |
|  | } |
|  | fields[ID] = {squares.first, squares.second}; |
|  | send\_message(reply, socket); |
|  | } |
|  | else if (command == "Amount") |
|  | { |
|  | std::cout << "Amount: " << amount[ID].first << "\n"; |
|  | send\_message("OK", socket); |
|  | } |
|  | else if (command == "Turns") |
|  | { |
|  | std::vector<std::pair<unsigned, unsigned>> turns = possible\_turns[ID]; |
|  | for (int i = 0; i < turns.size(); i++) |
|  | { |
|  | std::cout << turns[i].first << " " << turns[i].second << "\n"; |
|  | } |
|  | std::cout << "Length is " << turns.size() << "\n"; |
|  | send\_message("Ok", socket); |
|  | } |
|  | else if ((command == "Do") || (command == "Killed")) |
|  | { |
|  | if (command == "Killed") |
|  | { |
|  | std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>> squares = fields[ID]; |
|  | std::vector<std::pair<unsigned, unsigned>> turns = possible\_turns[ID]; |
|  | unsigned vertical = last\_commands[ID].first, horizontal = last\_commands[ID].second; |
|  | squares.second[vertical][horizontal] = 'K'; |
|  | //turns.erase(turns.begin() + std::distance(turns.begin(), std::find(turns.begin(), turns.end(), std::make\_pair(vertical, horizontal)))); |
|  | for (int i = -1; i < 2; ++i) |
|  | { |
|  | for (int j = -1; j < 2; ++j) |
|  | { |
|  | if ((vertical + i > 0) && (vertical + i < 11) && (horizontal + j > 0) && (horizontal + j < 11)) |
|  | { |
|  | std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | } |
|  | if (squares.second[vertical + i][horizontal + j] == '.') |
|  | { |
|  | squares.second[vertical + i][horizontal + j] = 'w'; |
|  | /\*std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | }\*/ |
|  | } |
|  | } |
|  | } |
|  | } |
|  | fields[ID] = {squares.first, squares.second}; |
|  | possible\_turns[ID] = turns; |
|  | finishing[ID] = false; |
|  | variants[ID] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; |
|  | amount[ID] = {amount[ID].first, ++amount[ID].second}; |
|  | if (amount[ID].second == 10) |
|  | { |
|  | reply = "Lost"; |
|  | amount[ID] = {0, 0}; |
|  | } |
|  | } |
|  | if (!finishing[ID]) |
|  | { |
|  | int length = possible\_turns[ID].size(); |
|  | srand(time(0)); |
|  | int number = rand() % length; |
|  | std::vector<std::pair<unsigned, unsigned>> coordinates = possible\_turns[ID]; |
|  | std::pair<unsigned, unsigned> turn = coordinates[number]; |
|  | std::cout << "Turn is " << turn.first << " " << turn.second << "\n"; |
|  | coordinates.erase(coordinates.begin() + number); |
|  | possible\_turns[ID] = coordinates; |
|  | last\_commands[ID] = turn; |
|  | reply = "Try" + std::to\_string(turn.first) + std::to\_string(turn.second); |
|  | std::cout << "Tried coordinates " << turn.first << " " << turn.second << "\n"; |
|  | } |
|  | else |
|  | { |
|  | int length, number, k = 1; |
|  | std::vector<std::pair<unsigned, unsigned>> positions; |
|  | std::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)))); |
|  | std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>> squares = fields[ID]; |
|  | std::vector<std::pair<unsigned, unsigned>> turns = possible\_turns[ID]; |
|  | unsigned vertical, horizontal; |
|  | do |
|  | { |
|  | send\_message("Try" + std::to\_string(last\_commands[ID].first + turn.first \* k) + std::to\_string(last\_commands[ID].second + turn.second \* k), socket); |
|  | ++k; |
|  | zmq::message\_t answer; |
|  | socket.recv(&answer); |
|  | std::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() + std::distance(turns.begin(), std::find(turns.begin(), turns.end(), std::make\_pair(vertical, horizontal)))); |
