qtmud Documentation

Release 0.1.0

emsenn

Dec 21, 2018

Contents

Resources:

CHAPTER 1

Using qtMUD

On its own, qtMUD is a talker-style MUD. This means it provides basic user accounts and the ability for them to communicate with each other directly or through channels, along with some utility functions.

qtMUD is

While this is intended to be a framework for more complex MUDs,

1.1 Quickstart

The easiest way to get started with qtMUD is by installing the PyPi package:

```
$ pip install qtmud
```

If this fails for any reason, check the *Installation* seciton for help. If it works:

```
$ qtmud_run
```

should produce output like:

```
qtmud_run preparing to start qtmud 0.0.6
qtmud        WARNING  No valid config found, using default values
qtmud        INFO     qtmud.load() called
qtmud        INFO     adding qtmud.subscriptions to qtmud.subscribers
qtmud        INFO     adding qtmud.services to qtmud.active_services
qtmud        INFO     qtmud.load()ed
qtmud        INFO     filling qtmud.client_accounts from ./qtmud_client_accounts.p
qtmud        INFO     start()ing active_services
qtmud        INFO     talker start()ed
qtmud        INFO     start()ing MUDSocket
qtmud        INFO     MUDSocket successfully bound to ('localhost', 5787)
qtmud        INFO     MUDSocket successfully bound to ('localhost', 5788, 0, 0)
qtmud        INFO     mudsocket start()ed
qtmud        INFO     qtmud.run()ning
```

This means that qtMUD is up and running and you can connect to it as a client. The easiest way to do this is with telnet:

```
$ telnet localhost 5787
```

If everything is working, you should see output similar to this::

```
Trying ::1...
Connection failed: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

qtmud                0.0.6

Successfully connected to qtmud, press enter to continue login...
```

From there, you can log in (qtMUD has *very* basic user accounts) and play around with the default commands.

1.2 Installation

Todo: in-depth explanation of installation procedure for non-developers.

1.3 Configuration

```
[SERVERS]
IPv4_HOSTNAME = localhost
IPv6_HOSTNAME = localhost
IPv4_MUDPORT = 5787
IPv6_MUDPORT = 5788

[PICKLES]
CLIENT_ACCOUNTS_PICKLE = './client_accounts.p'
```

Todo: verbose explanation of the qtmud.conf file

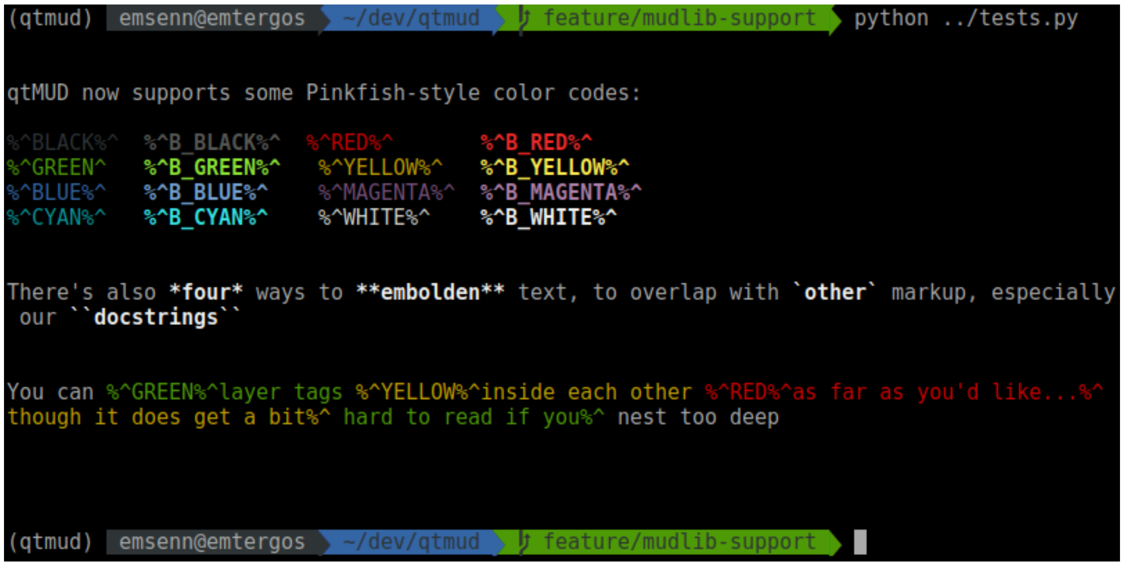
1.4 Client Features

Todo: explanation of what clients can do, and how they're represented on the backend.

