Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по предмету «Операционные сети»
Тема: Игра-аналог Dino-chrome.

Студент: Д. С. Ляшун
Преподаватель: Е. С. Миронов
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель курсового проекта:

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.

Проектирование интерактивной (консольной|графической) клиент-серверной игры на основе любой из выбранных технологий:

1. Pipes.
2. Sockets.
3. Сервера очередей.
4. И другие.

Создать собственную игру для нескольких пользователей. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

Вариант игры: Аналог Dino-chrome. Игра для одного или двух пользователей, связь клиент-клиент, взаимодействие процессов посредством очереди сообщений.

2 Алгоритм решения

Для создания графического пользовательского интерфейса игры будет использоваться графическая библиотека ncurses, предназначенная для управления вводом-выводом в терминал с возможностью создания внутренних окон, изменением цветов выводимых символов и работой с координатной сеткой окна.

Обмен данными между игроками при кооперативном режиме будет обеспечиваться с помощью технологии очереди сообщений ZeroMQ следующим образом: устанавливается парная связь между игроками, далее при отрисовке каждого кадра игры игроки будут обращаться к очереди сообщений и получать информацию о движении соперника, если она есть, в противном случае будут использована старые или первоначальные данные движения. В конце игрового цикла каждый игрок будет также посылать сообщение о собственном перемещении.

Сама программа будет устроена так: в начале игры запускается процесс с образом стартового меню, где пользователю предлагается создать новую игру на одного или два человека, а также присоединиться к уже существующей игре при указании соответствующего порта. В соответствии с выбранным режимом изменяется образ процесса на игровой, где происходит установка связи с игроком-соперником (при кооперативной игре), а также выполнение самого хода игры: получение информации о движении соперника, отрисовки игрового поля, игроков и препятствий, чтение команды движения, изменение положения игрока, отправка информации о движении текущего игрока сопернику. В конце игры выводится сообщение о набранных игроком очков и смена образа выполнения обратно на стартовое меню для возможной новой игры.

3 Листинг программы

Исходный код interface.hpp:

```
#pragma once

#include <string.h>
#include <zmq.h>
#include <stdio.h>
#include <stdlib.h>
#define check_ok1(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { fprintf(stderr, MSG); exit(-1); }
#define check_wrong1(VALUE, WRONGVAL, MSG) if (VALUE == WRONGVAL) { fprintf(stderr, MSG); exit(-1); }
const int WAIT_TIME = 1000000;

const int DATA_SIZE = 5;
const int start = 1;
const int info = 2;
struct Token {
  int action;
  int data[DATA_SIZE];
};

void CreateSocket(void* & context, void* & socket) {
  int res;
  context = zmq_ctx_new();
  check_wrong1(context, NULL, "Error creating context!");
  socket = zmq_socket(context, ZMQ_PAIR);
  check_wrong1(socket, NULL, "Error creating socket!");
  res = zmq_setsockopt(socket, ZMQ_RCVTIMEO, &WAIT_TIME, sizeof(int));
  check_ok1(res, 0, "Error changing options of socket!");
  res = zmq_setsockopt(socket, ZMQ_SNDTIMEO, &WAIT_TIME, sizeof(int));
  check_ok1(res, 0, "Error changing options of socket!");
}

void DeleteSocket(void* & context, void* & socket) {
    int res;
    res = zmq_close(socket);
    check_ok1(res, 0, "Error when socket closed!");
  res = zmq_ctx_term(context);
  check_ok1(res, 0, "Error when context closed!");
}

bool SendMessage(Token* token, void* socket, int type_work) { // ZMQ_DONTWAIT - dont wait, 0 - with waiting
  int res;
  zmq_msg_t message;
  res = zmq_msg_init_data(&message, token, sizeof(Token), NULL, NULL);
  check_ok1(res, 0, "Error creating message!");
```

```
  res = zmq_msg_send(&message, socket, type_work);
  if (res == -1) {
      fprintf(stderr, "Error sending message!\n");
    zmq_msg_close(&message);
    return false;
  }
  check_ok1(res, sizeof(Token), "Error getting wrong message!");
  return true;
}

bool RecieveMessage(Token& reply_data, void* socket, int type_work) {
  int res = 0;
  zmq_msg_t reply;
  zmq_msg_init(&reply);
  check_ok1(res, 0, "Error creating message-reply!");
  res = zmq_msg_recv(&reply, socket, type_work);
  if (res == -1) {
    if (type_work == 0) {
      fprintf(stderr, "Error getting message!\n");
      res = zmq_msg_close(&reply);
      check_ok1(res, 0, "Error closing message!");
    }
    return false;
  }
  check_ok1(res, sizeof(Token), "Error getting wrong message!");
  reply_data = *(Token*)zmq_msg_data(&reply);
  res = zmq_msg_close(&reply);
  check_ok1(res, 0, "Error closing message!");
  return true;
}

bool DialogMessages(Token* send, Token& reply, void* socket) {
  if (SendMessage(send, socket, 0) && RecieveMessage(reply, socket, 0)) {
      return true;
  }
  return false;
}
```

