БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ВЕБ-ТЕХНОЛОГИЙ И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

Курсовая работа

Создание клиент-серверного игрового приложения под операционную систему Android

Научный руководитель:
Дедков Даниил Юрьевич

Исполнители:
студенты 3 курса, 2 группы
Бабеня Вячеслав Игоревич
Кочурко Александр Александрович
Яковчик Антон Евгеньевич

Минск 2015

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В наше время сложно представить себе человека без сотового телефона, планшетного компьютера, смартфона или любого другого портативного мультимедийного устройства. Мы привыкли к тому, что всегда под рукой не только средство связи, но и множество полезных функций, таких как: калькулятор, органайзер, конвертер, календарь, часы. Смартфоны становятся новой мобильной игровой платформой, соревнуясь с классическими карманными игровыми системами вроде Nintendo DS или Playstation Portable. В устройстве смартфона все довольно просто.

Главным образом он состоит из нескольких отдельных блоков - памяти, процессора, который занимается вычислениями, хранилища данных, радиомодуля, который в свою очередь состоит из приемника и передатчика и отвечает за связь. Самое интересное здесь - операционная система, установленная на встроенную память. От операционной системы и ее версии зависят все основные возможности устройства. Смартфоны, как и персональные компьютеры, существуют в абсолютно разных комплектациях и под управлением разных операционных систем, разновидности которых мы рассмотрим далее. По мере роста продаж мобильных устройств во всем мире, растет и спрос на различные приложения для них.

Каждая уважающая себя компания, стремится иметь хотя бы одно мобильное приложение, чтобы быть у своего клиента ”всегда под рукой”. А существование некоторых компаний и вовсе сложно представить без мобильных устройств и специализированных программ, при помощи которых можно, например, управлять базами данных или следить за состоянием своего продукта на рынке в любой момент времени. К сожалению, на сегодняшний день не существуют определенного стандарта средства разработки мобильных приложений. Каждый производитель пытается сделать операционную систему в своем устройстве более уникальной и запоминающейся пользователю, и как следствие возникают вопросы совместимости различных приложений на разных операционных системах.

Основной целью данной работы является разработка клиент-серверного игрового приложения на примере игры жанра Shooter для мобильных устройств на базе операционной системы Android.

Android - портативная (сетевая) операционная система для коммуникаторов, планшетных компьютеров, электронных книг, цифровых проигрывателей, наручных часов и нетбуков основанная на ядре Linux. Изначально разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android Native Development Kit позволяет портировать (но не отлаживать)

библиотеки и компоненты приложений, написанные на C и других языках;

Для достижение поставленной цели необходимо решить следующие задачи:

- Создание инфраструктуры сервера
- Проектирование базы данных
- Разработка и проектирование User Interface
- Создание логики и описание поведения игровых моделей
- Разработка дизайна приложения

В качестве языка разработки для сервера и клиента был выбран язык Java. Мы посчитали его наиболее приемлимым для клиентской части нашего приложения, поскольку наиболее развивающиеся и поддерживаемые средства разработки под ОС Android (Android SDK) основаны на использовании Java. А для максимальной совместимости сервера и клиента для реализации сервера был также выбран этот язык.

ГЛАВА 1
НЕКОТОРЫЕ СВЕДЕНИЯ О ТЕХНОЛОГИЯХ, ЗАДЕЙСТВОВАННЫХ В РАЗРАБОТКЕ 2D ИГР

1.1. Процессы и потоки в Android OS. Управление памятью приложений

Когда запускается компонент приложения и приложение не имеет других запущенных компонентов, Android создает новый процесс для приложения с одним потоком исполнения. По умолчанию все компоненты одного приложения запускаются в одном процессе, в потоке называемом «главный». Если компонент приложения запускается и уже существует процесс для данного приложения(какой-то компонент из приложения существует), тогда компонент запущен в этом процессе и использует его поток выполнения. Вы можете изменить данное поведение, задав разные процессы для разных компонентов вашего приложения. Кроме того вы можете добавить потоки в любой процесс.

Задать отдельный процесс для компонента можно с помощью файла манифеста. Каждый тег компонента(activity, service, receiver и provider) поддерживает атрибут android:process. Данный атрибут позволяет задать процесс, в котором будет выполняться компонент. Также вы можете задать процесс в котором будут выполняться компоненты разных приложений. Также данный атрибут поддерживается тегом application, что позволяет задать определенный процесс для всех компонентов приложения.

Android пытается поддерживать процесс приложения как можно дольше, но когда потребуются ресурсы старые процессы будут вытеснены по иерархии важности.

