Team Study

Getting Started with Stendhal

COMP23311- Software Engineering I

University of Manchester

School of Computer Science

2018/2019

written by

Suzanne M. Embury

Contents

1 Introduction

By the time you meet your team, you should have access to the GitLab repository containing the code base that you will work on in the three coursework exercises for this semester. Your repository contains a version of the source code for Stendhal: a multi-player online adventure game. It is built on top of the Marauroa game engine that you worked with in the workshops in week 1. Although it is a game, Stendhal functions much like the kinds of software systems that keep industry and governments functioning. It is a multi-user, multi-threaded, client-server based system, that uses secure sockets to communicate over the Web. It also implements many business rules, and must manage stored data using a database. It therefore provides us with a good training base for learning modern software engineering skills.

You can find lots more information about the code and the game on the main Stendhal website at:

```
stendhalgame.org
```

On this website, you'll find a development guide plus lots of end-user documentation and a wiki describing the world in which the game is set.

Your task for the team study sessions this week is to make sure that each team member has a local clone of your team's Git repository in their own filespace, accessible through your team's preferred IDE. Within your preferred IDE, you will each need to:

- Acquire a local copy of the code (by cloning your team's Stendhal repository from GitLab).
- Figure out how to build the Stendhal components (both client and server).
- Figure out how to run the tests for Stendhal, and how to see the results.
- Figure out how to run the code for Stendhal on your local machine.

This document will guide you through the steps needed to achieve all of these goals.

2 Choosing an IDE

Probably the first thing your team needs to do is agree what IDE you will use. Within the School, we have access to versions of IntelliJ IDEA, NetBeans and Eclipse. You are welcome to use any IDE for your team, but you must use a proper IDE. Glorified text editors are not

accepted for this coursework. This is not because IDEs are always better for all tasks. The aim is to ensure that you all have experience of working in depth with at least one IDE, so that you can understand the strengths and weaknesses, compared to other, simpler development environments.

It is recommended that you choose a single IDE that the whole team uses. It is perfectly possible for different team members to use different IDEs, and we don't disallow this. But it adds an extra layer of complexity to the process of shared coding, and requires a certain level of confidence with Git and the IDEs in question to make work. Since you'll be facing plenty of other technical and team work challenges in this coursework, you may wish to avoid adding extra, unnecessary, complications from your team's workflow. All agreeing to use the same IDE is one way to do this.

As mentioned elsewhere, although teams are welcome to choose any of the standard IDEs, we are only able to provide technical support for Eclipse Oxygen at this time. We'll do our best to help teams using other IDEs, but can't guarantee we'll be able to fix the problems in the timescales needed by the coursework deadlines.

3 Acquiring the Code

We have set up a repository for each team on the school's GitLab server. This contains a close copy of the actual Stendhal code base. (Each year, we make some changes to make the code base more suitable for the work you are going to be doing with it. The extent and nature of the changes will be different every year.)

You will have been e-mailed with a link to your team's repository when it was created. The first step is for each team member to make a local clone of this repository in their own account. You might want to do this as a team exercise, initially, with team members grouped around a single machine. **But all of you will need to carry out these steps in your own filespace before you can get started on the coursework.**

To make a local clone, you should follow the same set of steps we used to create a Marauroa clone in the workshops in week 1. You can get the URI for your team's repository by logging onto GitLab, finding the Stendhal project in your Projects list and navigating to its main page. You'll be able to copy the URI from here.

You will need to select a branch to checkout when you create the clone. The Stendhal team use the `master` branch as their main development branch. We'll choose that as our starting point, too.

Use the HTTPS Protocol on School Machines

You should use the HTTPS protocol URI for GitLab repositories when working from School machines.

You may use the SSH protocol when working from your own machine/device, but within the School the version of Java we are using does not support large enough keys to satisfy GitLab's security requirements when connecting through Eclipse. So, the HTTPS protocol is the more reliable of the two.

To prevent repeatedly having to enter your username and password when using the HTTPS protocol, it is okay to ask for them to be stored by Eclipse. This option is presented whenever GitLab asks you to enter your GitLab credentials.

If you have difficulties in creating the clone from Eclipse, see the trouble shooting guide in our online Lab Help wiki:

- Index for trouble-shooting from error messages: `https://wiki.cs.manchester.ac.uk/index.php/LabHelp:Errors`
- Index for trouble-shooting from more general symptoms: `https://wiki.cs.manchester.ac.uk/index.php/LabHelp:Symptoms`

4 Building the Code

Since the version of Stendhal we are working with uses Java as the main implementation technology, the built-in Eclipse Java compiler will be able to compile the source files for you, without your needing to request it. (In fact, like most IDEs, Eclipse will automatically recompile source files after each small code change, so that errors can be highlighted in the source, while you are typing — a very useful facility.)

But, as we saw in the workshops in week 1, there is more to building a system than just compiling the Java classes. The Stendhal team use an Ant script to describe how to build their system. We showed you how to do this for the Marauroa code base in the workshops in week 1. You can apply the same approach to invoke Ant to build the Stendhal code base.