Исходный код start_menu.cpp:

```
#include <ncurses.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#define check_ok(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { fprintf(log_file, MSG);
    fprintf(log_file, "\n"); exit(-1); }
#define check_wrong(VALUE, WRONGVAL, MSG) if (VALUE == WRONGVAL) { fprintf(log_file,
    MSG); fprintf(log_file, "\n"); exit(-1); }
const char* TEST_OUTPUT_FILE = "./logs/log.txt";
```

```
9  const int COUNT_SECTIONS_MAIN_MENU = 2;
10 const int COUNT_SECTIONS_NEW_GAME = 4;
11 const char* MENU_SECTION[COUNT_SECTIONS_MAIN_MENU] = { "New game", "Exit" };
12 const char* NEW_GAME_SECTION[COUNT_SECTIONS_NEW_GAME] = { "New game with 1 player", "
       New game with 2 players", "Connect to existing game", "Back to main menu" };
13 enum Menu {
14   NEW_GAME = 0,
15   EXIT = 1,
16   NEW_GAME_1_PLAYER = 2,
17   NEW_GAME_2_PLAYERS = 3,
18   CONNECT_TO_EXIST_GAME = 4,
19   BACK_TO_MAIN_MENU = 5
20 };
21 const int WINDOW_SIZE_LINES = 25;
22 const int WINDOW_SIZE_COLUMNS = 90;
23 const int WINDOW_LEFT_UP_X = 0;
24 const int WINDOW_LEFT_UP_Y = 0;
25 const int CENTER_WINDOW_Y = (WINDOW_SIZE_LINES - WINDOW_LEFT_UP_Y) / 2 - 2;
26 const int CENTER_WINDOW_X = (WINDOW_SIZE_COLUMNS - WINDOW_LEFT_UP_X) / 2;
27
28 const char* NAME_PLAYER_EXECUTION = "player";
29 const int PORT_SIZE = 5;
30
31 bool ReadPort(WINDOW* window, std::string& port) {
32   box(window, 0, 0);
33   wclear(window);
34   mvwprintw(window, CENTER_WINDOW_Y+1, CENTER_WINDOW_X-10, "Input port game: ");
35   echo();
36   wrefresh(window);
37   int input = 1;
38   int index = 0;
39   while (input) {
40     char c = wgetch(window);
41     if (c == 10) {
42       break;
43     }
44     else if (c == 27) {
45       input = 0;
46       break;
47     }
48     else if (0 <= c - '0' && c - '0' <= 9 && index < PORT_SIZE) {
49       port.push_back(c);
50     }
51   }
52   noecho();
53   wclear(window);
54   if (!input) {
55     return false;
56   }
```

```
57    return true;
58  }
59
60  int main() {
61    FILE* log_file = NULL;
62    log_file = fopen(TEST_OUTPUT_FILE, "w");
63    if (log_file == NULL) {
64      printf("Error creating log file!\n");
65      return -1;
66    }
67
68    WINDOW* terminal = initscr();
69    check_wrong(terminal, NULL, "Error initialising ncurses!");
70    check_ok(start_color(), OK, "Error initialising colors mode!");
71    WINDOW* menu = newwin(WINDOW_SIZE_LINES, WINDOW_SIZE_COLUMNS, WINDOW_LEFT_UP_X,
          WINDOW_LEFT_UP_Y);
72    check_wrong(menu, NULL, "Error creating new window!");
73    if (has_colors() && COLOR_PAIRS >= 13) {
74      check_ok(init_pair(1, COLOR_BLACK, COLOR_WHITE), OK, "Error creating new color pair
            !");
75    }
76    else {
77      fprintf(log_file, "Error working colors mode!");
78      return -1;
79    }
80    check_ok(raw(), OK, "Error changing raw mode of working!");
81    check_ok(wbkgd(menu, COLOR_PAIR(1)), OK, "Error changing background color of window!
          ");
82    check_ok(keypad(menu, TRUE), OK, "Error of enabling mode processing special keys!");
83    check_ok(noecho(), OK, "Error make noecho mode of working!");
84    curs_set(0);
85
86    int option_now = 0;
87    int button;
