СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сегодня словосочетание «скачать в Интернете» так же популярно и понятно, как «купить в магазине». Поэтому многим, особенно начинающим пользователям, глобальная сеть представляется как некий всемирный «супермаркет файлов», в котором есть все, что душе угодно, и большей частью – бесплатно.

Курсовая работа посвящена разработке приложения «Torrent-клиент», являющейся оболочкой для работы с сетью BitTorrent и torrent-файлами. Приложение должно отличаются удобством интерфейса, «экономией» системных ресурсов, оптимизацией скорости скачивания, возможностями настройки. Выбор подходящих алгоритмов основан на простоте реализации и эффективности работы в программе.

В рамках данной работы необходимо овладение практическими навыками проектирования и разработки законченного, отлаженного и протестированного программного продукта с использованием методик объектно-ориентированного проектирования и языка высокого уровня C++. Закрепить теоретические знания, полученные при изучении курса “Системное программное обеспечение вычислительных машин”. Также необходимо осуществить разработку удобного пользовательского интерфейса.

Помимо практического интереса, тема имеет широкие возможности для последующей модернизации проекта с применением навыков, полученных в ходе изучения курса “Компьютерное проектирование и языки программирования”.

Данная тема является актуальной, так как разработанный программный продукт предоставляет пользователю возможность получить удобный и оптимизированный torrent-клиент для ОС Linux, который может использоваться для удобного обмена большим количеством файлов, что является отличным вариантом доставки даже для мировых IT-гигантов.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ аналогов программного средства

Существует множество торрент-клиентов для всех популярных операционных систем, большинство из них распространяется бесплатно.

µTorrent - самый популярный торрент-клиент под операционные системы Windows и Mac. По информации портала TorrentFreak к концу декабря 2008 года его месячная аудитория составила 28 млн человек. Программа была выпущена сентябре 2005 года. А сейчас µTorrent установлен на 11,6% компьютеров в Европе и на чуть более 5% — В США. Всего на пользователей µTorrent приходится от 40 до 60% от общего числа пользователей торрент-клиентов. Пример µTorrent приведет на рисунке 1.1:

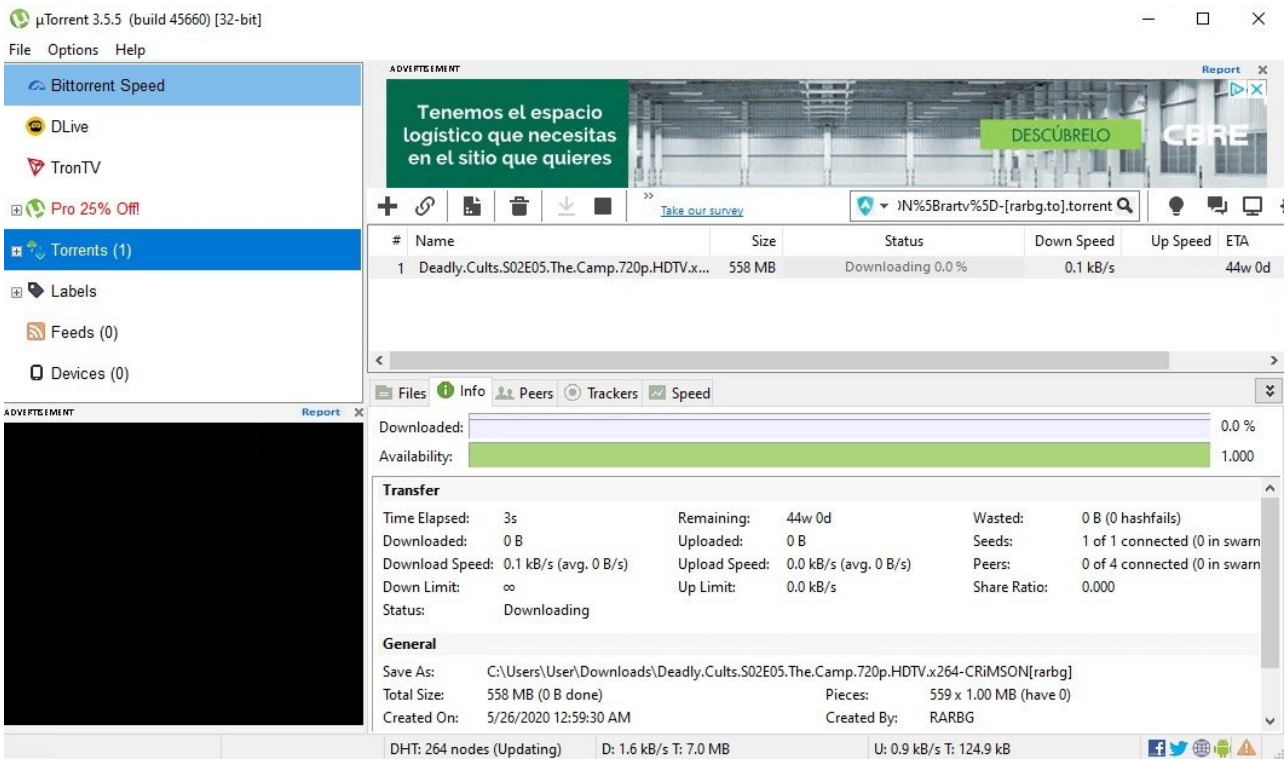

Рисунок 1.1 – Скриншот приложения “µTorrent”

MediaGet - популярный BitTorrent клиент для удобного поиска и загрузки такого контента, как программное обеспечение, аудиозаписи, компьютерные игры, фильмы. Русская версия поддерживает различные торрент-трекеры, а также имеет доступ в закрытые пиринговые сети. Пример MediaGet приведет на рисунке 1.2:

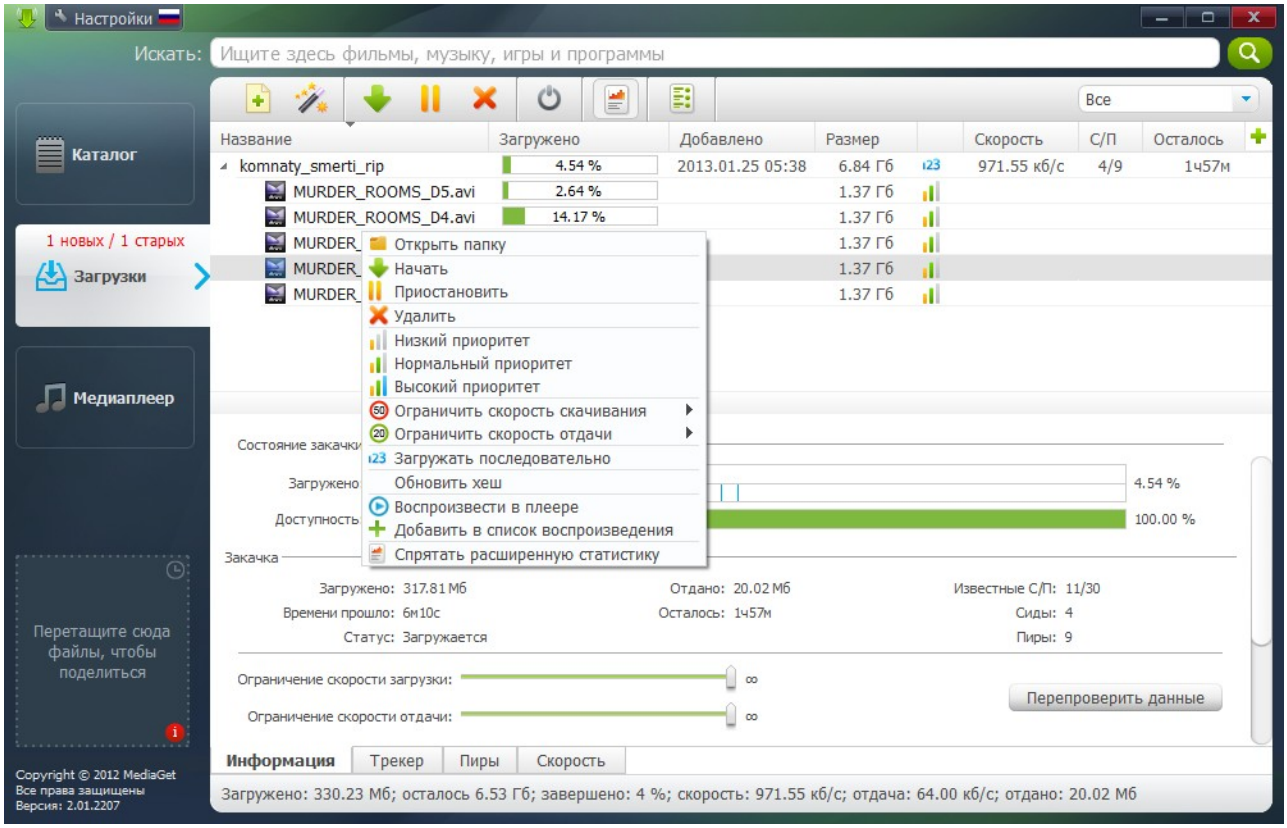

Рисунок 1.2 – Скриншот приложения “MediaGet”

BitTorrent - предок всех современных торрент клиентов, так как это была первая программа, которая использовала протокол, на котором нынче базируются все программы подобного типа. На данный момент этот клиент, хоть не является лидером на рынке, все равно считается самым надежным и проверенным приложением среди всех конкурентов. Пример BitTorrent приведет на рисунке 1.3:

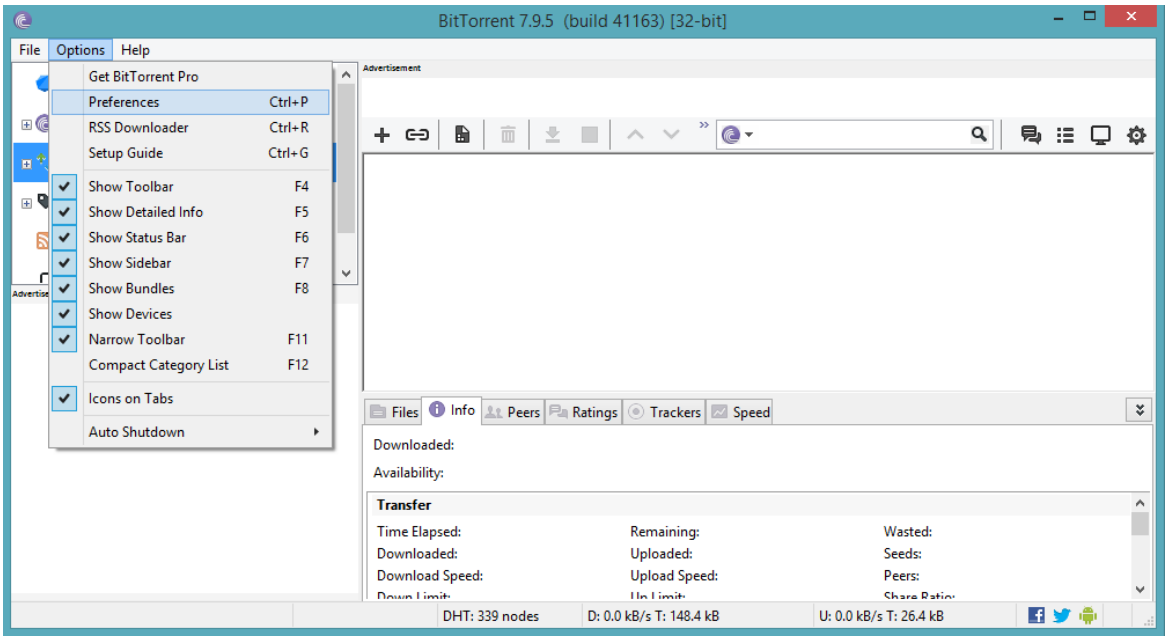

Рисунок 1.3 – Скриншот приложения “BitTorrent”

Новое приложение Torrent-клиент для ОС Linux будет иметь открытый исходный код, поддерживать практически все современные BitTorrent-технологии. Для пользователей будет разработан дружественный, интуитивно понятный интерфейс, без наличия навязчиво рекламы, как у аналогов. Буду разработаны эффективные алгоритмы для экономии системных ресурсов и оптимизации скорости скачивания.

Исходя из всего этого можно сделать вывод, что работать в приложении будет проще и удобнее, что увеличит продуктивность пользователя и самого клиента. А открытый исходный код, будет полезен каждому разработчику, чтобы на базе нового приложения реализовать свой творческий потенциал.

1.2 Постановка задачи

BitTorrent - пиринговый (P2P) сетевой протокол для кооперативного обмена файлами через Интернет. Файлы передаются частями, каждый torrent-клиент, получая (скачивая) эти части, в то же время отдаёт (закачивает) их другим клиентам, что снижает нагрузку и зависимость от каждого клиента-источника и обеспечивает избыточность данных.

Программа должна иметь графический интерфейс, для удобного взаимодействия пользователя с программой.

В программе будут реализованы главные методы:

- Добавления торрент-файла.
- Сортировки данных.
- Просмотр информации о загружаемых файлах.
- Загрузка торрент-файлов.
- Сохранения загруженных файлов.

Для реализации данного программного обеспечения используется объектно-ориентированный язык программирования C++, среда разработки и реализации графического интерфейса используется кроссплатформенная IDE – Qt.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения ее следует разбить на функциональные блоки. Такой подход упростит понимание системы, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость системы в будущем путем добавления новых блоков. Структурная схема системного проектирования приведена в приложении Б.

2.1 Модуль основной работы приложения

Модуль основной работы приложения предназначен для манипуляций с данными и взаимодействием с ними вне зрения пользователя. Модуль нацелен на обработку добавленных торрент-файлов, а также тех, которые находятся на стадии загрузки. Является основной частью приложения, которая работает со всеми остальными модулями программы.

Таким образом модуль основной работы приложения должен следующие функции:

- Поиск данных.
- Обработка торрент-файлов.
- Поддержка остальных модулей.

2.2 Модуль базы данных

Модуль базы данных предназначен для хранения добавленных торрент-файлов, а также выполняет работу по поиску и загрузке ранее добавленных торрент-файлов

Главные методы базы данных будут выполнять следующие важные функции:

- Хранение торрент-файлов.
- Загрузка данных о ранее добавленных торрент-файлах.

2.3 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предназначен для взаимодействия пользователя с компьютером, основываясь на представлении и визуализации данных.

Для разрабатываемого проекта создается удобный и понятный пользовательский интерфейс, для реализации которого используется кроссплатформенная IDE – Qt.

Основные методы реализованные в модуле пользовательского интерфейса:

- Добавление торрент-файлоов.
- Вывод информации о торрент-файлах.
- Взаимодействие с доббавленными торрент-файлами.

2.2 Модуль работы с сетью

Модуль работы с сетью предназначен для соединения с сетью для загрузки торрент-файлов

Главный метод базы данных будет выполнять следующую функцию:

- Загрузка торрент-файлов.
- Взаимодействие с интернет-протоколами.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемой программы.

