

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 ИУ7-71 Петухов И. С. _____

Руководитель: преподаватель ИУ7 Рогозин Н. О. _____

Москва, 2016

Содержание

Введение	4
1 Аналитический раздел	5
1.1 Игра «Pinball для двух игроков»	5
1.2 Gameplay игры	5
1.3 Передаваемые данные	5
1.4 Требования к серверу	6
2 Конструкторский раздел	9
2.1 Архитектура программного комплекса	9
2.2 Протокол взаимодействия	9
2.2.1 Пакеты сообщений	9
2.3 Алгоритм серверной части	10
2.3.1 Поток добавления игроков	10
2.3.2 Поток соединения	10
2.3.3 Поток сессии	10
2.4 Алгоритм клиентской части	12
2.4.1 Интерфейс взаимодействия для клиента	12
2.5 Диаграммы классов клиента и сервера	12
2.6 Выводы	13
3 Технологический раздел	18
3.1 Выбор языка программирования	18
3.2 Выбор протокола транспортного уровня	18
3.3 Проблемы многопоточности	18
3.4 Используемые шаблоны проектирования	18
3.4.1 Паттерн singleton	18
3.4.2 Паттерн шаблонный метод	20
3.5 Примеры кода	21
3.6 Интерфейс взаимодействия с клиентом	25
3.7 Логгер сервера	25
3.8 Инструкция по компиляции и запуску игры	26
3.9 Запуск сервера	28
3.10 Интерфейс взаимодействия сервера с пользователем	28
Заключение	30
Список использованных источников	31

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 Игра «Pinball для двух игроков»

Игра «Pinball для двух игроков» написана на языке C++ с помощью фреймворка cocos2d.

Cocos2d — кросс-платформенный фреймворк, используемый для разработки интерактивных приложений и игр (преимущественно для мобильных устройств). Является открытым программным обеспечением. [1]

Cocos2d-x появился в 2010 году, это проект с открытым исходным кодом, распространяющийся под лицензией MIT. Cocos2d-x позволяет писать на таких языках как C++, Lua и Javascript. Cocos2d-x быстрый, простой и обладает большими возможностями. В настоящее время много игр, написанных с помощью этого фреймворка, находятся в топе AppStore и Google Play. [2]

1.2 Gameplay игры

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

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

На рисунке 1.2 показан gameplay игры. 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. На рисунке 1.3 показан момент перелета шара от одного игрока к другому. Поражение наступит либо при утере 5 шаров, либо при отставании в счете от соперника более чем на 12 000 000 очков.

1.3 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) шар;
- б) пружина;
- в) рычаг;
- г) игрок.

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;
- в) рычаг:
- 1) какой рычаг (левый или правый);
 - 2) флаг активации;
- г) игрок:
- 1) счет;
 - 2) имя;

1.4 Требования к серверу

При разработке сервера игры необходимо учесть следующие требования:

- а) возможность одновременной игры более 1 пары игроков;
- б) поиск соперника происходит простым ожиданием первого подключившегося игрока;
- в) запись лога всех событий в файл с числом и временем запуска сервера.