88    int type = 0;
89    while (TRUE) {
90      box(menu, 0, 0);
91      for (int i = 0; i < (type == 0? COUNT_SECTIONS_MAIN_MENU : COUNT_SECTIONS_NEW_GAME)
            ; ++i) {
92        if (i == option_now) {
93          check_ok(wattron(menu, A_BLINK | A_BOLD), OK, "Error changing window attribute!
                ");
94        }
95        check_ok(mvwprintw(menu, CENTER_WINDOW_Y + i*2, CENTER_WINDOW_X - 10, (type == 0?
              MENU_SECTION[i]: NEW_GAME_SECTION[i])), OK, "Error output menu info!");
96        check_ok(wattroff(menu, A_BLINK | A_BOLD), OK, "Error changing window attribute!"
              );
97      }
98      wrefresh(menu);
```

```
99      button = wgetch(menu);
100     check_wrong(button, ERR, "Error input button in program!");
101     fprintf(log_file, "Getting key -> %d\n", button);
102     if (button == 27) {
103       if (type == 0) {
104         break;
105       }
106       else {
107         type = 0;
108       }
109     }
110     else if (button == 10) {
111       if (option_now+type == EXIT) {
112         break;
113       }
114       else if (option_now+type == NEW_GAME) {
115         type = 2;
116         option_now = 0;
117       }
118       else if (option_now+type == NEW_GAME_1_PLAYER) {
119         delwin(menu);
120         endwin();
121         fclose(log_file);
122         check_wrong(execl(NAME_PLAYER_EXECUTION, NAME_PLAYER_EXECUTION, std::to_string
                (0).c_str(), std::to_string(2).c_str(), NULL), -1, "Error changing
                execution program!");
123       }
124       else if (option_now+type == NEW_GAME_2_PLAYERS) {
125         std::string port;
126         if (ReadPort(menu, port)) {
127           delwin(menu);
128           endwin();
129           fclose(log_file);
130           check_wrong(execl(NAME_PLAYER_EXECUTION, NAME_PLAYER_EXECUTION, port.c_str(),
                    std::to_string(0).c_str(), NULL), -1, "Error changing execution program!
                   ");
131         }
132       }
133       else if (option_now+type == CONNECT_TO_EXIST_GAME) {
134         std::string port;
135         if (ReadPort(menu, port)) {
136           delwin(menu);
137           endwin();
138           fclose(log_file);
139           check_wrong(execl(NAME_PLAYER_EXECUTION, NAME_PLAYER_EXECUTION, port.c_str(),
                    std::to_string(1).c_str(), NULL), -1, "Error changing execution program!
                   ");
140         }
141       }
```

```
142       else if (option_now+type == BACK_TO_MAIN_MENU) {
143         option_now = 0;
144         type = 0;
145       }
146     }
147     else if (button == KEY_UP) {
148       --option_now;
149       if (option_now < 0) {
150         option_now = (type == 0? COUNT_SECTIONS_MAIN_MENU : COUNT_SECTIONS_NEW_GAME) -
                1;
151       }
152     }
153     else if (button == KEY_DOWN) {
154       ++option_now;
155       if (option_now == (type == 0? COUNT_SECTIONS_MAIN_MENU : COUNT_SECTIONS_NEW_GAME)
              ) {
156         option_now = 0;
157       }
158     }
159     wclear(menu);
160   }
161
162   check_ok(endwin(), OK, "Error closing ncurses!");
163   fprintf(log_file, "Exiting from program!");
164   fclose(log_file);
165 }
```