|  | if ((reply == "Wounded") || (reply == "Killed")) |
|  | { |
|  | squares.second[vertical][horizontal] = 'K'; |
|  | for (int i = -1; i < 2; ++i) |
|  | { |
|  | for (int j = -1; j < 2; ++j) |
|  | { |
|  | if ((vertical + i > 0) && (vertical + i < 11) && (horizontal + j > 0) && (horizontal + j < 11)) |
|  | { |
|  | std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | } |
|  | if (squares.second[vertical + i][horizontal + j] == '.') |
|  | { |
|  | squares.second[vertical + i][horizontal + j] = 'w'; |
|  | /\*std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | }\*/ |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | else if (reply == "Missed") |
|  | { |
|  | squares.second[vertical][horizontal] = 'w'; |
|  | } |
|  | //fields[ID] = {squares.first, squares.second}; |
|  | } |
|  | while (reply == "Wounded"); |
|  | fields[ID] = {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] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; |
|  | amount[ID] = {amount[ID].first, ++amount[ID].second}; |
|  | if (amount[ID].second == 10) |
|  | { |
|  | reply = "Lost"; |
|  | amount[ID] = {0, 0}; |
|  | } |
|  | else |
|  | { |
|  | int length = possible\_turns[ID].size(); |
|  | srand(time(0)); |
|  | int number = rand() % length; |
|  | std::vector<std::pair<unsigned, unsigned>> coordinates = possible\_turns[ID]; |
|  | std::pair<unsigned, unsigned> turn = coordinates[number]; |
|  | coordinates.erase(coordinates.begin() + number); |
|  | possible\_turns[ID] = coordinates; |
|  | last\_commands[ID] = turn; |
|  | reply = "Try" + std::to\_string(turn.first) + std::to\_string(turn.second); |
|  | std::cout << "Tried coordinates " << turn.first << " " << turn.second << "\n"; |
|  | } |
|  | } |
|  | } |
|  | send\_message(reply, socket); |
|  | } |
|  | else if (command == "Missed") |
|  | { |
|  | std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>> squares = fields[ID]; |
|  | squares.second[last\_commands[ID].first][last\_commands[ID].second] = 'w'; |
|  | fields[ID] = {squares.first, squares.second}; |
|  | send\_message("Do", socket); |
|  | } |
|  | else if (command == "Wounded") |
|  | { |
|  | finishing[ID] = true; |
|  | int length, number, k = 1; |
|  | std::vector<std::pair<unsigned, unsigned>> positions; |
|  | std::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)))); |
|  | std::pair<std::vector<std::vector<char>>, std::vector<std::vector<char>>> squares = fields[ID]; |
|  | std::vector<std::pair<unsigned, unsigned>> turns = possible\_turns[ID]; |
|  | unsigned vertical, horizontal; |
|  | do |
|  | { |
|  | send\_message("Try" + std::to\_string(last\_commands[ID].first + turn.first \* k) + std::to\_string(last\_commands[ID].second + turn.second \* k), socket); |
|  | ++k; |
|  | zmq::message\_t answer; |
|  | socket.recv(&answer); |
|  | std::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() + std::distance(turns.begin(), std::find(turns.begin(), turns.end(), std::make\_pair(vertical, horizontal)))); |
|  | if ((reply == "Wounded") || (reply == "Killed")) |
|  | { |
|  | squares.second[vertical][horizontal] = 'K'; |
|  | for (int i = -1; i < 2; ++i) |
|  | { |
|  | for (int j = -1; j < 2; ++j) |
|  | { |
|  | if ((vertical + i > 0) && (vertical + i < 11) && (horizontal + j > 0) && (horizontal + j < 11)) |
|  | { |
|  | std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | } |
|  | if (squares.second[vertical + i][horizontal + j] == '.') |
|  | { |
|  | squares.second[vertical + i][horizontal + j] = 'w'; |