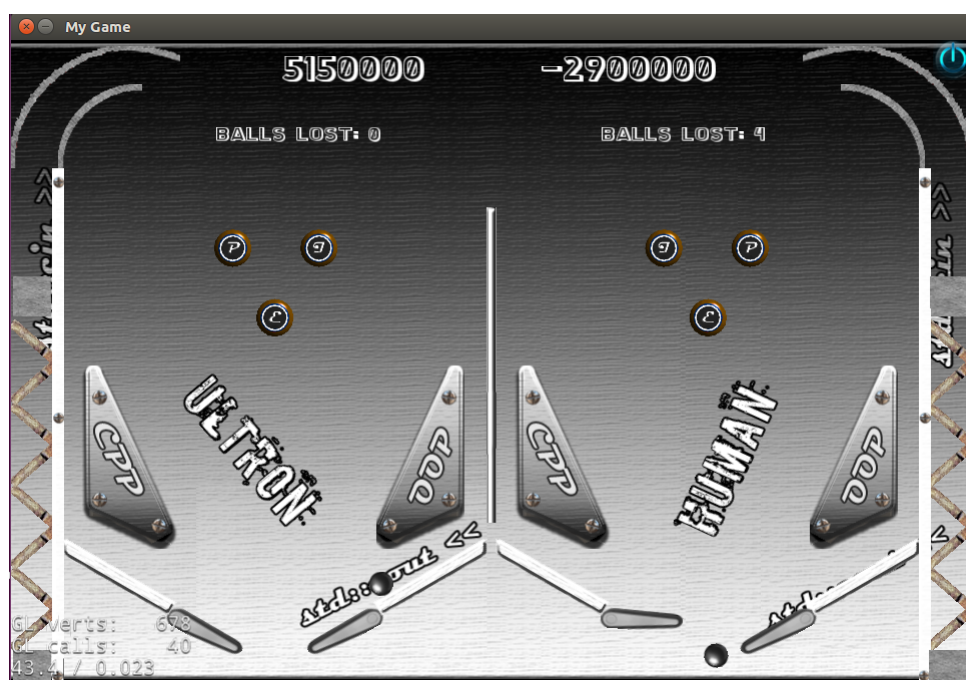


Рисунок 1.2 — Gameplay игры.

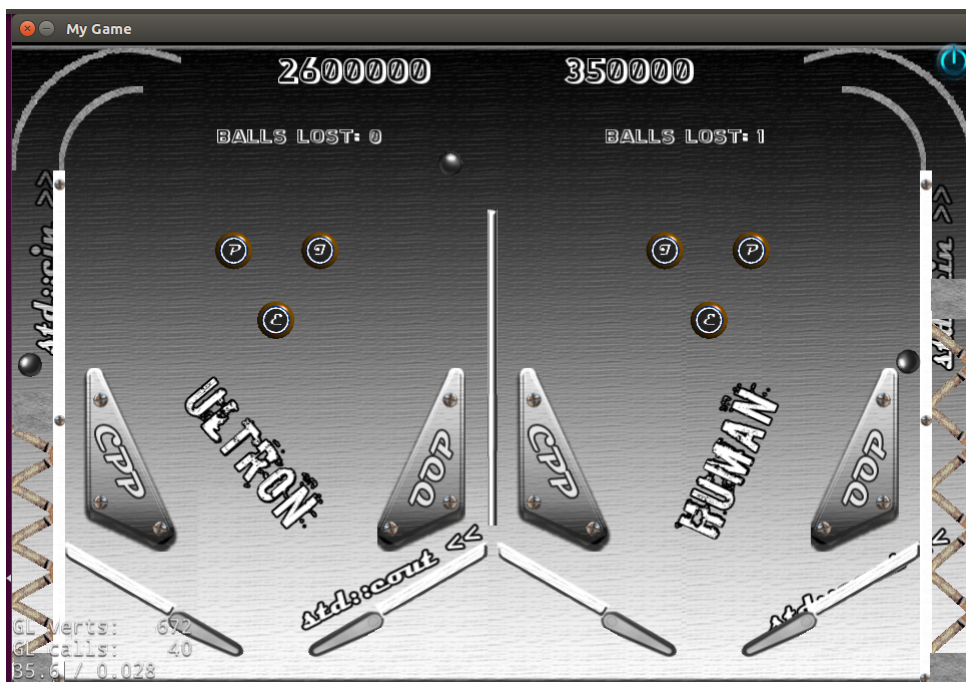


Рисунок 1.3 — Перелет шара.

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 шар
- е) ARR BALL - массив шаров
- ж) FLIPPER TRIGGERED - дополнительные данные

2.2.1 Пакеты сообщений

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 (дополнительные данные):

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

2.3 Алгоритм серверной части

Сервер игры построен по типу «thread per session», т.е. для каждой игровой сессии создается новый поток.

Сервер состоит из следующих потоков

- а) добавления игроков;
- б) соединения;
- в) сессий.

2.3.1 Поток добавления игроков

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. Блок схема потока представлена на рисунке [2.1](#).

