**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

КУРСОВАЯ РАБОТА

По дисциплине: Сети ЭВМ и телекоммуникации

Тема: “ Разработка клиент-сервер приложения “Крестики-нолики” для сетей Bluetooth”

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Федотов Е.А.

Белгород 2021

# Содержание

Оглавление

[Содержание 2](#_Toc72242727)

[Введение 4](#_Toc72242728)

[1.1. Краткая характеристика протокола TCP 5](#_Toc72242729)

[1.2. Краткая характеристика модели TCP/IP 6](#_Toc72242730)

[1.3. BLE 7](#_Toc72242731)

[Глава 2. Разработка программы 9](#_Toc72242732)

[2.1. Описание модулей. 9](#_Toc72242733)

[2.1.1. Massage. 9](#_Toc72242734)

[Описание модуля 9](#_Toc72242735)

[Спецификация модуля 9](#_Toc72242736)

[ Message 9](#_Toc72242737)

[2.1.2. LobbyScreen. 9](#_Toc72242738)

[Описание модуля 9](#_Toc72242739)

[Спецификация модуля 9](#_Toc72242740)

[ LobbyScreen 9](#_Toc72242741)

[2.1.3. GameScreen. 9](#_Toc72242742)

[Описание модуля 9](#_Toc72242743)

[Спецификация модуля 10](#_Toc72242744)

[ GameScreen 10](#_Toc72242745)

[2.1.4. BluetoothService. 10](#_Toc72242746)

[Описание модуля 10](#_Toc72242747)

[Спецификация модуля 10](#_Toc72242748)

[ BluetoothService 10](#_Toc72242749)

[2.2. Организация сетевого взаимодействия. 10](#_Toc72242750)

[2.2.1. Формат передачи данных. 10](#_Toc72242751)

[2.2.2. Логика взаимодействия клиента и сервера. 11](#_Toc72242752)

[2.3. Структура проекта 12](#_Toc72242753)

[Глава 3. Тестирование ПО. 13](#_Toc72242754)

[Скриншоты 14](#_Toc72242755)

[Заключение 14](#_Toc72242756)

[Список литературы 15](#_Toc72242757)

[ПРИЛОЖЕНИЯ 16](#_Toc72242758)

[Листинг TicTacToe.java 16](#_Toc72242759)

[Листинг LobbyScreen.java 18](#_Toc72242760)

[Листинг GameScreen.java 22](#_Toc72242761)

[Листинг BluetoothService.java 28](#_Toc72242762)

[Листинг Message.java 28](#_Toc72242763)

# Введение

Сегодня все крупные и успешные коммерческие приложения активно взаимодействуют с интернетом и между собой, эти приложения могут быть как веб, так и нативными, могут работать на различных протоколах (например, WebSocket). Мы будем работать с протоколом Bluetooth.

Во все время актуальность протокола Bluetooth заключается в надёжной передаче данных между двумя удалёнными устройствами или программами, находящимися в зоне действия сети.

Bluetooth обеспечивает обмен информацией между такими устройствами, как персональные компьютеры (настольные, карманные, ноутбуки), мобильные телефоны, интернет-планшеты, принтеры, цифровые фотоаппараты, мыши, клавиатуры, джойстики, наушники, гарнитуры и акустические системы на надёжной, бесплатной, повсеместно доступной радиочастоте для ближней связи.

В курсовой работе мы подробно рассмотрим протокол Bluetooth, разработаем программу “Крестики-нолики” на языке программирования java, для того, чтобы на примере показать сетевое Bluetooth взаимодействие клиент-сервера Bluetooth

**Глава 1. Протокол Bluetooth, модель TCP/IP, BLE.**

# Краткая характеристика протокола TCP

TCP (transmission control protocol) – «протокол управления передачей», предназначенный для управления передачей данных.

Разработка практически любого сетевого программного продукта как правило требует наличия сетевого протокола передачи данных, который позволит передать требуемые данные между программами.

Определить протокол для создания сетевого взаимодействия между программами следует осмысленно, так как есть TCP и UDP, и далеко не во всех случаях следует выбирать один и тот же протокол, например, для приложений, работающих в режиме “Реального времени” подошёл бы скорее UDP протокол, а для работы сервера лицензирования какого-нибудь коммерческого программного обеспечения идеально бы подошёл TCP протокол, так как там больше важна не скорость, а надёжность передачи.

Механизм TCP осуществляет установление надежного соединения между двумя устройствами и надежную передачу данных (потока байтов) от одной программы на некотором компьютере к другой программе на другом компьютере. Протокол TCP позволяет пересылать большие объемы информации, разделяя ее на отдельные части – пакеты, – которые доставляются получателю независимо. Также TCP отвечает за правильное восстановление данных из пакетов, полученных после передачи. В частности, TCP контролирует размер и целостность передаваемого пакета данных, автоматически осуществляя повторную пересылку при сбое передачи (при потере или повреждении данных), а также устраняет дублирование при получении двух копий одного пакета информации. Таким образом, TCP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи.

Абсолютное ограничение на размер пакета TCP составляет 64 КБ (65535 байт), включая заголовки, то есть полезная передаваемая нагрузка порядком меньше, чем 65535 байт.

TCP пакет имеет следующую структуру:



Рис 1.1. Структура TCP пакета.

# Краткая характеристика модели TCP/IP

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. Название TCP/IP происходит из двух важнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были первыми разработаны и описаны в данном стандарте.

Протоколы TCP/IP делятся на 4 уровня.