|  | std::vector<std::pair<unsigned, unsigned>>::iterator it; |
|  | /\*if ((it = std::find(turns.begin(), turns.end(), std::make\_pair(vertical + i, horizontal + j))) != turns.end()) |
|  | { |
|  | turns.erase(turns.begin() + std::distance(turns.begin(), it)); |
|  | }\*/ |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | else if (reply == "Missed") |
|  | { |
|  | squares.second[vertical][horizontal] = 'w'; |
|  | } |
|  | //fields[ID] = {squares.first, squares.second}; |
|  | } |
|  | while (reply == "Wounded"); |
|  | fields[ID] = {squares.first, squares.second}; |
|  | variants[ID] = positions; |
|  | if (reply == "Missed") |
|  | { |
|  | reply = "Do"; |
|  | } |
|  | else if (reply == "Killed") |
|  | { |
|  | finishing[ID] = false; |
|  | variants[ID] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; |
|  | amount[ID] = {amount[ID].first, ++amount[ID].second}; |
|  | if (amount[ID].second == 10) |
|  | { |
|  | reply = "Lost"; |
|  | amount[ID] = {0, 0}; |
|  | statistics[ID] = {statistics[ID].first, ++statistics[ID].second}; |
|  | } |
|  | else |
|  | { |
|  | int length = possible\_turns[ID].size(); |
|  | srand(time(0)); |
|  | int number = rand() % length; |
|  | std::vector<std::pair<unsigned, unsigned>> coordinates = possible\_turns[ID]; |
|  | std::pair<unsigned, unsigned> turn = coordinates[number]; |
|  | coordinates.erase(coordinates.begin() + number); |
|  | possible\_turns[ID] = coordinates; |
|  | last\_commands[ID] = turn; |
|  | reply = "Try" + std::to\_string(turn.first) + std::to\_string(turn.second); |
|  | std::cout << "Tried coordinates " << turn.first << " " << turn.second << "\n"; |
|  | } |
|  | } |
|  | send\_message(reply, socket); |
|  | } |
|  | /\* |
|  | else if (command == "Left") |
|  | { |
|  | std::pair<unsigned, unsigned> last\_command = last\_commands[ID]; |
|  | last\_command = {last\_command.first, last\_command.second - 1}; |
|  | last\_commands[ID] = last\_command; |
|  | reply = "Try" + std::to\_string(last\_command.first) + std::to\_string(last\_command.second - 1); |
|  | } |
|  | else if (command == "Right") |
|  | { |
|  | std::pair<unsigned, unsigned> last\_command = last\_commands[ID]; |
|  | last\_command = {last\_command.first, last\_command.second + 1}; |
|  | last\_commands[ID] = last\_command; |
|  | reply = "Try" + std::to\_string(last\_command.first) + std::to\_string(last\_command.second + 1); |
|  | } |
|  | else if (command == "Up") |
|  | { |
|  | std::pair<unsigned, unsigned> last\_command = last\_commands[ID]; |
|  | last\_command = {last\_command.first - 1, last\_command.second}; |
|  | last\_commands[ID] = last\_command; |
|  | reply = "Try" + std::to\_string(last\_command.first - 1) + std::to\_string(last\_command.second); |
|  | } |
|  | else if (command == "Down") |
|  | { |
|  | std::pair<unsigned, unsigned> last\_command = last\_commands[ID]; |
|  | last\_command = {last\_command.first + 1, last\_command.second}; |
|  | last\_commands[ID] = last\_command; |
|  | reply = "Try" + std::to\_string(last\_command.first + 1) + std::to\_string(last\_command.second); |
|  | } |
|  | send\_message(reply, socket);\*/ |
|  | } |
|  | return 0; |
|  | } |

**client.cpp**

|  |
| --- |
| #include <zmq.hpp> |
|  | #include <iostream> |
|  | #include <unistd.h> |
|  | #include <vector> |
|  | #include <algorithm> |
|  |  |
|  | void send\_message(std::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)) |
|  | { |
|  | std::cout << "Error: can't send message from node with pid " << getpid() << "\n"; |