Исходный код player.cpp:

```
1  #include "interface.hpp"
2  #include <ncurses.h>
3  #include <unistd.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #define check_ok(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { fprintf(log_file, MSG);
       fprintf(log_file, "\n"); exit(-1); }
8  #define check_wrong(VALUE, WRONGVAL, MSG) if (VALUE == WRONGVAL) { fprintf(log_file,
       MSG); fprintf(log_file, "\n"); exit(-1); }
9  const char* LOG_FILE = "./logs/plog.txt";
10
11 const int WINDOW_SIZE_LINES = 25;
12 const int WINDOW_SIZE_COLUMNS = 90;
13 const int WINDOW_LEFT_UP_X = 0;
14 const int WINDOW_LEFT_UP_Y = 0;
15 const int CENTER_WINDOW_Y = (WINDOW_SIZE_LINES - WINDOW_LEFT_UP_Y) / 2;
16 const int CENTER_WINDOW_X = (WINDOW_SIZE_COLUMNS - WINDOW_LEFT_UP_X) / 2;
17
18 int PLAYER_POS = 1;
19 const int CONNECT_PLAYER_POS = 12;
```

```
20 const int PLAYER_POS_LINES = 15;
21 const int GROUND_LINES = 20;
22 const int SPACE_STRENGTH = 10;
23
24 const int GAME_SPEED = 50000;
25 const int GAME_CHANGE_SPEED_DELAY = 10;
26 const int GAME_CHANGE_SPEED_VALUE = 1000;
27
28 const int SCORE_DIGITS = 5;
29 const int SCORE_DELAY = 10;
30
31 const int TREES_TYPES_COUNT = 3;
32 const int TEXTURE_HEIGHT = 5;
33 const char* TREES[TREES_TYPES_COUNT][TEXTURE_HEIGHT] = {{ " Y ", "Y Y Y", " YYYYY ", "
       Y ", " Y " },
34 {" ", " ", "Y Y Y", " YYYYY ", " Y "}, {" ", " ", " Y Y Y", " YYY", " Y" }};
35 const int MAX_COUNT_TREES = 10;
36 const int TREES_DISTANCE = 10;
37 const int TREE_PROPABILITY = 10;
38 const int TREE_SUCCESS = 9;
39
40 const char* MAIN_MENU_PROGRAM_NAME = "game";
41
42 void DinoDraw(WINDOW* window, int x, int y, int bend_down, int type) {
43   static int step1 = 0;
44   static int step2 = 1;
45   wattron(window, COLOR_PAIR(type));
46   if (bend_down == 0) {
47     mvwprintw(window, x, y, " ^");
48     mvwprintw(window, x+1, y, "I OO=");
49     mvwprintw(window, x+2, y, "I OOOOOO");
50     mvwprintw(window, x+3, y, " OOOOOO");
51     if (x == PLAYER_POS_LINES) {
52       mvwprintw(window, x+4, y, " O O");
53     }
54   }
55   else {
56     mvwprintw(window, x+2, y, " ^");
57     mvwprintw(window, x+3, y, "I OOOOOOO=");
58     mvwprintw(window, x+4, y, "I OOOOOO");
59   }
60   if (x == PLAYER_POS_LINES) {
61     if ((type == 1 && step1 == 0) || (type == 2 && step2 == 0)) {
62       mvwprintw(window, x+5, y, " __ ==");
63     }
64     else {
65       mvwprintw(window, x+5, y, " == __");
66     }
67     if (type == 1) {
```

```
68        step1 = (step1 == 0? 1 : 0);
69      }
70      else if (type == 2) {
71        step2 = (step2 == 0? 1 : 0);
72      }
73    }
74    else {
75      mvwprintw(window, x+4, y, " == ==");
76    }
77    wattroff(window, COLOR_PAIR(type));
78  }
79  void TreeDraw(WINDOW* window, int tree_num, int x, int y, int* interact) {
80    for (int i = 0; i < TEXTURE_HEIGHT; ++i) {
81      for (int j = 0; TREES[tree_num][i][j] != '\0'; ++j) {
82        if (0 <= x+i && x+i < WINDOW_SIZE_LINES && 0 <= y+j && y+j < WINDOW_SIZE_COLUMNS)
              {
83          char c = mvwinch(window, x+i, y+j) & A_CHARTEXT;
84          if (TREES[tree_num][i][j] != ' ') {
85            if (TREES[tree_num][i][j] == 'Y') {
86              int color = mvwinch(window, x+i, y+j) & A_COLOR;
87              if (color == COLOR_PAIR(1)) {
88                if (c == 'O' || c == '=' || c == '^' || c == '_') {
89                  *interact = 1;
90                }
91              }
92            }