* Самый нижний (уровень IV) соответствует физическому и канальному уровням модели OSI. Этот уровень в протоколах TCP/IP не регламентируется, но поддерживает все популярные стандарты физического и канального уровня: для локальных сетей это Ethernet, Token Ring, FDDI, Fast Ethernet, 100VG-AnyLAN, для глобальных сетей - протоколы соединений «точка- точка» SLIP и PPP, протоколы территориальных сетей с коммутацией пакетов X.25, frame relay.
* Следующий уровень (уровень III) - это уровень межсетевого взаимодействия, который занимается передачей пакетов с использованием различных транспортных технологий локальных сетей, территориальных сетей, линий специальной связи и т.п.

В качестве основного протокола сетевого уровня (в терминах модели OSI) в стеке используется протокол IP, который изначально проектировался как протокол передачи пакетов в составных сетях, состоящих из большого количества локальных сетей, объединенных как локальными, так и глобальными связями. Поэтому протокол IP хорошо работает в сетях со

сложной топологией, рационально используя наличие в них подсистем и экономно расходуя пропускную способность низкоскоростных линий связи. Протокол IP является дейтаграммным протоколом, то есть он не гарантирует доставку пакетов до узла назначения, но старается это сделать.

К уровню межсетевого взаимодействия относятся и все протоколы, связанные с составлением и модификацией таблиц маршрутизации, такие как протоколы сбора маршрутной информации RIP (Routing Internet Protocol) и OSPF (Open Shortest Path First), а также протокол межсетевых управляющих сообщений ICMP (Internet Control Message Protocol). Последний протокол предназначен для обмена информацией об ошибках между маршрутизаторами сети и узлом

- источником пакета. С помощью специальных пакетов ICMP сообщается о невозможности доставки пакета, о превышении времени жизни или продолжительности сборки пакета из фрагментов, об аномальных величинах параметров, об изменении маршрута пересылки и типа обслуживания, о состоянии системы и т.п.

* Следующий уровень (уровень II) называется основным. На этом уровне функционируют протокол управления передачей TCP (Transmission Control Protocol) и протокол дейтаграмм пользователя UDP (User Datagram Protocol). Протокол TCP обеспечивает надежную передачу сообщений между удаленными прикладными процессами за счет образования виртуальных соединений. Протокол UDP обеспечивает передачу прикладных пакетов дейтаграммным способом, как и IP, и выполняет только функции связующего звена между сетевым протоколом и многочисленными прикладными процессами.
* Верхний уровень (уровень I) называется прикладным. За долгие годы использования в сетях различных стран и организаций стек TCP/IP накопил большое количество протоколов и сервисов прикладного уровня. К ним относятся такие широко используемые протоколы, как протокол копирования файлов FTP, протокол эмуляции терминала telnet, почтовый протокол SMTP, используемый в электронной почте сети Internet, гипертекстовые сервисы доступа к удаленной информации, такие как WWW и многие другие. Остановимся несколько подробнее на некоторых из них.

# BLE

Принцип работы BLE описан уже в его названии: Low Energy. Протокол подразумевает передачу данных короткими пакетами по необходимости, затем – выключение передатчика. Низкое энергопотребление частично достигается применением именно этого принципа. Вместо классического тандема в обычном Bluetooth, устройства BLE связываются друг с другом лишь при необходимости отправки или получения информации.

Протокол BLE строго структурирован по принципу своей коммуникации с другими устройствами. Вначале девайсы изучают доступные сервисы для отправки/принятия данных; неотъемлемая часть этих сервисов – их характеристики (characteristics), определяющие тип данных для будущей передачи. Характеристики, из соображений наглядности, могут иметь в своём составе описания-дескрипторы (descriptors), которые помогают определить тип данных. К примеру, разберём сервис под названием «Heart Rate Monitor» (монитор частоты сердцебиения) – среди его характеристик присутствуют такие, как «измерение пульса».

Большинство API для Bluetooth LE позволяют искать локальные устройства и определять доступные в них сервисы, характеристики и дескрипторы.

# Глава 2. Разработка программы

# Описание модулей.

Наша программа состоит из основного исходного файла TicTacToe.java и других вспомогательных модулей (GameScreen.java, LobbyScreen.java, BluetoothService.java, Message.java.

Рассмотрим все модули по подробнее.

# Massage.

### Описание модуля

Massage отвечает за вывод всплывающих сообщений на экран пользователя таких как: соединение, разрыв соединения, победа, ничья и др.

### Спецификация модуля

Модуль Massage состоит из 1 класса:

### Message

Назначение: представляет собой структуру данных, которая будет использоваться при выводе соообщений.

# LobbyScreen.

### Описание модуля

LobbyScreen представляет собой модуль - интерфейс, предназначенный для присоединения к серверному устройству как клиент, или для создания своего сервера и ожидания клиента.

### Спецификация модуля

Модуль LobbyScreen из 1 класса:

### LobbyScreen

Назначение: предназначен для включения и выключения Bluetooth на устройстве и соединения между игроками.

# GameScreen.

### Описание модуля

GameScreen представляет собой модуль в котором происходит основная игра и взаимодействие между клиентом и сервером.

### Спецификация модуля

Модуль GameScreen из 1 класса:

### GameScreen

Назначение: отвечает за все игровые моменты между игроками

# BluetoothService.

### Описание модуля

BluetoothService представляет собой модуль, отвечающий за Bluetooth.

### Спецификация модуля

Модуль BluetoothService из 1 класса:

### BluetoothService

Назначение: содержит все нужные функции для работы с Bluetooth

# Организация сетевого взаимодействия.

# Формат передачи данных.

Сперва следует определить структуру элементарного пакета, который позволил бы нам структурировать какие-то данные нашей предметной области таким образом, чтобы мы могли в будущем просто, удобно и однозначно передавать, и обрабатывать какие-то данные.