|  | } |
|  | } |
|  |  |
|  | std::string recieve\_message(zmq::socket\_t& socket) |
|  | { |
|  | zmq::message\_t reply; |
|  | if (!socket.recv(&reply)) |
|  | { |
|  | std::cout << "There's no answer from server\n"; |
|  | exit(1); |
|  | } |
|  | std::string message(static\_cast<char\*>(reply.data()), reply.size()); |
|  | return message; |
|  | } |
|  |  |
|  | void random(std::vector<std::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(); |
|  | 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) |
|  | { |
|  | std::replace(p[i].begin(), p[i].end(), '/', '.'); |
|  | } |
|  | } |
|  |  |
|  | void flood(std::vector<std::vector<char>>& p) |
|  | { |
|  | for (int i = 0; i < 12; i++) |
|  | { |
|  | p[i].clear(); |
|  | p[i] = std::vector<char>(12, '.'); |
|  | } |
|  | } |
|  |  |
|  | void print(std::vector<std::vector<char>>& p) |
|  | { |
|  | for (int i = 1; i < 11; ++i) |
|  | { |
|  | for (int j = 1; j < 11; ++j) |
|  | { |
|  | std::cout << p[i][j]; |
|  | } |
|  | std::cout << "\n"; |
|  | } |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | zmq::context\_t context (1); |
|  | zmq::socket\_t socket (context, ZMQ\_REQ); |
|  | std::string port; |
|  | std::cout << "Enter the port\n"; |
|  | std::cin >> port; |
|  | std::cout << "Connecting to hello world server…" << std::endl; |
|  | unsigned milliseconds; |
|  | std::cout << "Enter the time that socket should wait for answer from server\n"; |
|  | std::cin >> milliseconds; |
|  | socket.setsockopt(ZMQ\_SNDTIMEO, (int)milliseconds); |
|  | socket.setsockopt(ZMQ\_RCVTIMEO, (int)milliseconds); |
|  | socket.connect ("tcp://localhost:" + port); |
|  | send\_message("ID " + std::to\_string(getpid()), socket); |
|  | zmq::message\_t reply; |
|  | /\*if (!socket.recv(&reply)) |
|  | { |
|  | std::cout << "There's no answer from server\n"; |
|  | return 0; |
|  | }\*/ |
|  | recieve\_message(socket); |
|  | std::cout << "If you want to start a game, input Begin\n"; |
|  | std::vector<std::vector<char>> my\_field(12, std::vector<char> (12, '.')); |
|  | std::vector<std::vector<char>> server\_field (12, std::vector<char> (12, '.')); |
|  | std::string command; |
|  | bool playing = false; |
|  | while (std::cin >> command) |
|  | { |
|  | if (command == "Begin") |
|  | { |
|  | playing = true; |
|  | int number; |
|  | flood(my\_field); |
|  | flood(server\_field); |
|  | std::cout << "Do you want to arrange the ships by yourself or generate a random combination? If the first, input 1, else - 2\n"; |
|  | do |
|  | { |
|  | std::cin >> number; |
|  | if ((number < 1) || (number > 2)) |
|  | { |
|  | std::cout << "Error, input 1 or 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; |
|  | std::cout << "You should input 10 fours of symbols: for example, A 1 A 4, or B 3 E 3\n"; |
|  | while (amount < 10) |
|  | { |
|  | std::cin >> h1 >> v1 >> h2 >> v2; |
|  | if ((v1 < 1) || (v2 < 1) || (v1 > 10) || (v2 > 10)) |
|  | { |
|  | std::cout << "Number must be greater than 0 and less than 11\n"; |
|  | continue; |
|  | } |
|  | if (!((h1 >= 'A') && (h1 <= 'J') && (h2 >= 'A') && (h2 <= 'J'))) |
|  | { |
|  | std::cout << "Letters must be not less than A and not greater than J\n"; |
|  | continue; |
|  | } |
|  | if ((v1 != v2) && (h1 != h2)) |
|  | { |
|  | std::cout << "Ship must be parallel to one of the coordinate axis\n"; |
|  | continue; |
|  | } |
|  | if ((v1 - v2 > 4) || (h1 - h2 > 4)) |
|  | { |
|  | std::cout << "These ships are too long\n"; |
|  | continue; |