5 Running the Test Suite

There are two ways to run the test suite for Stendhal, and you should become familiar with both of them.

5.1 Running the Test Suite Through Ant

First, run the test suite using the Ant build file. This will set up everything needed to run the test suite (including the necessary database configuration files).

To do this, follow the same procedures we used in the week 1 workshop to run the Maraurao tests, using the Ant view within Eclipse. There are several targets that appear to be related to tests in the Stendhal Ant build file. The target you should use is the one called `test`. All other targets containing the word "test" are intermediate targets and you should not run them directly.

Notice that the process of compiling the code and running the full test suite takes a little while (as did cloning the repository). Depending on the load on the network, this can take anything between 3 and 10 minutes to run. With such a large code base and test suite, we have to factor compilation and test running times into the way we work, since they introduce a noticable delay.

5.2 Run the Tests Using the Eclipse JUnit Plugin

The second approach is to run the tests within Eclipse using the JUnit plug-in and a Run Configuration. This is a little more complicated than running the tests through JUnit in the GitLab Access Check activity, because we need to configure the path and the database. Instructions on how to do this can be found on the Stendhal wiki at:

`https://stendhalgame.org/wiki/Stendhal_on_Eclipse#Running_JUnit_Tests_in_Eclipse`

Be sure to follow the instructions to the letter, or you may find that the test suite does not run.

Run the Tests Using Ant First

If you decided to rub the tests through the JUnit plugin before running them through Ant, you may have been surprised to notice a significant number of errors cropping up, causing many of the tests to fail or be blocked. This is because you do not have the correct configuration files set up to be able to access the database that forms the back-end storage of the game. The Ant build script includes instructions to check whether the required configuration file exists, and to create it if it does not. This shows a advantage of an automated build tool like Ant, over a simple test harness like JUnit.

You should immediately notice some benefits to running the test suite through Eclipse with

a Run Configuration, compared to running the tests from Ant. On the console window, you will see the logger output from running the tests, which can be very helpful in tracking down the cause of failing tests. And the results of *all* the tests will be visible in the JUnit window, rather than having to load and examine them test class by test class.

Take a look at the results from executing the test suite through the Eclipse JUnit plugin. We've set up the repository so that all the tests should pass, so you should see a green bar in JUnit. However, as is not unusual in large complex code bases, there is one test that seems to be failing intermittently on some platforms. If this happens with your team, please report it to Suzanne.

Failing Tests in a Public Repository?

Although in an ideal world, no code versions would be committed to a public repository with failing tests, in practice this can be hard to achieve. It is actually very common for real test suites to contain a few tests that don't pass. This is because writing tests for some kinds of functionality can be tricky and expensive. Sometimes, the cheapest and easiest way to write a unit test isn't always the most reliable. Tests can be *unreliable*; that is, they can sometimes pass and sometimes fail, even when no changes have been made to the code under test. Tests can also be *brittle*. This means that they are highly sensitive to small changes in the environment in which the code runs. Dependencies of this kind can mean that failing tests sometimes indicate a problem with the way the test is written, rather than a problem with the production code being tested.

Brittle and unreliable tests are a problem because they mean that developers stop interpreting a red bar from a test suite execution as a problem they need to fix now. Ideally, we need to keep the test suite status green most of the time, so that we sit up and take notice whenever it turns red. If JUnit always (or often) gives a red bar, because of the brittle tests, then we lose this useful early warning signal when bugs enter the code. Later in the course unit, we'll look at ways to minimise the kinds of dependencies that cause problems of this kind. In the next team study session, we're going to try out a much simpler and very pragmatic, though imperfect, approach to dealing with these failing tests.

6 Running the Code

The final step is to run the code on your local machine. Note that **you should not run the public version of the Stendhal game**. Instead, you must run the version that is built from your local copy of the code, and that connects to your local copy of the game database. Otherwise, any changes you make to the code will not be visible in the code that we run.

To run your local Stendhal, you'll need to set up two new run configurations in Eclipse, in addition to the run configuration you set up for the JUnit tests earlier. One of these run configurations will start a Stendhal server running on your local machine, and the other will run a client on your local machine.

Help with doing both these things in Eclipse is available from the Stendhal wiki, at:

`https://stendhalgame.org/wiki/Stendhal_on_Eclipse#Start_StendhalServer`

`https://stendhalgame.org/wiki/Stendhal_on_Eclipse#Start_Stendhal_Client`

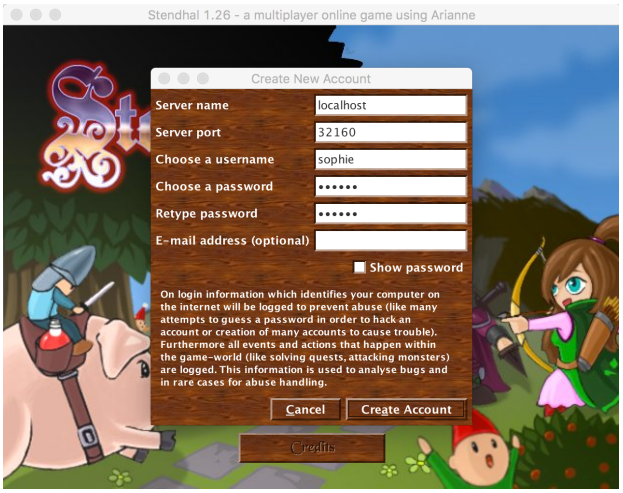
The process is straightforward if you follow the instructions carefully.