1.4.1 Pinkfish Parser

New in version 0.0.10.

Players (and developers) can use Pinkfish-style tags to make up their text. Not too much to explain here. Enjoy abusing it on the talker.



```
(qtmud)  emsennn@emtergos  ~/dev/qtmud  feature/mudlib-support  python ../tests.py

qtMUD now supports some Pinkfish-style color codes:

%^BLACK%^   %^B_BLACK%^  %^RED%^       %^B_RED%^
%^GREEN^    %^B_GREEN%^   %^YELLOW%^   %^B_YELLOW%^
%^BLUE%^    %^B_BLUE%^    %^MAGENTA%^  %^B_MAGENTA%^
%^CYAN%^    %^B_CYAN%^    %^WHITE%^    %^B_WHITE%^


There's also *four* ways to **embolden** text, to overlap with `other` markup, especially
 our ``docstrings``


You can %^GREEN%^layer tags %^YELLOW%^inside each other %^RED%^as far as you'd like...%^
though it does get a bit%^ hard to read if you%^ nest too deep



(qtmud)  emsennn@emtergos  ~/dev/qtmud  feature/mudlib-support 
```

CHAPTER 2

Developing with qtMUD

qtMUD is meant to be used as a framework for developing your own MUD. This document contains an explanation of how qtMUD works, as well as detailed explanations of our development process and some tutorials on how to expand the library.

2.1 Getting Started

2.1.1 Download Source

The easiest way to get the source code for qtMUD is by cloning the GitHub repository:

```
$ git clone https://github.com/emsenn/qtmud.git
```

This will create a new folder with

2.1.2 How it Works

The best way to explain how qtMUD functions is probably by looking, in detail, at what happens when you execute `./bin/qtmud_run`:

```
qtmud_run preparing to start qtmud 0.0.6
qtmud        WARNING  No valid config found, using default values
```

If `qtmud_run` isn't passed the `--conf` argument (`qtmud_run --conf ./fireside.conf`), it will look in *./qtmud.conf*, *./.qtmud.conf*, *~/qtmud.conf*, and *~/.qtmud.conf*, in that order. If it can't find one there, it will rely on defaults set in the `qtmud` module.:

```
qtmud        INFO     qtmud.load() called
qtmud        INFO     adding qtmud.subscriptions to qtmud.subscribers
qtmud        INFO     adding qtmud.services to qtmud.active_services
qtmud        INFO     qtmud.load()ed
```

`qtmud.load()` populates:

```
qtmud         INFO     filling qtmud.client_accounts from ./qtmud_client_accounts.p
qtmud         INFO     start()ing active_services
qtmud         INFO     talker start()ed
qtmud         INFO     start()ing MUDSocket
qtmud         INFO     MUDSocket successfully bound to ('localhost', 5787)
qtmud         INFO     MUDSocket successfully bound to ('localhost', 5788, 0, 0)
qtmud         INFO     mudsocket start()ed
qtmud         INFO     qtmud.run()ning
```

2.2 Development Flow

This section is intended to be an outline of the steps that I take to bring a part of qtMUD from idea to release. Each section is, roughly, broken up by the console command used to handle the task.

2.2.1 GitHub Issues

Todo: Coming by 0.1.0!

2.2.2 Flow Feature Start

```
$ git flow feature start womble
```

This creates a new git branch, `feature/womble`, based on the `develop` branch. Once you're on the feature branch, you can make whatever changes to the code you want. Let's pretend I added a womble method to `qtmud.cmds`. Once I'm done writing it, I'll want to go ahead and run:

```
$ python setup.py test
```

Even though I haven't written any test cases for my code (yet), it's still a good idea to check and make sure your changes didn't inadvertently break something existing. Once you're sure your code doesn't hurt existing functionality, you can go ahead and commit the changes.

2.2.3 Git Commit

```
action: [audience:] Summary Of Commit [!tag]
```

In order to use git-changelog qtMUD commit messages have a specific format:

- `action` represents what purpose the commit serves and is one of:
- – `chg` - You've refactored existing code, removed technical debt, or expanded test coverage
- – `fix` - You've fixed an issue. Ideally, prove the GitHub Issue # in the summary.
- – `new` - You've added new features, documentation, or test cases.