Наша предметная область связана с разработкой игры, а в игре всё основывается на состояниях и действиях. Состояния наших игроков будут изменяться под воздействием каких-либо действий, поэтому в нашем пакете следует выделить как минимум одно поле, которое будет удалённому клиенту сообщать какое действие стоит ему выполнить.

Ввиду того, что существует огромное множество ситуаций, когда только типа действия просто недостаточно для выполнения какого действия и требуется какая-то дополнительная информация, нам следует добавить ещё одно поле, которое будет являться полезной нагрузкой в нашем пакете и содержать информацию, которая будет необходима при выполнении действия определённого в пакете удалённым пользователем.

# Логика взаимодействия клиента и сервера.

Логику работы клиента и сервера описать достаточно просто. Наша программа запустившись автоматически запускает сервер.

Так же учтём возможность подключения нашей программы к удалённому серверу, но стоит отметить, что сперва следует остановить сервер для предотвращения проблем в дальнейшем.

При соединении клиента и сервера, клиент и сервер должны будут отправить друг другу свои никнеймы заданные изначально при запуске программы, для того, чтобы можно было во время игры вести игровой счёт и видеть, какому игроку какой счёт соответствует.

Далее происходит сама игра. Определим правило, что первый игрок, который будет ходить и будет в протяжении всей игровой сессии крестиком это будет сервер. В протяжении всей игры поочерёдно клиент будет отправлять серверу, а сервер клиенту пакет с действием о постановке крестика или нолика, который программа должна моментально же выполнить.

# Структура проекта

Наш проект будет состоять из следующих файлов:

|  |  |
| --- | --- |
| Название | Назначение |
| build.gradle | Файлы проекта с информацией о том:   * Как построить проект. * Настройки проекта. * О сборках, на которые ссылается проект. |
| settings. gradle |
| LobbyScreen.java | Лобби нашей игры |
| GameScreen.java | Основная игра |
| Message.java | Сообщения игры |
| BluetoothService.java | Функции для работы с Bluetooth |

И следующих файлов ресурсов:

|  |  |
| --- | --- |
| Название | Назначение |
| ic\_launcher.pnd | иконка программы |
| X.png | Картинка х |
| O.pnd | Картинка о |
| Bluetooth.png | Картинка Bluetooth |
| Field.png | Картинка поля |

# Глава 3. Тестирование ПО.

Тестирование программного обеспечения (ПО) - это процесс исследования ПО с целью выявления ошибок и определения соответствия между реальным и ожидаемым поведением ПО, осуществляемый на основе набора тестов, выбранных определённым образом. В более широком смысле, тестирование ПО - это техника контроля качества программного продукта и анализ полученных результатов в результате тестирования.

Для проверки нашей уже скомпилированной программы на наличие ошибки, мы проведём 10 тестовых и 5 контрольных запусков нашей программы, на основе полученных результатов сделаем вывод.

В случае возникновения аварийной ситуации на контрольном тесте, произведём правки исходного кода нашей программы и проведём тестовый цикл заново.

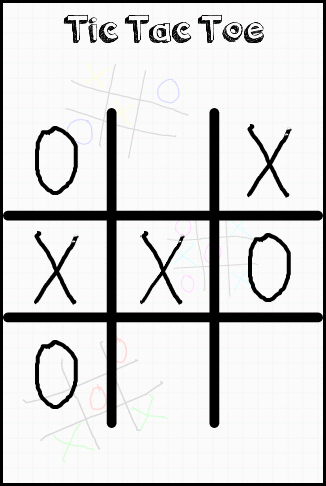
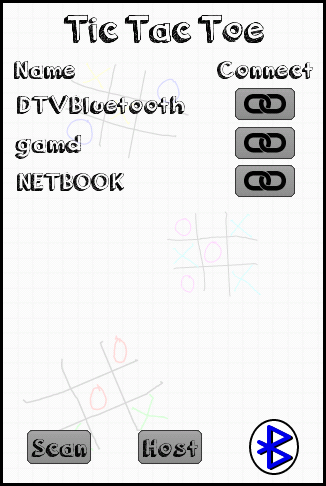
|  |  |  |
| --- | --- | --- |
| Номер запуска | Ошибки | Статус |
| 1 | Отсутствуют | Тест |
| 2 | Отсутствуют | Тест |
| 3 | Отсутствуют | Тест |
| 4 | Отсутствуют | Тест |
| 5 | Отсутствуют | Тест |
| 6 | Отсутствуют | Тест |
| 7 | Отсутствуют | Тест |
| 8 | Отсутствуют | Тест |
| 9 | Отсутствуют | Тест |
| 10 | Отсутствуют | Тест |
| 11 | Отсутствуют | Контроль |
| 12 | Отсутствуют | Контроль |
| 13 | Отсутствуют | Контроль |
| 14 | Отсутствуют | Контроль |
| 15 | Отсутствуют | Контроль |

Рис 4.1. Результаты проведённых тестов.

В результате проведённых тестов ошибок выявлено не было, все алгоритмы были отработаны правильно, звуковые эффекты и графические элементы были проверены, а также были проверены возможности сетевой игры как в локальной, так и глобальной сети интернет.

В заключении можно сделать вывод, что в результате успешного тестирования нашего ПО, можно переходить к процедуре внедрения программного обеспечения в эксплуатацию.

# Скриншоты

****

# 

# Заключение

В данной работе мы подробно разобрали, что такое Bluetooth и TCP/IP, где этот протокол сегодня используют и вообще следует использовать, а также его особенности, после краткого теоретического материала мы описали структуру нашей будущей программы в виде спецификаций, по которым далее мы разработали программу “крестики-нолики”, написав её на языке Java.