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 $K2$

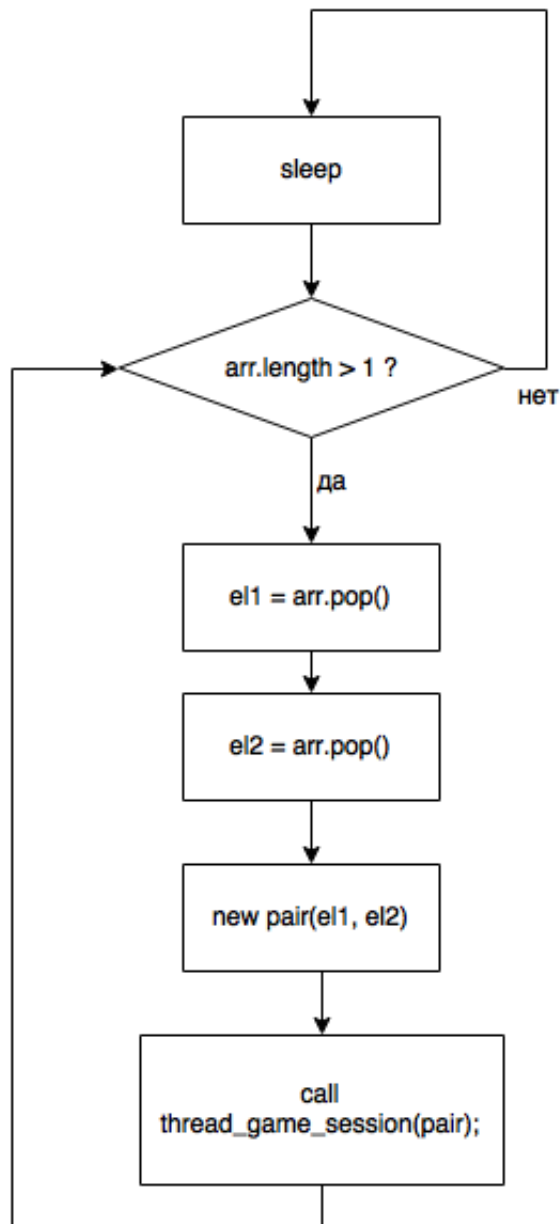


Рисунок 2.1 — Блок схема потока соединения.

- 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

Последовательность действий сервера представлена на рисунке [2.2](#)

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

Последовательность действий клиента представлена на рисунке [2.3](#)

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 Диаграммы классов клиента и сервера

На рисунках [2.4](#) и [2.5](#) представлены диаграммы 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.