93            mvwaddch(window, x+i, y+j, TREES[tree_num][i][j]);
94          }
95        }
96      }
97    }
98  }
99  void GroundDraw(WINDOW* window) {
100   static int mode = 0;
101   mode = (mode == 0? 1 : 0);
102   for (int i = 0; i < WINDOW_SIZE_COLUMNS; ++i) {
103     if (mode) {
104       mvwprintw(window, GROUND_LINES, i, (i % 2 == 0? "-" : "~"));
105     }
106     else {
107       mvwprintw(window, GROUND_LINES, i, (i % 2 == 0? "~" : "-"));
108     }
109   }
110 }
111 void ScoreDraw(WINDOW* window, int score1, int score2) {
112   char answer1[SCORE_DIGITS+1];
113   char answer2[SCORE_DIGITS+1];
114   for (int i = SCORE_DIGITS-1; i >= 0; --i) {
115     answer1[i] = '0' + score1 % 10;
```

```
116     answer2[i] = '0' + score2 % 10;
117     score1 /= 10;
118     score2 /= 10;
119   }
120   answer1[SCORE_DIGITS] = '\0';
121   answer2[SCORE_DIGITS] = '\0';
122   mvwprintw(window, 2, 40, "YOUR SCORE: %s\tENEMY SCORE: %s", answer1, answer2);
123 }
124 int main(int argc, char* argv[]) {
125   int res;
126   srand(time(0));
127   FILE* log_file = fopen(LOG_FILE, "w");
128   check_wrong(log_file, NULL, "Error open log file!");
129   check_ok(argc, 3, "Incorrect count arguments in child process!");
130   int port = atoi(argv[1]);
131   int mode = atoi(argv[2]);
132
133   void* my_context, *my_socket;
134
135   if (mode != 2) {
136     char address[104];
137     CreateSocket(my_context, my_socket);
138     if (mode == 0) {
139       snprintf(address, sizeof(address), "tcp://*:%d", port);
140       res = zmq_bind(my_socket, address);
141       check_ok(res, 0, "Error creating bind connection!");
142       fprintf(log_file, "%s\n", address);
143     }
144     else if (mode == 1) {
145       snprintf(address, sizeof(address), "tcp://localhost:%d", port);
146       res = zmq_connect(my_socket, address);
147       check_ok(res, 0, "Error creating outgoing connection!");
148       fprintf(log_file, "%s\n", address);
149     }
150   }
151
152   WINDOW* terminal = initscr();
153   check_wrong(terminal, NULL, "Error initialising ncurses!");
154   check_ok(start_color(), OK, "Error initialising colors mode!");
155   WINDOW* gamewin = newwin(WINDOW_SIZE_LINES, WINDOW_SIZE_COLUMNS, WINDOW_LEFT_UP_X,
        WINDOW_LEFT_UP_Y);
156   check_wrong(gamewin, NULL, "Error creating new window!");
157   if (has_colors() && COLOR_PAIRS >= 13) {
158     check_ok(init_pair(1, COLOR_BLACK, COLOR_WHITE), OK, "Error creating new color pair
          !");
159     check_ok(init_pair(2, COLOR_RED, COLOR_WHITE), OK, "Error creating new color pair!"
          );
160   }
161   else {
```

```
162     fprintf(log_file, "Error working colors mode!");
163     return -1;
164   }
165   check_ok(raw(), OK, "Error changing raw mode of working!");
166   check_ok(wbkgd(gamewin, COLOR_PAIR(1)), OK, "Error changing background color of
          window!");
167   check_ok(keypad(gamewin, TRUE), OK, "Error of enabling mode processing special keys!
          ");
168   check_ok(noecho(), OK, "Error make noecho mode of working!");
169   nodelay(gamewin, TRUE);
170   curs_set(0);
171
172   Token reply;
173   Token* request = new Token;
174   if (mode == 0) {
175     RecieveMessage(reply, my_socket, 0);
176     if (reply.action == start) {
177       request->action = start;
178       request->data[0] = CONNECT_PLAYER_POS;
179       SendMessage(request, my_socket, ZMQ_DONTWAIT);
180     }
181   }
182   else if (mode == 1) {