Run Configurations vs Debug Configurations

The instructions from the Stendhal team suggest you create Debug Configurations. Instead, we suggest for now you create Run Configurations. The only difference is that when we run the code using debug configuration, Eclipse will stop and launch the built-in debugger whenever a breakpoint is reached, while with a run configuration breakpoints are ignored. You shouldn't need to use the debugger for most of the work you do with Stendhal, so run configurations are the most appropriate. And you can always run the configurations you create as debug configurations later, should you need to.

7 Creating a Local Stendhal Account

When you have run both the server and the client, you will need to create an account on your server to use in manual testing. Use "localhost" as the name of the server to connect to (since you want to connect to the server running on the machine you are using, and not the public Stendhal game server). Keep the default port number that is already filled in on the form. You can choose your own username and password. These will be stored in the local database for your server, and will be available for you to use next time you log in. (Obviously, you should **not** use your university username and password for this purpose.)



When you login with this account, be sure to use `localhost` as the server address:

If you use the main Stendhal game server address, you will be connecting your local client to the public version of the server, running on a machine outside the University, and you won't see the effects of any changes you have made to your copy of the code.

Important Note for Stendhal Players

It is not necessary for this coursework to run Stendhal in true multiplayer mode, on a public server. Nor is it necessary for you to set up and host a private server. You should run a local test server on your machine and connect to that for all manual testing needed in this coursework. If you need to test a feature that requires the interaction between two players, you can run two copies of your client on your local machine, connecting both to your local server. (The good news about this is that you won't use a network at all, so you can run the game regardless of any network downtime or overloading.)

Finally, it is not necessary to play the Stendhal game to complete this coursework, any more than it would be necessary to take out house insurance when maintaining software for an insurance company, or to be x-rayed when working on software that controls medical scanning equipment. You'll need to run the game, and try out specific features, but later we'll look at special testing features embedded in the game to allow you to bypass the need for anything but the shortest trial sessions with the game.

8 Getting to Know the Code

When you can run the version of Stendhal that is in your repository, you can try out some code changes and see if you can see their effects when you run the code. Here are some examples you can try.

Can you work out how to make Hayunn Naratha, the first non-player character you meet in the game, give lots of experience points when you talk to him? Can you change his greeting?

How about changing the prices of the items being sold in Semos city—by Carmen, Margaret (in the Inn) and Xoderos (in the Blacksmiths near to Carmen), so everything is really cheap? To get to Semos from the game starting point, leave the guard house and follow the path down and left.

Can you change how many health points you get from eating or drinking the items sold by Margaret? Or make the player receive the health points faster?

A suggested approach to making these changes is:

1. Run the game to see how the component you are trying to change works. All the suggested changes are based around items and characters that you will encounter within the first few minutes of playing the game. Don't try to complete any quests or interactions with the non-player characters. Just focus on seeing the specific functionality

you are trying to change in action.

2. While interacting with the game, note down keywords that can help you locate the code you are looking for. Character names, item names and location names are all useful pointers.

3. Use the keyword search facility in your IDE to locate candidate files/lines to examine. In Eclipse, use the *Search* $\rightarrow$ *File Search* menu command to search for keywords occurring anywhere within your repository.

   Hint: remember to look at configuration files as well as program code. Not all of the game's functionality is described in Java code.

4. Make the changes to the code, save and rebuild. Don't commit, and certainly don't push your changes, as we are just experimenting at this stage.

5. Run the tests to make sure you haven't broken anything that used to work.

6. Shutdown and restart your local Stendhal server, so that the version you are running definitely contains the changes you made.

7. Shutdown and restart your local Stendhal client, so that the version you are running definitely contains any changes you made. Make sure you connect to your local server.

8. Go to the place in the game where you made the change, and see if the new behaviour is as you expect.

When you've worked through these exercises as a team, you are ready to get working on the coursework. Don't forget to undo any changes you've made while experimenting, so that you can start the coursework from a clean version of the code. (The quickest way to do this is to checkout any files you have changed, so that they are overwritten with the version from the `master` branch.)

Important Note for Open Source Coders

The code base that we have provided you with for this coursework is *not* guaranteed to be the same as the public Stendhal project code base. Each year, we make some small changes to suit the purposes of the coursework.

All this means that, if you are interested in contributing some code to the Stendhal project, you should first get a clean and up-to-date copy of their actual code base to work from. The bulk of the code base we are working from is the same, but there are just enough differences to mean that any patch you create might not be usable (or even recognisable as useful) by the Stendhal team.

The Stendhal dev team are very welcoming, and say they will be very happy to receive contributions from any student taking COMP23311. Information about how to contribute can be found on the Stendhal Game website.

Good luck!