3.1 Описание классов приложения

3.1.1 BencodeParser

Класс для обработки ошибок связанных с данными и из анализа.
Поля:

- `QString _errorString` – ошибка.
- `QByteArray _bencodeData` – данные.
- `QList<BencodeValue *> _mainList` – главный список содержащий `<BencodeValue *>`.

Методы:

- `void setError(const QString &errorString)` – инициализация переменной `_errorString`.
- `void clearError()` – очистка переменной `_errorString`.
- `QString errorString()` – возвращает переменную `_errorString`.
- `void setData(const QByteArray &data)` – передаем `data` и инициализируем переменную `_bencodeData`.
- `bool readFile(const QString &fileName)` – Сохраняет данные из `fileName` в `_bencodeData`. Возвращает `false` при ошибке и устанавливает `_errorString`.
- `bool parse(const QByteArray &data)` – анализирует `data`, возвращает `false` при ошибке.
- `bool parse()` – анализирует `_bencodeData`. Возвращает `false` при ошибке.
- `QByteArray& rawBencodeData()` – Возвращает не проанализированные `_bencodeData`.
- `QList <BencodeValue *> list()` – возвращает `_mainList`.
- `~BencodeParser ()` – деструктор.
- `BencodeParser ()` – конструктор.

3.1.2 BencodeValue

Класс для обработки и определения разных типов данных.
Поля:

- `Type _type` – тип данных `Integer, String, List, Dictionary`.
- `int _dataPosBegin` – начальное расположение данных в массиве.
- `Int _dataPosEnd` – конечное расположение данных в массиве.
- `QByteArray *_bencodeData` – указатель на данные.

Методы:

– `void loadFromByteArray (const QByteArray &data, int &position)` – инициализация переменной `_errorString`.

– `BencodeValue (Type type)` – конструктор.

– `~BencodeValue ()` – деструктор.

– `Type type ()` – возвращает тип данных.

– `bool isInteger ()` – возвращает `true` если `_type` это `Integer`.

– `bool isString ()` – возвращает `true` если `_type` это `String`.

– `bool isList ()` – возвращает `true` если `_type` это `List`.

– `bool isDictionary()` – возвращает `true` если `_type` это `Dictionary`.

– `BencodeInteger *toBencodeInteger()` – возвращает `BencodeInteger` если `Integer`.

– `BencodeString *toBencodeString()` – возвращает `BencodeString` если `String`.

– `BencodeList *toBencodeList()` – возвращает `BencodeList` если `List`.

– `BencodeDictionary *toBencodeDictionary()` – возвращает `BencodeDictionary` если `Dictionary`.

– `qint64 toInt()` – преобразование к типу `qint64`.

– `QByteArray toByteArray()` – преобразование к типу `QByteArray`.

– `QList<BencodeValue *> toList()` – преобразование к типу `QList`.

– `QByteArray bencode(bool includeMetadata = true)` – кодирует значение.

– `QByteArray getRawBencodeData(bool includeMetadata = true)` – возвращает закодированную версию значения.

– `BencodeValue *createFromByteArray(const QByteArray &data, int &position)` – создает новый `BencodeValue` путем чтения данных из индекса позиции.

– `void print(QTextStream &out)` – вывод.

– `bool equalTo(BencodeValue *other)` – проверяет сходство типов.

3.1.3 Block

Класс для работы с запрошенными данными.

Поля:

– `Piece *_piece` – данные.

– `int _begin` – начало

– `int _size` – размер.

– `bool _isDownloaded` – состояние загрузки.

– `QList<Peer *> _assignees` – список пиров.

Методы:

– `Block(Piece *piece, int begin, int size)` – конструктор.

– `~Block()` – деструктор.
– `Piece *piece()` – возвращает `_piece`.
– `int begin()` – возвращает `_begin`.
– `int size()` – возвращает `_size`.
– `bool isDownloaded()` – возвращает состояние `_isDownloaded`.
– `QList<Peer *> &assignees ()` – возвращает `_assignees`.
– `bool hasAssignees()` – возвращает состояние `_assignees`.
– `void setDownloaded(bool isDownloaded)` – инициализация переменной `_isDownloaded`.
– `void setData(const Peer *peer, const char *data)` – инициализация данных.
– `void addAssignee(Peer *peer)` – добавляет `*peer` в список.
– `void removeAssignee(Peer *peer)` – удаляет `*peer` из списка.
– `void clearAssignees()` – очистка списка.

3.1.4 FileControllerWorker

Класс для контроля работы торрент файлов.
Поля:
– `Torrent *_torrent` – указатель на торрент.
Методы:
– `void checkTorrent()` – проверка торрента.
– `FileControllerWorker(Torrent *torrent)` – конструктор.

3.1.5 FileController

Класс для контроля торрент файлов.
Поля:
– `Torrent *_torrent` – указатель на торрент.
– `QThread *_workerThread` – указатель на поток.
Методы:
– `FileController(Torrent *torrent)` – конструктор.
– `~FileController()` – деструктор.

3.1.6 LocalServiceDiscoveryClient

Класс для работы с сокетами и протоколами.
Поля:
– `QTimer *_announceTimer` – указатель на таймер.
– `QElapsedTimer _elapsedTimer` – истекший таймер.
– `QUdpSocket *_socketIPv4` – указатель на сокет IPv4.
– `QUdpSocket *_socketIPv6` – указатель на сокет IPv6.

– `QByteArray _cookie` – данные cookie.

Методы:

– `LocalServiceDiscoveryClient(QObject *parent = nullptr)` – конструктор.

– `void announceAll()` – объявление как для IPv4, так и для IPv6.

– `void announceIPv4()` – объявление для IPv4.

– `void announceIPv6()` – объявление для IPv6.

– `void processPendingDatagrams()` – получение дейтаграммы.

– `void announce(QUdpSocket *socket, const char *address, int port)` – объявление.

3.1.7 Peer

Класс для работы с пирами.

Поля:

– `Torrent *_torrent` – указатель на торрент.

– `QHostAddress _address` – адрес.

– `int _port` – порт.

– `int _piecesDownloaded` – загруженные данные.

– `bool *_bitfield` – состояние поля бит.

– `QByteArray _protocol` – протокол.

– `QByteArray _reserved` – регистрация данных.

– `QByteArray _infoHash` – хэш данные.

– `QByteArray _peerId` – id пира.

– `ConnectionInitiator _connectionInitiator` – подключение.

– `bool _amChoking` – заглушенный пир.

– `bool _amInterested` – найден нужный пир.

– `bool _peerChoking` – пиры заглушены.

– `bool _peerInterested` – заинтересованные пиры.

– `QTcpSocket *_socket` – указатель на сокет.

– `QByteArray _receivedDataBuffer` – полученные данные.

– `QTimer _replyTimeoutTimer` – таймер для повтора.

– `QTimer _handshakeTimeoutTimer` – таймер для подключения.

– `QTimer _reconnectTimer` – таймер переподключения.

– `QTimer _sendMessagesTimer` – таймер отправки.

– `bool _hasTimedOut` – состояние узла.

– `QList<Block *> _blocksQueue` – запрошенные блоки.

– `bool _isPaused` – состояние паузы.

Методы:

– `Peer(ConnectionInitiator connectionInitiator, QTcpSocket *socket)` – конструктор.

– ~Peer(ConnectionInitiator connectionInitiator, QTcpSocket *socket) – деструктор.
– Torrent *torrent() – возвращает *_torrent.
– QHostAddress address()– возвращает _address.
– int port()– возвращает _port.
– int piecesDownloaded()– возвращает _piecesDownloaded.
– bool *bitfield()– возвращает *_bitfield.
– QByteArray &protocol()– возвращает _protocol.
– QByteArray &reserved возвращает _reserved.
– QByteArray &infoHash()– возвращает _infoHash.
– QByteArray &peerId()– возвращает _peerId.
– State state()– возвращает состояние.
– ConnectionInitiator connectionInitiator()– возвращает _connectionInitiator.
– bool amChoking()– возвращает _amChoking.
– bool amInterested()– возвращает _amInterested.
– bool peerChoking()– возвращает _peerChoking.
– bool peerInterested()– возвращает _peerInterested.
– QTcpSocket *socket()– возвращает *_socket.
– bool hasTimedOut()– возвращает _hasTimedOut.
– QList<Block *> &blocksQueue()– возвращает _blocksQueue.
– bool isPaused()– возвращает _isPaused.
– QString addressPort()– возвращает преобразованный адрес порта.
– bool isDownloaded()– возвращает true данные загрузились.
– bool hasPiece(Piece *piece)– возвращает _address.
– bool isConnected()– возвращает true если подключился.
– bool isInteresting()– возвращает true нашло нужные пиры.
– bool readHandshakeReply(bool *ok)– объявление для IPv4.
– void initServer(Torrent *torrent, QHostAddress address, int port) – инициализация сервера.
– void initBitfield()– инициализация поля бит.
– void initClient()– инициализация клиента.
– Peer *createClient(QTcpSocket *socket)– создание клиента.
– void Peer* createServer(Torrent *torrent, QHostAddress address, int port)– создание сервера.
– void uploadedData(qint64 bytes)– загруженные данные.
– void downloadedData(qint64 bytes)– объявление для IPv4.
– void startConnection()– начало подключения.
– void start()– отправка, если подключились.
– void pause()– пауза в отправке.
– void sendHandshake()– отправка в консоль, что нашли связь.

– `void sendChoke()` – отправляем состояние Choke в консоль.
– `void sendUnchoke()` – отправляем состояние Choke в консоль..
– `void sendInterested()` – отправляем в консоль, что нашли нужный пир.
– `void sendNotInterested()` – отправка в консоль, что нет нужных пиров.
– `void sendHave(int index)` – отправка в консоль, что имеется пир по index.
– `void sendBitfield()` – отправка в консоль, состояния поля бит.
– `void sendRequest(Block *block)` – уведомление о запросе в консоли.
– `void sendPiece(int index, int begin, const QByteArray &blockData)` – объявление для IPv4.
– `void sendCancel(Block *block)` – отмена отправки блока данных.
– `bool requestBlock()` – возвращает true если отправило блок данных.
– `void releaseBlock(Block *block)` – удаление блока данных.
– `void releaseAllBlocks()` – удаление ненужных блоков данных.
– `void fatalError()` – уведомление при потере соединения.
– `void sendMessages()` – отправка учитывая таймеры.
– `void connected()` – проверка подключения.
– `void readyRead()` – работа с чтением данных при готовности.
– `void finished()` – выполняется при завершении.
– `void error(QAbstractSocket::SocketErrorsocketEror)` - ошибка.
– `void replyTimeout()` – перезапуск таймера.
– `void handshakeTimeout()` – таймер для соединения.
– `void reconnect()` – переподключение.

3.1.8 Piece

Класс для работы с частями данных.
Поля:
– `Torrent *_torrent` – указатель на торрент файл.
– `int _pieceNumber` – номер части данных.
– `int _size` – размер.
– `bool _isDownloaded` – состояние загрузки.
– `char *_pieceData` – указатель на части данных.
– `QList<Block *> _blocks` – список блоков данных.
Методы:
– `Piece(Torrent *torrent, int pieceNumber, int size)` – конструктор.
– `~Piece()` – деструктор.
– `bool isDownloaded()` – возвращает `_isDownloaded`.

– `int pieceNumber()` – возвращает `_pieceNumber`.

– `char *data()` – возвращает `*_pieceData`.

– `int size()` – возвращает `_size`.

– `bool getBlockData(int begin, int size, QByteArray &blockData)` – получение блока данных.

– `bool getPieceData(QByteArray &pieceData)` – получение части данных.

– `Block *getBlock(int begin, int size)` – возвращает указатель на существующий блок.

– `Block *requestBlock(int size)` – возвращает блок данных из части, который не был загружен или запрошен.

– `void addBlock(Block *block)` – добавляет блок данных.

– `bool checkIfFullyDownloaded()` – просмотр состояния полной загрузки

– `void updateState()` – обновляет состояние.

– `void deleteBlock(Block *block)` – удаляет ненужный блок данных.

– `void unloadFromMemory()` – загрузка данных из памяти.

– `void setDownloaded(bool isDownloaded)` – установка состояния загрузки.

3.1.9 Remote

Класс для стабилизации работы приложения и предотвращения двойного запуска.

Поля:

– `QLocalServer *_server` – указатель на сервер.

– `QLocalSocket *_socket` – указатель на сокет.

– `QByteArray _buffer` – буфер данных.

Методы:

– `Remote()` – конструктор.

– `~Remote()` – деструктор.

– `bool start()` – запуск сервера.

– `void sendShowWindow()` – показ окна после записи.

– `void showWindow()` – показ окна.

– `void newConnection()` – сделать новое подключение.

– `void disconnected()` – отключится.

– `void readyRead()` – подготовка к чтению.

– `void readMessages()` – объявление о готовности.

3.1.10 ResumeInfo

Класс который работает с информацией, для возобновления торрентов

после загрузки приложения.

Поля:

- `TorrentInfo *_torrentInfo` – указатель на информацию о торренте.
- `QString _downloadLocation` – путь загрузки.
- `qint64 _totalBytesDownloaded` – кол-во скачанных байт.
- `qint64 _totalBytesUploaded` – кол-во загруженных байт.
- `bool _paused` – пауза.
- `QVector<bool> _aquiredPieces` – вектор приобретенных частей данных.

Методы:

- `ResumeInfo(TorrentInfo *torrentInfo)` – конструктор.
- `bool loadFromBencode(BencodeDictionary *dict)` – возвращает `true`, если успешно преобразовало данные.
- `void addToBencode(BencodeDictionary*mainResumeDictionary)` – добавляет данные.
- `TorrentInfo *torrentInfo()` – возвращает `*_torrentInfo`.
- `QString &downloadLocation()` – возвращает `_downloadLocation`.
- `qint64 totalBytesDownloaded()` – возвращает `_totalBytesDownloaded`.
- `qint64 totalBytesUploaded()` – возвращает `_totalBytesUploaded`.
- `QVector<bool> &aquiredPieces()` – возвращает `_aquiredPieces`.
- `QByteArray aquiredPiecesArray()` – проверка и преобразование частей данных.
- `void setDownloadLocation(const QString &downloadLocation)` – инициализация `_downloadLocation`.
- `void setTotalBytesDownloaded(qint64 totalBytesDownloaded)` – инициализация `_totalBytesDownloaded`.
- `void setTotalBytesUploaded(qint64 totalBytesUploaded)` – инициализация `_totalBytesUploaded`.
- `void setPaused(bool paused)` – инициализация `_paused`.
- `void setAquiredPieces(const QVector<bool> &aquiredPieces)` – инициализация `_aquiredPieces`.
- `QVector<bool> toBitArray(const QByteArray &data)` – проверка частей данных.

3.1.11 Torrent

Класс работы самого торрента.

Поля:

- `State _state` – состояние торрента.
- `QList<Peer *> _peers` – список пиров.
- `QList<Piece *> _pieces` – список частей данных.