Написав и скомпилировав нашу программу, мы провели её тестирование, устранили недостатки и получили конечный продукт, который является результатом нашей курсовой работы и показывает, что мы освоили материал, связанный с протоколами и позволяющий нам разрабатывать сетевые приложения.

# Список литературы

1. Шилдт Г. - Java 8. Полное руководство. 9-е издание (2015)
2. Сайт <http://developer.alexanderklimov.ru/android/>
3. Сайт <https://developers.google.com/maps/documentation/android-sdk/marker>
4. Сайт https://javatalks.ru/topics/42851

# ПРИЛОЖЕНИЯ

## Листинг TicTacToe.java

package com.merltech.tictactoe;  
  
import com.badlogic.gdx.Game;  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.InputMultiplexer;  
import com.badlogic.gdx.files.FileHandle;  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.BitmapFont;  
import com.badlogic.gdx.graphics.g2d.SpriteBatch;  
import com.badlogic.gdx.graphics.g2d.TextureAtlas;  
import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator;  
import com.badlogic.gdx.scenes.scene2d.ui.Skin;  
import com.merltech.tictactoe.Screens.GameScreen;  
import com.merltech.tictactoe.Screens.LobbyScreen;  
import com.merltech.tictactoe.graphics.SVGService;  
import com.merltech.tictactoe.network.BluetoothService;  
  
  
import java.io.InputStream;  
import java.io.OutputStream;  
  
public class TicTacToe extends Game {  
 private SVGService svgService;  
 private final float density;  
 public SpriteBatch batch;  
 public Skin skin;  
 public Texture background;  
 public InputMultiplexer inputMultiplexer;  
 public BluetoothService bluetoothService;  
  
 public LobbyScreen lobbyScreen;  
 public GameScreen gameScreen;  
  
 private final String Tag = "TicTacToe";  
  
 public TicTacToe(BluetoothService bluetoothService, SVGService svgService, float density) {  
 this.bluetoothService = bluetoothService;  
 this.svgService = svgService;  
 this.density = density;  
 }  
  
 @Override  
 public void create () {  
 this.batch = new SpriteBatch();  
  
 generateGraphics();  
 generateFonts();  
  
 background = new Texture(Gdx.*files*.local("background.png"));  
  
 inputMultiplexer = new InputMultiplexer();  
 Gdx.*input*.setInputProcessor(inputMultiplexer);  
 Gdx.*input*.setCatchBackKey(true);  
  
 *// create screens* lobbyScreen = new LobbyScreen(this);  
 gameScreen = new GameScreen(this);  
  
 this.setScreen(lobbyScreen);  
 }  
  
 private void generateGraphics() {  
 InputStream inputStream;  
 OutputStream outputStream;  
 if(!Gdx.*files*.local("bluetooth.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/bluetooth.svg").read();  
 outputStream = Gdx.*files*.local("bluetooth.png").write(false);  
 svgService.scaleToDPI(inputStream, outputStream);  
 }  
 if(!Gdx.*files*.local("connect.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/connect.svg").read();  
 outputStream = Gdx.*files*.local("connect.png").write(false);  
 svgService.scaleToDPI(inputStream, outputStream);  
 }  
 if(!Gdx.*files*.local("background.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/background.svg").read();  
 outputStream = Gdx.*files*.local("background.png").write(false);  
 svgService.fill(inputStream, outputStream, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
 }  
 if(!Gdx.*files*.local("field.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/field.svg").read();  
 outputStream = Gdx.*files*.local("field.png").write(false);  
 svgService.fit(inputStream, outputStream, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
 }  
 if(!Gdx.*files*.local("X.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/X.svg").read();  
 outputStream = Gdx.*files*.local("X.png").write(false);  
 svgService.scaleToDPI(inputStream, outputStream);  
 }  
 if(!Gdx.*files*.local("O.png").exists()) {  
 inputStream = Gdx.*files*.internal("ui/O.svg").read();  
 outputStream = Gdx.*files*.local("O.png").write(false);  
 svgService.scaleToDPI(inputStream, outputStream);  
 }  
 }  
  
 private void generateFonts() {  
 FreeTypeFontGenerator orangeJuiceGenerator = new FreeTypeFontGenerator(Gdx.*files*.internal("ui/orangejuice.ttf"));  
 FreeTypeFontGenerator unispaceGenerator = new FreeTypeFontGenerator(Gdx.*files*.internal("ui/unispace.ttf"));  
 FreeTypeFontGenerator.FreeTypeFontParameter parameter = new FreeTypeFontGenerator.FreeTypeFontParameter();  
  
 *// this has to be related to density on screen* parameter.size = (int)(density / 14);  
 BitmapFont defaultFont = unispaceGenerator.generateFont(parameter);  
 parameter.size = (int)(density / 6);  
 BitmapFont smallFont = orangeJuiceGenerator.generateFont(parameter);  
 parameter.size = (int)(density / 4);  
 BitmapFont bigFont = orangeJuiceGenerator.generateFont(parameter);  
  
 skin = new Skin();  
 skin.add("default-font", defaultFont, BitmapFont.class);  
 skin.add("small-font", smallFont, BitmapFont.class);  
 skin.add("big-font", bigFont, BitmapFont.class);  
 FileHandle fileHandle = Gdx.*files*.internal("ui/uiskin.json");  
 FileHandle atlasFile = Gdx.*files*.internal("ui/uiskin.atlas");  
 if(atlasFile.exists()) {  
 skin.addRegions(new TextureAtlas(atlasFile));  
 }  
 skin.load(fileHandle);  
 orangeJuiceGenerator.dispose();  
 unispaceGenerator.dispose();  
 }  
  
