

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Э. БАУМАНА

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»



РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ ПО КОМПЬЮТЕРНЫМ СЕТЯМ НА ТЕМУ

Режим сетевого мультимедиа для игры «Pinball для двух игроков»

Исполнитель: студент ИУ7-61 Петухов И. С. _____

Руководитель: преподаватель ИУ7 Рогозин Н. О. _____

Москва, 2016

Содержание

Введение	4
1 Аналитический раздел	5
1.1 Gameplay игры	5
1.2 Передаваемые данные	5
1.3 Требования к серверу	6
2 Конструкторский раздел	7
2.1 Архитектура программного комплекса	7
2.2 Протокол взаимодействия	7
2.2.1 Пакеты сообщения	7
2.3 Алгоритм серверной части	8
2.3.1 Поток добавления	8
2.3.2 Поток соединения	8
2.3.3 Поток сессии	8
2.4 Алгоритм клиентской части	9
2.4.1 Интерфейс взаимодействия для клиента	9
2.5 Схема	10
2.6 Выводы	10
3 Технологический раздел	11
3.1 Выбор языка программирования	11
3.2 Выбор протокола транспортного уровня	11
3.3 Используемые шаблоны проектирования	11
3.3.1 Паттерн singleton	11
3.3.2 Паттерн шаблонный метод	12
3.4 Примеры кода	13
3.5 Интерфейс взаимодействия с клиентом	17
3.6 Интерфейс взаимодействия сервера с пользователем	18
3.7 Логгер сервера	18
Заключение	21

Введение

Целью данной работы является создание режима сетевого мультиплеера для игры «Pinball для двух игроков». Для этого необходимо создать серверную часть приложения и предоставить интерфейс для клиентской части приложения.

1 Аналитический раздел

1.1 Gameplay игры

Игра «Pinball для двух игроков» имеет 2 режима работы:

- а) Два пользователя на одной клавиатуре.
- б) Пользователь против бота.

Экран игры разбит на два поля. У каждого игрока - своё поле для игры в Pinball. Взаимодействие игроков заключается в перебрасывании шаров через отверстие в стене, разделяющее два поля. Поражение наступит либо при утере 5 шаров, либо при отставании в счете от соперника более чем на 5000000 очков. На рисунке 1.1 показан gameplay игры.