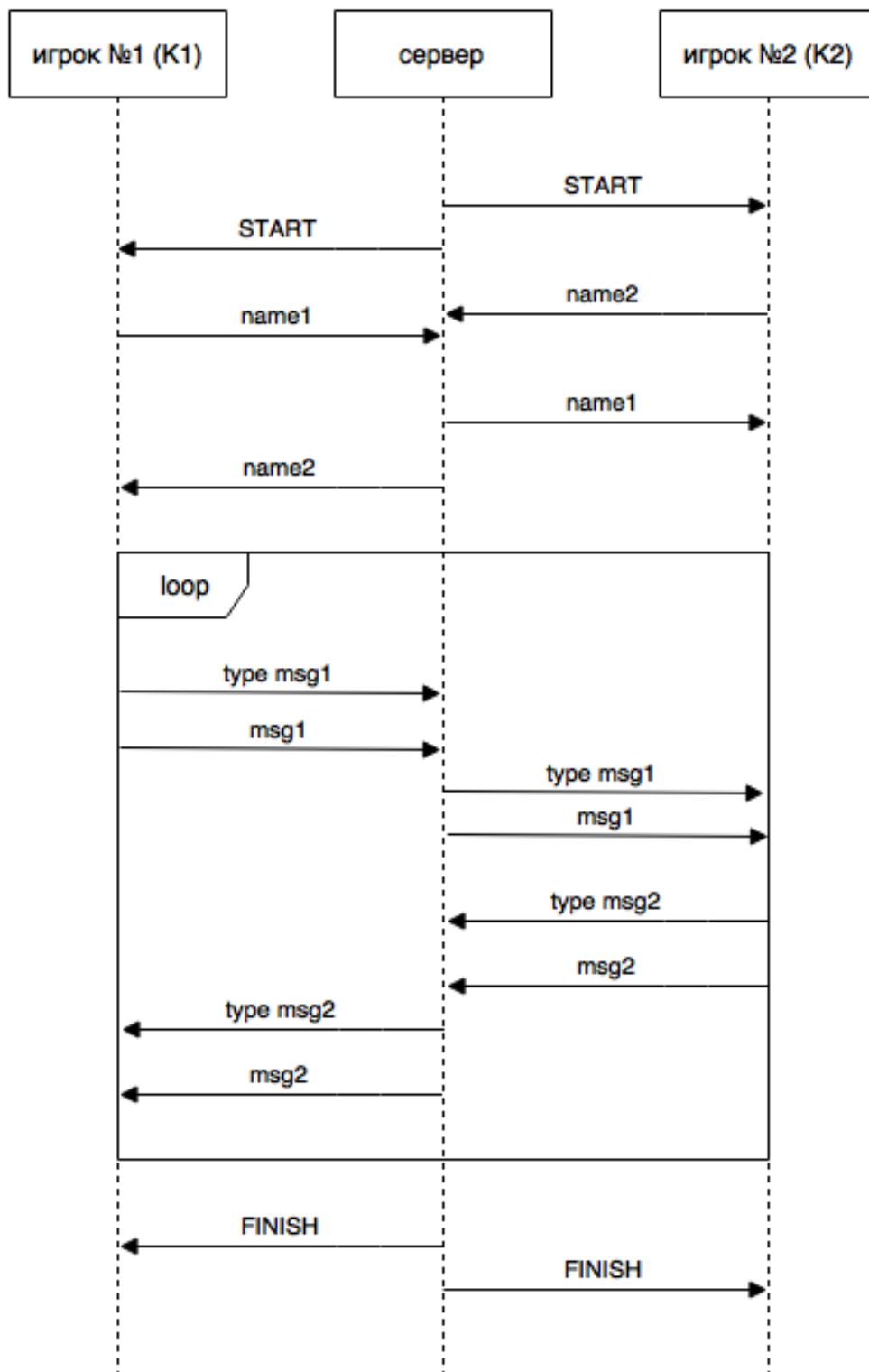


Рисунок 2.2 — Поток сессии.

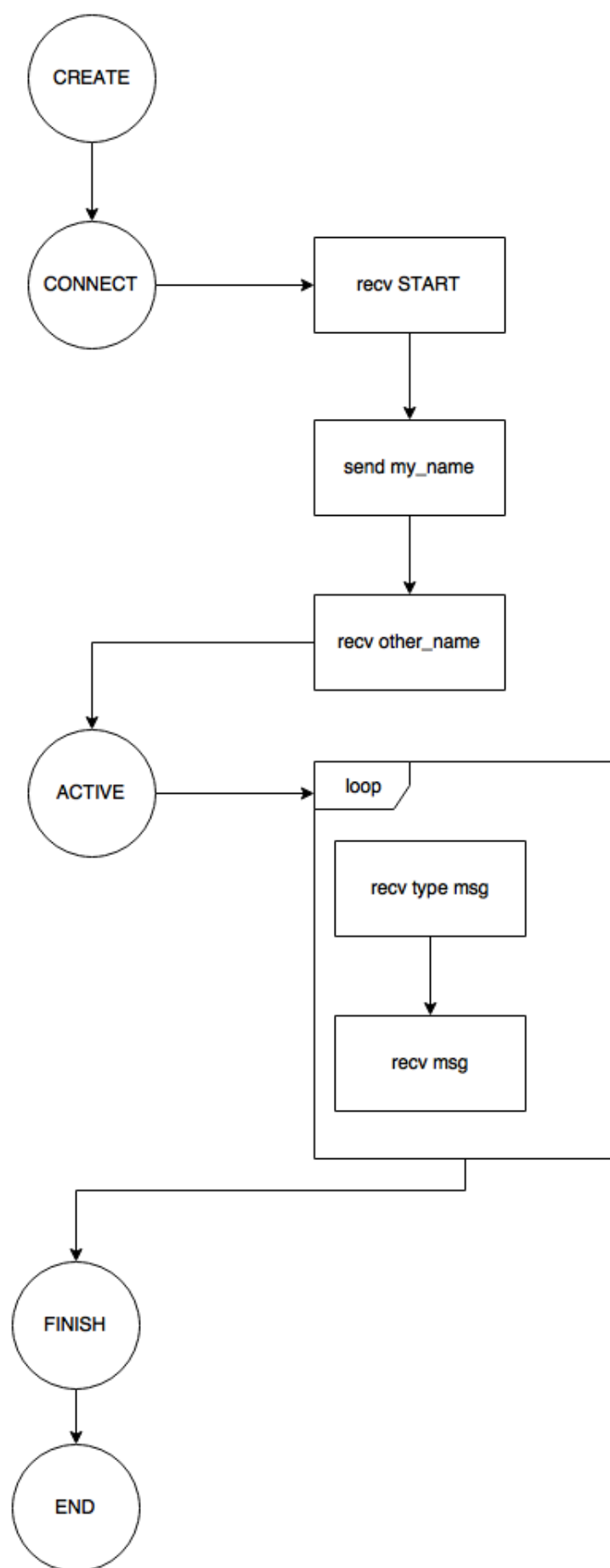


Рисунок 2.3 — Последовательность действий клиента.