 @Override  
 public void render () {  
 super.render();  
 }  
  
  
 @Override  
 public void dispose () {  
 bluetoothService.dispose();  
 }  
}

## Листинг LobbyScreen.java

package com.merltech.tictactoe.Screens;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.Input;  
import com.badlogic.gdx.Screen;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.GL20;  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.TextureRegion;  
import com.badlogic.gdx.scenes.scene2d.InputEvent;  
import com.badlogic.gdx.scenes.scene2d.Stage;  
import com.badlogic.gdx.scenes.scene2d.ui.Dialog;  
import com.badlogic.gdx.scenes.scene2d.ui.ImageButton;  
import com.badlogic.gdx.scenes.scene2d.ui.Label;  
import com.badlogic.gdx.scenes.scene2d.ui.Skin;  
import com.badlogic.gdx.scenes.scene2d.ui.Table;  
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
import com.badlogic.gdx.scenes.scene2d.ui.Value;  
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
import com.badlogic.gdx.scenes.scene2d.utils.Drawable;  
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
import com.badlogic.gdx.utils.viewport.ScreenViewport;  
import com.merltech.tictactoe.TicTacToe;  
import com.merltech.tictactoe.network.BluetoothService;  
import com.merltech.tictactoe.network.Message;  
  
public class LobbyScreen implements Screen {  
 private final TicTacToe game;  
 private final BluetoothService bluetoothService;  
 private Skin skin;  
 private Stage stage;  
 float runningTime = 0;  
  
 *// UI* private Table peerTable;  
 private Dialog noBluetoothDialog;  
 private ImageButton bluetoothButton;  
 private Label connectLabel;  
 private Label nameLabel;  
 boolean animateBluetooth = false;  
  
 private final String Tag = "Lobby";  
  
 public LobbyScreen(TicTacToe game) {  
 this.game = game;  
 this.skin = game.skin;  
 this.bluetoothService = game.bluetoothService;  
  
 setupUi();  
 }  
  
 private void setupUi() {  
 stage = new Stage(new ScreenViewport());  
  
 Texture bluetoothIcon = new Texture(Gdx.*files*.local("bluetooth.png"));  
 Drawable drawable = new TextureRegionDrawable(new TextureRegion(bluetoothIcon));  
 bluetoothButton = new ImageButton(drawable);  
  
 noBluetoothDialog = new Dialog("No Bluetooth", skin) {  
 protected void result(Object object) {  
  
 };  
 };  
 Label enableBluetooth = new Label("Please enable bluetooth first", skin);  
 noBluetoothDialog.text(enableBluetooth);  
 noBluetoothDialog.button("Okay").center();  
 noBluetoothDialog.setMovable(false);  
  
 *// default style* TextButton.TextButtonStyle style = new TextButton("a", skin).getStyle();  
 style.font = skin.getFont("small-font");  
 final TextButton scanButton = new TextButton("Scan", style);  
 scanButton.getLabel().setColor(Color.*BLACK*);  
 final TextButton hostButton = new TextButton("Host", style);  
 hostButton.getLabel().setColor(Color.*BLACK*);  
  
 hostButton.addListener(new ClickListener(){  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 if(!bluetoothService.isEnabled()) {  
 noBluetoothDialog.show(stage);  
 } else {  
 bluetoothService.makeDiscoverable(30);  
 game.setScreen(game.gameScreen);  
 }  
 }  
 });  
  
 scanButton.addListener(new ClickListener(){  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 if(!bluetoothService.isEnabled()) {  
 noBluetoothDialog.show(stage);  
 } else {  
 bluetoothService.scan();  
 peerTable.clear();  
 peerTable.add(nameLabel).left().expandX();  
 peerTable.add(connectLabel).right();  
 }  
 }  
 });  
  
 bluetoothButton.addListener(new ClickListener(){  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 if(bluetoothService.isEnabled()) {  
 bluetoothService.disable();  
 *// starts at running time = PI/4* runningTime = (float)Math.*PI* / 4;  
 } else {  
 bluetoothService.enable();  
 runningTime = 0.0f;  
 }  
 animateBluetooth = true;  
 }  
 });  
  
 Table rootTable = new Table(skin);  
 rootTable.setFillParent(true);  
  
 Label titleLabel = new Label("Tic Tac Toe", skin, "big-font", Color.*BLACK*);  
 rootTable.add(titleLabel).center().expandX();  
  
 rootTable.row().expandY().top();  
 peerTable = new Table(skin);  
 nameLabel = new Label("Name", skin, "small-font", Color.*BLACK*);  
 peerTable.add(nameLabel).left().expandX();  
 connectLabel = new Label("Connect", skin, "small-font", Color.*BLACK*);  
 peerTable.add(connectLabel).right();  
 peerTable.pad(Value.*zero*, Value.*percentWidth*(0.03f, rootTable), Value.*zero*, Value.*percentWidth*(0.03f, rootTable));  
 rootTable.add(peerTable).fillX();  
  
 rootTable.row().bottom();  
 Table buttonTable = new Table(skin);  
 buttonTable.add(scanButton).center().expandX();  
 buttonTable.add(hostButton).center().expandX();  
 buttonTable.add(bluetoothButton).center().expandX();  
 rootTable.add(buttonTable).fillX();  
  