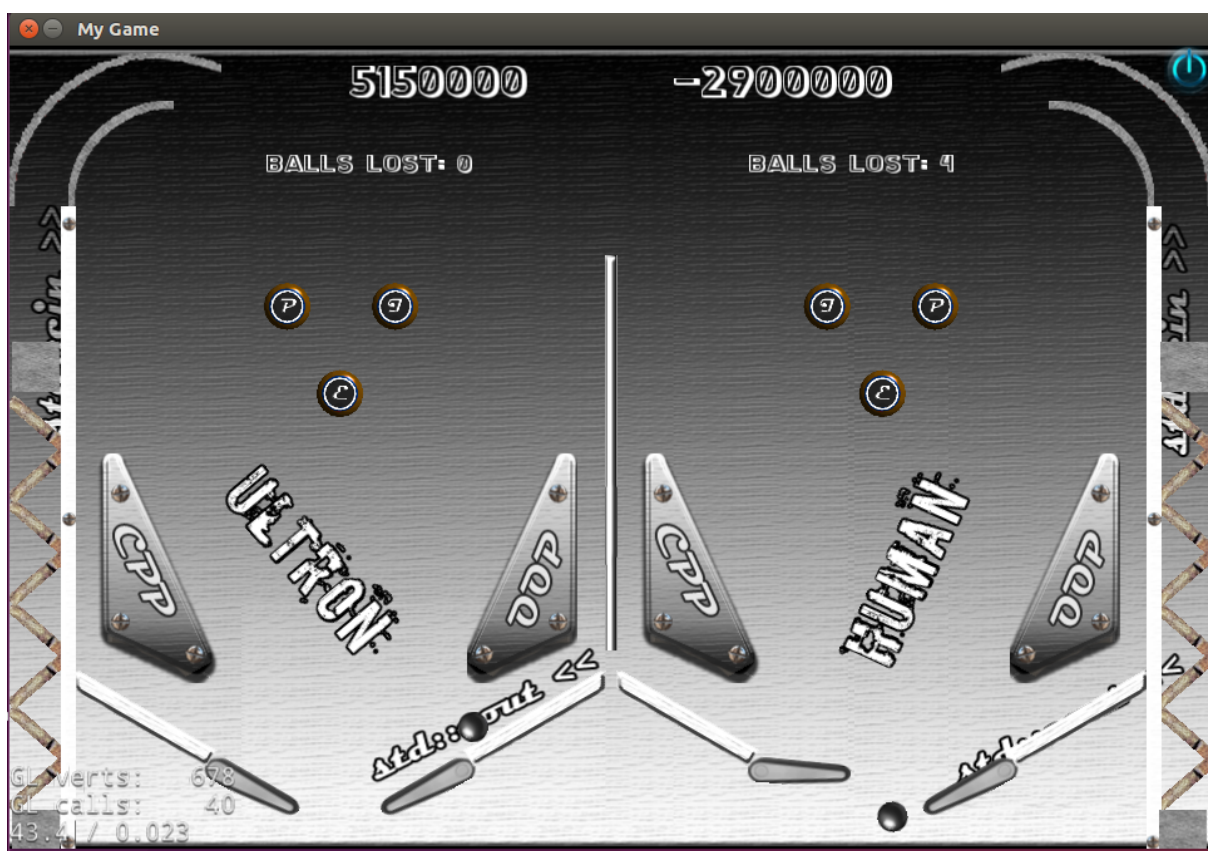


Рисунок 1.1 — Gameplay игры.

1.2 Передаваемые данные

Для того, чтобы создать сетевой мультиплеер для данной игры необходимо изучить, какие данные необходимо передавать между клиентами. Между двумя игроками будут передаваться сведения о следующих объектах:

- а) шары;
- б) пружина;

в) рычаги.

В связи со спецификой игры, для каждого объекта будут передаваться следующие данные:

а) шар:

- 1) координата X;
- 2) координата Y;
- 3) скорость по X;
- 4) скорость по Y;
- 5) угол поворота.

б) пружина:

- 1) координата Y;

в) рычаг:

- 1) какой рычаг (левый или правый);
- 2) флаг активации;

Дополнительная информация, необходимая для игроков:

а) счет;

б) имя;

1.3 Требования к серверу

а) возможность одновременной игры более 1 пары игроков;

б) поиск соперника происходит простым ожиданием первого подключившегося игрока;

в) запись лога событий.

2 Конструкторский раздел

Конструкторская часть содержит рассмотрение архитектуры разрабатываемого программного комплекса, структуры собственного протокола передачи данных между частями комплекса, а также основные алгоритмы, используемые в отдельных частях комплекса.

2.1 Архитектура программного комплекса

Разрабатываемый программный комплекс должен придерживаться модели взаимодействия «клиент» - «сервер». Таким образом, в программном комплексе дифференцируются две части.

Серверная часть отвечает за соединение игроков и обмен информацией между клиентами.

Клиентская часть передает серверу данные о себе и получает данные от сервера во время игры. Клиенты могут взаимодействовать друг с другом только посредством серверной коммуникации, т.е. если клиент желает отправить данные другому клиенту, из этих данных вначале формируется пакет, соответствующий разработанному для обмена информацией протоколу, затем этот пакет отправляется серверу, и уже сервер пересылает его конечному получателю.

2.2 Протокол взаимодействия

Типы сообщений T:

- а) FINISH - закончить игру
- б) START - начать игру
- в) RECEIVE NAME - получить имя
- г) SEND NAME - отправить имя
- д) NEW BALL - новый мяч
- е) ARR BALL - массив мячей
- ж) FLIPPER TRIGGERED - движение рычага, координата пружины, счет

2.2.1 Пакеты сообщения

Пакет NEW BALL новый мяч:

- а) x - координата x;
- б) y - координата y;
- в) speedX - скорость по x;
- г) speedY - скорость по y;
- д) rotation - угол поворота;

Пакет ARR BALL массив мячей:

- а) количество элементов;
- б) сами элементы:
 - 1) x - координата x ;
 - 2) y - координата y ;
 - 3) rotation - угол поворота;

Пакет FLIPPER TRIGGERED движение рычага:

- а) left - активен левый;
- б) right - активен правый;
- в) spring - координата пружины;
- г) score - счет;

2.3 Алгоритм серверной части

Сервер построен по типу «thread per game session», т.е. для каждой игровой сессии создается новый поток. Т.е. сервер состоит из потоков добавления, соединения и потоков сессий.