– TorrentInfo *_torrentInfo – указатель на информацию о торренте.
– TrackerClient *_trackerClient – указатель на трекер.
– FileController *_fileController – указатель на контроль файлов.
– TrafficMonitor *_trafficMonitor – указатель на мониторинг трафика.
– qint64 _bytesDownloadedOnStartup – скачанные байты.
– qint64 _bytesUploadedOnStartup – загруженные байты.
– qint64 _totalBytesDownloaded – всего байт скачано.
– qint64 _totalBytesUploaded – всего байт загружено.
– qint64 _bytesAvailable – доступно байт.
– int _downloadedPieces – скачано частей данных.
– bool _isDownloaded – состояние скачивания.
– bool _isPaused – состояние паузы.
– bool _startAfterChecking – состояние после проверка.
– QString _downloadLocation – директория скачивания.
– QString _errorString – сообщение о ошибках.

Методы:

– Torrent() – конструктор.
– ~Torrent() – деструктор.
– bool createNew(TorrentInfo *torrentInfo, const QString &downloadLocation) – создание нового объекта скачивания.
– bool createFromResumeInfo(TorrentInfo *torrentInfo, ResumeInfo *resumeInfo)– возобновить уже имеющий торрент.
– void loadFileDescriptors() – загрузка данных из дескриптора.
– Block *requestBlock(Peer *client, int size) – блок запросов.
– bool savePiece(Piece *piece) – состояние сохранение частей данных.
– QList<Peer *> &peers() – возвращает _peers.
– QList<Piece *> &pieces() – возвращает _peers.
– QList<QFile *> &files() – возвращает _pieces.
– TorrentInfo *torrentInfo() – возвращает *_torrentInfo.
– TrackerClient *trackerClient() – возвращает *_trackerClient.
– TrafficMonitor *trafficMonitor() – возвращает *_trafficMonitor.
– qint64 bytesDownloaded() – возвращает _bytesDownloadedOnStartup.
– qint64 bytesUploaded() – возвращает _bytesUploadedOnStartup.
– qint64 totalBytesDownloaded() – возвращает _totalBytesDownloaded.
– qint64 totalBytesUploaded() – возвращает _totalBytesUploaded.
– qint64 bytesAvailable() – возвращает _bytesAvailable.

– `qint64 bytesLeft()` – возвращает оставшееся кол-во байт.
– `int downloadedPieces()` – возвращает `_downloadedPieces`.
– `bool isDownloaded()` – возвращает `_isDownloaded`.
– `bool isPaused()` – возвращает `_isPaused`.
– `bool isStarted()` – возвращает `true`, если началось скачивание.
– `int connectedPeersCount()` – кол-во подключенных пиров.
– `int allPeersCount()` – общее кол-во пиров.
– `State state()` – возвращает `_state`.
– `QString stateString()` – возвращает строку состояния.
– `QString &downloadLocation()` – возвращает `_downloadLocation`.
– `float percentDownloaded()` – возвращает процент скачанных данных.
– `QVector<bool> bitfield()` – возвращает текущее битовое поле этого торрента.
– `ResumeInfo getResumeInfo()` – возвращает `resumeInfo`.
– `QString errorString()` – возвращает `_errorString`.
– `void fullyDownloaded()` – вызывается при полном скачивании.
– `void onChecked()` – вызывается при проверке торрента.
– `void onPieceDownloaded(Piece *piece)` – вызывается, когда часть данных успешно загружена.
– `void onBlockUploaded(int bytes)` – вызывается, когда блок данных успешно загружен.
– `void onFullyDownloaded()` – вызывается при полной загрузке торрента.
– `void onSuccessfullyAnnounced(TrackerClient::Event event)` – Вызывается при успешном объявлении.
– `Peer *connectToPeer(QHostAddress address, int port)` – создает пира и подключается к нему.
– `void addPeer(Peer *peer)` – добавление пира, который подключился к нам, в список.
– `void setPieceAvailable(Piece *piece, bool available)` – устанавливает состояние загруженного или доступного фрагмента.
– `void start()` – Начать скачивание / закачку.
– `void pause()` – приостановить торрент.
– `void stop()` – остановить торрент.
– `void check()` – просмотреть торрент.

3.1.12 TorrentInfo

Класс содержащий информацию о торрент файле.
Поля:
– `QList<QByteArray> _announceUrlsList` – список url.

– qint64 _length – длинна торрент файла.
– QByteArray _torrentName – название торрент файла.
– qint64 _pieceLength – длинна частей данных.
– QList<QByteArray> _pieces – список частей данных.
– QDateTime *_creationDate – указатель на дату создания торрент файла.
– QString *_comment – указатель на примечание.
– QString *_createdBy – указатель на создателя.
– QString *_encoding – указатель на кодировку.
– QList<FileInfo> _fileInfos – список информации о торрент файле.
– QByteArray _infoHash – Hash информация.
– QString _creationFileName – имя файла.
– int _numberOfPieces – кол-во частей данных.
– QString _errorString – строка ошибки.

Методы:

– TorrentInfo() – конструктор.
– ~ TorrentInfo() – деструктор.
– QString clearError() – очистка _errorString.
– QString setError() – инициализация _errorString.
– QString errorString() – возвращает _errorString.
– bool loadFromTorrentFile(QString filename) – загрузка данных из торрент файла.
– QList<QByteArray> &announceUrlsList() – возвращает _announceUrlsList.
– QByteArray &torrentName() – возвращает _torrentName.
– qint64 pieceLength() – возвращает _pieceLength.
– QList<QByteArray> &pieces() – возвращает _pieces.
– QDateTime *creationDate() – возвращает *_creationDate.
– QString *comment() – возвращает *_comment.
– QString *createdBy() – возвращает *_createdBy.
– QString *encoding() – возвращает *_encoding.
– QList<FileInfo>& fileInfos() – возвращает _fileInfos.
– bool isSingleFile() – возвращает _errorString.
– QByteArray &infoHash() – возвращает _infoHash.
– QString &creationFileName() – возвращает _creationFileName.
– int numberOfPieces() – возвращает _numberOfPieces.
– int bitfieldSize() – возвращает размер поля данных.

3.1.13 TorrentManager

Класс сохранения торрентов в файловой базе данных.

Поля:

– `QList<Torrent *> _torrents` – указатель на список торрентов.

– `TorrentManager *_torrentManager` – указатель на торрент менеджер.

Методы:

– `TorrentManager()` – конструктор.

– `~TorrentManager()` – деструктор.

– `TorrentManager* instance()` – возвращает `*_torrentManager`.

– `QList<Torrent *> &torrents()` – возвращает `_torrents`.

– `void addTorrentFromInfo(TorrentInfo *torrentInfo, const TorrentSettings &settings)` – добавляет торрент по его данным.

– `void saveTorrentsResumeInfo()` – сохранения торрента, чтобы в будущем продолжить его загрузку.

– `bool saveTorrentFile(const QString &filename, TorrentInfo *torrentInfo)` – сохранение торрент файла.

– `bool removeTorrent(Torrent *torrent, bool deleteData)` – удаление торрент файла.

– `void resumeTorrents()` – продолжить загрузку торрентов.

3.1.14 TorrentMessage

Класс, используемый для генерации сообщений BitTorrent.

Поля:

– `QByteArray _data` – данные.

Методы:

– `TorrentMessage(Type type)` – конструктор.

– `QByteArray &getMessage()` – возвращает `_data`.

– `void addByte(unsigned char value)` – добавляет данные в `_data`.

– `void addInt32(qint32 value)` – добавляет преобразованные данные в `_data`.

– `void addByteArray(QByteArray value)` – добавляет данные в `_data`.

– `void keepAlive(QAbstractSocket *socket)` – вывод логирования, что Alive.

– `void choke(QAbstractSocket *socket)` – вывод логирования, что choke.

– `void unchoke(QAbstractSocket *socket)` – вывод логирования, что unchoke.

– `void interested(QAbstractSocket *socket)` – вывод логирования, что interested.

– `void notInterested(QAbstractSocket *socket)` – вывод логирования, что `notInterested`.

– `void have(QAbstractSocket *socket, int pieceIndex)` – вывод логирования, что have.

– `void bitfield(QAbstractSocket *socket, const QVector<bool> &bitfield)` – вывод логирования о bitfield.

– `void request(QAbstractSocket *socket, int index, int begin, int length)` – вывод логирования о request.

– `void piece(QAbstractSocket *socket, int index, int begin, const QByteArray &block)` – вывод логирования о piece.

– `void cancel(QAbstractSocket *socket, int index, int begin, int length)` – вывод логирования, что cancel.

– `void port(QAbstractSocket *socket, int listenPort)` – вывод логирования о port.

3.1.15 TorrentServer

Класс используется для приема и обработки входящих соединений.
Поля:
– `QTcpServer _server` – сервер.
– `QList<Peer *> _peers` – список пиров.
Методы:
– `TorrentServer()` – конструктор.
– `~TorrentServer()` – деструктор.
– `bool startServer()` – возвращает состояние запуска сервера.
– `QTcpServer& server()` – возвращает `_server`.
– `int port()` – возвращает порт сервера.
– `QHostAddress address()` – возвращает адрес сервера.
– `QList<Peer *> &peers()` – возвращает `_peers`.
– `void newConnection()` – новое подключение.
–

3.1.16 TorrentSettings

Класс для связи с трекером.
Поля:
– `Torrent *_torrent` – указатель на торрент.
– `QNetworkAccessManager _accessManager` – менеджер доступа.
– `QNetworkReply *_reply` – указатель на ответ.
– `int _reannounceInterval` – интервал повторного подключения.
– `QTimer _reannounceTimer` – таймер для повторного подключения.
– `int _currentAnnounceListIndex` – индекс в списке объявлений.
– `bool _hasAnnouncedStarted` – состояние отправки на трекер.
– `int _numberOfAnnounces` – количество успешных объявлений.
– `Event _lastEvent` – Последнее событие.
Методы:
– `TrackerClient(Torrent *torrent)` – конструктор.

– `~TrackerClient()` – деструктор.

– `void announce(Event event)` – Используется для отправки сообщения на трекер.

– `void reannounce()` – интервал для переподключения, учитывая таймер.

– `void httpFinished()` – работа с сетью.

– `QByteArray &currentAnnounceUrl()` – Возвращает текущий URL.

– `void resetCurrentAnnounceUrl()` – Сбрасывает текущий URL .

– `void announceFailed()` – используется, если возникли проблемы с анонсом.

– `void announceSucceeded()` – используется, если с анонсом все хорошо.

– `int numberOfAnnounces()` – Возвращает количество успешных анонсов.

– `bool hasAnnouncedStarted()` – состояние анонса.

3.1.17 TrafficMonitor

Класс для работы со скоростью скачивания и загрузки.
Поля:

– `qint64 _uploadSpeed` – скорость загрузки.

– `qint64 _downloadSpeed` – скорость скачивания.

– `QSet<Peer *> _peers` – список пиров.

– `QTimer _timer` – таймер.

– `qint64 _bytesUploaded` – загружено байт.

– `qint64 _bytesDownloaded` – скачано байт.

Методы:

– `TrafficMonitor(QObject *parent = nullptr)` – конструктор.

– `qint64 uploadSpeed()` – возвращает `_uploadSpeed`.

– `qint64 downloadSpeed()` – возвращает `_downloadSpeed`.

– `void onDataSent(qint64 bytes)` – инициализирует `_bytesUploaded`.

– `void onDataReceived(qint64 bytes)` – инициализирует `_bytesDownloaded`.

– `void addPeer(Peer *peer)` – добавить пир.

– `void removePeer(Peer *peer)` – удалить пир.

– `void update()` – обновить информацию о трафике.

3.1.18 Global

Класс для конвертации байт.
Методы:

– `QString formatSize(qint64 size)` – возвращает отформатированный размер данных.

– `QByteArray percentEncode(const QByteArray &data)` – возвращает закодированную версию данных.

3.1.19 JTorrent

Класс для работы с пользовательским интерфейсом и ядром программы.

Поля:

– `QByteArray _peerId` – id пира.

– `TorrentManager *_torrentManager` – указатель на торрент менеджер.

– `TorrentServer *_server` – указатель на сервер.

– `LocalServiceDiscoveryClient *_LSDClient` – указатель на локальный клиент.

– `MainWindow *_mainWindow` – указатель на пользовательский интерфейс.

– `JTorrent *_instance` – скачано байт.

Методы:

– `JTorrent()` – конструктор.

– `~JTorrent()` – возвращает `_uploadSpeed`.

– `bool startServer()` – возвращает состояние сервера.

– `void startLSDClient()` – запускает локальный клиент.

– `void shutDown()` – завершает работу и сохраняет данные торрента.

– `void showMainWindow()` – показывает главное окно.

– `void critical(const QString &text)` – вывод критической ошибки.

– `void information(const QString &text)` – вывод информационного окна.

– `bool question(const QString &text)` – вывод вопроса.

– `void warning(const QString &text)` – вывод предупреждения.

– `QList<Torrent *> &torrents()` – возвращает список торрентов.

– `TorrentManager *torrentManager()` – возвращает `*_torrentManager`.

– `TorrentServer *server()` – возвращает `*_server`.

– `MainWindow *mainWindow()` – возвращает `*_mainWindow`.

– `JTorrent *instance()` – возвращает `*_instance`.

– `void LSDPeerFound(QHostAddress address, int port, Torrent *torrent)` – подключается к пиру.

3.1.20 MainWindow : public QMainWindow

Класс основного пользовательского интерфейса.

Поля:

– `Panel *_panel` – указатель на панель.
– `QStackedWidget *_stackedWidget`– указатель на виджет.
– `SettingsWindow *_settingsWindow` – указатель на меню настроек.
– `TorrentsList *_torrentsList` – указатель на список торрентов.
– `QTimer _refreshTimer` – таймер.
– `MainWindow *_mainWindow` – указатель на пользовательский интерфейс.

Методы:

– `MainWindow()` – конструктор.
– `~MainWindow()` – деструктор.
– `void createMenus()` – создает меню.
– `QString getDownloadLocation()` – выбор директории установки.
– `void failedToAddTorrent(QString errorString)` – ошибка добавления торрента.
– `failedToResumeTorrents(QString errorString)` – ошибка продолжения скачивания торрента.
– `void addTorrentAction()` – добавления торрента.
– `void exitAction()` – выход из меню действий.
– `void addTorrentFromUrl(QUrl url)` – добавления торрента по URL.
– `void torrentFullyDownloaded(Torrent *torrent)` – уведомление, что торрент загружен.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Метод disconnect() класса Peer.

Шаг 1. Начало.
Шаг 2. Сравниваем, что вернула функция isConnected().
Шаг 3. Если true, то с помощью метода close() закрываем _socket, иначе срабатывает метод finished().
Шаг 4. Конец.

4.2 Метод pause() класса Torrent.

Шаг 1. Начало.
Шаг 2. Проходим по каждому `peer`.
Шаг 3. Ставим каждый пир на паузу с помощью `peer->pause()`.
Шаг 4. Меняем состояние `_isPaused` на `true`.
Шаг 5. Конец.

4.3 Метод start() класса Torrent.