- `audience` represents who would care about the change

- - `dev` - API revisions/additions, refactors
- - `usr` - Client experience revisions/additions
- - `pkg` - Packaging/metadata revisions/additions
- - `test` - Test case revisions/additions
- - `doc` - documentation revisions/additions

- `Summary Of Commit` is a short (very short) summary of what the commit does. If you find yourself struggling to come up with this, you probably should be committing more frequently. Capitalize every word, unless it references a part of the code which is not capitalized.

- Each `tag` should be prefixed with an exclamation point, and should be one of:
- - `!refactor` marks that the commit deals with existing code.
- - `!minor` marks that the commit has a very minor change - adding a one-line comment or fixing a typo.
- - `!cosmetic` marks a commit as dealing with code practice adhesion - fixing pep8 violations or other small stuff
- - `!wip` marks a commit as being a work-in-progress - the committed code is functional, but doesn't contain all of the eventual changes the function will require.

If it wasn't clear from the tags, commits should have a single thing that they deal with. My commit for my new `womble` command might look something like this:

```
$ git commit -m "new: usr: Addition Of womble To qtmud.cmds !wip"
```

Once you've committed the functional code, it's time to test it.

2.2.4 Unittests

```
from unittest import TestCase
import qtmud

class TestWomble(TestCase):
def test_Womble(self):
    cmd = cmds.womble
    client = qtmud.Client()
    self.assertTrue(cmd(client))
```

qtMUD uses unittest for testing. For a more comprehensive guide on how to write qtMUD unittests, check the respective tutorials for commands, subscriptions, and services.

I use coverage to make sure my test cases are comprehensive, and strongly suggest you do the same:

```
$ coverage run setup.py test
$ coverage report
$ coverage annotate qtmud/cmds.py
```

I change and expand my test cases until the things I changed in this feature are properly covered. Once my test cases are built, I'll want to go ahead and do another commit:

```
$ git commit -m "new: test: Test Cases For qtmud.cmds.womble !wip"
```

2.2.5 Pylint

```
$ pylint -rn ./qtmud/cmds.py ./tests/test_cmds.py
```

Now that the `womble` command has been written, and we have tests to verify it works, it's time to make sure the addition follows good Python practices.

2.2.6 Documentation

D

```
$ cd ./docs && make html
```

After you've linted your code, build the html documentation (goes into ./docs/build/html by default) and look for any errors caused by your docstrings.

2.2.7 Flow Feature Finish

```
$ git flow feature finish womble
```

2.2.8 Flow Release Start

```
$ git flow release start 0.0.4
$ gitchangelog > ./docs/source/changelog.rst
$ cd ./docs && make html && cd ..
```

Also, manually (for now) change the version in these three locations: * `docs/conf.py` * `qtmud/__init__.py` * `setup.py`

2.2.9 Flow Release Finish

```
$ git flow release finish 0.0.4
$ python setup.py sdist bdist_wheel upload
$ git push --all
```

The end!

2.3 Making & Using Things

Todo: Coming by version 0.1.0!

2.4 Making & Using Subscriptions

Todo: Coming by version 0.1.0!

2.5 Making & Using Services

2.6 Changelog

2.6.1 Changelog

%%version%% (unreleased)

- Merge branch ‘release/0.0.9’ into develop. [emsenn]

0.0.9 (2016-10-04)

New

- Talker -t And talker -d To Tune & Drop Talker Channels. [emsenn]
- Whatami Command, Tells Client Their Class. [emsenn]
- Added qtmud.services.Talker.replace_client_object(client, object) [emsenn]
- Connected Clients Informed Of New Connections. [emsenn]
- Addition Of *tell* Client Command. [emsenn]

Changes

- Documentation & Linting of ./qtmud/__init__.py. [emsenn]

Fix

- Fixed Issue #30. [emsenn]

  Clients are now added to `qtmud.connected_clients` after they get their password right.
- Fixed Issue #31, Multiword Tells Now Work. [emsenn]
- Fixed Clients Tuning In Repeatedly To Talker Channels. [emsenn]

  Also bolded some output and stopped making log directories if one wasn’t given.