2.3.1 Поток добавления

Имеется массив ожидающих игроков. При появлении соединения, новый сокет добавляется в массив ожидающих игроков.

2.3.2 Поток соединения

Если количество элементов в массиве ожидающих игроков больше одного, то взять два последних элемента, соединить их в пару, удалить из массива ожидающих игроков. Запустить поток сессии для этой пары игроков.

2.3.3 Поток сессии

В паре игроков имеем два сокета ($K1$ и $K2$), через которые сервер будет общаться с этими двумя клиентами.

Последовательность действий сервера:

- а) отправить сообщение START $K1$
- б) отправить сообщение START $K2$
- в) получить подтверждение от $K1$ и $K2$ о начале игры;
- г) получить имена от $K1$ и $K2$;
- д) отправить имя соперника $K1$ и $K2$;
- е) для каждого $K1$ и $K2$, пока не получено сообщение о завершении игры:

- 1) получить тип сообщения T;
 - 2) получить структуру информации для типа T и отправить её сопернику;
- ж) отправить сообщение FINISH сопернику

2.4 Алгоритм клиентской части

Клиент может находиться в одном из состояний:

- а) CREATE - только создан,
- б) CONNECT- подключен к серверу,
- в) ACTIVE - идет игра,
- г) FINISH - закончена игра,
- д) END - отправлено сообщение о конце игры

Последовательность действий клиента:

- а) создать соединение с сервером,
- б) поздороваться с соперником:
 - 1) получить сообщение о старте игры;
 - 2) отправить своё имя;
 - 3) получить имя соперника;
- в) пока состояние игрока - ACTIVE и пока не получено сообщение о завершении игры:
 - 1) получить тип сообщения T;
 - 2) получить структуру информации для типа T;
- г) отправить сообщение FINISH

2.4.1 Интерфейс взаимодействия для клиента

На клиенте могут быть вызваны следующие операции:

- а) createConnection - создать соединение,
- б) handshake - поздороваться с соперником,
- в) setFinish - закончить игру,
- г) getOpponentName - взять имя соперника,
- д) getNewBall - взять новый шар соперника
- е) getArrBallsOpponent - взять массив шаров соперника
- ж) getFlipperTriggered - взять информацию о рычаге соперника
- з) sendArrBalls - отправить массив своих шаров
- и) sendNewBall - отправить свой новый шар
- к) sendFlipperTriggered - отправить информацию о своем рычаге и счете