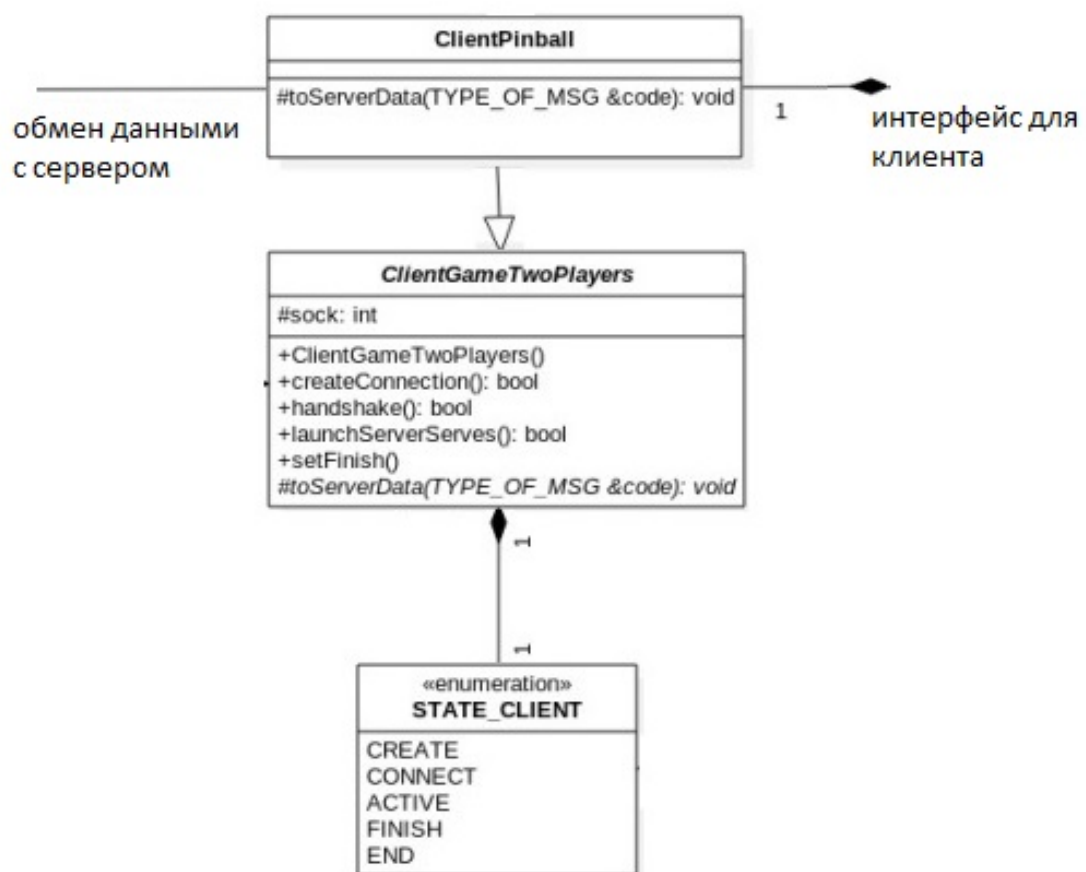


Рисунок 2.4 — Диаграмма классов клиента.