 stage.addActor(rootTable);  
 }  
  
 private void processMessages() {  
 Message message = null;  
 while((message = bluetoothService.getMessage()) != null) {  
 Gdx.*app*.log(Tag, "received message: " + message.code.name());  
 switch(message.code) {  
 case *BLUETOOTH\_ADAPTER\_ENABLED*:  
 Gdx.*app*.log(Tag, "bluetooth enabled");  
 animateBluetooth = false;  
 bluetoothButton.getImage().setColor(1.0f, 1.0f, 1.0f, 1.0f);  
 break;  
 case *BLUETOOTH\_ADAPTER\_DISABLED*:  
 Gdx.*app*.log(Tag, "bluetooth disabled");  
 animateBluetooth = false;  
 bluetoothButton.getImage().setColor(0.3f, 0.3f, 0.3f, 1.0f);  
 break;  
 case *BLUETOOTH\_CONNECTED*:  
 Gdx.*app*.log(Tag, "bluetooth connected");  
 game.setScreen(game.gameScreen);  
 break;  
 case *BLUETOOTH\_DEVICE\_FOUND*:  
 Gdx.*app*.log(Tag, "bluetooth device found");  
 addPeer((BluetoothService.BluetoothPeer)message.data);  
 break;  
 case *BLUETOOTH\_ERROR*:  
 Gdx.*app*.log(Tag, "ERROR: " + message.data);  
 default:  
 Gdx.*app*.log(Tag, "message not handled, discarded");  
 }  
 }  
 }  
  
 private void addPeer(BluetoothService.BluetoothPeer peer) {  
 Texture background = new Texture(Gdx.*files*.local("connect.png"));  
 Drawable drawable = new TextureRegionDrawable(new TextureRegion(background));  
  
 ImageButton.ImageButtonStyle imageButtonStyle = new ImageButton.ImageButtonStyle();  
 TextButton.TextButtonStyle textButtonStyle = new TextButton("a", skin).getStyle();  
 imageButtonStyle.up = textButtonStyle.up;  
 imageButtonStyle.down = textButtonStyle.down;  
 imageButtonStyle.checked = textButtonStyle.checked;  
 imageButtonStyle.imageUp = drawable;  
 imageButtonStyle.imageDown = drawable;  
 imageButtonStyle.imageChecked = drawable;  
  
  
 peerTable.row().pad(Value.*percentWidth*(0.01f, peerTable), Value.*zero*, Value.*percentWidth*(0.01f, peerTable), Value.*zero*);  
 String name = peer.Name;  
 Label label = new Label(name, skin, "small-font", Color.*BLACK*);  
  
 *// I am a lazy fuck...* for(int n = name.length(); label.getWidth() > peerTable.getWidth() \* 0.65f; --n) {  
 if(n == 0) {  
 break;  
 }  
 name = peer.Name.substring(0, n - 1) + "...";  
 label = new Label(name, skin, "small-font", Color.*BLACK*);  
 }  
  
 peerTable.add(label).left().pad(Value.*zero*, Value.*percentWidth*(0.01f, peerTable), Value.*zero*, Value.*zero*);  
 *// The address makes everything to large  
 //peerTable.add(peer.Address);* final ImageButton connectButton = new ImageButton(imageButtonStyle);  
 final String buttonAddress = peer.Address;  
 connectButton.addListener(new ClickListener(){  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 Gdx.*app*.log(Tag, "connecting to: " + buttonAddress);  
 bluetoothService.connect(buttonAddress);  
 }  
 });  
 peerTable.add(connectButton).center();  
 }  
  
 @Override  
 public void show() {  
 Gdx.*app*.log(Tag, "showing");  
 game.inputMultiplexer.addProcessor(stage);  
 float brightness = bluetoothService.isEnabled() ? 1.0f : 0.3f;  
 animateBluetooth = false;  
 bluetoothButton.getImage().setColor(brightness, brightness, brightness, 1.0f);  
 }  
  
 @Override  
 public void render(float delta) {  
 if(Gdx.*input*.isKeyJustPressed(Input.Keys.*BACK*)) {  
 *// back key was pressed in lobby -> exit application* Gdx.*app*.exit();  
 }  
 runningTime += Math.*min*(0.03f, Gdx.*graphics*.getDeltaTime());  
 if(animateBluetooth) {  
 float brightness = (float)(Math.*sin*(runningTime\*4)+1) / 2 \* 0.7f + 0.3f;  
 bluetoothButton.getImage().setColor(brightness, brightness, brightness, 1.0f);  
 }  
 processMessages();  
  
 Gdx.*gl*.glClearColor(0.8f, 0.8f, 0.8f, 1);  
 Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);  
 game.batch.begin();  
 game.batch.draw(game.background, 0, 0);  
 game.batch.end();  
 stage.act();  
 stage.draw();  
 }  
  
 @Override  
 public void resize(int width, int height) {  
  
 }  
  
 @Override  
 public void pause() {  
  
 }  
  
 @Override  
 public void resume() {  
  
 }  
  
 @Override  
 public void hide() {  
 Gdx.*app*.log(Tag, "hiding");  
 game.inputMultiplexer.clear();  
  
 }  
  
 @Override  
 public void dispose() {  
  
 }  
}

## Листинг GameScreen.java

package com.merltech.tictactoe.Screens;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.Input;  
import com.badlogic.gdx.Screen;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.GL20;  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.Sprite;  
import com.badlogic.gdx.math.Rectangle;  
import com.badlogic.gdx.math.Vector2;  
import com.badlogic.gdx.scenes.scene2d.Stage;  
import com.badlogic.gdx.scenes.scene2d.ui.Dialog;  
import com.badlogic.gdx.scenes.scene2d.ui.Label;  
import com.badlogic.gdx.scenes.scene2d.ui.Skin;  
import com.badlogic.gdx.scenes.scene2d.ui.Table;  
import com.badlogic.gdx.utils.Timer;  
import com.badlogic.gdx.utils.viewport.ScreenViewport;  
import com.merltech.tictactoe.TicTacToe;  
import com.merltech.tictactoe.network.BluetoothService;  
import com.merltech.tictactoe.network.Message;  
  