2.5 Схема

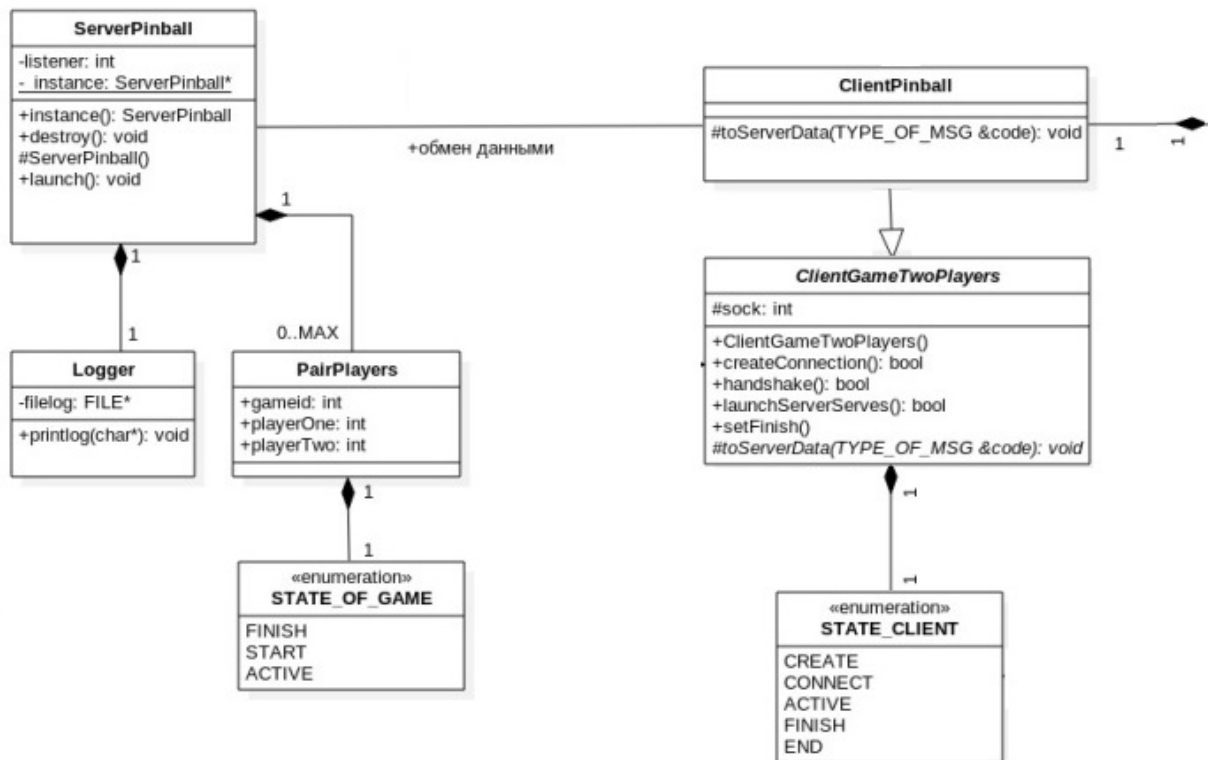


Рисунок 2.1 — Диаграмма классов клиента и сервера.

2.6 Выводы

В данном разделе были рассмотрены архитектура разрабатываемого программного комплекса, структура пакетов собственного протокола обмена данными, а также основные алгоритмы взаимодействия и обмена данными между клиентом и сервером.

3 Технологический раздел

3.1 Выбор языка программирования

Для создания программного комплекса был выбран язык программирования C++. Этот выбор был сделан на основании того, что данный язык позволяет делать всё то, что необходимо в рамках этой работы, а именно: создание и управление потоками, соединение и передача данных по сокетам, блокировка на мьютексах, работа с контейнерами.

3.2 Выбор протокола транспортного уровня

Взаимодействие между сервером и клиентами должно осуществляться на основании собственного протокола, основанного на протоколе ТСП. Данный протокол выбран в связи с тем, что потеря данных между клиентами - недопустима.

3.3 Используемые шаблоны проектирования

3.3.1 Паттерн singleton

Класс сервера игры основан на паттерне singleton. Это обосновывается тем, что в одной программе больше одного объекта сервера создавать не имеет смысла.

Листинг 3.1 — Header файл.

```
1 class ServerPinball {
2 public:
3     static ServerPinball* instance();
4     static void destroy();
5     ~ServerPinball();
6     //остальные методы...
7
8 protected:
9     ServerPinball();
10
11 private:
12     //остальные методы...
13
14 private:
15     static ServerPinball* _instance;
16     //остальные поля...
17 };
```

Листинг 3.2 — Реализация.

```
1 ServerPinball* ServerPinball::_instance = NULL;
2
3 ServerPinball* ServerPinball::instance() {
```

```

4     if ( _instance == 0 ) {
5         _instance = new ServerPinball;
6     }
7     return _instance;
8 }
9
10 ServerPinball::ServerPinball() {
11     stateServer = false;
12     stateListening = false;
13     stateCopulation = false;
14     numberWaitingPlayers = 0;
15     numberPairPlayers = 0;
16
17     logger = new Logger;
18
19     logger->printlog("a server's object is created");
20 }
21
22 ServerPinball::~ServerPinball() {
23     this->deleteConnection();
24     logger->printlog("a server's object has been deleted");
25     delete logger;
26 }
27
28 void ServerPinball::destroy() {
29     delete _instance;
30     _instance = NULL;
31 }

```

3.3.2 Паттерн шаблонный метод

Необходимо написать класс клиента игры и сделать его абстрактным. В этом классе реализуем методы, нужные для общения с сервером (обмен сообщениями о начале и конце игры). В этом классе будет чисто абстрактная функция, которая сделает класс абстрактным. Этот метод нужно реализовать в классе наследнике, он будет реализовывать обмен сообщениями конкретной игры.

В 3.3 создаем абстрактный класс.

В 3.4 создаем класс ClientPinball и наследуем его от класса ClientGameTwoPlayers. Реализуем абстрактную функцию из родительского класса, которая будет отвечать за передачу сообщений и данных, специфичные для данной игры.