Существует 5 уровней иерархии важности: (процессы первого уровня из списка будут удалены последними)

- Процесс с которым взаимодействует пользователь(Foreground process) К таким процессам относится например: активити с которым взаимодействует пользовать; сервис(экземпляр Service), с которым взаимодействует пользователь; сервис запущенный методом startForeground(); сервис, который выполняет один из методов своего жизненного цикла; BroadcastReceiver который выполняет метод onReceive().
- Видимый процесс Процесс, в котором не выполнены условия из пункта №1, но который влияет на то, что пользователь видит на экране. К примеру, вызван метод onPause() активити.

- Сервисный процесс Служба запущенная методом startService()
- Фоновый процесс Процесс выполняемый в фоновом режиме, который невиден пользователю.
- Пустой процесс

Отмечу, что в компонентах приложения существует метод onLowMemory(), но полагаться на то, что данный метод будет вызван нельзя, также как нельзя на 100

Когда запускается приложение, система создает «главный» поток выполнения для данного приложения, который также называется UI-потоком. Этот поток очень важен, так как именно в нем происходит отрисовка виджетов(кнопочек, списков), обработка событий вашего приложения. Система не создает отдельный поток для каждого экземпляра компонента. Все компоненты, которые запущенны в одном процессе будут созданы в потоке UI. Библиотека пользовательского интерфейса Android не является потоково-безопасной, поэтому необходимо соблюдать два важных правила:

- Не блокировать поток UI
- Не обращаться к компонентам пользовательского интерфейса не из UI-потока

AsyncTask позволяет выполнить асинхронную работу и делать обновления пользовательского интерфейса. Для обновления реализуйте метод onPostExecute(), а всю фоновую работу заключите в метод doInBackground(). После того, как вы реализуете свою собственную задачу, необходимо ее запустить методом execute().

Параметры передаваемые в AsyncTask:

- Параметры
- Прогресс(единицы задающие ход изменения задачи)
- Результат выполнения задачи

Отмечу пару важных моментов, которые нужно учитывать:

- Метод doInBackGround() выполняется в фоновом потоке, потому доступа к потоку UI внутри данного метода нет.
- Методы onPostExecute() и onProgressUpdate() выполняются в потоке UI, потому мы можем смело обращаться к нашим компонентам UI.

Android не поддерживает swap памяти. Это означает, что любые манипуляции, связанные с памятью, например, создание новых объектов, никак не влияют на выделенную память : она постоянна в RAM. Поэтому, единственно верный способ полного освобождения памяти текущего приложения - это освободить ссылки на объекты, таким образом делая память доступной сборщику мусора.

С целью обеспечения всех своих текущих потребностей в RAM, Android старается поделить её между между процессами. Это может быть достигнуто следующими путями :

Каждый процесс приложения ответвляется от Zygote процесса. Zygote про-

цесс начинает работу при запуске системы и загружает общие ресурсы (например, activity themes). Для того, чтобы новый процесс приложения, система ответвляет Zygote процесс, после чего загружает и запускает код приложения в новом процессе. Это позволяет большей части страничной памяти RAM, выделенной для ресурсов, быть поделённой между всеми процессами.

1.2. TCP и UDP протоколы

TCP и UDP

Протокол UDP (User Datagram Protocol) – протокол транспортного уровня, входящий в стек протоколов TCP/IP, обеспечивающий негарантированную доставку данных без установления виртуального соединения.

Поскольку на протокол не возлагается задач по обеспечению гарантированной доставки, а лишь требуется обеспечивать связь между различными программами, то структура заголовка дейтаграммы UDP (так называется пакет протокола) выглядит достаточно просто – она включает в себя всего четыре поля. Первые два поля содержат номера UDP-портов программы-отправителя и программы-получателя. Два остальных поля в структуре заголовка дейтаграммы предназначены для управления обработкой – это общая длина дейтаграммы и контрольная сумма заголовка.

Протокол TCP (Transmission Control Protocol) является транспортным протоколом стека протоколов TCP/IP, обеспечивающим гарантированную доставку данных с установлением виртуального соединения.

Протокол предоставляет программам, использующим его, возможность передачи непрерывного потока данных. Данные, подлежащие отправке в сеть, разбиваются на порции, каждая из которых снабжается служебной информацией, то есть формируются пакеты данных. В терминологии TCP пакет называется сегментом.

В соответствии с функциональным назначением протокола структура TCP-сегмента предполагает наличие следующих информационных полей:

- номер порта-отправителя и номер порта-получателя – номера портов, идентифицирующие программы, между которыми осуществляется взаимодействие;
- поля, предназначенные для обеспечения гарантированной доставки: размер окна, номер последовательности и номер подтверждения
- управляющие флаги – специальные битовые поля, управляющие протоколом.