public class GameScreen implements Screen {  
 private final TicTacToe game;  
 private final BluetoothService bluetoothService;  
 private Skin skin;  
 private Stage stage;  
 private Sprite fieldSprite;  
 private Sprite oSprite;  
 private Sprite xSprite;  
  
 private int playerValue;  
 private final int[][] field;  
 private Dialog dialog;  
 private Label dialogLabel;  
 private boolean gameRunning = false;  
 private boolean hasTurn = false;  
 private Rectangle[][] collisionRectangles;  
  
 private final String Tag = "Game";  
  
 public GameScreen(TicTacToe game) {  
 this.game = game;  
 this.skin = game.skin;  
 this.bluetoothService = game.bluetoothService;  
 playerValue = 0;  
 field = new int[3][3];  
  
 setupUi();  
 oSprite = new Sprite(new Texture(Gdx.*files*.local("O.png")));  
 xSprite = new Sprite(new Texture(Gdx.*files*.local("X.png")));  
 }  
  
 private int getWinner() {  
 int row\_sum = 0;  
 int column\_sum = 0;  
 int diag\_sum\_1 = 0;  
 int diag\_sum\_2 = 0;  
 for(int a = 0; a < 3; ++a) {  
 diag\_sum\_1 += field[a][a];  
 diag\_sum\_2 += field[2-a][a];  
 }  
  
 for(int a = 0; a < 3; ++a) {  
 row\_sum = 0;  
 column\_sum = 0;  
 for(int b = 0; b < 3; ++b) {  
 row\_sum += field[b][a];  
 column\_sum += field[a][b];  
 }  
 if(Math.*min*(row\_sum, Math.*min*(column\_sum, Math.*min*(diag\_sum\_1, diag\_sum\_2))) == -3) {  
 return -1;  
 }  
 if(Math.*max*(row\_sum, Math.*max*(column\_sum, Math.*max*(diag\_sum\_1, diag\_sum\_2))) == 3) {  
 return 1;  
 }  
 }  
 return 0;  
 }  
  
 private void setupUi() {  
 stage = new Stage(new ScreenViewport());  
  
 Table rootTable = new Table(skin);  
 rootTable.setFillParent(true);  
  
 Label titleLabel = new Label("Tic Tac Toe", skin, "big-font", Color.*BLACK*);  
 rootTable.add(titleLabel);  
 rootTable.center().top();  
  
 stage.addActor(rootTable);  
  
 dialog = new Dialog("End Game", skin) {  
 protected void result(Object object) {  
 bluetoothService.disconnect();  
 game.setScreen(game.lobbyScreen);  
 };  
 };  
 dialogLabel = new Label("error", skin);  
 dialog.text(dialogLabel);  
 dialog.button("Okay", 1);  
  
 fieldSprite = new Sprite(new Texture(Gdx.*files*.local("field.png")));  
 fieldSprite.setPosition(Gdx.*graphics*.getWidth() / 2 - fieldSprite.getWidth() / 2, (Gdx.*graphics*.getHeight() - titleLabel.getHeight()) / 2 - fieldSprite.getHeight() / 2);  
 Rectangle fieldRect = fieldSprite.getBoundingRectangle();  
 float width = fieldRect.getWidth() / 3;  
 float height = fieldRect.getHeight() / 3;  
 collisionRectangles = new Rectangle[3][3];  
 for(int column = 0; column < 3; ++ column) {  
 for(int row = 0; row < 3; ++row) {  
 collisionRectangles[column][row] = new Rectangle(fieldRect.getX() + column \* width, fieldRect.getY() + row \* height, width, height);  
 }  
 }  
 }  
  
 private void processMessages() {  
 Message message = null;  
 while((message = bluetoothService.getMessage()) != null) {  
 Gdx.*app*.log(Tag, "received message: " + message.code.name());  
 switch(message.code) {  
 case *BLUETOOTH\_CONNECTED*:  
 Gdx.*app*.log(Tag, "we have a connection");  
 gameRunning = true;  
 *// cancel the connection timeout* Timer.*instance*().clear();  
 break;  
 case *BLUETOOTH\_ADAPTER\_DISABLED*:  
 case *BLUETOOTH\_DISCONNECTED*:  
 case *BLUETOOTH\_ERROR*:  
 *// this is bad, we have to stop the game* if(!gameRunning) {  
 *// we have no problem if the game is already over* break;  
 }  
 gameRunning = false;  
 showDialog("Error", "bluetooth error: " + message.code.name());  
 break;  
 case *BLUETOOTH\_SENT*:  
 Gdx.*app*.log(Tag, "bluetooth was sent");  
 break;  
 case *BLUETOOTH\_RECEIVED*:  
 String received = new String((byte[])message.data);  
 int column = Integer.*parseInt*(received.substring(0, 1));  
 int row = Integer.*parseInt*(received.substring(1, 2));  
 String opponentString = received.substring(2,3);  
 int opponentValue = opponentString.equals("X") ? 1 : -1;  
 field[column][row] = opponentValue;  
 checkWinner();  
 hasTurn = true;  
 break;  
 default:  
 Gdx.*app*.log(Tag, "message not handled, discarded");  
 }  
 }  
 }  
  
 private void checkWinner() {  
 int winner = getWinner();  
 if(winner == 0) {  
 boolean draw = true;  
 for(int column = 0; column < 3; ++column) {  
 for(int row = 0; row < 3; ++row) {  
 if(field[column][row] == 0)  
 draw = false;  
 }  
 }  
 if(draw) {  
 showDialog("Game over", "It is a draw");  
 gameRunning = false;  
 }  
 return;  
 }  
 if(winner == playerValue) {  
 showDialog("Game over", "You won :)");  
 gameRunning = false;  
 return;  
 }  
 showDialog("Game over", "You lost like a noob");  
 gameRunning = false;  
 }  
  
 private void showDialog(String title, String message) {  
 dialog.getTitleLabel().setText(title);  
 dialogLabel.setText(message);  
 dialog.show(stage);  
 }  
  