Шаг 1. Начало.
Шаг 2. Рассматриваем `_state`.
Шаг 3. Если `_state` равен `Checking`, то присваиваем true к `_startAfterChecking` и делаем `return`, иначе шаг 4.
Шаг 4. Рассматриваем `_trackerClient->hasAnnoncedStarted()`.
Шаг 5. Если `false`, то делаем анонс `_trackerClient->announce(TrackerClient::Started)`, иначе если не было анонсов ни одного `_trackerClient->announce(TrackerClient::None)`.
Шаг 6. Затем включаем все пиры.
Шаг 7. Присваиваем к `isPaused false`.
Шаг 8. Изменяем состояние `_state` на `Started`.
Шаг 9. Конец.

4.4 Метод getDownloadLocation () класса MainWindow.

Шаг 1. Начало.
Шаг 2. Создаем переменную `downloadPath` типа `QString`.
Шаг 3. После открытия диалогового окна, пользователь выбирает нужный путь загрузки.
Шаг 4. После выбора пользователя мы возвращаем `downloadPath`.
Шаг 5. Конец.

4.5 Метод readFile (const QString &fileName) класса BencodeParser.

Шаг 1. Начало.
Шаг 2. С помощью `clearError()` очищаем значение `_errorString`.
Шаг 3. Создаем файл, по имени `fileName`.
Шаг 4. Пытаемся открыть файл в режиме чтения.
Шаг 5. Если файл нельзя открыть, то выводим сообщение об ошибке, а после возвращаем `false`.
Шаг 6. Если файл можно открыть в режиме чтения, то присваиваем к `_bencodeData` значение, которое вернет метод `file.readAll()`.
Шаг 7. Затем закрываем файл.
Шаг 8. Конец.

4.5 Метод addTorrentFromInfo (TorrentInfo *torrentInfo, *const* TorrentSettings &settings) класса TorrentManager.

Шаг 1. Начало.
Шаг 2. В цикле рассматриваем все имеющиеся торренты по хэш-данным.
Шаг 3. Если нашли совпадение, то информируем об этом и удаляем переданное `torrentInfo`, затем `return`, иначе шаг 4.
Шаг 4. С помощью метода `CreateNew()` пытаемся создать новый торрент, если не получилось, то информируем об этом, затем удаляем данные и делаем `return`, иначе шаг 5.
Шаг 5. Если получилось создать новый торрент, то сохраняем его.
Шаг 6. Если сохранить не получилось, информируем об этом, затем удаляем данные о торренте и выполняем return, иначе шаг 7
Шаг 7. Добавляем новый торрент в список всех торрентов.
Шаг 8. Рассматриваем настройки торрента, если был пропущен просмотр хэша, то просматриваем заново с помощью метода `check()`, иначе шаг 9.
Шаг 9. Если метод `settings.startImmediately()` вернул `true`, запускаем загрузку торрента с помощью метода `torrent->start()`, иначе ставим торрент на паузу с помощью метода `torrent->pause()`.
Шаг 10. Сохраняем информацию о торренте.
Шаг 11. Конец.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Для тестирования и отработки возможных ошибок разработанного приложения был использован Qt Creator.

5.1 Тестирование работы приложения при добавлении торрент файла, который был ранее добавлен

При добавлении пользователем торрент файла, который был ранее добавлен будет выведена ошибка (рисунок 5.1).

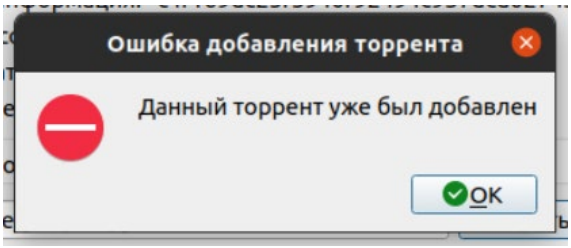

Рисунок 5.1 – Сообщение, что данный торрент был ранее добавлен

5.2 Тестирование работы приложения при вводе неправильного начального порта

Если пользователь решил изменить в настройках начальный порт на тот, который не смог пройти проверки, будет выведена ошибка (рисунок 5.2).

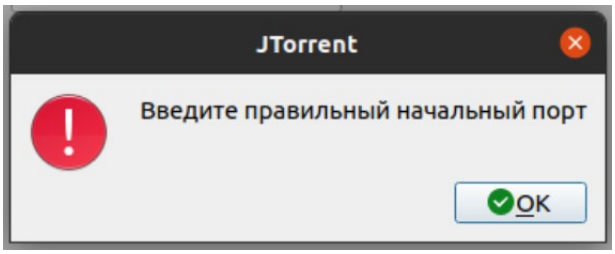

Рисунок 5.2 – Ошибка при вводе неверного начального порта

5.3 Тестирование работы приложения при вводе неправильного конечного порта

Если пользователь решил изменить в настройках конечный порт на тот, который не смог пройти проверки, будет выведена ошибка (рисунок 5.3).

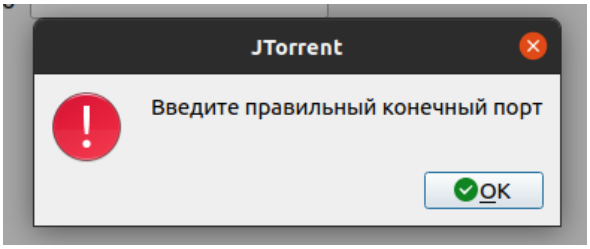

Рисунок 5.3 – Ошибка при вводе неверного конечного порта

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо открыть «JTorrent». После этого откроется основное окно программы (рисунок 6.1).

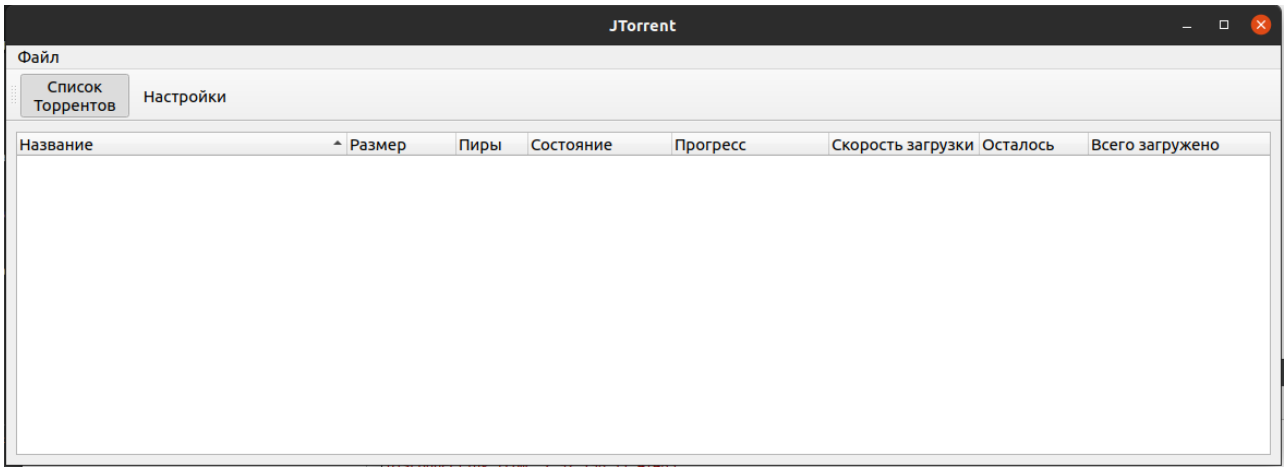

Рисунок 6.1 – Основное окно приложения

После того, как пользователь выберет торрент файл для загрузки, будет выведено диалоговое окно, где покажут информацию о торрент файле и предложат пользователю выбрать путь сохранения скачанного файла (рисунок 6.2).

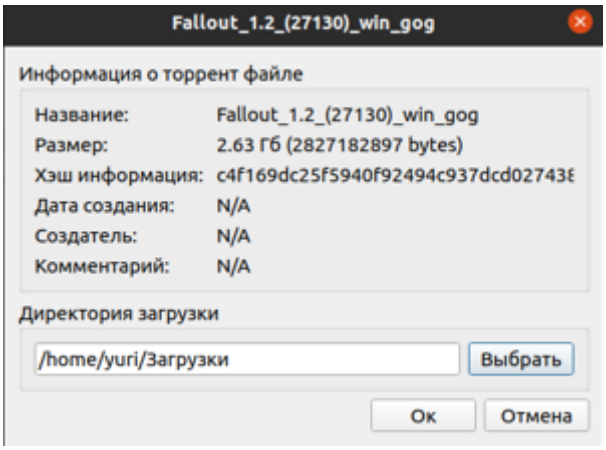

Рисунок 6.2 – Диалоговое окно после добавления торрент файла

При нажатии на кнопку «Файл» появляется виджет (рисунок 6.3). В нем пользователь может выбрать необходимое действие.

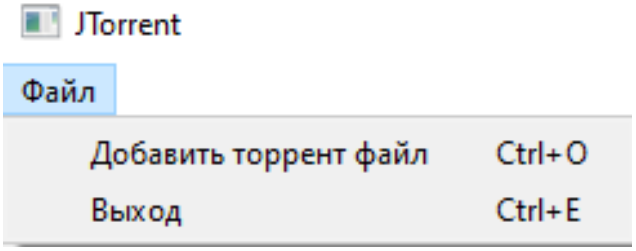

Рисунок 6.3 – Виджет действий при нажатии на “Файл”

Когда пользователь подтвердит добавление торрента, информация о процессе загрузки будет выведена в таблицу (рисунок 6.4).

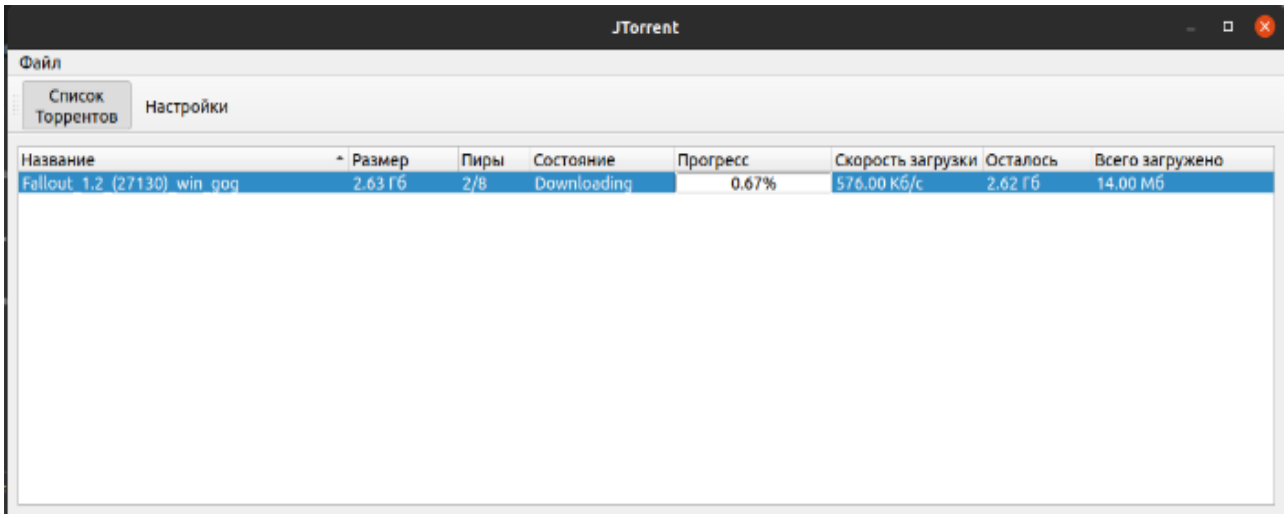

Рисунок 6.4 – Демонстрация работы таблицы со списком торрентов

При нажатии правой кнопкой мыши на торрент файл в списке торрентов, можно рассмотреть функционал манипуляций над ним. (Рисунок 6.5)

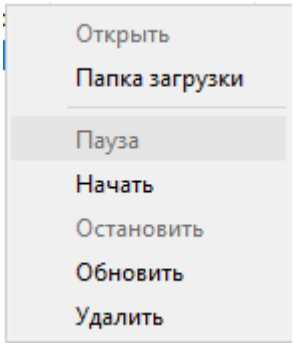

Рисунок 6.5 – Возможные действия над торрент файлами в списке торрентов

При нажатии на кнопку “Настройки” переходит в окно настроек, где может изменить необходимые параметры для лучшей работы приложения (рисунок 6.6)

Рисунок 6.6 – Кнопка настроек

Для сохранения настроек, пользователь может подтвердить свои действия или отменить (рисунок 6.7)

Рисунок 6.7 – Сохранение новых настроек или отмена действий

При удалении торрента будет выведено диалоговое окно для подтверждения действия пользователя (рисунок 6.8)

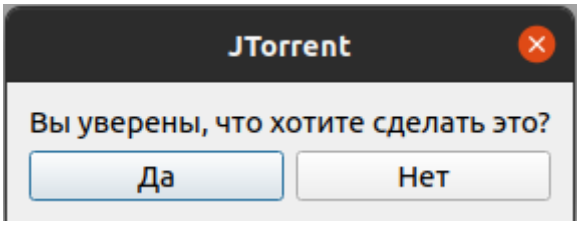

Рисунок 6.8 – Кнопки закрытия виджета

Для сортировки загружаемых торрент файлов, пользователь может нажать на оглавления колонок в таблице с торрентами и отсортировать их по нужной категории (рисунок 6.9)

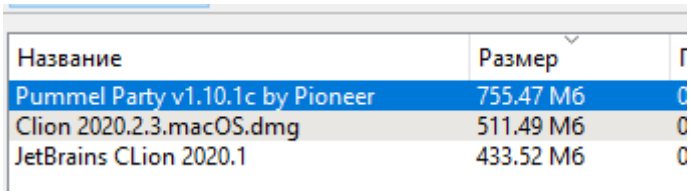

Рисунок 6.9 – Сортировка загружаемых файлов по нужной категории

При выходе из приложения пользователю нужно нажать на кнопку “Выход” в меню действий, затем подтвердить выход в диалоговом окне (рисунок 6.10).

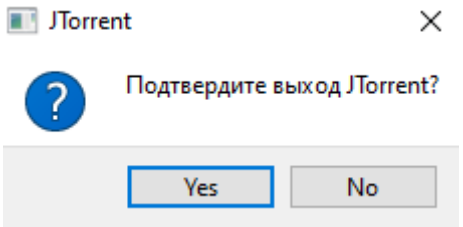

Рисунок 6.10 – Выход и диалоговое окно

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом была разработана работоспособная программа Torrent-клиент с необходимым набором функций. Данный курсовой проект был разработан в соответствии с поставленными задачами, весь функционал был реализован в полном объеме, а также достигнута кроссплатформенность.

В ходе разработки были изучены возможности языка программирования C++, а также получены навыки для проектирования и реализации графического пользовательского интерфейса в Qt.

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному средству, системное и функциональное проектирование, конструирование программного средства, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем планируется усовершенствование программы, а именно, усовершенствование графического интерфейса и расширения функционала, добавление разных виджетов с расширенным набором действий. Одна из главных целей на будущее – увеличение возможных манипуляций над загружаемыми торрент файлами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Герберт, Ш. Самоучитель C++/Ш. Герберт. – М.: Санкт-Петербург, 2003г. – 678 с.