Реализация режима гарантированной доставки Для обеспечения гарантированной доставки протокол TCP использует механизм отправки подтверждения. С целью снижения загрузки сети протокол TCP допускает посылку одного подтверждения сразу для нескольких полученных сегментов. Объем данных, которые могут

быть переданы в сеть отправителем до получения подтверждения, определяется специальным параметром протокола TCP - размером окна. Размер окна согласуется при установлении соединения между отправителем и получателем и может автоматически изменяться программными модулями протокола TCP в зависимости от состояния канала связи. Если в процессе передачи данных потери происходят достаточно часто, то размер окна уменьшается, и наоборот – окно может иметь большой размер, если высока надежность канала данных.

Для того, чтобы данные могли быть правильно собраны получателем в нужном порядке, в заголовке TCP-сегмента присутствует информация, определяющая положение вложенных данных в общем потоке. Отправляя подтверждение, получатель указывает положение данных, которые он ожидает получить в следующем сегменте, тем самым косвенно сообщая отправителю, какой фрагмент общего потока был успешно принят. Соответствующие поля заголовка TCP-сегмента получили название номер последовательности и номер подтверждения.

Установление соединения Перед началом передачи потока данных абоненты должны согласовать параметры передачи: размер окна и начальные номера последовательностей, относительно которых будет отсчитываться положение передаваемых в сегментах данных внутри общего потока. Очевидно, что такое согласование предполагает обмен специальными сегментами и выделение ресурсов, в частности, блоков памяти, необходимых для приема и обработки данных и подтверждений. Соответствующая последовательность действий называется установлением виртуального соединения.

1.3. Netty

Netty - это non-blocking I/O (NIO) клиент-серверный фреймворк предназначенный для разработки сетевых приложений на языке Java. Асинхронный событийно-ориентированный фрэймворк используемый для упрощения написания сетевых програм таких как TCP и UDP серверы. В отличии от традиционных Java-реализаций для работы с сетевыми протоколами, использующих синхронную модель передачи данных, Netty позволяет использовать асинхронную передачу, а также службы уровня операционной системы для достижения максимальной скорости передачи данных.

Для своей работы NIO использует:

- буферы — типы для хранения данных;
- каналы — аналоги потоков для быстрой записи или чтения данных.

1.4. Физика???

ГЛАВА 2
СОЗДАНИЕ ИГРЫ

2.1. Специфика игрового приложения

Разработанная нами игровое приложение представляет собой игру жанра shooter. По запуску приложения вы регистрируетесь, после чего получаете во владение танк. Вы управляете танком (вид сверху). Доступные действия: движение танка, стрельба из танка. Цель игры: уничтожить соперника раньше, чем он уничтожит вас. Танк можно модифицировать, модификация оплачивается внутриигровой валютой, начисляемой за достижения в игре (общее время игры, количество выигранных схваток). Также можно купить другой танк. Изюминка игры – множество доступных танков, интересные модификации, динамичность игры. Также доступен режим игры по Bluetooth вне учёта сервером.

2.2. Технологии, выбранные для разработки под Android OS

В ходе проведённой работы, были рассмотрены следующие способы промышленной разработки под упомянутую платформу :

- PhoneGap — это OpenSource платформа, позволяющая разрабатывать мобильные приложения на HTML, JavaScript и CSS под различные платформы (практически без изменения кода приложения) в их число входят: iOS, Android, Blackberry, WebOS, Symbian и Windows Mobile на подходе. Прелесть его в том, что он не требует навыков разработки под конкретную платформу. Вы пишете свое приложение на JavaScript, используете HTML и CSS для разметки. Вы пишете мобильное приложение как обычный сайт или вебсервис. Движок PhoneGap расширяет API браузера и добавляет следующие возможности: доступ к акселометру, доступ к камере (пока только фото), доступ к компасу, доступ к списку контактов, запись и прослушивание аудио файлов, предоставляет доступ к файловой системе, позволяет работать с разными HTML5 хранилищами localStorage, Web SQL и т.п а также позволяет безболезненно обращаться к любому кросс-доменному адресу. Кроме платформы PhoneGap имеет, пока бесплатный, билдер приложений под все устройства в один клик.
- Android SDK включает в себя разнообразные библиотеки, документацию и инструменты, которые помогают разрабатывать мобильные приложения для платформы Android. API Android SDK — APIбиблиотеки Android, предо-

ставляемые для разработки приложений. Документация SDK— включает обширную справочную информацию, детализирующую, что включено в каждый пакет и класс и как это использовать при разработке приложений. AVD (Android Virtual Device)— интерактивный эмулятор мобильного устройства Android. Используя эмулятор, можно запускать и тестировать приложения без использования реального Androidустройства. Development Tools — SDK включает несколько инструментальных средств для разработки, которые позволяют компилировать и отлаживать создаваемые приложения. Sample Code — Android SDK предоставляет типовые приложения, которые демонстрируют некоторые из возможностей Android, и простые программы, которые показывают, как использовать индивидуальные особенности API в вашем коде.