|  | } |
|  | int ih1 = int(h1) - int('A') + 1, ih2 = int(h2) - int('A') + 1; |
|  | if (v2 < v1) |
|  | { |
|  | std::swap(v1, v2); |
|  | } |
|  | if (ih2 < ih1) |
|  | { |
|  | std::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) |
|  | { |
|  | std::cout << "It is impossible to put the ship, as it will come into contact with another\n"; |
|  | continue; |
|  | } |
|  | else |
|  | { |
|  | int number = ih2 - ih1; |
|  | if (amounts[number] == 4 - number) |
|  | { |
|  | std::cout << "You already have enough ships of this type\n"; |
|  | continue; |
|  | } |
|  | ++amounts[number]; |
|  | for (int i = ih1; i < ih2 + 1; ++i) |
|  | { |
|  | my\_field[v1][i] = 'X'; |
|  | } |
|  | ++amount; |
|  | std::cout << "Successfully created ship\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) |
|  | { |
|  | std::cout << "It is impossible to put the ship, as it will come into contact with another\n"; |
|  | continue; |
|  | } |
|  | else |
|  | { |
|  | int number = v2 - v1; |
|  | if (amounts[number] == 4 - number) |
|  | { |
|  | std::cout << "You already have enough ships of this type\n"; |
|  | continue; |
|  | } |
|  | ++amounts[number]; |
|  | for (int i = v1; i < v2 + 1; ++i) |
|  | { |
|  | my\_field[i][ih1] = 'X'; |
|  | } |
|  | ++amount; |
|  | std::cout << "Successfully created ship\n"; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | else if (number == 2) |
|  | { |
|  | std::cout << "Random generations of ships will be displayed, if you choose input 1, else 2 - then another arrangement will be displayed\n"; |
|  | int indicator; |
|  | do |
|  | { |
|  | flood(my\_field); |
|  | random(my\_field); |
|  | print(my\_field); |
|  | /\*for (int i = 1; i < 11; ++i) |
|  | { |
|  | for (int j = 1; j < 11; ++j) |
|  | { |
|  | std::cout << my\_field[i][j]; |
|  | } |
|  | std::cout << "\n"; |
|  | }\*/ |
|  | do |
|  | { |
|  | std::cin >> indicator; |
|  | if ((indicator < 1) || (indicator > 2)) |
|  | { |
|  | std::cout << "Indicator must be 1 or 2\n"; |
|  | } |
|  | } |
|  | while ((indicator < 1) || (indicator > 2)); |
|  | } |
|  | while (indicator != 1); |
|  | } |
|  | send\_message("Begin " + std::to\_string(getpid()), socket); |
|  | recieve\_message(socket); |
|  | std::cout << "Input move\n"; |
|  | continue; |
|  | } |
|  | else if (command == "Get") |
|  | { |
|  | send\_message("Get " + std::to\_string(getpid()), socket); |
|  | recieve\_message(socket); |
|  | } |
|  | if (command == "Exit") |
|  | { |
|  | send\_message("Exit " + std::to\_string(getpid()), socket); |
|  | std::string reply = recieve\_message(socket); |
|  | std::cout << reply; |
|  | std::cout << "Input move\n"; |
|  | return 0; |
|  | } |
|  | if (command == "Statistics") |
|  | { |
|  | send\_message("Statistics " + std::to\_string(getpid()), socket); |
|  | std::string reply = recieve\_message(socket); |
|  | std::cout << "You have " + reply.substr(0, reply.find(" ")) + " wons and " + reply.substr(reply.find(" ") + 1) + " loses\n"; |
|  | std::cout << "Input move\n"; |
|  | continue; |
|  | } |
|  | if (command == "My") |
|  | { |
|  | std::cout << "Here is your field\n"; |
|  | print(my\_field); |
|  | std::cout << "Input move\n"; |
|  | continue; |
|  | } |
|  | if (command == "Amount") |
|  | { |
|  | send\_message("Amount " + std::to\_string(getpid()), socket); |
|  | recieve\_message(socket); |
|  | continue; |
|  | } |
|  | if (command == "Server") |
|  | { |
|  | std::cout << "Here is server's field\n"; |
|  | print(server\_field); |
|  | std::cout << "Input move\n"; |