Листинг 3.3 — Header файл для родительского класса

```

1 class ClientGameTwoPlayers {
2 public:

```

```

3     ClientGameTwoPlayers();
4     ~ClientGameTwoPlayers();
5     STATE_CLIENT getState();
6     //остальные методы...
7
8 protected:
9     virtual void toListenServerData(TYPE_OF_MSG &code) = 0;
10    //остальные методы...
11
12 protected:
13     int sock;
14     //остальные поля...
15 };

```

Листинг 3.4 — Реализация для класса потомка

```

1 class ClientPinball : public ClientGameTwoPlayers {
2     virtual void toListenServerData(TYPE_OF_MSG &code) {
3         do {
4             if (code == TYPE_OF_MSG::ARR_BALL) {
5                 recv(sock, &countSimpleBalls, sizeof(countSimpleBalls),
6                     0);
7                 recv(sock, arrBallsOpponent, countSimpleBalls * sizeof
8                     (SimpleBall), 0);
9             } else if (code == TYPE_OF_MSG::NEW_BALL) {
10                recv(sock, &newBall, sizeof(newBall), 0);
11                this->setPresenceNewBall(true);
12            }
13
14            code = receiveTypeMessage(sock);
15        } while (code != TYPE_OF_MSG::FINISH and this->getState() ==
16            STATE_CLIENT::ACTIVE);
17    }
18 };

```

3.4 Примеры кода

Листинг 3.5 — Создание соединения с сервером на клиенте

```

1 bool ClientGameTwoPlayers::initConnection(char *ipServer, int portServer) {
2     mutexArrBalls = PTHREAD_MUTEX_INITIALIZER;
3     mutexNewBall = PTHREAD_MUTEX_INITIALIZER;
4     mutexFlipper = PTHREAD_MUTEX_INITIALIZER;
5
6     if (this->getState() != STATE_CLIENT::CREATE) {
7         return false;
8     }

```

```

9
10 sock = socket(AF_INET, SOCK_STREAM, 0);
11 if (sock < 0) {
12     perror("socket");
13     return false;
14 }
15
16 addr.sin_family = AF_INET;
17 addr.sin_port = htons(portServer);
18 addr.sin_addr.s_addr = inet_addr(ipServer);
19
20 if (connect(sock, (struct sockaddr *) &addr, sizeof (addr)) < 0) {
21     perror("connect");
22     return false;
23 }
24
25 this->setState(STATE_CLIENT::CONNECT);
26 return true;
27 }

```

Листинг 3.6 — Подготовка сервера к слушанию клиентов

```

1 bool ServerPinball::initConnection(int port) {
2     w_lock = PTHREAD_MUTEX_INITIALIZER;
3     p_lock = PTHREAD_MUTEX_INITIALIZER;
4
5     struct sockaddr_in addr;
6
7     listener = socket(AF_INET, SOCK_STREAM, 0);
8     if (listener < 0) {
9         perror("socket");
10        return false;
11    }
12    addr.sin_family = AF_INET;
13    addr.sin_port = htons(port);
14    addr.sin_addr.s_addr = htonl(INADDR_ANY);
15    if (bind(listener, (struct sockaddr *) &addr, sizeof (addr)) < 0) {
16        perror("bind");
17        return false;
18    }
19
20    this->setPort(port);
21    listen(listener, MAX_COUNT_PAIR_PLAYERS * 2 +
22        MAX_LENGTH_QUEUE_WAITING_PLAYERS);
23    logger->printlog("server is created");
24    printf("server is created\n");
25    return true;

```

25 }

Листинг 3.7 — Структуры данных передаваемых пакетов

```
1 typedef struct {
2     float x;
3     float y;
4     float rotation;
5 } SimpleBall;
6
7 typedef struct {
8     float x;
9     float y;
10    float speedX;
11    float speedY;
12    float rotation;
13 } PhysicsBall;
14
15 typedef struct {
16     bool left;
17     bool right;
18     float spring;
19     int score;
20 } FlipperTriggered;
```

Листинг 3.8 — Отправка/прием типа сообщения

```
1 static void sendTypeMessage(int sock, TYPE_OF_MSG code) {
2     send(sock, &code, sizeof (code), 0);
3 }
4
5 static TYPE_OF_MSG receiveTypeMessage(int sock) {
6     TYPE_OF_MSG code;
7     recv(sock, &code, sizeof (code), 0);
8     return code;
9 }
```