- Android NDK (native development kit) – это набор инструментов, которые позволяют реализовать часть вашего приложения используя такие языки как C/C++. Google рекомендует прибегать к использованию NDK только в редчайших случаях. Зачастую это такие случаи:
  - Нужно увеличить производительность (например, сортировка большого объема данных);
  - Использовать стороннюю библиотеку. Например, много уже чего написано на C/C++ языках и нужно просто заиспользовать существующий материал. Пример таких библиотек, как, Ffmpeg, OpenCV;
  - Программирование на низком уровне (например, всё что выходит за рамки Dalvik);
- Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования. Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработки исходного кода (в общем-то, Qt — это библиотека не для чистого C++, а для его особого наречия, с которого и «переводит» MOC для последующей компиляции любым стандартным C++ компилятором). MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах на C++ описания классов, и создаёт дополнительный исходный файл на C++,

содержащий метаобъектный код.

Ввиду написанного выше, мы остановили свой выбор на Android SDK, для возможных оптимизаций было решено использовать вставки модулей написанных на C++ с использованием выше упомянутого Android NDK.

Существует два наиболее распространённых утилиты для сборки для OS Android :

- Apache Ant — утилита для автоматизации процесса сборки программного продукта. Является платформонезависимым аналогом утилиты make, где все команды записываются в XML-формате. Ant был создан в рамках проекта Jakarta, сегодня — самостоятельный проект первого уровня Apache Software Foundation. Первая версия была разработана инженером Sun Microsystems Джеймсом Дэвидсоном (James Davidson (англ.)русск.), который нуждался в утилите, подобной make, разрабатывая первую референтную реализацию J2EE. В отличие от make, утилита Ant полностью независима от платформы, требуется лишь наличие на применяемой системе установленной рабочей среды Java — JRE. Отказ от использования команд операционной системы и формат XML обеспечивают переносимость сценариев. Управление процессом сборки происходит посредством XML-сценария, также называемого Build-файлом. В первую очередь этот файл содержит определение проекта, состоящего из отдельных целей (Targets). Цели сравнимы с процедурами в языках программирования и содержат вызовы команд-заданий (Tasks). Каждое задание представляет собой неделимую, атомарную команду, выполняющую некоторое элементарное действие. Между целями могут быть определены зависимости — каждая цель выполняется только после того, как выполнены все цели, от которых она зависит (если они уже были выполнены ранее, повторного выполнения не производится).
- Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая DSL на языке Groovy вместо традиционной XML-образной формы представления конфигурации проекта. В отличие от Apache Maven, основанного на концепции жизненного цикла проекта, и Apache Ant, в котором порядок выполнения задач (targets) определяется отношениями зависимости (depends-on), Gradle использует направленный ациклический граф для определения порядка выполнения задач. Gradle был разработан для расширяемых многопроектных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска. Основные плагины предназначены для разработки и развертывания Java, Groovy и Scala приложений, но готовятся плагины и для других языков программирования.

2.3. Компоненты User Interface

UI— разновидность интерфейсов, в котором одна сторона представлена человеком (пользователем), другая — машиной/устройством. Представляет собой совокупность средств и методов, при помощи которых пользователь взаимодействует с различными, чаще всего сложными, машинами, устройствами и аппаратурой.

При создании приложения, нами были спроектированы и реализованы следующие компоненты пользовательского интерфейса боя :

- Игровое поле - представляет собой модель двухмерной плоскости с нулевым углом наклона относительно вертикали
- Индикатор стрельбы - графическая реализация логической составляющей игры, de facto является таймером до готовности нанесения стрельбы игроком
- Кнопка стрельбы - графический элемент, позволяющий вести стрельбу юнитом игрока
- Индикатор повреждения танка - графический элемент, являющийся идентификатором текущего физического состояния танка игрока
- Таймер окончания боя - элемент графики, отображающий время до конца поединка
- Игровой джойстик - графическая компонента, реализующая логику движения юнита (танка) игрока по игровому полю
- Кнопка включения / выключения звука - графическая компнента, позволяющая пользователю включать и выключать звук

Также, при создании приложения, был создан ряд ”окон” - android activity, и соответственно графические компоненты их сопровождающие (кнопки, ”обои”).