Other

- Merge branch 'feature/mudlib-support' into develop. [emsenn]
- Merge branch 'release/0.0.8' into develop. [emsenn]
- Merge branch 'release/0.0.8' [emsenn]

0.0.8 (2016-10-02)

Changes

- Added Development Flow to Development Docs. [emsenn]
- Documented Parameters in `qtmud.cmds` [emsenn]
- Added Tests for qtmud.cmds. [emsenn]

  Also preparing to use gitchangelog to aid with versioning moving forward.

Other

- Merge branch 'master' into develop. [emsenn]
- Cleaner Shutdown/Startup. [emsenn]

  Added –logdir and –datadir options to *./bin/qtmud_run*

  Also fixed bugs #21 and #23
- Codecov YML Addition. [emsenn]
- Merge branch 'master' of github.com:emsenn/qtmud. [emsenn]
- Update .travis.yml. [emsenn]
- Update .travis.yml. [emsenn]
- Start of qtmud.cmds Test Cases. [emsenn]
- Added Code Climate Integration to Travis CI. [emsenn]
- Merge branch 'master' of github.com:emsenn/qtmud. [emsenn]
- Delete environment.pickle. [emsenn]
- Merge branch 'release/0.0.7' [emsenn]
- Merge branch 'release/0.0.6' [emsenn]

  First real package, I think? I hope. I'm sorry to anyone reading this from the future, I am a newbie.
- Merge branch 'release/0.0.7' into develop. [emsenn]

0.0.7 (2016-10-01)

- V0.0.7: Doc Refactor & client_disconnect Fix. [emsenn]
- Doc Refactoring, Fixed client_disconnect Bug. [emsenn]
- Merge branch 'release/0.0.6' into develop. [emsenn]

0.0.6 (2016-10-01)

- First “Good” Packaging Release. [emsenn]
- Merge branch ‘master’ into release/0.0.6. [emsenn]

  I messed up the git flow whoops
- Some random cleanups. [emsenn]
- Documentation Update - Removed mudlib docs. [emsenn]
- Fixed Broken MUDSocket Test. [emsenn]
- Update README.md. [emsenn]
- Update README.md. [emsenn]
- Update README.md. [emsenn]
- Update README.md. [emsenn]
- Update README.md. [emsenn]
- Update README.md. [emsenn]
- Added CodeClimate Badge to README.md. [emsenn]
- Transition to Package. [emsenn]

  Shifting qtMUD into being an acceptable Python package.
- Code Coverage? [emsenn]
- Fixed ./data/ not being in git. [emsenn]
- More Cards. [emsenn]

  And a few other trivial changes.
- More Fireside Cards. [emsenn]

  Cleaning up of Fireside code, too
- Require sudo in Travis CI. [emsenn]

  Travis-CI doesn’t like that we require brlapi, and while I could just remove the requirement, we are going to need it eventually.
- Added requirements.txt. [emsenn]
- Merge branch ‘master’ of github.com:emsenn/qtmud. [emsenn]
- Update README.md. [emsenn]
- Travis CI Scripts. [emsenn]

  Simple tests for Travis CI? Maybe? I don’t get how it works.
- Fireside Documentation. [emsenn]

  Built and added Fireside documentation.
- Attempted Fix for ReadTheDocs error. [emsenn]

  error was Could not import extension sphinx.ext.githubpages (exception: No module named githubpages)

  this is what google said would help
- PEP8 Updates & Fireside Cards. [emsenn]

- Fireside Mudlib. [emsenn]

  Simple cardgame mudlib and some edits to qtmud methods
- Basic Talker Service. [emsenn]

  A really basic and lazy implementation of a talker service.
- Reduction of Dependence on Starhopper. [emsenn]

  qtMUD, through refactoring, became dependent on Starhopper methods.