Листинг 3.9 — Отправка/прием сообщения о старте игры

```
1 static void sendStartGame(PairPlayers* pair) {
2     TYPE_OF_MSG code = TYPE_OF_MSG::START;
3     sendTypeMessage(pair->playerOne, code);
4     sendTypeMessage(pair->playerTwo, code);
5 }
6
7 static bool receiveStartGame(PairPlayers* pair) {
8     TYPE_OF_MSG code = receiveTypeMessage(pair->playerOne);
9     if (code != TYPE_OF_MSG::START)
```

```

10     return false;
11
12     return receiveTypeMessage(pair->playerTwo) == TYPE_OF_MSG::START;
13 }

```

Листинг 3.10 — Отправка/прием сообщения с именем

```

1  static bool receiveNamePlayers(PairPlayers* pair, char
    namePlayer1[MAX_LENGTH_NAME_PLAYER], char
    namePlayer2[MAX_LENGTH_NAME_PLAYER]) {
2  sendTypeMessage(pair->playerOne, TYPE_OF_MSG::RECEIVE_NAME);
3  recv(pair->playerOne, namePlayer1, MAX_LENGTH_NAME_PLAYER * sizeof
    (char), 0);
4
5  sendTypeMessage(pair->playerTwo, TYPE_OF_MSG::RECEIVE_NAME);
6  recv(pair->playerTwo, namePlayer2, MAX_LENGTH_NAME_PLAYER * sizeof
    (char), 0);
7
8  return true;
9  }
10
11 static void sendNamePlayers(PairPlayers* pair, char
    namePlayer1[MAX_LENGTH_NAME_PLAYER], char
    namePlayer2[MAX_LENGTH_NAME_PLAYER]) {
12 sendTypeMessage(pair->playerOne, TYPE_OF_MSG::SEND_NAME);
13 send(pair->playerOne, namePlayer2, MAX_LENGTH_NAME_PLAYER * sizeof
    (char), 0);
14
15 sendTypeMessage(pair->playerTwo, TYPE_OF_MSG::SEND_NAME);
16 send(pair->playerTwo, namePlayer1, MAX_LENGTH_NAME_PLAYER * sizeof
    (char), 0);
17 }

```

Листинг 3.11 — Отправка массива шаров

```

1 void ClientGameTwoPlayers::sendArrBalls(int count, SimpleBall
    arr[MAX_COUNT_BALLS]) {
2 sendTypeMessage(sock, TYPE_OF_MSG::ARR_BALL);
3 send(sock, &count, sizeof (int), 0);
4 send(sock, arr, count * sizeof (SimpleBall), 0);
5 }

```

Листинг 3.12 — Отправка информации о рычаге, пружине, счете

```

1 void ClientGameTwoPlayers::sendFlipperTriggered(FlipperTriggered
    &flipperTriggered) {
2 sendTypeMessage(sock, TYPE_OF_MSG::FLIPPER_TRIGGERED);
3 send(sock, &flipperTriggered, sizeof (flipperTriggered), 0);

```

4 }

Листинг 3.13 — Получение данных типа T

```
1 virtual void toListenServerData(TYPE_OF_MSG code) {
2     if (code == TYPE_OF_MSG::FLIPPER_TRIGGERED) {
3         FlipperTriggered *flipperTriggered = new FlipperTriggered;
4         mutexFlipperLock();
5         recv(sock, flipperTriggered, sizeof (FlipperTriggered), 0);
6         queueFlipperTriggered.push(flipperTriggered);
7         mutexFlipperUnlock();
8     } else if (code == TYPE_OF_MSG::ARR_BALL) {
9         ArrayBalls *arrayBalls = new ArrayBalls;
10        mutexArrBallsLock();
11        recv(sock, &(arrayBalls->countSimpleBalls), sizeof
12        (arrayBalls->countSimpleBalls), 0);
13        recv(sock, arrayBalls->arrBallsOpponent, arrayBalls->countSimpleBalls
14        * sizeof (SimpleBall), 0);
15        queueArrBallsOpponent.push(arrayBalls);
16        mutexArrBallsUnlock();
17    } else if (code == TYPE_OF_MSG::NEW_BALL) {
18        PhysicsBall *physicsBall = new PhysicsBall;
19        mutexNewBallLock();
20        recv(sock, physicsBall, sizeof (PhysicsBall), 0);
21        queueNewBall.push(physicsBall);
22        mutexNewBallUnlock();
23    }
24 }
```