2.4. Сервер

2.4.1. СУБД

Для хранения данных о пользователях нами была использована СУБД MariaDB. MariaDb - это свободная(под лицензией GPL) реляционная система управления базами данных являющаяся ответвлением СУБД MySQL.

В нашей игре присутствуют сущности танков, оружия, двигателей и брони для их хранения, а также хранения информации о пользователе и его вооружения используется следующая архитектура базы данных.

Таблицы:

- users - базовой информации о пользователе такие как имя, пароль и т.д.
- tanks - информация о танке, название и чем (по умолчанию)оснащен

- engine - информация о двигателе, название и характеристики
- armor - информация о броне, название и характеристики
- weapon - информация о оружии, название и характеристики
- garage - информация о танках имеющихся у пользователя
- garage_armor - информация о броне имеющейся у пользователя
- garage_engine - информация о двигателях имеющихся у пользователя
- garage_weapon - информация о оружии имеющемся у пользователя

Таблица users:
- id - первичный ключ
- username - уникальное поле
- password
- mail - уникальное поле
- scores
- money
- tank - танк находящийся в гараже (последний использованный танк)

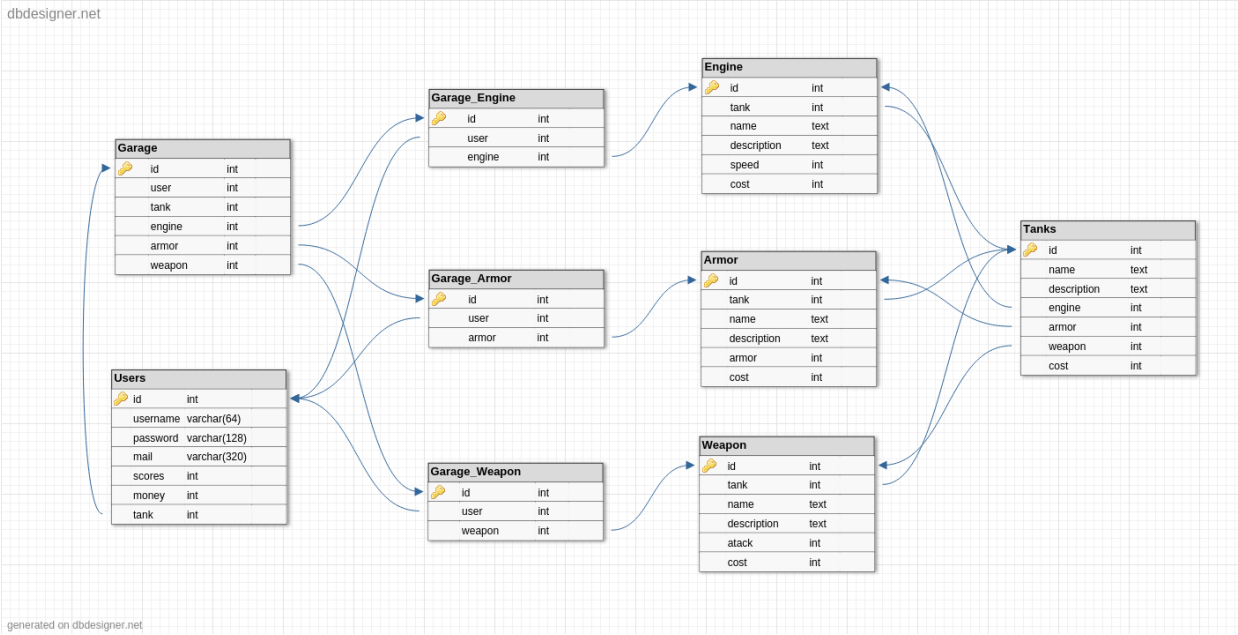
Таблица tanks:
- id - первичный ключ
- name
- description
- weapon - базовое оружие
- armor - базовая броня
- engine - базовый двигатель
- cost

Таблицы armor/engine/weapon:
- id - первичный ключ
- name
- description
- tank - базовый танк (танк совместимый с этим оружием)
- armor/speed/atack - значение характеристик
- cost

Таблица garage:
- id - первичный ключ
- user - пользователь которому принадлежит танк
- tank - тип танка
- weapon - оружие экипированное на танк

