B Maintenance Manual

The Maintenance Manual provides details of the software implementation. It gives an advanced overview of the system setup and installation and lists the main dependencies in Section B.1. A detailed guide to developing the software is given in Section B.2, including testing, documentation and suggested areas of further development.

To readers unfamiliar with Ruby, it is important to know that the term *gems* are Ruby software packages. The project is available online at this address: `https://github.com/vencha90/taxi-sim`. You might also have received a `tar` archive file of the software.

B.1 Installation of Development Environment

All commands in this section are supposed to be ran on your main operating system.

B.1.1 Installation Without Virtualisation

It is recommended to use virtualisation as described in User Manual in Appendix A.1.2. Virtualisation will ensure that only the correct required dependencies are installed and your operating system is not corrupted by potentially conflicting software. If you still do not wish to use virtualisation, there are alternatives for Unix-like systems.

You can install Ruby Version Manager (RVM) (Papis and Seguin 2014) from `http://rvm.io/` and then install Ruby by running `rvm install 2.1.1` in terminal. You also need to install Bundler by running `gem install bundler && bundle install --local` to manage dependencies.

Instead of RVM you could use `rbenv` (Stephenson 2014). Unfortunately RVM and rbenv are mutually exclusive and could cause conflicts with other parts of an OS. The worst option is installing Ruby completely natively as this provides no isolation whatsoever from other OS components.

B.1.2 Developer Setup

Even if you have the source code, you should use the latest version-controlled software from the official code repository at `https://github.com/vencha90/taxi-sim`.

To acquire the source code for development, please follow the instructions below. Commands that should be used in command line terminal are marked *(cmd)*, commands that require downloading software from the world wide web are marked *(web)*.

- *(web)* Install Git (Git 2014) version control from `http://git-scm.com/`.
- *(cmd)* Open a command line terminal.
- *(cmd)* Clone source code to a suitable directory on your local hard drive using this command: `git clone git@github.com:vencha90/taxi-sim.git`
- *(cmd)* Change the working directory to the project's source code: `cd taxi-sim`.

B.1.3 Dependencies

Top-level gems are listed in `Gemfile` and explained below. Some gems are grouped together and labelled as for development and testing, meaning that they are not necessary for simply using the

program. Potentially these labels could be used to release two separate software versions – standard and developer. The full list of dependencies is listed in `Gemfile.lock` and not explained as they only play a supporting role.

`require_all` (Pertman 2013) eases source code management by allows to require all source files in a directory on a single line in an OS- agnostic way (not supported by Ruby natively).

`plexus` (community 2014) is a graphing gem. It is discussed in more detail in Section 5.1.3

`thread_safe, '0.3.1'` is actually a dependency for other top-level gems, but was listed explicitly to specify a compatible release version (0.3.1).

`guard` and `guard-rspec` (Guillaume-Gentil 2014) can be used for monitoring file system events. A recommended work flow is suggested in Section B.2.1

`rake` (Weirich 2014) is a common tool used for Ruby command line applications.

`simplecov, ' > 0.7.1'` (Olszowka 2014) monitors code test coverage as discussed in Sections 4.3.3 and 5.2.1. The version number is specified due to a bug in the latest version.

`rspec` (Marston 2014) is a testing framework. It is discussed in Sections 4.3.3 and 5.2.1. Development with Rspec is discussed in Section B.2.1.

B.2 Developer's Guide

All commands in this section are supposed to be ran from a terminal in your development environment. This section explains how to work with the automated tests, where to find documentation and how to customise the software. Class and module names are used in this section, please see Section B.2.2 for reference.

B.2.1 Tests

Test-Driven Development (TDD) was followed as much as was reasonable when developing this project, and you are recommended to do the same. Please see Sections 4.1.1 and 5.2.1 to find out more about TDD.

Documentation and support is readily available online for Rspec. An example Rspec scenario was shown in Figure 3, but the simplistic language does not limit the functionality of the framework. It can even be easily extended if needed.

All Rspec scenarios can be run by entering `bundle exec rspec` in terminal, and help displayed by entering `bundle exec rspec -h`.

A typical developer's TDD work flow with Rspec would go as follows: write a test in text editor, switch context to command line, run the test in question (because running all tests would take a lot of time in a bigger codebase), switch context back...

This process can be automated by Guard, ordering it to monitor changes in test files and the linked code files. Guard can also be assigned conditions when the full test suite needs to be ran. It has already been configured for this project and can be started by entering `bundle exec guard`. Unfortunately this functionality is not available on virtual machines running on Windows. For virtual machines on other OSs, Guard and Vagrant needs configuration as shown here: https://github.com/guard/guard#-o--listen-on-option

lib/	Directory for source code files
spec/	Directory for Rspec test files
lib/file_parser.rb spec/file_parser_spec.rb	Input file processing – `FileParser` class
lib/graph/ spec/graph/	Road Network – `Graph` and `Graph::Vertex` classes
lib/logging.rb	Logging module – `Logging`
lib/passenger/ spec/passenger/	Passenger – `Passenger` and `Passenger::Characteristic` classes
lib/taxi/ spec/taxi/	Taxi representation, including the Q-Learner – `Taxi`, `Taxi::Action`, `Taxi::State` and `Taxi::Learner` classes
lib/taxi_learner.rb spec/taxi_learner_spec.rb	Main class and top-level namespace for the simulation – `Runner`
lib/world.rb spec/world_spec.rb	Environment – `World` class.

Table 4: Core Software Files

All tests are located in `specs/` directory. Please see Section B.2.2 for an overview of files and locations.

B.2.2 Documentation

Documentation for this software is in two forms. Table 4 in this section provides an overview of classes and modules with their related test files, and Table 5 lists the remainder of noteworthy files. Secondly, detailed documentation for each class is the automated Rspec tests. Additionally, Git commit history may prove useful for understanding the developer's intentions at the time of writing code.

B.2.3 Customising Output

Customised log details is one of the simplest changes to make to the system. It will not break automated tests if changes are made only to the output parameters, as tests only check that `any` output is produced.

Logs are currently being written from World, Taxi and Passenger classes and are handled by Logger module. To change the variables being written, you simply have to include them in `log_params` method.

Logger module can easily be included in other classes as needed, and variables to output can be specified just like in World, Taxi or Passenger classes.

B.2.4 Future Work

Some issues with the software were identified in Section 5.1.7. This section will give suggest how they could be fixed.

To transparently manage default values, they can be stored in a single `.yml` configuration file and retrieved when necessary. YAML parsing is already included in the system in FileParser class, so the

analysis.R	R language script for data analysis
bin/taxi_learner	Executable file for shell
.gitignore	File list for Git to ignore from version control
spec/fixtures/	Fixture files for automated tests
Guardfile	Specifications for Guard runtime
Gemfile Gemfile.lock	Bundler gem specifications
logs/	Log files from simulation – `simulation.log` and `summary.log`
Rakefile	Specifications for Rake tasks
.rspec spec/spec_helper.rb	Rspec configuration and support
Vagrantfile vagrant_manifest.pp	Vagrant configuration
.vagrant	Vagrant-created files for virtualisation
vendor/	Gems stored for offline usage of the software

Table 5: Supporting Software Files

hardest part of this task would be referencing the old variables to the new file. It is easier to identify the old variables, as they are currently stored as ALL_CAPS constants in source code files.

Fixing the taxi action sequence is aided by automated tests. However, this task will require serious refactoring as it would likely change the Taxi class significantly, and may need to change its subclasses as well.

Probably the easiest issue to fix is providing a progress indicator to users. This can easily be done by Runner or World classes that have access to the global time.