  This fixes some of that./
- Documentation Hotfix Part Three. [emsenn]
- Merge branch 'master' of github.com:emsenn/qtmud. [emsenn]
- Merge pull request #16 from emsenn/develop. [emsenn]

  Develop
- Merge pull request #14 from emsenn/develop. [emsenn]

  Develop
- Merge branch 'release/0.0.4' [emsenn]
- Merge branch 'release/0.0.3' [emsenn]

  Release of version 0.0.3 to master woooo
- Merge branch 'release/0.0.4' into develop. [emsenn]

0.0.4 (2016-09-26)

- Bump to version 0.0.4. [emsenn]
- .gitignore hotfix. [emsenn]
- Documentation Hotfix. [emsenn]
- Documentation Cleanup, Separating Client and Ship in Starhopper. [emsenn]

  Title about says it all.
- Mirrored Starhopper Structure in Qtmud. [emsenn]

  Updated qtmud to use a package structure more in line with the updated starhopper structure.
- Bringing Back Documentation. [emsenn]

  It's back! and less messy than ever!
- Deleted Broken Documentation, Refactored Starhopper. [emsenn]

  The documentation wasn't rendering right so I just got rid of it.

  also, refactored starhopper. Need to shuffle qtmud to match, unfortunately. New system is way better, though.
- Merge branch 'feature/diceroller' into develop. [emsenn]

  Got a little carried away with this feature
- Starhopper Update. [emsenn]

  Got frustrated with trying to buy a full MMORPG in one go so made a dinky little space adventure game.

- Migration to Game Library. [emsenn]

  I realized a lot of stuff was in qtmud that was better suited for the specific libraries - not every game that gets built is going to want a "say" command, for instance.

- Refactor. [emsenn]

  It finally clicked with me what people were saying about organizing the engine differently, so this is me shuffling around toward doing that.

  A lot of functionality is broken but I like the new direction.

- Changed Thing's search methods, restructured lib. [emsenn]

  I know it looks like a lot of changes but it's really not much.

- Swordsmanship, Healthful, Acting Qualities in Lib. [emsenn]

  A few qualities to make use of the diceroller.

  Not pleased with any of this code but it's better than nothing.

- Merge branch 'feature/noise' into develop. [emsenn]

  Noises basically work, even if their trigger mechanism is a bit simple.

- Failed to add changes to last commit. [emsenn]

  Whooops!

- Fixed Issue #9 & Added Documentation Theme. [emsenn]

  Fixed Issue #9, where clients weren't removed from their location when they disconnect.

  Also, added cute little Tumblebeasts to the documentation!

- Additions to Library: Ye Olde Tavern. [emsenn]

  made ye olde tavern less of a filler thing and more of a real thing.

- Documentation for Noisemaker. [emsenn]

- Noisy quality, Noisemaker service. [emsenn]

  Noisy things randomly send messages to things in their environment through the Noisemaker service.

  This is a rough draft and probably hella buggy, and also has like NO documentation, but hey, it's progress.

- Learning, Teaching Qualities. [emsenn]

  Learning quality which lets things use learn from qualities with the

  Teaching quality which adds qualities in the teacher's teachable_qualities to the learner.

- Merge branch 'release/0.0.3' into develop. [emsenn]

  NLTK-based parser, Prehensile, Hearing Qualities, Sender service

0.0.3 (2016-09-16)

- Missed adding updated __init__.py. [emsenn]

  Forgot to add this to the last commit ffs

- Bumping things up to version 0.0.3. [emsenn]

  Note to self: remember to rebuild documentation during *this* part of the release process, not when closing a feature branch.

- Merge branch ‘feature/textblob’ into develop. [emsenn]

  Fancier parsing, more qualities, expanded library.

- Documentation Update. [emsenn]

  Rebuilt the Sphinx autodocumentation.

- Prehensile Quality, Hearing Quality. [emsenn]

  Fixed adjectives, added a Prehensile quality that lets Things with it ‘take’ objects, which moves them from where they are into the contents of the prehensile thing.

  Also added the Hearing quality, which lets things listen. Added the sounds string to Renderable quality.

- Sender Service, Fixing Commands. [emsenn]

  A lot of commands broke when I set up the new parser, this fixes a fair chunk of them, but certainly not all.

  I also created the Sender service, which does basically what the Renderer service does. Leaving the Renderer service for now, because it will probably be used to format scenes (which maybe should be called frames lol) for users.

- Implementing Natural Language Toolkit. [emsenn]

  Changed qtmud.services.Parser to have the parse_line() function, which uses the TextBlob package to do some basic parts of speech tagging on player lines, to try and suss out what things the player might be talking about.

  It’s functional in this commit, but uncommented and not fully implemented. Check the Sighted quality’s look() method for an example usage.

- Merge branch ‘feature/nametags’ into develop. [emsenn]

  The basic nametags code is finished. There’s probably some parts of the code which don’t use it, though, so be careful.