183     request->action = start;
184     DialogMessages(request, reply, my_socket);
185     if (reply.action == start) {
186       PLAYER_POS = reply.data[0];
187     }
188   }
189   int button;
190
191   int bend_down = 0;
192   int space_vers = 0;
193   int player_pos = PLAYER_POS_LINES;
194
195   int my_score = 0;
196   int enemy_score = 0;
197   int delay = 0;
198
199   int trees[MAX_COUNT_TREES];
200   int trees_pos[MAX_COUNT_TREES];
201
202   int interact = 0;
203   int enemy_lose = 0;
204   int game_speed = GAME_SPEED;
205   for (int i = 0; i < MAX_COUNT_TREES; ++i) {
206     trees[i] = -1;
207   }
208
```

```
209   if (mode == 1) {
210     delete request;
211     request = new Token;
212     request->action = info;
213     request->data[0] = player_pos;
214     request->data[1] = PLAYER_POS;
215     request->data[2] = bend_down;
216     request->data[3] = -1;
217     request->data[4] = 1;
218     SendMessage(request, my_socket, ZMQ_DONTWAIT);
219   }
220   reply.action = info;
221   reply.data[0] = player_pos;
222   reply.data[1] = (mode == 0? CONNECT_PLAYER_POS : 1);
223   reply.data[2] = 0;
224   reply.data[3] = -1;
225   reply.data[4] = 1;
226
227   if (mode == 2) {
228     enemy_lose = 1;
229   }
230   while (!interact) {
231     if (!enemy_lose) {
232       Token new_data;
233       if (RecieveMessage(new_data, my_socket, ZMQ_DONTWAIT) && new_data.action == info)
            {
234         reply = new_data;
235       }
236     }
237     usleep(game_speed);
238     GroundDraw(gamewin);
239     ScoreDraw(gamewin, my_score, enemy_score);
240     if (space_vers != 0) {
241       --space_vers;
242       --player_pos;
243     }
244     else if (player_pos != PLAYER_POS_LINES) {
245       ++player_pos;
246     }
247
248     if (bend_down != 0) {
249       --bend_down;
250     }
251     DinoDraw(gamewin, player_pos, PLAYER_POS, bend_down, 1);
252     if (!enemy_lose) {
253       DinoDraw(gamewin, reply.data[0], reply.data[1], reply.data[2], 2);
254     }
255     for (int i = 0; i < MAX_COUNT_TREES; ++i) {
256       if (trees[i] != -1) {
```

```
257         TreeDraw(gamewin, trees[i], GROUND_LINES-TEXTURE_HEIGHT+1, trees_pos[i], &
                interact);
258         trees_pos[i] -= 1;
259         if (trees_pos[i] <= -10) {
260           trees[i] = -1;
261         }
262       }
263     }
264
265     wrefresh(gamewin);
266
267     button = wgetch(gamewin);
268     if (button == 10) {
269       break;
270     }
271     else if (button == ' ') {
272       if (player_pos == PLAYER_POS_LINES && space_vers == 0 && bend_down == 0) {
273         space_vers = SPACE_STRENGTH;
274       }
275     }
276     else if (button == KEY_DOWN) {
277       if (player_pos == PLAYER_POS_LINES && space_vers == 0) {
278         bend_down = 7;
279       }
280     }
281
282     ++delay;
283     if (delay % SCORE_DELAY == 0) {
284       ++my_score;
285       if (my_score % GAME_CHANGE_SPEED_DELAY == 0) {
286         game_speed += GAME_CHANGE_SPEED_VALUE;
287       }
288       if (!enemy_lose) {
289         ++enemy_score;
290       }
291       delay = 0;
292     }
293
294     if (!enemy_lose) {
295       delete request;
296       request = new Token;
297       request->action = info;
298       request->data[0] = player_pos;
299       request->data[1] = PLAYER_POS;
300       request->data[2] = bend_down;
301       request->data[3] = -1;
302       if (interact) {
303         request->data[4] = -1;
304       }
```