[2] Страуструп, Б. Программирование. Принципы и практика использования C++/ Б. Страуструп. – М.: Вильямс, 2018 г. – 991 с.

[3] Стивен П. Язык программирования C++. Лекции и упражнения, 6-е изд. : Пер. с англ. – М.: Вильямс, 2012. – 1248 с.

ПРИЛОЖЕНИЕ А
(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ В
(обязательное)

Листинг кода

main.cpp

```
#include "jtorrent.h"
#include "core/remote.h"
#include <QApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
          QApplication app(argc, argv);
    app.setOrganizationName("JTorrent");
    app.setApplicationName("JTorrent");

          Remote remote;
          if (!remote.start()) {
        qDebug() << "Запущено";
                  return 0;
          }

    JTorrent JTorrent;

          app.exec();
    JTorrent.shutDown();
          return 0;
}
```

bencodeparser.h

```
#ifndef BENCODE_H
#define BENCODE_H

#include "bencodevalue.h"
#include <QByteArray>
#include <QString>
#include <QList>
#include <QTextStream>

class BencodeParser
{

    QString _errorString;
          void setError(const QString &errorString);
          void clearError();

    QByteArray _bencodeData;

    QList<BencodeValue *> _mainList;

public:
          BencodeParser();
          ~BencodeParser();

          QString errorString() const;

          void setData(const QByteArray &data);

          bool readFile(const QString &fileName);

          bool parse(const QByteArray &data);

          bool parse();

          const QByteArray& rawBencodeData() const;

          QList<BencodeValue *> list() const;
};

#endif
```

bencodeparser.cpp

```
#include "bencodeparser.h"
#include "bencodevalue.h"
#include <QFile>

void BencodeParser::setError(const QString &errorString)
{
    _errorString = errorString;
}

void BencodeParser::clearError()
{
    _errorString.clear();
}


BencodeParser::BencodeParser()
{

}

BencodeParser::~BencodeParser()
{
    for (BencodeValue *value : _mainList) {
                delete value;
          }
}


QString BencodeParser::errorString() const
{
    return _errorString;
}


void BencodeParser::setData(const QByteArray &data)
{
    _bencodeData = data;
}

bool BencodeParser::readFile(const QString &fileName)
{
          clearError();

          QFile file(fileName);
          if (!file.open(QIODevice::ReadOnly)) {
                    setError(file.errorString());
                    return false;
          }
    _bencodeData = file.readAll();
          file.close();
          return true;
}

bool BencodeParser::parse(const QByteArray &data)
{
          setData(data);
          return parse();
}

bool BencodeParser::parse()
{
          clearError();

    for (BencodeValue *value : _mainList) {
                delete value;
          }
    _mainList.clear();

          int i = 0;
    while (i < _bencodeData.size()) {
                    BencodeValue *value;
                    try {
            value = BencodeValue::createFromByteArray(_bencodeData, i);
                    } catch (BencodeException &ex) {
                              setError(ex.what());
                              return false;
                    }
```

```
        _mainList.push_back(value);
          }

          return true;
}


const QByteArray &BencodeParser::rawBencodeData() const
{
    return _bencodeData;
}


QList<BencodeValue *> BencodeParser::list() const
{
    return _mainList;
}


bencodevalue.h

#ifndef BENCODEVALUE_H
#define BENCODEVALUE_H

#include <QByteArray>
#include <QTextStream>
#include <QString>
#include <QList>
#include <QMap>

class BencodeParser;
class BencodeInteger;
class BencodeString;
class BencodeList;
class BencodeDictionary;

class BencodeException
{
    QString _errorString;

public:
    BencodeException(const QString &errorString) : _errorString(errorString) {}
          BencodeException() {}

          template<typename T>
          BencodeException &operator<<(const T &toAppend)
          {
         QTextStream stream(&_errorString);
                    stream << toAppend;
                    return *this;
          }

          const QString &what() const
          {
         return _errorString;
          }
};

class BencodeValue
{
public:
          enum class Type
          {
                    Integer, String, List, Dictionary
          };

protected:
    Type _type;

    int _dataPosBegin;
    int _dataPosEnd;
    const QByteArray *_bencodeData;

          virtual void loadFromByteArray(const QByteArray &data, int &position) = 0;

public:
          BencodeValue(Type type);
          virtual ~BencodeValue();
```

```
        Type type() const;

        bool isInteger() const;
        bool isString() const;
        bool isList() const;
        bool isDictionary() const;

        BencodeInteger *toBencodeInteger();
        BencodeString *toBencodeString();
        BencodeList *toBencodeList();
        BencodeDictionary *toBencodeDictionary();

        virtual qint64 toInt();
        virtual QByteArray toByteArray();
        virtual QList<BencodeValue *> toList();

        virtual QByteArray bencode(bool includeMetadata = true) const = 0;

        QByteArray getRawBencodeData(bool includeMetadata = true);

        static BencodeValue *createFromByteArray(const QByteArray &data, int &position);

        virtual void print(QTextStream &out) const = 0;
        virtual bool equalTo(BencodeValue *other) const = 0;
};

class BencodeInteger : public BencodeValue
{
protected:
    qint64 _value;

        void loadFromByteArray(const QByteArray &data, int &position);

public:
        BencodeInteger();
        BencodeInteger(qint64 value);
        ~BencodeInteger();

        qint64 toInt();
        void setValue(qint64 value);
        QByteArray bencode(bool includeMetadata = true) const;
        void print(QTextStream &out) const;
        bool equalTo(BencodeValue *other) const;
};

class BencodeString : public BencodeValue
{
protected:
    QByteArray _value;

        void loadFromByteArray(const QByteArray &data, int &position);

public:
        BencodeString();
        BencodeString(const QByteArray &value);
        ~BencodeString();

        QByteArray toByteArray();
        void setValue(const QByteArray &value);
        QByteArray bencode(bool includeMetadata = true) const;
        void print(QTextStream &out) const;
        bool equalTo(BencodeValue *other) const;
};

class BencodeList : public BencodeValue
{
protected:
    QList<BencodeValue *> _values;

        void loadFromByteArray(const QByteArray &data, int &position);

public:
        BencodeList();
        ~BencodeList();

        QList<BencodeValue *> toList();
        void setValues(const QList<BencodeValue *> &values);
        void add(BencodeValue *value);
        QByteArray bencode(bool includeMetadata = true) const;
```

```
        void print(QTextStream &out) const;
        bool equalTo(BencodeValue *other) const;
};

class BencodeDictionary : public BencodeValue
{
protected:
    QMap<QByteArray, BencodeValue *> _values;

        void loadFromByteArray(const QByteArray &data, int &position);

public:
        BencodeDictionary();
        ~BencodeDictionary();

        void print(QTextStream &out) const;
        bool equalTo(BencodeValue *other) const;

        QList<QByteArray> keys() const;
        QList<BencodeValue *> values() const;
        bool keyExists(const QByteArray &key) const;
        BencodeValue *value(const QByteArray &key) const;
        void add(const QByteArray &key, BencodeValue *value);
        QByteArray bencode(bool includeMetadata = true) const;
};

#endif
```

bencodevalue.cpp

```
#include "bencodevalue.h"
#include <QDataStream>
#include <QDebug>

BencodeValue::BencodeValue(Type type)
    : _type(type)
    , _dataPosBegin(0)
    , _dataPosEnd(0)
    , _bencodeData(nullptr)
{
}

BencodeValue::~BencodeValue()
{
}


BencodeValue::Type BencodeValue::type() const
{
    return _type;
}


bool BencodeValue::isInteger() const
{
    return _type == Type::Integer;
}

bool BencodeValue::isString() const
{
    return _type == Type::String;
}

bool BencodeValue::isList() const
{
    return _type == Type::List;
}

bool BencodeValue::isDictionary() const
{
    return _type == Type::Dictionary;
}


BencodeInteger *BencodeValue::toBencodeInteger()
{
        if (!isInteger()) {
                QString errorString;
```

```
                    QTextStream err(&errorString);
                    err << "BencodeValue::toBencodeInteger(): Value is not an integer: ";
                    print(err);
                    throw BencodeException(errorString);
          }
          return static_cast<BencodeInteger *>(this);
}

BencodeString *BencodeValue::toBencodeString()
{
          if (!isString()) {
                    QString errorString;
                    QTextStream err(&errorString);
                    err << "bencodeValue::toBencodeString(): Value is not an string: ";
                    print(err);
                    throw BencodeException(errorString);
          }
          return static_cast<BencodeString *>(this);
}

BencodeList* BencodeValue::toBencodeList()
{
          if (!isList()) {
                    QString errorString;
                    QTextStream err(&errorString);
                    err << "BencodeValue::toBencodeList(): Value is not an list: ";
                    print(err);
                    throw BencodeException(errorString);
          }
          return static_cast<BencodeList *>(this);
}

BencodeDictionary *BencodeValue::toBencodeDictionary()
{
          if (!isDictionary()) {
                    QString errorString;
                    QTextStream err(&errorString);
                    err << "BencodeValue::toBencodeDictionary(): Value is not an dictionary";
                    print(err);
                    throw BencodeException(errorString);
          }
          return static_cast<BencodeDictionary *>(this);
}


qint64 BencodeValue::toInt()
{
          return toBencodeInteger()->toInt();
}

QByteArray BencodeValue::toByteArray()
{
          return toBencodeString()->toByteArray();
}

QList<BencodeValue *> BencodeValue::toList()
{
          return toBencodeList()->toList();
}


QByteArray BencodeValue::getRawBencodeData(bool includeMetadata)
{
          QByteArray returnData;
    int begin = _dataPosBegin;
    int end = _dataPosEnd;

          if (!includeMetadata) {
                    begin++;
                    end--;
          }

          for (int i = begin; i < end; i++) {
        returnData.push_back(_bencodeData->at(i));
          }

          return returnData;
}
```

```
BencodeValue *BencodeValue::createFromByteArray(const QByteArray &data, int &position)
{
          BencodeException ex("BencodeValue::createFromByteArray(): ");

          if (position >= data.size()) {
                    throw ex << "Unexpectedly reached end of the data stream";
          }

          BencodeValue *value;
          char firstByte = data[position];
          if (firstByte == 'i') {
                    value = new BencodeInteger;
          } else if (firstByte >= '0' && firstByte <= '9') {
                    value = new BencodeString;
          } else if (firstByte == 'l') {
                    value = new BencodeList;
          } else if (firstByte == 'd') {
                    value = new BencodeDictionary;
          } else {
                    throw ex << "Invalid begining character for bencode value: "
                                        << "'" << firstByte << "'."
                                        << "Expected 'i', 'l', 'd' or a digit.";
          }
          try {
                    value->loadFromByteArray(data, position);
          } catch (BencodeException &ex2) {
                    delete value;
                    throw ex << "Failed to load value" << endl
                                        << ex2.what();
          }
          return value;
}



BencodeInteger::BencodeInteger() : BencodeValue(Type::Integer)
{
}

BencodeInteger::BencodeInteger(qint64 value)
          : BencodeValue(Type::Integer)
    , _value(value)
{
}

BencodeInteger::~BencodeInteger()
{
}

qint64 BencodeInteger::toInt()
{
    return _value;
}

void BencodeInteger::loadFromByteArray(const QByteArray &data, int &position)
{
          BencodeException ex("BencodeInteger::loadFromByteArray(): ");

    _bencodeData = &data;
    _dataPosBegin = position;
          int &i = position;
          if(i >= data.size()) {
                    throw ex << "Unexpectedly reached end of the data stream";
          }

          char firstByte = data[i++];
          if(firstByte != 'i') {
                    throw ex << "First byte of Integer must be 'i', insted got '" << firstByte <<
"'";
          }

          QString valueString;
          for(;;) {
                    if(i == data.size()) {
                              throw ex << "Unexpectedly reached end of the data stream";
                    }
                    char byte = data[i++];
                    if(byte == 'e') {
```

```
                        break;
                }
                if((byte < '0' || byte > '9') && byte != '-') {
                        throw ex << "Illegal character: '" << byte << "'";
                }
                valueString += byte;
        }
        bool ok;
    _value = valueString.toLongLong(&ok);
    _dataPosEnd = i;
        if(!ok) {
                throw ex << "Value not an integer: '" << valueString << "'";
        }
}



BencodeString::BencodeString() : BencodeValue(Type::String)
{
}

BencodeString::BencodeString(const QByteArray& value)
        : BencodeValue(Type::String)
    , _value(value)
{
}

BencodeString::~BencodeString()
{
}

QByteArray BencodeString::toByteArray()
{
    return _value;
}

void BencodeString::loadFromByteArray(const QByteArray &data, int &position)
{
        BencodeException ex("BencodeString::loadFromByteArray(): ");

    _bencodeData = &data;
    _dataPosBegin = position;
        int& i = position;
        if(i >= data.size()) {
                throw ex << "Unexpectedly reached end of the data stream";
        }

        char firstByte = data[i];
        if(firstByte < '0' || firstByte > '9') {
                throw ex << "First byte must be a digit, but got '" << firstByte << "'";
        }

        QString lengthString;
        for(;;) {
                if(i == data.size()) {
                        throw ex << "Unexpectedly reached end of the data stream";
                }
                char byte = data[i++];
                if(byte == ':') {
                        break;
                }
                if((byte < '0' || byte > '9') && byte != '-') {
                        throw ex << "Illegal character: '" << byte << "'";
                }
                lengthString += byte;
        }
        bool ok;
        int length = lengthString.toInt(&ok);
        if(!ok) {
                throw ex << "Length not an integer: '" << lengthString << "'";
        }

        for(int j = 0; j < length; j++) {
                if(i == data.size()) {
                        throw ex << "Unexpectedly reached end of the data stream";
                }
                char byte = data[i++];
       _value += byte;
        }
```

```
    _dataPosEnd = i;
}



BencodeList::BencodeList() : BencodeValue(Type::List)
{
}

BencodeList::~BencodeList()
{
    for(auto value : _values) {
                delete value;
        }
}

QList<BencodeValue*> BencodeList::toList()
{
    return _values;
}

void BencodeList::loadFromByteArray(const QByteArray &data, int &position)
{
        BencodeException ex("BencodeList::loadFromByteArray(): ");

    _bencodeData = &data;
    _dataPosBegin = position;
        int& i = position;
        if(i >= data.size()) {
                throw ex << "Unexpectedly reached end of the data stream";
        }

        char firstByte = data[i++];
        if(firstByte != 'l') {
                throw ex << "First byte of list must be 'l', instead got '" << firstByte << "'";
        }

        for(;;) {
                if(i >= data.size()) {
                        throw ex << "Unexpectedly reached end of the data stream";
                }
                if(data[i] == 'e') {
                        i++;
                        break;
                }

                BencodeValue* element;
                try {
                        element = BencodeValue::createFromByteArray(data, i);
                } catch(BencodeException& ex2) {
                        throw ex << "Failed to create element" << endl << ex2.what();
                }

        _values.push_back(element);
          }
    _dataPosEnd = i;
}



BencodeDictionary::BencodeDictionary() : BencodeValue(Type::Dictionary)
{
}

BencodeDictionary::~BencodeDictionary()
{
    for(BencodeValue* value : _values.values()) {
                delete value;
        }
}

QList<QByteArray> BencodeDictionary::keys() const
{
    return _values.keys();
}

QList<BencodeValue*> BencodeDictionary::values() const
{
    return _values.values();
```

```
}

bool BencodeDictionary::keyExists(const QByteArray& key) const
{
    return _values.keys().contains(key);
}

BencodeValue* BencodeDictionary::value(const QByteArray& key) const
{
         if(keyExists(key)) {
        return _values.value(key);
         }
         throw BencodeException("BencodeDictionary::value(): No such key: '" + key + "'");
}

void BencodeDictionary::loadFromByteArray(const QByteArray &data, int &position)
{
         BencodeException ex("BencodeDictionary::loadFromByteArray(): ");

    _bencodeData = &data;
    _dataPosBegin = position;
         int& i = position;
         if(i == data.size()) {
                  throw ex << "Unexpectedly reached end of the data stream";
         }
         char firstByte = data[i++];
         if(firstByte != 'd') {
                  throw ex << "First byte of a dictionary must be 'd', instead got '" << firstByte
<< "'";
         }
         for(;;) {
                  if(i >= data.size()) {
                           throw ex << "Unexpectedly reached end of the data stream";
                  }
                  if(data[i] == 'e') {
                           i++;
                           break;
                  }
                  BencodeValue* key = nullptr;
                  QByteArray keyString;
                  BencodeValue* value;
                  try {
                           key = BencodeValue::createFromByteArray(data, i);
                           keyString = key->toByteArray();
                           delete key;
                           key = nullptr;
                  } catch(BencodeException& ex2) {
                           if(key) {
                                    delete key;
                           }
                           throw ex << "Failed to load key" << endl << ex2.what();
                  }

                  try {
                           value = BencodeValue::createFromByteArray(data, i);
                  } catch(BencodeException& ex2) {
                           throw ex << "Failed to load value" << endl << ex2.what();
                  }

        _values[keyString] = value;
         }
    _dataPosEnd = i;
}


void BencodeInteger::setValue(qint64 value)
{
    _value = value;
}

void BencodeString::setValue(const QByteArray &value)
{
    _value = value;
}

void BencodeList::setValues(const QList<BencodeValue *> &values)
{
    _values = values;
}
```

```
void BencodeList::add(BencodeValue *value)
{
    _values.push_back(value);
}

void BencodeDictionary::add(const QByteArray& key, BencodeValue* value)
{
    _values[key] = value;
}


QByteArray BencodeInteger::bencode(bool includeMetadata) const
{
          QByteArray data;
          if(includeMetadata) {
        data.append('i').append(QString::number(_value)).append('e');
          } else {
        data.append(QString::number(_value));
          }
          return data;
}

QByteArray BencodeString::bencode(bool includeMetadata) const
{
          QByteArray data;
          if(includeMetadata) {
        data.append(QString::number(_value.size())).append(':').append(_value);
          } else {
        data.append(_value);
          }
          return data;
}

QByteArray BencodeList::bencode(bool includeMetadata) const
{
          QByteArray data;
          if(includeMetadata) {
                    data.append('l');
          }
    for(BencodeValue* value : _values) {
                    data.append(value->bencode());
          }
          if(includeMetadata) {
                    data.append('e');
          }
          return data;
}

QByteArray BencodeDictionary::bencode(bool includeMetadata) const
{
          QByteArray data;
          if(includeMetadata) {
                    data.append('d');
          }
    for(const QByteArray& key : _values.keys()) {
                    BencodeString bkey(key);
        data.append(bkey.bencode()).append(_values.value(key)->bencode());
          }
          if(includeMetadata) {
                    data.append('e');
          }
          return data;
}


void BencodeInteger::print(QTextStream& out) const
{
    out << _value;
}

void BencodeString::print(QTextStream& out) const
{
    out << _value;
}

void BencodeList::print(QTextStream& out) const
{
          out << "List {" << endl;
```

```
    for(auto v : _values) {
                    QString s;
                    QTextStream stream(&s);
                    v -> print(stream);
                    while(!stream.atEnd()) {
                              QString line = stream.readLine();
                              out << '\t' << line << endl;
                    }
          }
          out << "}";
}

void BencodeDictionary::print(QTextStream& out) const
{
          out << "Dictionary {" << endl;
    for(const QByteArray& key : _values.keys()) {
                    out << key;
                    out << " : ";
                    QString s;
                    QTextStream stream(&s);
        _values[key] -> print(stream);
                    out << stream.readLine() << endl;
                    while(!stream.atEnd()) {
                              out << '\t' << stream.readLine() << endl;
                    }
          }
          out << "}";
}


bool BencodeInteger::equalTo(BencodeValue *other) const
{
          try {
         return other->toInt() == _value;
          } catch(BencodeException& ex) {
                    return false;
          }
}

bool BencodeString::equalTo(BencodeValue *other) const
{
          try {
         return other->toByteArray() == _value;
          } catch(BencodeException& ex) {
                    return false;
          }
}

bool BencodeList::equalTo(BencodeValue *other) const
{
          try {
                    auto list = other->toList();
         if(list.size() != _values.size()) {
                              return false;
                    }
                    for(int i = 0; i < list.size(); i++) {
            if(!list[i]->equalTo(_values[i])) {
                                        return false;
                              }
                    }
                    return true;
          } catch(BencodeException& ex) {
                    return false;
          }
}

bool BencodeDictionary::equalTo(BencodeValue *other) const
{
          try {
                    BencodeDictionary* otherDict = other->toBencodeDictionary();
                    if(keys() != otherDict->keys()) {
                              return false;
                    }
                    if(values() != otherDict->values()) {
                              return false;
                    }
                    return true;
          } catch(BencodeException& ex) {
                    return false;
```

```
        }
}
```