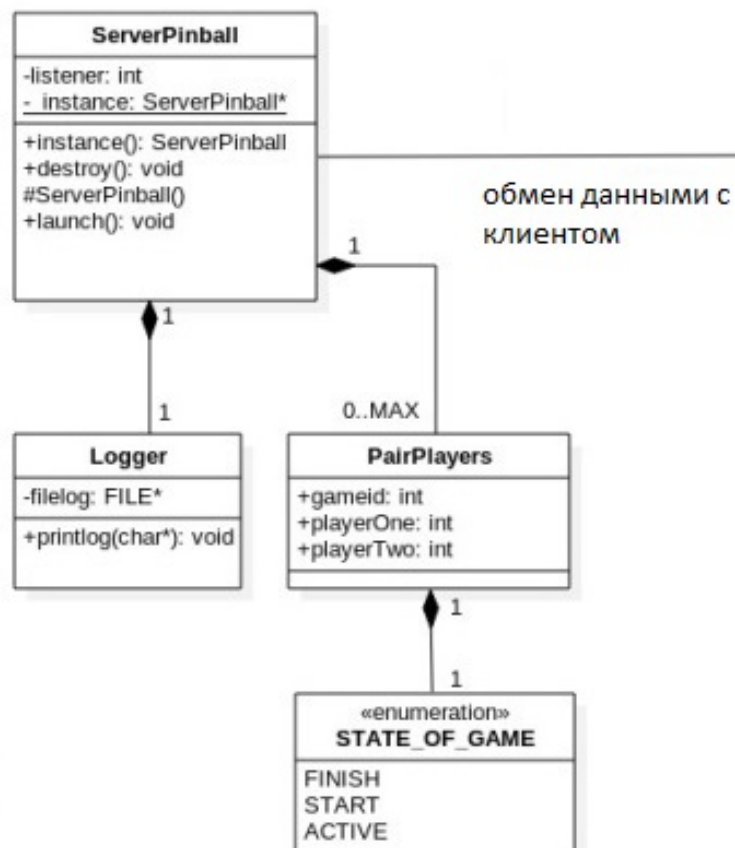


Рисунок 2.5 — Диаграмма классов сервера.

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 TCP. Данный протокол выбран в связи с тем, что потеря данных между клиентами - недопустима. Протокол TCP (Transmission Control Protocol — протокол управления передачей) был специально разработан для обеспечения надежного сквозного байтового потока по ненадежной интернет-сети. [3]

3.3 Проблемы многопоточности

При работе сервера приложения несколько потоков работают с одними и теми же данными. Для взаимоисключений используются мьютексы.

Мьютекс (mutex) - это фактически блокировка, которая устанавливается (запирается) перед обращением к разделяемому ресурсу и снимается (отпирается) после выполнения требуемой последовательности операций. Если мьютекс заперт, то любой другой поток, который попытается запереть его, будет заблокирован до тех пор, пока мьютекс не будет отперт. Если в момент, когда отпирается мьютекс, заблокированными окажутся несколько потоков, все они будут запущены и первый из них, который успеет запереть мьютекс, продолжит работу. Все остальные потоки обнаружат, что мьютекс по-прежнему заперт, и опять перейдут в режим ожидания. Таким образом, доступ к ресурсу сможет получить одновременно только один поток. [4]

3.4 Используемые шаблоны проектирования

При реализации программного комплекса использовались шаблоны проектирования singleton и шаблонный метод. [5]

3.4.1 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 }

```

3.4.2 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             }
10        } while (code != TYPE_OF_MSG::END_GAME);
11    }
12 };

```

```

7         } else if (code == TYPE_OF_MSG::NEW_BALL) {
8             recv(sock, &newBall, sizeof (newBall), 0);
9             this->setPresenceNewBall(true);
10        }
11
12        code = receiveTypeMessage(sock);
13    } while (code != TYPE_OF_MSG::FINISH and this->getState() ==
14    STATE_CLIENT::ACTIVE);
15    };

```

3.5 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;
26 }

```

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
            (arrayBalls->countSimpleBalls), 0);
12        recv(sock, arrayBalls->arrBallsOpponent, arrayBalls->countSimpleBalls
            * sizeof (SimpleBall), 0);
13        queueArrBallsOpponent.push(arrayBalls);
14        mutexArrBallsUnlock();
15    } else if (code == TYPE_OF_MSG::NEW_BALL) {
16        PhysicsBall *physicsBall = new PhysicsBall;
17        mutexNewBallLock();
18        recv(sock, physicsBall, sizeof (PhysicsBall), 0);
19        queueNewBall.push(physicsBall);

```

```

20     mutexNewBallUnlock();
21 }
22 }

```

3.6 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);

```

3.7 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 */

```

3.8 Инструкция по компиляции и запуску игры

Исходные тексты игры можете скачать из данного репозитория <https://github.com/iproha94/MultiplayerPinball>.

а) Установите Ubuntu.

