# GoRogue Documentation

***Release 0.0.1***

**Matthew Fisher**

October 22, 2014

# Contents

GoRogue is a roguelike clone built in Go.

If you are new to GoRogue, you should start with the basic *Usage* of the project. Interested in contributing to the GoRogue project? Check out our *Contributing Documentation*.

CHAPTER 1

# Community

GoRogue is a fully open source project. As such, the GoRogue community consists of anyone who uses the GoRogue software and participates in its evolution, whether by answering questions, finding bugs, suggesting enhancements, or writing documentation or code.

GoRogue development is coordinated through its project on GitHub. Anyone can check out the source code for GoRogue, fork the project, make improvements, and create a pull request to incorporate those changes into GoRogue.

GoRogue uses the timeless, highly efficient, and totally unfair system known as “Benevolent Dictator for Life” (BDFL). Matthew Fisher, the creator of GoRogue, is our BDFL and has final say over all decisions related to GoRogue.

CHAPTER 2

# Conduct

The GoRogue community welcomes and encourages participation by everyone. We welcome contributions from everyone as long as they interact constructively with our community.

As the developer community continues to grow, it is inevitable that disagreements and conflicts will arise. We ask that participants conduct themselves according to this very clear, simple concept:

1. **Don't be a dick.**

CHAPTER 3

# Contributing Standards

GoRogue is a Go project. We chose Go over other compelling languages because it is well-documented and friendly to a large number of developers. Source code benefits from many eyes upon it.

While this is a Go project, we follow the teachings of The Zen of Python, which emphasizes simple over clever, and we agree. Readability counts. GoRogue also aims for complete test coverage.

Contributors to GoRogue should feel welcome to make changes to any part of the codebase, as well as throw any issues on our bug tracker. To create a proper GitHub pull request for inclusion into the official repository, your code must pass `make test`.

## 3.1 Testing

GoRogue adheres to the testing documentation followed in most Go projects. We also add coverage reports to our tests to ensure that our code is well-tested.

## 3.2 Merge Approval

GoRogue maintainers add "LGTM" (Looks Good To Me) in code review comments to indicate that a pull request is acceptable. Any code change requires one of GoRogue's maintainers to accept the change in this manner before it can be merged.

## 3.3 Commit Style Guide

There are several reasons why we try to follow a specific style guide for commits:

- it allows us to generate CHANGELOG.md by script
- it allows us to recognize unimportant commits like formatting
- it provides better information when browsing the git history

### 3.3.1 Generating CHANGELOG.md

We use these three sections in the changelog: new features, bug fixes, and breaking changes. This list could be generated by script when doing a release, along with links to related commits.

To list all subjects (first lines in commit message) since last release:

```
$ git log <last tag> HEAD --pretty=format:%s
```

To list all new features in this release:

```
$ git log <last release> HEAD --grep feature
```

### 3.3.2 Recognizing Unimportant Commits

These commits are usually just formatting changes like adding/removing spaces/empty lines, fixing indentation, or adding comments. So when you are looking for some change in the logic, you can ignore these commits - there's no logic change inside this commit.

When bisecting, you can ignore these by running:

```
$ git bisect skip $(git rev-list --grep irrelevant <commit ID> HEAD)
```

### 3.3.3 Providing more Information when Browsing the History

This adds extra context to our commit logs. Look at these messages (taken from the last few AngularJS commits):

- Fix small typo in docs widget (tutorial instructions)
- Fix test for scenario.Application - should remove old iframe
- docs - various doc fixes
- docs - stripping extra new lines
- Replaced double line break with single when text is fetched from Google
- Added support for properties in documentation

All of these messages try to specify where the change occurs, but they don't share any convention. Now look at these messages:

- fix comment stripping
- fixing broken links
- Bit of refactoring
- Check whether links do exist and throw exception
- Fix sitemap include (to work on case sensitive linux)

Are you able to guess what's inside each commit diff?

It's true that you can find this information by checking which files had been changed, but that's slow. When looking in the git history, we can see that all of the developers are trying to specify where the change takes place, but the message is missing a convention. Cue commit message formatting entrance stage left.

### 3.3.4 Format of the Commit Message

```
{type}({scope}): {subject}
<BLANK LINE>
{body}
<BLANK LINE>
{footer}
```

Any line of the commit message cannot be longer than 90 characters, with the subject line limited to 70 characters. This allows the message to be easier to read on github as well as in various git tools.

#### Subject Line

The subject line contains a succinct description of the change to the logic.

The allowed {types} are as follows:

- feat -> feature
- fix -> bug fix
- docs -> documentation
- style -> formatting
- refactor
- test -> adding missing tests
- chore -> maintenance

The {scope} can be anything specifying place of the commit change e.g. the controller, the client, the logger, etc.

The {subject} needs to use imperative, present tense: "change", not "changed" nor "changes". The first letter should not be capitalized, and there is no dot (.) at the end.

#### Message Body

Just like the {subject}, the message {body} needs to be in the present tense, and includes the motivation for the change, as well as a contrast with the previous behavior.

#### Message Footer

All breaking changes need to be mentioned in the footer with the description of the change, the justification behind the change and any migration notes required. For example:

```
BREAKING CHANGE: this relies on a new screen resolution library that is in active
    development, well tested and maintained. Because it introduces a new dependency,
    users will have to run `make && make install` again to receive this change.
```

#### Referencing Issues

Closed bugs should be listed on a separate line in the footer prefixed with the "closes" keyword like this:

```
closes #123
```

Or in the case of multiple issues:

```
closes #123, #456, #789
```

### 3.3.5 Examples

```
feat(screen): add support for 2880x1800 resolutions

This adds support for the 2880x1800 resolution, which is the default for Macbook
Pro Retina Late 2013 model.

closes #123

BREAKING CHANGE: this relies on a new screen resolution library that is in active
    development, well tested and maintained. Because it introduces a new dependency,
    users will have to run `make && make install` again to receive this change.
-------------------------------------------------------------------------------------
test(client): check if monster dies after HP <= 0

GoRogue does not currently test if a monster dies when their health drops to or below
zero. This commit adds that functionality to the test suite.

closes #392
```

CHAPTER 4

# Release Checklist

These instructions are to assist the GoRogue maintainers with creating a new product release. Please keep this document up-to-date with any changes in this process.

- Create the next milestone
- Move any open issues from the current release to the next milestone
- Close the current milestone
- **Commit and push the bacongobbler/gorogue release and tag**
  - `git commit -a -m 'bump to vX.Y.Z'`
  - `git push origin master`
  - `git tag vX.Y.Z`
  - `git push --tags origin vX.Y.Z`
- update version fields in packages to *next* version

## 4.1 Documentation

- Docs are automatically published to http://gorogue.readthedocs.org
- **Log in to the http://gorogue.readthedocs.org admin**
  - add the current release to the list of published builds
  - rebuild all published versions so their "Versions" index links are updated
- **Generate release notes**
  - follow the format of previous release notes
  - summarize all work done since the previous release
  - include "what's next" and "future directions" sections

# Usage

Once you have cloned the repository:

```
$ cd gorogue
$ make install
$ gorogue
```

Alternatively, you can just build from master:

```
$ go get github.com/bacongobbler/gorogue
$ go install github.com/bacongobbler/gorogue
$ gorogue
```