3.5 Интерфейс взаимодействия с клиентом

Интерфейс взаимодействия с клиентом представлен следующими функциями:

Листинг 3.14 — Интерфейс взаимодействия с клиентом

```
1 ClientGameTwoPlayers();
2 ~ClientGameTwoPlayers();
3 bool createConnection(char *ipServer);
4 bool createConnection(char *ipServer, int portServer);
5 bool handshake(char name[MAX_LENGTH_NAME_PLAYER]);
6 bool launchServesServer();
7 STATE_CLIENT getState();
8 void setFinish();
9 char *getMyName();
10 char *getOpponentName();
11
12 bool getNewBall(PhysicsBall &ball);
```



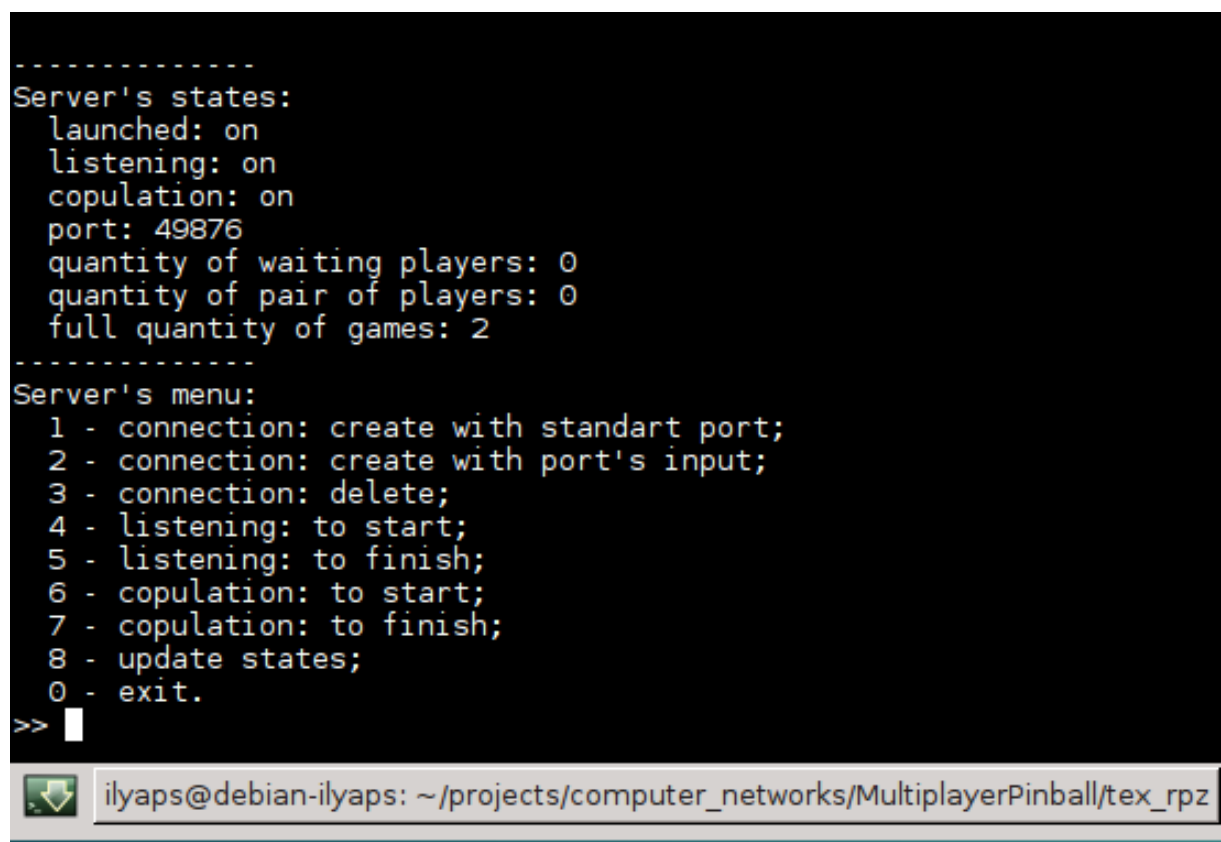
```

13  bool getArrBallsOpponent(SimpleBall arr[MAX_COUNT_BALLS], int &count);
14  bool getFlipperTriggered(FlipperTriggered &flipper);
15
16  void sendArrBalls(int count, SimpleBall arr[MAX_COUNT_BALLS]);
17  void sendNewBall(PhysicsBall &newBall);
18  void sendFlipperTriggered(FlipperTriggered &flipperTriggered);
19
20  void clearArrBalls(int countOpp);