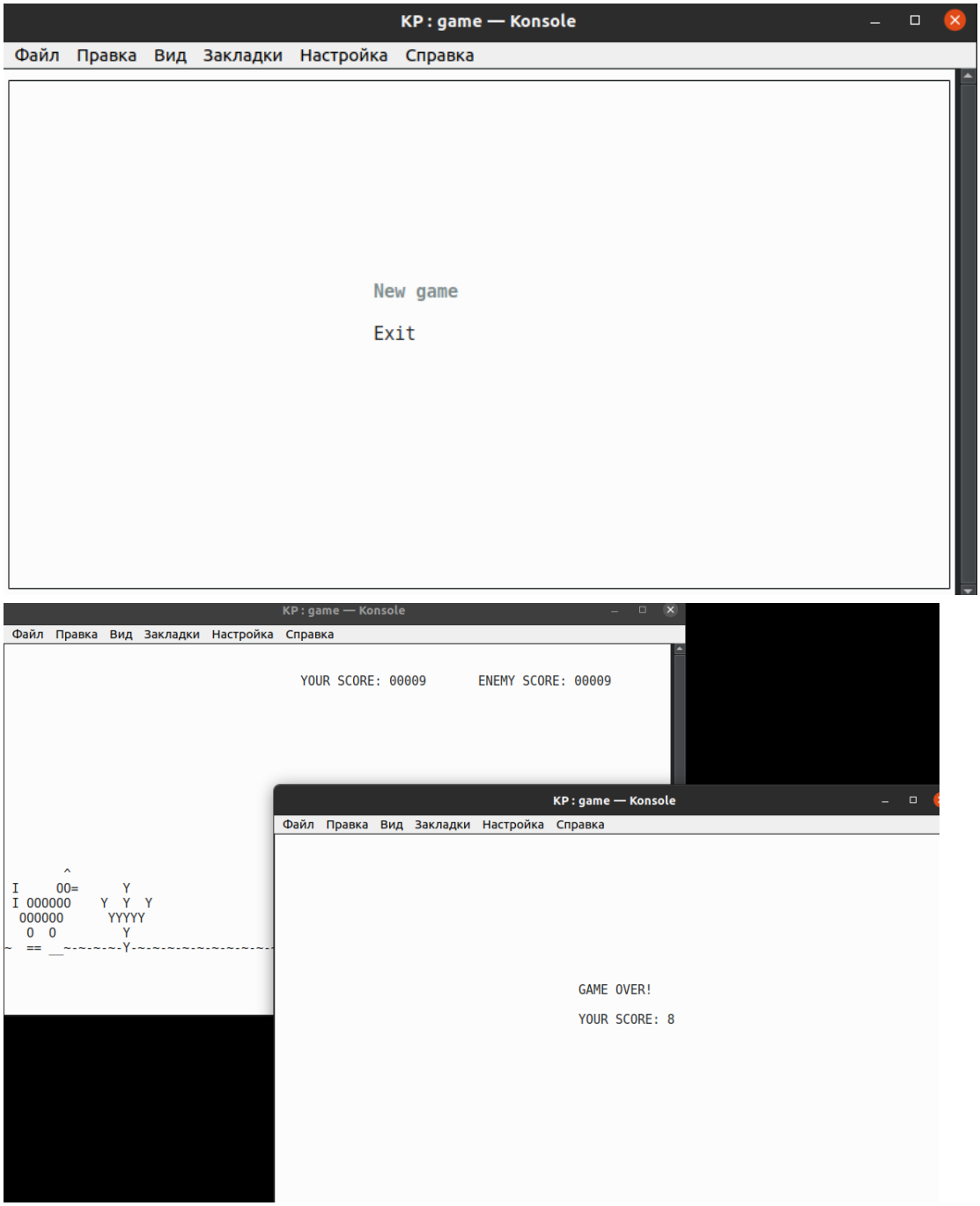
```
305       else {
306         request->data[4] = 1;
307       }
308     }
309     if (mode == 0 || enemy_lose) {
310       int create_tree = rand() % TREE_PROPABILITY;
311       if (create_tree >= TREE_SUCCESS) {
312         create_tree = rand() % TREES_TYPES_COUNT;
313         if (!enemy_lose) {
314           request->data[3] = create_tree;
315         }
316         for (int i = 0; i < MAX_COUNT_TREES; ++i) {
317           if (trees[i] == -1) {
318             trees[i] = create_tree;
319             trees_pos[i] = WINDOW_SIZE_COLUMNS + 10;
320           }
321         }
322       }
323     }
324     else {
325       if (reply.data[3] != -1) {
326         for (int i = 0; i < MAX_COUNT_TREES; ++i) {
327           if (trees[i] == -1) {
328             trees[i] = reply.data[3];
329             trees_pos[i] = WINDOW_SIZE_COLUMNS + 10;
330           }
331         }
332       }
333     }
334
335     if (reply.data[4] == -1) {
336       enemy_lose = 1;
337     }
338
339     if (!enemy_lose) {
340       SendMessage(request, my_socket, ZMQ_DONTWAIT);
341     }
342
343     wclear(gamewin);
344   }
345   if (interact) {
346     nodelay(gamewin, FALSE);
347     check_ok(wattron(gamewin, A_BLINK), OK, "Error changing window attribute!");
348     check_ok(mvwprintw(gamewin, CENTER_WINDOW_Y-2, CENTER_WINDOW_X-4, "GAME OVER!"), OK
          , "Error output text!");
349     check_ok(wattroff(gamewin, A_BLINK), OK, "Error changing window attribute!");
350     check_ok(mvwprintw(gamewin, CENTER_WINDOW_Y, CENTER_WINDOW_X-4, "YOUR SCORE: %d",
          my_score), OK, "Error output text!");
351     wrefresh(gamewin);
```

```
352     wgetch(gamewin);
353   }
354   check_ok(endwin(), OK, "Error closing ncurses!");
355   fprintf(log_file, "Exiting from program!");
356   fclose(log_file);
357   check_wrong(execl(MAIN_MENU_PROGRAM_NAME, MAIN_MENU_PROGRAM_NAME, NULL), -1, "Error
          changing program execution!");
358 }
```