block.h

```
#ifndef BLOCK_H
#define BLOCK_H

#include <QObject>
#include <QByteArray>
#include <QList>

class Piece;
class Peer;

class Block : public QObject
{
        Q_OBJECT

private:
    Piece *_piece;
    int _begin;
    int _size;
    bool _isDownloaded;

    QList<Peer *> _assignees;

public:
        Block(Piece *piece, int begin, int size);
        ~Block();
        Piece *piece();
        int begin() const;
        int size() const;
        bool isDownloaded();
        QList<Peer *> &assignees();
        bool hasAssignees() const;

signals:
        void downloaded(Block *block);

public slots:
        void setDownloaded(bool isDownloaded);
        void setData(const Peer *peer, const char *data);
        void addAssignee(Peer *peer);
        void removeAssignee(Peer *peer);
        void clearAssignees();
};

#endif
```

block.cpp

```
#include "block.h"
#include "piece.h"
#include "peer.h"

Block::Block(Piece *piece, int begin, int size)
    : _piece(piece)
    , _begin(begin)
    , _size(size)
    , _isDownloaded(false)
{
    connect(this, &Block::downloaded, _piece, &Piece::updateState);
}

Block::~Block()
{
}

Piece *Block::piece()
{
    return _piece;
}

int Block::begin() const
{
    return _begin;
}
```

```
int Block::size() const
{
    return _size;
}

bool Block::isDownloaded()
{
    return _isDownloaded;
}

QList<Peer *> &Block::assignees()
{
    return _assignees;
}

bool Block::hasAssignees() const
{
    return !_assignees.isEmpty();
}

void Block::setDownloaded(bool isDownloaded)
{
    _isDownloaded = isDownloaded;
          if (isDownloaded) {
                    emit downloaded(this);
          }
}

void Block::setData(const Peer *peer, const char *data)
{
          if (isDownloaded()) {
                    return;
          }

    char *p = _piece->data() + _begin;
    for (int i = 0; i < _size; i++) {
                    p[i] = data[i];
          }
          setDownloaded(true);
    QList<Peer *> assignees = _assignees;
          for (auto p : assignees) {
                    if (p != peer) {
                              p->sendCancel(this);
                    }
                    p->releaseBlock(this);
          }
}

void Block::addAssignee(Peer *peer)
{
    _assignees.push_back(peer);
}

void Block::removeAssignee(Peer *peer)
{
    for (int i = _assignees.size() - 1; i >= 0; i--) {
        if (_assignees[i] == peer) {
            _assignees.removeAt(i);
                    }
          }
}

void Block::clearAssignees()
{
    _assignees.clear();
}
```

filecontroller.h

```
#ifndef FILECONTROLLER_H
#define FILECONTROLLER_H

#include <QObject>

class QThread;
class Torrent;
class Piece;
```

```
class FileControllerWorker : public QObject
{
          Q_OBJECT

public:
          FileControllerWorker(Torrent *torrent);

public slots:
          void checkTorrent();

signals:
          void torrentChecked();
          void pieceAvailable(Piece* piece, bool available);

private:
    Torrent *_torrent;
};

class FileController : public QObject
{
          Q_OBJECT

public:
          FileController(Torrent *torrent);
          ~FileController();

signals:
          void checkTorrent();
          void torrentChecked();

private:
    Torrent *_torrent;
    QThread *_workerThread;
};

#endif
```

filecontroller.cpp

```
#include "filecontroller.h"
#include "torrent.h"
#include "torrentinfo.h"
#include "piece.h"
#include <QCryptographicHash>
#include <QThread>

FileController::FileController(Torrent *torrent)
    : _torrent(torrent)
    , _workerThread(new QThread)
{
          FileControllerWorker *worker = new FileControllerWorker(torrent);
    worker->moveToThread(_workerThread);
    connect(_workerThread, &QThread::finished, worker, &FileControllerWorker::deleteLater);

    _workerThread->start();

          connect(this, &FileController::checkTorrent, worker, &FileControllerWorker::checkTorrent);
          connect(worker, &FileControllerWorker::torrentChecked, this,
&FileController::torrentChecked);
    connect(worker, &FileControllerWorker::pieceAvailable, _torrent, &Torrent::setPieceAvailable);
}

FileController::~FileController()
{
    _workerThread->quit();
    _workerThread->wait();
    delete _workerThread;
}


FileControllerWorker::FileControllerWorker(Torrent *torrent)
    : _torrent(torrent)
{
}

void FileControllerWorker::checkTorrent()
{
```

```
    TorrentInfo *info = _torrent->torrentInfo();
    QList<Piece *> &pieces = _torrent->pieces();
          for (Piece *piece : pieces) {
                    emit pieceAvailable(piece, false);
          }
          for (Piece *piece : pieces) {
                    QByteArray pieceData, pieceHash;
                    if (!piece->getPieceData(pieceData)) {
                              continue;
                    }
                    pieceHash = QCryptographicHash::hash(pieceData, QCryptographicHash::Sha1);
                    bool pieceIsValid = (pieceHash == info->piece(piece->pieceNumber()));
                    if (pieceIsValid) {
                              emit pieceAvailable(piece, true);
                    }
          }
          emit torrentChecked();
}
```

localservicediscoveryclient.h

```
#ifndef LOCALSERVICEDISCOVERY_H
#define LOCALSERVICEDISCOVERY_H

#define LSD_ADDRESS_IPV4 "37.214.31.191"
#define LSD_PORT_IPV4 6771
#define LSD_ADDRESS_IPV6 "ff15::efc0:988f" //ff15::efc0:988f
#define LSD_PORT_IPV6 6771
#define LSD_INTERVAL 5*60*1000
#define LSD_MIN_INTERVAL 60*1000

#include <QObject>
#include <QElapsedTimer>
#include <QHostAddress>
#include <QByteArray>

class Torrent;
class QTimer;
class QUdpSocket;

class LocalServiceDiscoveryClient : public QObject
{
          Q_OBJECT

public:
          LocalServiceDiscoveryClient(QObject *parent = nullptr);

public slots:
          void announceAll();
          void announceIPv4();
          void announceIPv6();
          void processPendingDatagrams();

signals:
          void foundPeer(QHostAddress address, int port, Torrent *torrent);

private:
    QTimer *_announceTimer;
    QElapsedTimer _elapsedTimer;
    QUdpSocket *_socketIPv4;
    QUdpSocket *_socketIPv6;
    QByteArray _cookie;

          void announce(QUdpSocket *socket, const char *address, int port);
};

#endif
```