- Applicative Fix. [emsenn]

  After talking with a friend and having the difference between applicative and imperative methods explained, made some changes to make the applicative methods more, well, applicative. Also fixed some older lines that were outdated but not throwing errors.

- Thing.search(‘target’) method. [emsenn]

  Added a simple method for looking around a thing’s potential environment to find a match for ‘target’, intended to be a nametag’

- Library Expansion. [emsenn]

  Lots of MUDs let you ‘look at <thing in room>’, even if that thing isn’t a real “item”, something you can interact with. A cobblestone road might let you ‘look cobbles’, for example, even though you can’t do anything beside look at the cobbles. Normally this requires a weird archaic syntax to work.

  because of the granular nature of qualities, these fake-but-still-observable items are easy to make, by making a new item and applying the “Renderable” quality to it.

  The downside is that this means a new thing is instanced for every lookable thing in every room, which could cause memory problems down the line.

  However, I think the extensibility and power this gives the engine is way worth that potential cost. Normally it’s a big commitment in MUD development to move a thing from a lookable to a genuine item - normally a complete rewrite. In this case, however, it’s as simple as lookable.add_quality(Physical).

- Better Nametags Documentation. [emsenn]

  Added some notes on how to use nametags

- Implemented Nametags. [emsenn]

  Nametags are a new thing-level attribute, and are used to find a thing if you only have some names it might respond to. (For example a client has the nametags 'client', 'player', 'thing', and their name (if they've set one).

  I also added a __setattr__ function to qtmud.Thing, so that qtmud.qualities.Renderable. Essentially, if a thing has a set_attr() function, the thing will use that when attr is being set, rather than the type default.

- "inventory" Command & Method in Container Quality. [emsenn]

  Added the inventory() method to the Container quality, and changed its apply() method to add the 'inventory' command to that container if it is also Commandable.

- Never Forget Holiday Update. [emsenn]

  qtmud's first holiday update! Added a memorial to commemorate September 11th. Also modified the look command so that people can actually look at the memorial.

- Merge branch 'master' of github.com:emsenn/qtmud into develop. [emsenn]

- Merge branch 'develop' [emsenn]

  Documentation hotfix

- Merge branch 'release/0.0.2' [emsenn]

  continued shifting of core functions, establishment of real documentation using Sphinx, and starting to solidify library-building API.

- Merge branch 'release/0.0.1' [emsenn]

  First release version, though I use that term very loosely. It should run, and the documentation should explain what the code does, but don't expect anything close to gameplay.

- Merge branch 'feature/renderer' into develop. [emsenn]

  Set up a renderer service, among other small changes

- Documentation Update. [emsenn]

  Just some documentation expansion before bed.

- Cleaning up Qualities. [emsenn]

  The last commit rolled out changes to the command backend pretty quickly. This commit cleans a lot of that up, and expands the new Sphinx-friendly docstrings through more of the code.

- Added Scene Rendering. [emsenn]

  Created a new service at qtmud.services.renderer to handle scheduled events for sending information to clients. This makes sure clients aren't getting messages too soon - such as building a room description with 'look' for a room the player just left.

  Currently, only the 'look' function in the Sighted quality makes use of render. Other places where things are currently being sent through the Client's send() function will be fixed in later commits.

- Documentation Hotfix. [emsenn]

  Documentation wasn't linking to source properly, reworked the configuration files so it would.

- Merge branch 'release/0.0.2' into develop. [emsenn]

0.0.2 (2016-09-10)

- Bump to Version 0.0.2. [emsenn]

  bumped version number everywhere it occurs. (i think)

- Addition of Sphinx-Generated Documentation. [emsenn]

  shuffled documentation around, in part so the repo should (I think) work with Github Pages. Even if it doesn't, it's a better presentation of the information within the repo.

- Parser & Breaking Up Qualities. [emsenn]

  rewrote qtmud.services.parser.Parser to look for commands in a thing's commands attribute, and for the command's functions to live in the quality that gives them.

  This meant breaking up the qualities from qualities/__init__.py into individual files.

  I also started documenting things using Sphinx markup. The configuration files and such have been added to the repo. Going to try and build it as our github pages after this commit.

- Merge branch 'feature/environments' into develop. [emsenn]

  A super-simple way of handling things having locations.