(с Debian могут возникнуть проблемы при запуске скрипта install-deps-linux.sh, пишет что проблемы с OpenGL)

б) Скачайте cocos2d-x-3.6

(именно эта версия должна быть, более новые версия искажают физическое пространство для данной игры)

```

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 3.1 — Пример логга.

в) зайдите в папку с распакованным cocos2d

г) `./build/install-deps-linux.sh`

д) `./tools/travis-scripts/install_glfw.sh`

е) `./setup.py`

ж) `mkdir build/linux-build`

з) `cd build/linux-build`

и) `cmake ../..`

к) `make`

(на данном этапе будут выскакивать ошибки, связанные с `isnan`, в файлах, в которых происходят эти ошибки перед `isnan` надо написать `"std::"` без кавычек)

л) `./bin/cpp-tests/cpp-tests`

м) Возвращаемся в основную папку кокоса. Создание нового проекта:

`cocos new MyGame -p com.your_company.mygame -l cpp -d NEW_PROJECTS_DIR`

н) Запуск

`cocos run -s NEW_PROJECTS_DIR/MyGame -p linux`

о) В `NEW_PROJECTS_DIR/MyGame` удалите папку `Classes`, `Resources` и файл `CMakeLists.txt`

- п) Склонируйте данный репозиторий
- р) Переместите папки Classes, Resources и файл CMakeLists.txt в папку NEW_PROJECTS_DIR/MyGame
- с) Из папки Multiplayer переместите файлы, в которых нет слова "server" в папку Classes

3.9 Запуск сервера

Компиляция исходных кодов сервера осуществляется командами [3.16](#)

Листинг 3.16 — Компиляция исходных кодов сервера

```
1 g++ -c -std=c++11 mainServerPinball.cpp serverPinball.cpp
2 g++ -pthread -std=c++11 mainServerPinball.o serverPinball.o -o serexe
```

Программа сервера не написана в виде демона. При развертывании программы удобно будет использовать утилиту screen. Screen создает отдельные объекты, называемые иногда «скринами». Каждый скрин - это что-то вроде окна, которое можно свернуть-развернуть, если проводить аналогию с графическим интерфейсом. Только вместо окна вы получаете виртуальную консоль, которую можно отправить в фон или вывести на передний план, и в которой запускается указанное приложение.

3.10 Интерфейс взаимодействия сервера с пользователем

Интерфейс взаимодействия сервера с пользователем представлен дружелюбным консольным меню [3.2](#)

В меню отображается следующая информация:

- а) launched - создан ли серверный сокет
- б) listening - прослушиваются ли входящие соединения
- в) copulation - соединяются ли в пары подсоединяющиеся игроки
- г) port - номер порта сервера
- д) quantity of waiteng players - количество ожидающих соединения игроков
- е) quantity of pair of players - количество сессий (играющих пар игроков)
- ж) full quantity of games - количество сыгранных игр

Управление сервером осуществляется следующими командами:

- а) 1 - создать соединение с стандартным портом (49876)
- б) 2 - создать соединение с портом, которое нужно будет ввести с клавиатуры
- в) 3 - закрыть соединение
- г) 4 - начать слушать входящие соединения
- д) 5 - перестать слушать входящие соединения
- е) 6 - начать соединять в пары ожидающих игроков

- ж) 7 - перестать соединять в пары ожидающих игроков
- з) 8 - обновить информацию о состоянии сервера
- и) 0 - выход из программы

```
-----
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.
>> 
```

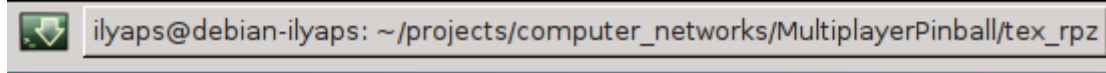


Рисунок 3.2 — Меню сервера.

Заключение

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

Список использованных источников

1. *Wikipedia*. Cocos2d. — <https://ru.wikipedia.org/wiki/Cocos2d>. — 2016. — [Online; accessed 25-October-2016].
2. *@Nucleotide*. Cocos2d-x — разработка простой игры. — <https://habrahabr.ru/post/270133/>. — [Online; accessed 25-October-2016].
3. *Эндрю Таненбаум, Дэвид Уэзеролл*. Компьютерные сети / Дэвид Уэзеролл Эндрю Таненбаум. — Питер, 2016.
4. *Стивен А. Раго, Уильям Ричард Стивенс*. UNIX. Профессиональное программирование / Уильям Ричард Стивенс Стивен А. Раго. — Символ-Плюс, 2015.
5. *Эрих Гамма, Ричард Хелм*. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Ричард Хелм Эрих Гамма. — Питер, 2016.