localservicediscoveryclient.cpp

```
#include "localservicediscoveryclient.h"
#include <QTimer>
#include <QUdpSocket>
#include <QString>
#include <QStringList>

#include "jtorrent.h"
```

```
#include "torrentserver.h"
#include "torrent.h"
#include "torrentinfo.h"

LocalServiceDiscoveryClient::LocalServiceDiscoveryClient(QObject *parent)
          : QObject(parent)
{
    _announceTimer = new QTimer(this);
    _socketIPv4 = new QUdpSocket(this);
    _socketIPv4->bind(QHostAddress::AnyIPv4, LSD_PORT_IPV4, QUdpSocket::ShareAddress);
    _socketIPv4->joinMulticastGroup(QHostAddress(LSD_ADDRESS_IPV4));
    _socketIPv4->setSocketOption(QAbstractSocket::MulticastTtlOption, 20);

    _socketIPv6 = new QUdpSocket(this);
    _socketIPv6->bind(QHostAddress::AnyIPv6, LSD_PORT_IPV6, QUdpSocket::ShareAddress);
    _socketIPv6->joinMulticastGroup(QHostAddress(LSD_ADDRESS_IPV6));
    _socketIPv6->setSocketOption(QAbstractSocket::MulticastTtlOption, 20);
          for (int i = 0; i < 20; i++) {
                    int q = qrand() % (10+26+26);
                    if (q < 10) {
            _cookie.append('0' + q);
                    } else if (q < 36) {
            _cookie.append('A' + q - 10);
                    } else {
            _cookie.append('a' + q - 36);
                    }
          }

    connect(_announceTimer, &QTimer::timeout, this, &LocalServiceDiscoveryClient::announceAll);
    connect(_socketIPv4, &QUdpSocket::readyRead, this,
&LocalServiceDiscoveryClient::processPendingDatagrams);
    connect(_socketIPv6, &QUdpSocket::readyRead, this,
&LocalServiceDiscoveryClient::processPendingDatagrams);
}

void LocalServiceDiscoveryClient::announce(QUdpSocket *socket, const char *address, int port)
{
          QHostAddress addr(address);
          QString addressString = address;
          if (addr.protocol() == QAbstractSocket::IPv6Protocol) {
                    addressString.prepend('[');
                    addressString.append(']');
          }
          QString datagramString;
          QTextStream datagramStream(&datagramString);
          datagramStream << "BT-SEARCH * HTTP/1.1\r\n"
                                        << "Host: " << addressString << ":" << port << "\r\n"
                    << "Port: " << JTorrent::instance()->server()->port() << "\r\n";

    for (Torrent *torrent : JTorrent::instance()->torrents()) {
                    QByteArray hash = torrent->torrentInfo()->infoHash().toHex().toLower();
                    datagramStream << "Infohash: " << hash << "\r\n";
          }

    datagramStream << "cookie: " << _cookie << "\r\n";
          datagramStream << "\r\n\r\n";
          socket->writeDatagram(datagramString.toLatin1(), addr, port);

    _announceTimer->start(LSD_INTERVAL);
    _elapsedTimer.start();
}

void LocalServiceDiscoveryClient::announceAll()
{
    int elapsed = _elapsedTimer.elapsed();
    if (elapsed < LSD_MIN_INTERVAL && _announceTimer->isActive()) {
        _announceTimer->start(LSD_MIN_INTERVAL - elapsed);
                    return;
          }

    if (JTorrent::instance()->torrents().isEmpty()) {
                    return;
          }

          announceIPv4();
          announceIPv6();
}

void LocalServiceDiscoveryClient::announceIPv4()
```

```
{
    announce(_socketIPv4, LSD_ADDRESS_IPV4, LSD_PORT_IPV4);
}

void LocalServiceDiscoveryClient::announceIPv6()
{
    announce(_socketIPv6, LSD_ADDRESS_IPV6, LSD_PORT_IPV6);
}

void LocalServiceDiscoveryClient::processPendingDatagrams()
{
          QList<QByteArray> infoHashes;
    for (Torrent *torrent : JTorrent::instance()->torrents()) {
                  infoHashes.append(torrent->torrentInfo()->infoHash().toHex().toLower());
          }

          for (;;) {
                  QUdpSocket *senderSocket = nullptr;
        if (_socketIPv4->hasPendingDatagrams()) {
            senderSocket = _socketIPv4;
        } else if (_socketIPv6->hasPendingDatagrams()) {
            senderSocket = _socketIPv6;
                  } else {
                            break;
                  }

                  QHostAddress sender;
                  QByteArray datagram;
                  datagram.resize(senderSocket->pendingDatagramSize());
                  senderSocket->readDatagram(datagram.data(), datagram.size(), &sender);

                  int port = 0;
                  QList<QByteArray> receivedInfoHashes;
                  QByteArray cookie;

                  QTextStream stream(datagram);
                  QString line;
                  stream.readLineInto(&line);
                  if (line != "BT-SEARCH * HTTP/1.1") {
                            continue;
                  }
                  while (stream.readLineInto(&line)) {
                            QStringList splitLine = line.split(": ");
                            if (splitLine.size() == 2) {
                                      QString header = splitLine.first();
                                      if (header == "Host") {
                                      } else if (header == "Port") {
                                                bool ok;
                                                port = splitLine.last().toInt(&ok);
                                                if(!ok) {
                                                          port = 0;
                                                }
                                      } else if (header == "Infohash") {

          receivedInfoHashes.append(splitLine.last().toLatin1().toLower());
                                      } else if (header == "cookie") {
                                                cookie = splitLine.last().toLatin1();
                                      }
                            }
                  }

        if (port == 0 || cookie == _cookie) {
                            continue;
                  }

                  for (QByteArray& hash : receivedInfoHashes) {
                            if (infoHashes.contains(hash)) {
                for (Torrent *torrent : JTorrent::instance()->torrents()) {
                                                if (torrent->torrentInfo()-
>infoHash().toHex().toLower() == hash) {
                                                          emit foundPeer(sender, port, torrent);
                                                }
                                      }
                            }
                  }
          }
}
```

peer.h

```
#ifndef PEER_H
#define PEER_H

#include <QByteArray>
#include <QHostAddress>
#include <QTimer>
#include <QObject>
#include <QAbstractSocket>

class Torrent;
class Piece;
class Block;
class QTcpSocket;


class Peer : public QObject
{
          Q_OBJECT

public:
          enum State {
         Created,
         Connecting,
         Handshaking,
         ConnectionEstablished,
         Disconnected,
         Error
          };

          enum class ConnectionInitiator
          {
         Client,
         Peer
          };

          Torrent *torrent();

          QHostAddress address();
          int port();
          int piecesDownloaded();
          bool *bitfield();
          QByteArray &protocol();
          QByteArray &reserved();
          QByteArray &infoHash();
          QByteArray &peerId();

          State state();
          ConnectionInitiator connectionInitiator();
          bool amChoking();
          bool amInterested();
          bool peerChoking();
          bool peerInterested();

          QTcpSocket *socket();
          bool hasTimedOut();
          QList<Block *> &blocksQueue();
          bool isPaused() const;

          QString addressPort();
          bool isDownloaded();
          bool hasPiece(Piece *piece);
          bool isConnected();
          bool isInteresting();

private:
    Torrent *_torrent;

    QHostAddress _address;
    int _port;
    int _piecesDownloaded;
    bool *_bitfield;
    QByteArray _protocol;
    QByteArray _reserved;
    QByteArray _infoHash;
    QByteArray _peerId;

    State _state;
```

```
    ConnectionInitiator _connectionInitiator;
    bool _amChoking;
    bool _amInterested;
    bool _peerChoking;
    bool _peerInterested;

    QTcpSocket *_socket;
    QByteArray _receivedDataBuffer;
    QTimer _replyTimeoutTimer;
    QTimer _handshakeTimeoutTimer;
    QTimer _reconnectTimer;

    QTimer _sendMessagesTimer;

    bool _hasTimedOut;

    QList<Block *> _blocksQueue;

    bool _isPaused;

          bool readHandshakeReply(bool *ok);

          bool readPeerMessage(bool *ok);

          void connectAll();

          void initBitfield();

          void initClient();

          void initServer(Torrent *torrent, QHostAddress address, int port);

public:
          Peer(ConnectionInitiator connectionInitiator, QTcpSocket *socket);
          ~Peer();

          static Peer *createClient(QTcpSocket *socket);

          static Peer* createServer(Torrent *torrent, QHostAddress address, int port);

signals:
          void uploadedData(qint64 bytes);
          void downloadedData(qint64 bytes);

public slots:
          void startConnection();

          void start();
          void pause();

          void sendHandshake();
          void sendChoke();
          void sendUnchoke();
          void sendInterested();
          void sendNotInterested();
          void sendHave(int index);
          void sendBitfield();
          void sendRequest(Block *block);
          void sendPiece(int index, int begin, const QByteArray &blockData);
          void sendCancel(Block *block);

          bool requestBlock();

          void releaseBlock(Block *block);
          void releaseAllBlocks();

          void disconnect();

          void fatalError();

          void sendMessages();

          void connected();
          void readyRead();
          void finished();
          void error(QAbstractSocket::SocketError socketError);
          void replyTimeout();
          void handshakeTimeout();
          void reconnect();
```

```
};

#endif
```