- Just Some Comments. [emsenn]

- Rough Environments. [emsenn]

  Clients can now 'look' and 'go' between rooms. Everything is real rough but I'm probably taking a break from this code binge so wanted to get it committed. It's functional, at least if you don't try to do anything outside of documented syntax.

0.0.1 (2016-09-07)

- Merge branch 'feature/organizing' into develop. [emsenn]

  Finished writing a base I think can be built up from, so closing this feature to open ones for specific additions.

- Updated Documentation & Mild Cleanup. [emsenn]

  Mostly just added documentation and cleaned up a few lines, to take it from "rough idea" to "workable base".

  Also to play with git flow a bit tbh

- Start of Environments & Movement. [emsenn]

  There is now a Mover service which listens for 'move' events.

  It works against the Room, Container, and Physical Qualities:

  Container - Give a thing contents, a list Room - If a thing doesn't have contents, give it contents

  > (this'll probably be fleshed out more to have code for in-built exits/entrances, which is why I went ahead and did it separate from Container.)

  Physical - gives attributes for name, description, and location.

  Now when a client logs in, their associated thing is given the Client and Physical qualities, leaving them with connection information, a name (for now a synonym for their identity), and moved into qtmud.manager.back_room, a lazy little hack to give incoming clients someplace to be until there's proper login.

  I also added the whereami command so users can find the name of their location.

- Start of Documentation & Say Command. [emsenn]

  Added some linese of documentation in case I put the project down for a couple years and don't want to be completely lost when I come back.

  Also added a super basic say command, mostly so there's something to play with during the next step, adding physical and container qualities

- Basic Schedule Service. [emsenn]

  I haven't fully tested it but qtmud.manager.tick() should call to every service, and pass on any 'events' that the service 'subscribed' to.

  All I've tested was gettinng it so the Parser service could intercept incoming client commands and, well, parse them. Seems to work, but I'm sure there's at least a dozen things awfully wrong in it.

- Basic MUDSocket Server. [emsenn]

  a super-basic attempt at a socket server for mud clients (telnet).

  also a few jabs toward implementing a basic schedule caller. doesn't do anything yet, but doesn't get in the way.

  next is writing a basic parser and tying it into the scheduler

- Rough Outline. [emsenn]

  This is more of a rough outline of how the engine might be structured.

  It's going to build up Things() with Qualities(), and those will be the user and objects around them.

  Going to set up a subscription-based central manager for issuing game updates.

- Initial Commit. [emsenn]

  First commit just to set up the git repository.

- Initial commit. [emsenn]

CHAPTER 3

qtMUD API

This is the complete API for qtMUD

3.1 qtmud module

3.1.1 qtmud.cmds module

3.1.2 qtmud.services module

3.1.3 qtmud.subscriptions module

3.1.4 qtmud.txt module

3.2 Glossary

MUD Multi-User Dimension. A more historic term for MMORPG, MUDs were some of the earliest multiplayer games. Wikipedia has a pretty comprehensive article on them.

MUD client A specialized client for playing MUDs. (duh)

MUD engine Also known as drivers, MUD engines are game engines meant for presenting *MUDs*. There have traditionally been two main classes of MUD engine: DIKU and LPC. DIKU MUDs came first, and used database-driven logic to present a simplistic hack-and-slash environment to the player. LPC MUDs used object-oriented programming techniques to render more detailed game worlds to the player. qtmud follows in the tradition of the latter.

mudlib short for Multi-User Dimension Library, a mudlib is a game written for a *MUD engine* to run. From qtmud's perspective, a mudlib is a Python package which is heavily reliant on the `qtmud` module.

References:

- genindex

- modindex
- search

---

qtMUD is a **Python 3** package for developing and managing a multi-user dimension, or *MUD*, written by emsenn and released under the Unlicense. It is currently in **early alpha**.

This documentation is intended to serve as documentation for those administrating or using a qtMUD server. The official version of this documentation is hosted at https://qtmud.readthedocs.io/en/latest/.

MUDs are a form of multi-player game where players receive verbose text descriptions of their environments and the activity around them, and interact by typing out commands for what they wish to do, such as `survey planet` or `fight dragon`.

By providing a limited set of features and focusing on documentation, qtMUD aims to be the non-programmer's way to create and run a MUD.

Note: qtMUD is just the core server for running a MUD, and is meant to be used with a MUD library, like Fyreside.

Index