```

3.6 Интерфейс взаимодействия сервера с пользователем

Интерфейс взаимодействия сервера с пользователем представлен дружественным консольным меню 3.2



```

-----
Server's states:
  launched: on
  listening: on
  copulation: on
  port: 49876
  quantity of waiting players: 0
  quantity of pair of players: 0
  full quantity of games: 2
-----
Server's menu:
  1 - connection: create with standart port;
  2 - connection: create with port's input;
  3 - connection: delete;
  4 - listening: to start;
  5 - listening: to finish;
  6 - copulation: to start;
  7 - copulation: to finish;
  8 - update states;
  0 - exit.
>> 

```

The screenshot shows a terminal window with a dark background. The text is displayed in a monospaced font. The terminal title bar at the bottom indicates the user is 'ilyaps@debian-ilyaps' and the current directory is '~/projects/computer_networks/MultiplayerPinball/tex_rpz'.

Рисунок 3.1 — Меню сервера.

3.7 Логгер сервера

Листинг 3.15 — Логгер

```

1  /*
2   * File:    logger.h
3   * Author:  ilyapc-ubuntu
4   */
5

```

```

6  #ifndef LOGGER_H
7  #define LOGGER_H
8
9  #include <stdio.h>
10 #include <string.h>
11 #define MAX_LENGTH_STRING_TIME 14
12
13 class Logger {
14 public:
15     Logger() {
16         char str[MAX_LENGTH_STRING_TIME + 12] = "logger/";
17         strcat(strcat(str, timeToString()), ".txt");
18         filelog = fopen(str, "w");
19     }
20
21     ~Logger() {
22         fclose(filelog);
23     }
24
25     FILE *filelog;
26
27     char *timeToString() {
28         time_t t = time(NULL);
29         static char *str = new char[MAX_LENGTH_STRING_TIME];
30         tm* aTm = localtime(&t);
31         sprintf(str, "%02d_%02d_%02d:%02d:%02d", aTm->tm_mday, aTm->tm_mon +
32             1, aTm->tm_hour, aTm->tm_min, aTm->tm_sec);
33         return str;
34     }
35
36     void printlog(const char *str) {
37         fprintf(filelog, "%s - ", timeToString());
38         fprintf(filelog, str);
39         fprintf(filelog, "\n");
40         fflush(filelog);
41     }
42 };
43 #endif  /* LOGGER_H */

```

```
11_10_22:21:38 - a server's object is created
11_10_22:21:38 - ready to start copulating
11_10_22:21:38 - ready to start listening
11_10_22:21:43 - server is created
11_10_22:21:46 - listening has started
11_10_22:21:49 - copulating has started
11_10_22:23:14 - waiting player has joined
11_10_22:23:17 - waiting player has joined
11_10_22:23:17 - a pair of waiting players is united
11_10_22:23:17 - game #1 has started
11_10_22:23:17 - game #1: samsung vs bubuntu

^G Get Help      ^O WriteOut
^X Exit          ^J Justify
```

Рисунок 3.2 — Пример логга.

Заключение

В ходе выполнения работы был реализован программный продукт, полностью отвечающий требованиям, изложенным в техническом задании, а именно сервер для мультиплеерной игры Pinball. По результатам аналитической части работы были сформулированы требования для разрабатываемого программного комплекса. В качестве архитектуры разрабатываемого программного комплекса была выбрана архитектура «клиент-сервер». Также были разработаны собственный сетевой протокол для передачи данных и алгоритмы функционирования серверной и клиентской части комплекса. Были реализованы все части программного комплекса. Было проведено тестирование отдельных частей комплекса, а также взаимодействие между ними. Методы написания приложения позволяют в дальнейшем развивать приложение, легко изменять существующую функциональность и добавлять новую.