- armor - броня экипированная на танк
- engine - двигатель экипированный на танк

Таблица garage_(armor/engine/weapon):
- id - первичный ключ
- user - пользователь которому принадлежит
- armor/engine/weapon - базовая экипировка



2.4.2. Архитектура сервера

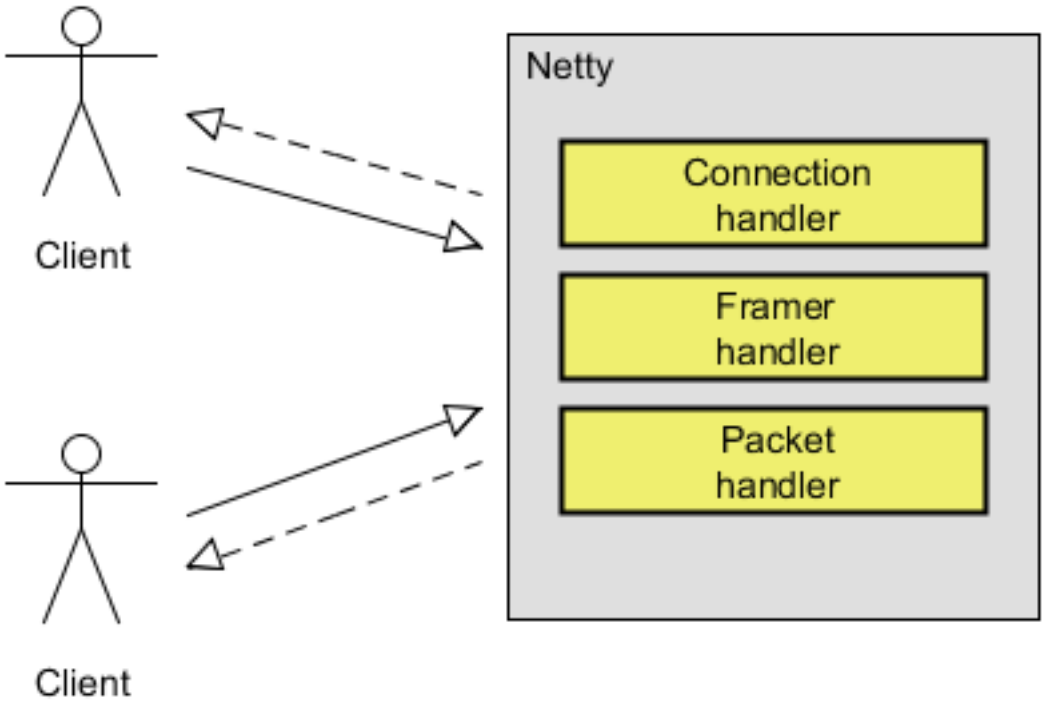
Сервер является одной из важнейших частей нашего приложения. От его производительности и стабильности зависит насколько пользователь будет доволен нашей игрой.

Для решения этих задач был использован сетевой фреймворк Netty. Он позволил нам просто и быстро обрабатывать входящие пакеты в асинхронном режиме не плодя большое число потоков ответственных за чтение пакетов приходящих от пользователя. Netty реализует очень удобную архитектуру. Например она позволяет подключить несколько обработчиков входящих данных. Т.е. в первом обработчике мы разделяем входящий поток данных на пакеты, а во втором уже обрабатываем эти пакеты. При этом можно гибко управлять настройками самой библиотеки, выделять ей необходимое число потоков или памяти и т.д.

Логика нашего приложения разделяется на две части. Первая часть ответственна за “мирную” логику, а вторая часть за “боевую” логику. Для “мирной” логики в отличии от “боевой” логики не сильно важна скорость. Поэтому для “боевой” логики мы решили вместо TCP протокола использовать более легковесный и быст-