peer.cpp

```
#include "peer.h"
#include "block.h"
#include "piece.h"
#include "jtorrent.h"
#include "torrent.h"
#include "torrentinfo.h"
#include "torrentmessage.h"
#include <QTcpSocket>
#include <QHostAddress>
#include <QTimer>
#include <QDebug>

const int BLOCK_REQUEST_SIZE = 16384;
const int REPLY_TIMEOUT_MSEC = 10000;
const int HANDSHAKE_TIMEOUT_MSEC = 20000;
const int BLOCKS_TO_REQUEST = 5;
const int MAX_MESSAGE_LENGTH = 65536;
const int RECONNECT_INTERVAL_MSEC = 30000;
const int SEND_MESSAGES_INTERVAL = 1000;

Peer::Peer(ConnectionInitiator connectionInitiator, QTcpSocket *socket)
    : _torrent(nullptr)
    , _bitfield(nullptr)
    , _state(Created)
    , _connectionInitiator(connectionInitiator)
    , _socket(socket)
    , _isPaused(false)
{
          connectAll();
}

Peer::~Peer()
{
    delete[] _bitfield;
    delete _socket;
}


void Peer::startConnection()
{
    if (_connectionInitiator == ConnectionInitiator::Peer) {
                  qDebug() << "Peer::startConnection(): Called, but connection was initiated by
the Peer";
                  return;
          }
    if (_socket->isOpen()) {
        _socket->close();
          }

    _piecesDownloaded = 0;
    for (int i = 0; i < _torrent->torrentInfo()->bitfieldSize() * 8; i++) {
        _bitfield[i] = false;
          }
    _protocol.clear();
    _reserved.clear();
    _infoHash.clear();
    _peerId.clear();

    _state = Connecting;
    _amChoking = true;
    _amInterested = false;
    _peerChoking = true;
    _peerInterested = false;

    _receivedDataBuffer.clear();
    _replyTimeoutTimer.stop();
    _handshakeTimeoutTimer.stop();
    _reconnectTimer.stop();

    _sendMessagesTimer.stop();

    _hasTimedOut = false;
```

```
    _blocksQueue.clear();

          qDebug() << "Connecting to" << addressPort();
    _socket->connectToHost(_address, _port);
}

void Peer::start()
{
    if(_connectionInitiator == ConnectionInitiator::Peer) {
                return;
          }

    _isPaused = false;
    if (_state == ConnectionEstablished) {
                sendMessages();
    } else if (_socket->state() == QAbstractSocket::UnconnectedState) {
                startConnection();
          }
}

void Peer::pause()
{
    _isPaused = true;
          sendMessages();
}

void Peer::sendHandshake()
{
          QByteArray dataToWrite;
          dataToWrite.push_back(char(19));
          dataToWrite.push_back("BitTorrent protocol");
          for (int i = 0; i < 8; i++) {
                dataToWrite.push_back(char(0));
          }
    dataToWrite.push_back(_torrent->torrentInfo()->infoHash());
    dataToWrite.push_back(JTorrent::instance()->peerId());
    _socket->write(dataToWrite);
}

void Peer::sendChoke()
{
    if (_state != ConnectionEstablished) {
                return;
          }
    _amChoking = true;
    TorrentMessage::choke(_socket);
}

void Peer::sendUnchoke()
{
    if (_state != ConnectionEstablished) {
                return;
          }
    _amChoking = false;
    TorrentMessage::unchoke(_socket);
}

void Peer::sendInterested()
{
    if (_state != ConnectionEstablished) {
                return;
          }
    _amInterested = true;
    TorrentMessage::interested(_socket);
}

void Peer::sendNotInterested()
{
    if (_state != ConnectionEstablished) {
                return;
          }
    _amInterested = false;
    TorrentMessage::notInterested(_socket);
}

void Peer::sendHave(int index)
{
    if (_state != ConnectionEstablished) {
                return;
```

```
        }
    TorrentMessage::have(_socket, index);
}

void Peer::sendBitfield()
{
    if (_state != ConnectionEstablished) {
                return;
        }
    TorrentMessage::bitfield(_socket, _torrent->bitfield());
}

void Peer::sendRequest(Block* block)
{
    if (_state != ConnectionEstablished) {
                return;
        }
        block->addAssignee(this);

        int index = block->piece()->pieceNumber();
        int begin = block->begin();
        int length = block->size();
        qDebug() << "Request" << index << begin << length << "from" << addressPort();
    TorrentMessage::request(_socket, index, begin, length);

    _replyTimeoutTimer.start();

    _blocksQueue.push_back(block);
}

void Peer::sendPiece(int index, int begin, const QByteArray &blockData)
{
    if (_state != ConnectionEstablished) {
                return;
        }
        qDebug() << "Sending piece" << index << begin << blockData.size() << "to" <<
addressPort();
    TorrentMessage::piece(_socket, index, begin, blockData);
    _torrent->onBlockUploaded(blockData.size());
        emit uploadedData(blockData.size());
}

void Peer::sendCancel(Block* block)
{
    if (_state != ConnectionEstablished) {
                return;
        }
        int      index = block->piece()->pieceNumber();
        int begin = block->begin();
        int length = block->size();
    TorrentMessage::cancel(_socket, index, begin, length);
}

bool Peer::requestBlock()
{
    Block *block = _torrent->requestBlock(this, BLOCK_REQUEST_SIZE);
        if (block == nullptr) {
                return false;
        }
        sendRequest(block);
        return true;
}

void Peer::disconnect()
{
        qDebug() << "Disconnecting from" << addressPort();
        if (isConnected()) {
       _socket->close();
        } else {
                finished();
        }
}

void Peer::fatalError()
{
        qDebug() << "Fatal error with" << addressPort() << "; Dropping connection";
    _state = Error;
    _socket->close();
}
```

```
Peer *Peer::createClient(QTcpSocket *socket)
{
          Peer *peer = new Peer(ConnectionInitiator::Peer, socket);
          peer->initClient();
          return peer;
}

Peer *Peer::createServer(Torrent *torrent, QHostAddress address, int port)
{
          Peer *peer = new Peer(ConnectionInitiator::Client, new QTcpSocket);
          peer->initServer(torrent, address, port);
          return peer;
}

void Peer::sendMessages()
{
    _sendMessagesTimer.stop();

    if (_state != ConnectionEstablished) {
                return;
          }

    if (_torrent->isDownloaded() && isDownloaded()) {
                disconnect();
                return;
          }

    _sendMessagesTimer.start(SEND_MESSAGES_INTERVAL);

    if (_isPaused) {

        if (_amInterested) {
                          sendNotInterested();
                }
        if (!_amChoking) {
                          sendChoke();
                }
        for (Block *block : _blocksQueue) {
                          sendCancel(block);
                }
                releaseAllBlocks();

          } else {

        if (!_amInterested) {
                          if (isInteresting()) {
                                    sendInterested();
                          }
                }
        if (_peerInterested && _amChoking) {
                          sendUnchoke();
                }
        if (!_peerChoking && _amInterested) {
            while (_blocksQueue.size() < BLOCKS_TO_REQUEST) {
                          if (!requestBlock()) {
                                    break;
                          }
                          }
                }

          }
}


bool Peer::readHandshakeReply(bool *ok)
{
          *ok = true;

    if (_receivedDataBuffer.isEmpty()) {
                return false;
          }
          int i = 0;
    int protocolLength = _receivedDataBuffer[i++];
    if (_receivedDataBuffer.size() < 49 + protocolLength) {
                return false;
          }

          for (int j = 0; j < protocolLength; j++) {
```

```
        _protocol.push_back(_receivedDataBuffer[i++]);
          }
          for (int j = 0; j < 8; j++) {
        _reserved.push_back(_receivedDataBuffer[i++]);
          }
          for (int j = 0; j < 20; j++) {
        _infoHash.push_back(_receivedDataBuffer[i++]);
          }
          for (int j = 0; j < 20; j++) {
        _peerId.push_back(_receivedDataBuffer[i++]);
          }
    _receivedDataBuffer.remove(0, 49 + protocolLength);

    if (_connectionInitiator == ConnectionInitiator::Client) {
        if (_infoHash != _torrent->torrentInfo()->infoHash()) {
                            qDebug() << "Info hash does not match expected one from peer" <<
addressPort();
                            *ok = false;
                            return false;
                    }
          } else {
        _torrent = nullptr;
        for (auto torrent : JTorrent::instance()->torrents()) {
            if (torrent->torrentInfo()->infoHash() == _infoHash) {
                _torrent = torrent;
                                    break;
                            }
                    }
        if (_torrent == nullptr) {
            qDebug() << "No torrents matching info hash" << _infoHash.toHex() << "for" <<
addressPort();
                            *ok = false;
                            return false;
                    }
          }
          return true;
}

bool Peer::readPeerMessage(bool *ok)
{
          *ok = true;

    if (_receivedDataBuffer.size() < 4) {
                    return false;
          }

          int i = 0;

          int length = 0;
          for (int j = 0; j < 4; j++) {
                    length *= 256;
        length += (unsigned char)_receivedDataBuffer[i++];
          }

          if (length > MAX_MESSAGE_LENGTH || length < 0) {
                    *ok = false;
                    return false;
          }

    if (length == 0) {
                    qDebug() << addressPort() << ": keep-alive";
        _receivedDataBuffer.remove(0, i);
                    return true;
          }

    if (_receivedDataBuffer.size() < 4 + length) {
                    return false;
          }

    int messageId = _receivedDataBuffer[i++];
          switch (messageId) {
          case TorrentMessage::Choke: {
                    qDebug() << addressPort() << ": choke";
        _peerChoking = true;
                    releaseAllBlocks();
        _replyTimeoutTimer.stop();
        _hasTimedOut = false;
                    break;
          }
```

```
          case TorrentMessage::Unchoke: {
                    qDebug() << addressPort() << ": unchoke";
          _peerChoking = false;
                    break;
          }
          case TorrentMessage::Interested: {
                    qDebug() << addressPort() << ": interested";
          _peerInterested = true;
                    break;
          }
          case TorrentMessage::NotInterested: {
                    qDebug() << addressPort() << ": not interested";
          _peerInterested = false;
                    break;
          }
          case TorrentMessage::Have: {
                    int pieceNumber = 0;
                    for (int j = 0; j < 4; j++) {
                              pieceNumber *= 256;
             pieceNumber += (unsigned char)_receivedDataBuffer[i++];
                    }
          if (!_bitfield[pieceNumber]) {
              _bitfield[pieceNumber] = true;
              _piecesDownloaded++;
                    }
                    break;
          }
          case TorrentMessage::Bitfield: {
                    int bitfieldSize = length - 1;
          if (bitfieldSize != _torrent->torrentInfo()->bitfieldSize()) {
                              qDebug() << "Error: Peer" << addressPort() << "sent bitfield of wrong
size:" << bitfieldSize*8
                                << "expected" << _torrent->torrentInfo()->bitfieldSize();
                              *ok = false;
                              return false;
                    } else {
                              for (int j = 0; j < bitfieldSize; j++) {
                    unsigned char byte = _receivedDataBuffer[i++];
                                        unsigned char pos = 0b10000000;
                                        for (int q = 0; q < 8; q++) {
                         _bitfield[j * 8 + q] = ((byte & pos) != 0);
                                                  pos = pos >> 1;
                                        }
                              }
             _piecesDownloaded = 0;
                              for (int j = 0; j < bitfieldSize * 8; j++) {
                    if (_bitfield[j]) {
                         _piecesDownloaded++;
                                        }
                              }
                    }
                    break;
          }
          case TorrentMessage::Request: {
                    unsigned int index = 0;
                    unsigned int begin = 0;
                    unsigned int blockLength = 0;

                    for (int j = 0; j < 4; j++) {
                              index *= 256;
             index += (unsigned char)_receivedDataBuffer[i++];
                    }
                    for (int j = 0; j < 4; j++) {
                              begin *= 256;
             begin += (unsigned char)_receivedDataBuffer[i++];
                    }
                    for (int j = 0; j < 4; j++) {
                              blockLength *= 256;
             blockLength += (unsigned char)_receivedDataBuffer[i++];
                    }

          QList<Piece *> &pieces = _torrent->pieces();

                    if (index >= (unsigned)pieces.size() || blockLength > MAX_MESSAGE_LENGTH) {
                              qDebug() << "Invalid request (" << index << begin << blockLength << ")"
                                                  << "from" << addressPort();
                              disconnect();
                              *ok = false;
                              return false;
```

```
                    }
                    Piece *piece = pieces[index];

                    if (begin + blockLength > (unsigned)piece->size() || begin > (unsigned)piece-
>size()) {
                              qDebug() << "Invalid request (" << index << begin << blockLength << ")"
                                                  << "from" << addressPort();
                              disconnect();
                              *ok = false;
                              return false;
                    }

                    QByteArray blockData;
                    if (!piece->getBlockData(begin, blockLength, blockData)) {
                              qDebug() << "Failed to get block (" << index << begin << blockLength <<
")"
                                                  << "for" << addressPort();
                              disconnect();
                              *ok = false;
                              return false;
                    }

                    sendPiece(index, begin, blockData);

                    break;
          }
          case TorrentMessage::Piece: {
                    int index = 0;
                    int begin = 0;
                    int blockLength = length - 9;
                    for (int j = 0; j < 4; j++) {
                              index *= 256;
            index += (unsigned char)_receivedDataBuffer[i++];
                    }
                    for (int j = 0; j < 4; j++) {
                              begin *= 256;
            begin += (unsigned char)_receivedDataBuffer[i++];
                    }
                    Block *block = nullptr;
                    int blockIndex = 0;
        for (auto b : _blocksQueue) {
                              if (b->piece()->pieceNumber() == index
                                        && b->begin() == begin
                                        && b->size() == blockLength) {
                                        block = b;
                                        break;
                              }
                              blockIndex++;
                    }

                    if (block == nullptr) {
            QList<Piece *> pieces = _torrent->pieces();
                              if (index >= 0 && index < pieces.size()) {
                                        block = pieces[index]->getBlock(begin, blockLength);
                              }
                    }

                    if (block == nullptr) {
                              qDebug() << "Error: received unrequested block from peer" <<
addressPort()
                                                  << ". Block(" << index << begin << blockLength <<
")";
                    } else {
            _hasTimedOut = false;
            const char *blockData = _receivedDataBuffer.data() + i;
                              if (!block->isDownloaded()) {
                                        block->setData(this, blockData);
                                        releaseBlock(block);
                                        emit downloadedData(blockLength);
                              }
            if (_blocksQueue.isEmpty()) {
                _replyTimeoutTimer.stop();
                              } else {
                _replyTimeoutTimer.start();
                              }
                    }
                    break;
          }
          case TorrentMessage::Cancel: {
```

```
                    break;
            }
            case TorrentMessage::Port: {
                    break;
            }
            default:
                    qDebug() << "Error: Received unknown message with id =" << messageId
                                    << " and length =" << length << "from" << addressPort();
                    *ok = false;
                    return false;
            }
    _receivedDataBuffer.remove(0, 4 + length);
            return true;
}

void Peer::connectAll()
{
    connect(_socket, SIGNAL(connected()), this, SLOT(connected()));
    connect(_socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
    connect(_socket, SIGNAL(disconnected()), this, SLOT(finished()));
    connect(_socket, SIGNAL(error(QAbstractSocket::SocketError)), this,
SLOT(error(QAbstractSocket::SocketError)));

    connect(&_replyTimeoutTimer, SIGNAL(timeout()), this, SLOT(replyTimeout()));
    connect(&_handshakeTimeoutTimer, SIGNAL(timeout()), this, SLOT(handshakeTimeout()));
    connect(&_reconnectTimer, SIGNAL(timeout()), this, SLOT(reconnect()));
    connect(&_sendMessagesTimer, SIGNAL(timeout()), this, SLOT(sendMessages()));

    _replyTimeoutTimer.setInterval(REPLY_TIMEOUT_MSEC);
    _handshakeTimeoutTimer.setInterval(HANDSHAKE_TIMEOUT_MSEC);
    _reconnectTimer.setInterval(RECONNECT_INTERVAL_MSEC);
}

void Peer::initBitfield()
{
    int bitfieldSize = _torrent->torrentInfo()->bitfieldSize();
    _bitfield = new bool[bitfieldSize * 8];
          for (int i = 0; i < bitfieldSize * 8; i++) {
        _bitfield[i] = false;
          }
}

void Peer::initClient()
{
    _torrent = nullptr;
    _address = _socket->peerAddress();
    _port = _socket->peerPort();
    _piecesDownloaded = 0;
    _bitfield = nullptr;
    _state = Handshaking;

    _protocol.clear();
    _reserved.clear();
    _infoHash.clear();
    _peerId.clear();

    _amChoking = true;
    _amInterested = false;
    _peerChoking = true;
    _peerInterested = false;

    _receivedDataBuffer.clear();
    _replyTimeoutTimer.stop();
    _handshakeTimeoutTimer.stop();
    _reconnectTimer.stop();

    _sendMessagesTimer.stop();

    _hasTimedOut = false;
    _blocksQueue.clear();
}

void Peer::initServer(Torrent *torrent, QHostAddress address, int port)
{
    _torrent = torrent;
    _address = address;
    _port = port;
    _piecesDownloaded = 0;
          initBitfield();
```

```
    _state = Created;
}

void Peer::releaseBlock(Block *block)
{
          block->removeAssignee(this);
    _blocksQueue.removeAll(block);
          if (!block->hasAssignees() && !block->isDownloaded()) {
                    block->piece()->deleteBlock(block);
          }
}

void Peer::releaseAllBlocks()
{
    QList<Block *> blocks = _blocksQueue;
          for (Block *block : blocks) {
                    block->removeAssignee(this);
          _blocksQueue.removeAll(block);
                    if (!block->hasAssignees() && !block->isDownloaded()) {
                              block->piece()->deleteBlock(block);
                    }
          }
}

void Peer::connected()
{
          qDebug() << "Connected to" << addressPort();

    _state = Handshaking;
          sendHandshake();
    _handshakeTimeoutTimer.start();
}

void Peer::readyRead()
{
    _receivedDataBuffer.push_back(_socket->readAll());

    switch (_state) {
          case Handshaking:
                    bool ok;
                    if (readHandshakeReply(&ok)) {
            if (_connectionInitiator == ConnectionInitiator::Peer) {
                if(_torrent->state() != Torrent::Started) {
                                                  disconnect();
                                                  break;
                                        }
                                        initBitfield();
                                        sendHandshake();
                    _torrent->addPeer(this);
                                }
                    } else {
                                if (!ok) {
                                        fatalError();
                                }
                                break;
                    }
          _handshakeTimeoutTimer.stop();
                    qDebug() << "Handshaking completed with peer" << addressPort();
          _state = ConnectionEstablished;
          _sendMessagesTimer.start(SEND_MESSAGES_INTERVAL);
                    sendBitfield();
            case ConnectionEstablished: {
                    int messagesReceived = 0;
                    while (readPeerMessage(&ok)) {
                              messagesReceived++;
                    }
                    if (!ok) {
                              fatalError();
                              break;
                    }

                    if (messagesReceived) {
                              sendMessages();
                    }
                    break;
          }
            default:
          _receivedDataBuffer.clear();
                    break;
```

```
        }
}

void Peer::finished()
{
    _handshakeTimeoutTimer.stop();
    _replyTimeoutTimer.stop();
    _sendMessagesTimer.stop();
         releaseAllBlocks();
    if (_state != Error) {
        _state = Disconnected;
          }

    if (_connectionInitiator == ConnectionInitiator::Client) {
        if (!isDownloaded() || !_torrent->isDownloaded()) {
            _reconnectTimer.start();
                   }
         }
    qDebug() << "Connection to" << addressPort() << "closed" << _socket->errorString();
}

void Peer::error(QAbstractSocket::SocketError socketError)
{
         qDebug() << "Socket error" << addressPort() << ":"
             << _socket->errorString() << "(" << socketError << ")";
         disconnect();
}

void Peer::replyTimeout()
{
         qDebug() << "Peer" << addressPort() << "took too long to reply";
    _hasTimedOut = true;
    _replyTimeoutTimer.stop();
}

void Peer::handshakeTimeout()
{
         qDebug() << "Peer" << addressPort() << "took too long to handshake";
    _handshakeTimeoutTimer.stop();
}

void Peer::reconnect()
{
         qDebug() << "Reconnecting to" << addressPort();
    _reconnectTimer.stop();
    if (_torrent->isStarted()) {
                  startConnection();
         }
}

Torrent *Peer::torrent()
{
    return _torrent;
}

QHostAddress Peer::address()
{
    return _address;
}

int Peer::port()
{
    return _port;
}

int Peer::piecesDownloaded()
{
    return _piecesDownloaded;
}

bool *Peer::bitfield()
{
    return _bitfield;
}

QByteArray &Peer::protocol()
{
    return _protocol;
}
```

```
QByteArray &Peer::reserved()
{
    return _reserved;
}

QByteArray &Peer::infoHash()
{
    return _infoHash;
}

QByteArray &Peer::peerId()
{
    return _peerId;
}

Peer::State Peer::state()
{
    return _state;
}

Peer::ConnectionInitiator Peer::connectionInitiator()
{
    return _connectionInitiator;
}

bool Peer::amChoking()
{
    return _amChoking;
}

bool Peer::amInterested()
{
    return _amInterested;
}

bool Peer::peerChoking()
{
    return _peerChoking;
}

bool Peer::peerInterested()
{
    return _peerInterested;
}

QTcpSocket *Peer::socket()
{
    return _socket;
}

bool Peer::hasTimedOut()
{
    return _hasTimedOut;
}

QList<Block *> &Peer::blocksQueue()
{
    return _blocksQueue;
}

bool Peer::isPaused() const
{
    return _isPaused;
}

QString Peer::addressPort()
{
    return _address.toString() + ":" + QString::number(_port);
}

bool Peer::isDownloaded()
{
    return _piecesDownloaded == _torrent->torrentInfo()->numberOfPieces();
}

bool Peer::hasPiece(Piece *piece)
{
    return _bitfield[piece->pieceNumber()];
```

```
}

bool Peer::isConnected()
{
    return _socket->state() == QAbstractSocket::ConnectedState;
}

bool Peer::isInteresting()
{
    if (_torrent->isDownloaded()) {
                return false;
          }
    for (Piece* piece : _torrent->pieces()) {
                if (!piece->isDownloaded() && hasPiece(piece)) {
                          return true;
                }
          }
          return false;
}


piece.h

#ifndef PIECE_H
#define PIECE_H

#include <QObject>
#include <QList>
#include <QByteArray>

class Torrent;
class Block;

class Piece : public QObject
{
          Q_OBJECT

private:
    Torrent *_torrent;
    int _pieceNumber;
    int _size;
    bool _isDownloaded;
    char *_pieceData;
    QList<Block *> _blocks;

          void addBlock(Block *block);
          bool checkIfFullyDownloaded();

public:
          Piece(Torrent *torrent, int pieceNumber, int size);
          ~Piece();

          bool isDownloaded() const;
          int pieceNumber() const;
          char *data() const;
          int size() const;

          bool getBlockData(int begin, int size, QByteArray &blockData);
          bool getPieceData(QByteArray &pieceData);
          Block *getBlock(int begin, int size) const;
          Block *requestBlock(int size);

signals:
          void availabilityChanged(Piece *piece, bool isDownloaded);

public slots:
          void updateState();
          void deleteBlock(Block *block);
          void unloadFromMemory();
          void setDownloaded(bool isDownloaded);
};
```

ПРИЛОЖЕНИЕ Г
(*обязательное*)

Ведомость документов