|  | continue; |
|  | } |
|  | if (command == "Turns") |
|  | { |
|  | send\_message("Turns " + std::to\_string(getpid()), socket); |
|  | recieve\_message(socket); |
|  | } |
|  | if (command == "Try") |
|  | { |
|  | if (!playing) |
|  | { |
|  | std::cout << "You aren't playing at the moment. Start a new game\n"; |
|  | continue; |
|  | } |
|  | else |
|  | { |
|  | int v; |
|  | char h; |
|  | while (true) |
|  | { |
|  | std::cin >> h >> v; |
|  | if (!((h >= 'A') && (h <= 'J'))) |
|  | { |
|  | std::cout << "Letters must be not less than A and not greater than J\n"; |
|  | continue; |
|  | } |
|  | else if ((v < 1) || (v > 10)) |
|  | { |
|  | std::cout << "Numbers must be greater than 0 and less than 11\n"; |
|  | continue; |
|  | } |
|  | break; |
|  | } |
|  | send\_message("Try" + std::to\_string(int(h) - int('A')) + std::to\_string(v - 1) + " " + std::to\_string(getpid()), socket); |
|  | std::string reply = recieve\_message(socket); |
|  | std::cout << "Reply: " << reply << "\n"; |
|  | if ((reply == "Killed") || (reply == "Wounded")) |
|  | { |
|  | server\_field[v][int(h) - int('A') + 1] = 'K'; |
|  | if (reply == "Killed") |
|  | { |
|  | std::cout << "You killed one of the server's ships\n"; |
|  | } |
|  | else |
|  | { |
|  | std::cout << "You wounded one of the server's ships\n"; |
|  | } |
|  | std::cout << "Input move\n"; |
|  | continue; |
|  | } |
|  | if (reply == "Another") |
|  | { |
|  | std::cout << "You have already entered these coordinates. Input something new\n"; |
|  | continue; |
|  | } |
|  | if (reply == "Won") |
|  | { |
|  | std::cout << "You won this game!\n"; |
|  | playing = false; |
|  | continue; |
|  | } |
|  | if (reply == "Missed") |
|  | { |
|  | server\_field[v][int(h) - int('A') + 1] = 'w'; |
|  | send\_message("Do " + std::to\_string(getpid()), socket); |
|  | while (true) |
|  | { |
|  | reply = recieve\_message(socket); |
|  | if (reply.substr(0, 3) == "Try") |
|  | { |
|  | std::cout << "Server's turn: " << char(int(reply[4] - int('0') + 'A')) << " " << int(reply[3]) - int('0') + 1 << "\n"; |
|  | //reply = std::to\_string(int(reply[4]) - int('0')) + " " + reply[3]; |
|  | } |
|  | else |
|  | { |
|  | std::cout << "Server's reply: " << 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"; |
|  | int v = ver, h = hor; |
|  | my\_field[v][h] = 'K'; |
|  | for (int i = -1; i < 2; i++) |
|  | { |
|  | for (int j = -1; j < 2; ++j) |
|  | { |
|  | if (my\_field[v + i][h + j] == '.') |
|  | { |
|  | my\_field[v + i][h + j] = 'w'; |
|  | } |
|  | } |
|  | } |
|  | 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'; |
|  | } |
|  | std::cout << "Our reply is " << reply << "\n"; |
|  | send\_message(reply + " " + std::to\_string(getpid()), socket); |
|  | } |
|  | if (reply == "Lost") |
|  | { |
|  | std::cout << "You lost this game\n"; |
|  | playing = false; |
|  | continue; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | return 0; |
|  | } |

**Демонстрация работы программы**

Её не будет, так как игра долгая.

**Минусы:** я попытался реализовать интеллектуальную систему, где если сервер ранит корабль, то старается его добить следующими ходами. Из-за этого программа иногда зацикливается, возможно это происходит с кораблями, соприкасающимися с границей, однако точного объяснения у меня нет.

**Выводы**Данный курсовой проект оказался довольно интересным. Я закрепил свои знания по zeromq, однако программа писалась хоть и на скорую руку, но в процессе довольно долго, поэтому получилась недоделка.