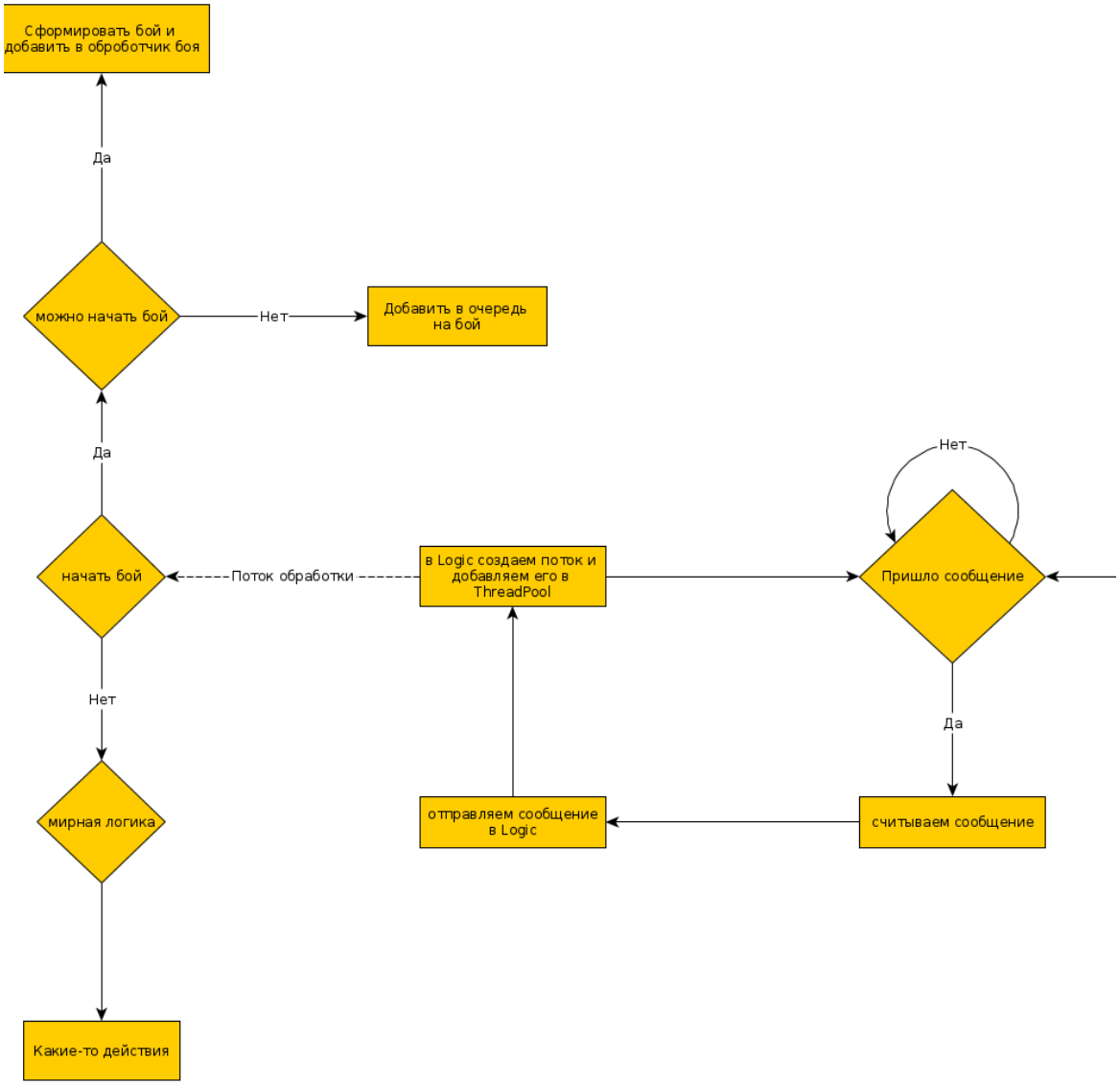
рый UDP.

Общая архитектура нашего серверного приложения (на TCP части)выглядит следующим образом:



У нас получается 3 обработчика:

- Connection handler — это обработчик который отвечает за подключения/отключения клиентов. В нашем случае используется для корректного отключение клиентов с закрытием всех используемых ресурсов
- Frame handler — это обработчик разделяет поток данных на отдельные пакеты
- Packet handler — это уже обработчик игровых сообщений. Здесь мы получаем данные из пакета и дальше их обрабатываем