 @Override  
 public void show() {  
 Gdx.*app*.log(Tag, "showing");  
 game.inputMultiplexer.addProcessor(stage);  
 *// reset field and buttons* for(int column = 0; column < 3; ++column) {  
 for(int row = 0; row < 3; ++row) {  
 field[column][row] = 0;  
 }  
 }  
  
 *// if we are not already connected we came here to host a game* if(!bluetoothService.isConnected()) {  
 bluetoothService.listen();  
 *// we are the host, so our id is -1* playerValue = -1;  
 *// the client always begins  
  
 // schedule a timeout because we are only visible for 30 seconds* Timer.*schedule*(new Timer.Task() {  
 @Override  
 public void run() {  
 showDialog("Timeout", "No client connected");  
 }  
 }, 30);  
  
 *// we need to show the user that he is waiting, could be some fancy animation someday :)* gameRunning = false;  
 hasTurn = false;  
 } else {  
 *// we are just a client so our id is 1* playerValue = 1;  
  
 *//no need to wait* gameRunning = true;  
 hasTurn = true;  
 }  
 }  
  
 @Override  
 public void render(float delta) {  
 if(Gdx.*input*.isKeyJustPressed(Input.Keys.*BACK*)) {  
 Gdx.*app*.log(Tag, "back key was pressed, returning to lobby");  
 bluetoothService.disconnect();  
 game.setScreen(game.lobbyScreen);  
 }  
 if(Gdx.*input*.justTouched() && gameRunning && hasTurn) {  
 *// the screen was tapped and we are allowed to take a turn* float x = Gdx.*input*.getX();  
 float y = Gdx.*graphics*.getHeight() - Gdx.*input*.getY();  
 for(int column = 0; column < 3; ++ column) {  
 for(int row = 0; row < 3; ++row) {  
 if(collisionRectangles[column][row].contains(new Vector2(x, y))) {  
 if(field[column][row] == 0) {  
 Rectangle rect = collisionRectangles[column][row];  
 hasTurn = false;  
 *// a play is allowed here* field[column][row] = playerValue;  
 String position = String.*valueOf*(column) + String.*valueOf*(row);  
 String data = playerValue == 1 ? position + "X" : position + "O";  
 bluetoothService.write(data.getBytes());  
 checkWinner();  
 }  
 }  
 }  
 }  
 }  
 processMessages();  
 Gdx.*gl*.glClearColor(0.8f, 0.8f, 0.8f, 1);  
 Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);  
 game.batch.begin();  
 game.batch.draw(game.background, 0, 0);  
 game.batch.end();  
 game.batch.begin();  
 fieldSprite.draw(game.batch);  
 for(int column = 0; column < 3; ++ column) {  
 for (int row = 0; row < 3; ++row) {  
 if(field[column][row] != 0) {  
 Sprite sprite = field[column][row] == 1 ? xSprite : oSprite;  
 Rectangle rect = collisionRectangles[column][row];  
 sprite.setPosition(rect.x + rect.width / 2 - sprite.getWidth() / 2, rect.y + rect.height / 2 - sprite.getHeight() / 2);  
 sprite.draw(game.batch);  
 }  
 }  
 }  
 game.batch.end();  
 stage.act();  
 stage.draw();  
 }  
  
 @Override  
 public void resize(int width, int height) {  
  
 }  
  
 @Override  
 public void pause() {  
  
 }  
  
 @Override  
 public void resume() {  
  
 }  
  
 @Override  
 public void hide() {  
 Gdx.*app*.log(Tag, "hiding");  
 game.inputMultiplexer.clear();  
  
 *// make sure no old dialogs are shown when we come back* dialog.remove();  
 }  
  
 @Override  
 public void dispose() {  
  
 }  
}

## Листинг BluetoothService.java

package com.merltech.tictactoe.network;  
  
public interface BluetoothService {  
 public boolean isAvailable();  
 public boolean isEnabled();  
 public boolean isConnected();  
 public Message getMessage();  
 public void dispose();  
 public void enable();  
 public void disable();  
 public void scan();  
 public void makeDiscoverable(int seconds);  
 public void setName(String name);  
 public void connect(String macAddress);  
 public void disconnect();  
 public void listen();  
 public void write(byte[] buffer);  
  
 public class BluetoothPeer {  
 public final String Name;  
 public final String Address;  
  
 public BluetoothPeer(final String name, final String address) {  
 Name = name;  
 Address = address;  
 }  
 }  
}

## Листинг Message.java

package com.merltech.tictactoe.network;  
  
*// Message class used for BluetoothService and (soon) WiFiService to communicate with the GameThread*public class Message {  
  
 public static enum Code {  
 *BLUETOOTH\_DEVICE\_FOUND*,  
 *BLUETOOTH\_DISCOVERY\_FINISHED*,  
 *BLUETOOTH\_ADAPTER\_ENABLED*,  
 *BLUETOOTH\_ADAPTER\_DISABLED*,  
 *BLUETOOTH\_CONNECTED*,  
 *BLUETOOTH\_DISCONNECTED*,  
 *BLUETOOTH\_RECEIVED*,  
 *BLUETOOTH\_SENT*,  
 *BLUETOOTH\_ERROR* }  
  
 public final Code code;  
 public final int arg2;  
 public final Object data;  
  
 public Message(Code code, int arg2, Object data) {  
 this.code = code;  
 this.arg2 = arg2;  
 this.data = data;  
 }  
}