Исходный код Makefile

```
1 all: start_menu.cpp player.cpp
2   g++ start_menu.cpp -o game -lncurses
3   g++ player.cpp -o player -lncurses -lzmq
```

4 Демонстрация работы программы



Выше приведены примеры выполнения программы: вывод стартового меню и ход игры для двух игроков.

5 Выводы

В результате выполнения данной курсовой работы я научился применять полученные знания по организации межпроцессорного взаимодействия на примере решения прикладной задачи – реализации собственной игры. В ходе её написания основную трудность для меня составило знакомство с библиотекой ncurses, изучением системных вызовов для работы с терминалом, а также организацией правильного взаимодействия между разными процессами-игроками посредством очереди сообщений ZeroMQ. Так, например, в первоначальной версии программы в игровом цикле получение информации о передвижении соперника из очереди сообщений происходило блокирующим образом, из-за чего игра работала заметно медленнее. Поэтому было принято решение сделать процесс получения сообщений неблокирующим, возможно, что из-за этого соперник отрисовывается не всегда предельно точно, однако для пользователя это выглядит незаметно из-за большой частоты смены кадров.

Также стоит отметить, что полученная игра получилась легко перенастраиваемой, т.е. с помощью именных числовых констант в ней были описаны такие значения как сила прыжка, длительность наклона, частота смены игрового счёта и т.д., в таком случае изменить динамичность игры можно довольно просто: для это нужно только поменять значения констант в исходном коде.

Список литературы

[1] *Linux Man Pages*
URL: `http://ru.manpages.org` (дата обращения: 27.12.2020).

[2] *Введение в ncurses*
URL: `http://dkhramov.dp.ua/Comp.NcursesTutorial.X_zGuNgzaM9` (дата обращения: 27.12.2020).