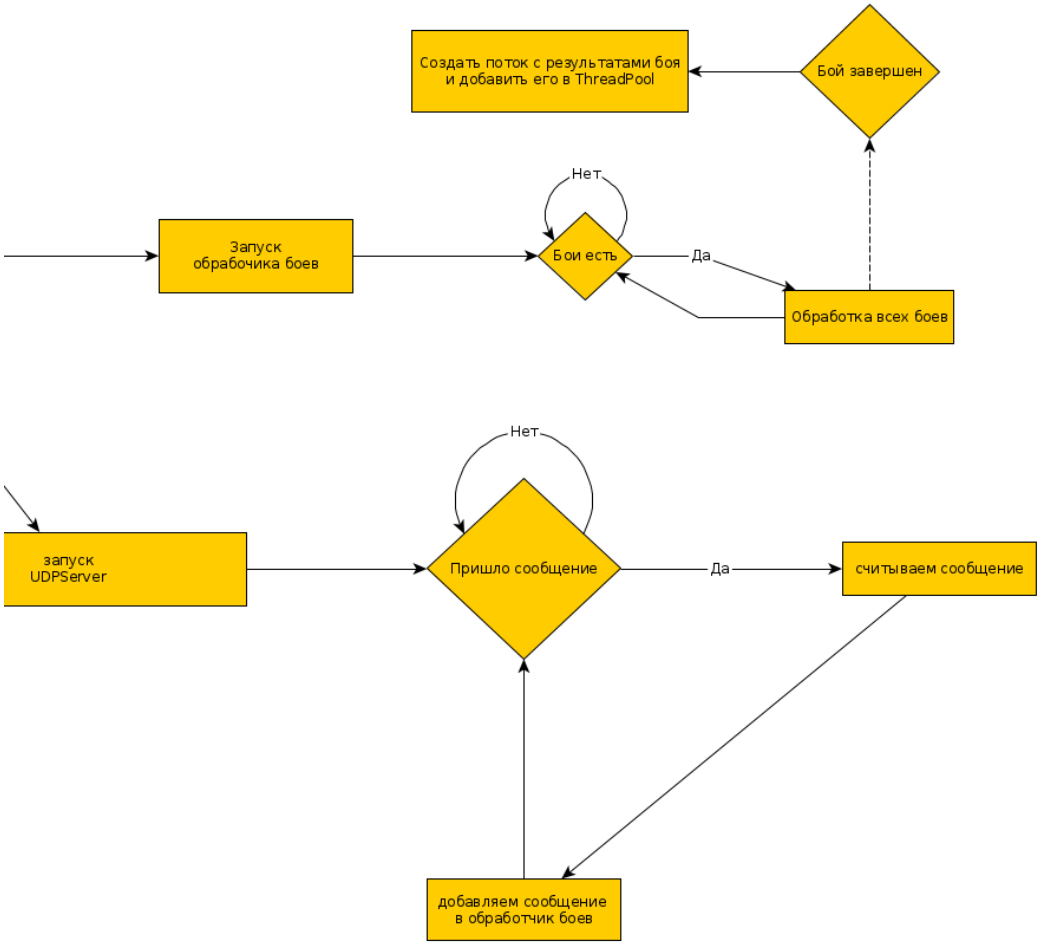


Получая пакет мы создаем отдельный поток обработчик для данного пакета и помещаем его в пул потоков. Это позволяет быстро освободить часть кода ответственную за получение данных.

Сам пакет данных используемый для общения между сервером и клиентом представляет собой контейнер содержащий массив байтов для которого определены функции получения из него или добавления в него данных определенного типа.

При обработке пакета из него последовательно извлекается информация. При не совпадении пришедших данных определенному шаблону клиенту отправляется сообщение об ошибке.

Структура “боевой” части приложения для большей производительности содержит в себе лишь код ответственный за обработку входящего запроса. В общем случае последовательность действий выполняемых сервером по отношению к пакетам приходящим по UDP выглядит следующим образом.



При получении пакета обработчик помещает его в очередь сообщений завязанных на определенного клиента. Дальше обработка этих сообщений происходит в однопоточном режиме для каждого боя в отдельности.

ЗАКЛЮЧЕНИЕ

В настоящее время игровая индустрия полна физическими движками разных типов. Но большинство из них жертвуют скоростью для достижения лучшей управляемости и поддержки кода, либо, наоборот, жертвуют понятностью и доступностью для разработчиков, но обеспечивая лучшую производительность.

Написание собственной физической библиотеки на языке программирования D по-прежнему остается актуальным, язык позволяет писать быстрые, удобные и интуитивно понятные для разработчика приложения.

- На основе изученных математических и физических законов была написана объемная библиотека математики, включающая в себя реализацию таких математических объектов как *кватернион, вектор, квадратная матрица*.
- Был реализован простейший геометрический примитив — сфера.
- Исследованы особенности пересечения шаров и их контактов в трехмерном пространстве.
- С помощью математической библиотеки реализована физическая библиотека, позволяющая моделировать физическое поведение сфер разной массы и диаметра, упругости.
- Реализовано реалистичное поведение коллизии сфер: изменение направления движения, кручения сфер.
- Реализована возможность создания как упругого так и абсолютно неупругого контакта сфер.
- В качестве интегратора для моделирования движения использовался интегратор Эйлера. Метод интегрирования Эйлера позволяет получить высокую скорость расчетов при допустимой погрешности.

Физическая библиотека прошла успешное первичное тестирование. Как и ожидалось шары вступают в контакт, обладая разными скоростями, вращениями и массами, а затем реалистично разлетаются приобретая новые скорости и вращения.