Software Testing

Y1481702

March 29, 2018

Contents

1 Test Plan

1.1 Introduction

Java MASON is an agent-based..
The software fits into the category of shrinkwrap meaning it may be used by many people in a variety of ways.
In particular,
The software is freely-available and open-source, which is a special case of shrinkwrap software. While some form of automated testing may have been conducted, it cannot be taken for granted as this has not been distributed with the source code, either on the MASON website, or its GitHub repository.

1.1.1 Tools

The IntelliJ IDE was used to explore the code and develop automated tests. JUnit has been used to create automated unit tests for the software. UI tests?

Git version control has been used to manage the code for the JUnit tests. For a long term project, this would mean that automated tests could be updated alongside any code changes.

1.2 Test Coverage

1.2.1 System Overview

As stated in the project brief, the testing only needs to cover the `sim.engine`, `sim.field` and `sim.field.grid` packages, but not their subpackages.

- `sim.engine` is responsible for the core simulation management, including the agent scheduling.
- `sim.field` provides abstract classes for the representations of space in MASON simulation models, with subpackages managing specific instances of these.
- `sim.field.grid` provides various 2D and 3D grid representations of simulation space.

The expected behaviour of the software has been determined using the extensive Java MASON documentation[1], first-party example implementations as well as a number of open-source examples[2].
The system behaviour has been tested against the inferred requirements of the client, a private biological research institute.

In order to design appropriate test cases an understanding of all levels of the system was needed.

IntelliJ's Diagram feature helped to show how different components of the system are connected. Code coverage has provided a good understanding of which parts of the [code] are regularly used while running the software. Other metrics, such as how often different source files have been changed provide an insight into which files are most subject to change.

Producing the right metrics to help test the application have been particularly difficult in this case. No real history of previous bug tracking GitHub commits are all credited to *eclab* rather than individual developers
Cyclomatic complexity can be a good measure of the most *dangerous* sections of applications.

1.2.2 Unit Testing

Unit Testing should NOT cover: -Trivial Code- getters, setters -Code that has non-deterministic results -Code that deals only with UI but SHOULD cover: -Core code that is accessed by (a lot of) other modules -Code that seems to gather a lot of bugs, how do we determine this? -Code that changes by multiple different developers, github commits may not tell whole story here? [3]
Aim for 60-70% code coverage of business logic, ˜20% of the overall application.

1.2.3 System Testing

1.2.4 Integration Testing

1.2.5 Acceptance Testing

Would help us to understand the domain and market better- particularly useful in this case -users have received a basic level of Java training, we need to know if MASON is an appropriate tool or too complex?
Should be meaningful for the customer- performed on potential users -not available as an option due to the requirement to find users... Acceptance Testing- I am NOT a domain expert..

1.2.6 Regression Testing

Unfortunately neither the website nor the GitHub repository for MASON provide any previously implemented automated testing. Either this testing has not been done, which is common with freely-distributed software, or it has not been publicly distributed. As such, it will not be possible to run any regression testing as part of the project.

5 References

[1] S. Luke, *Multiagent Simulation And the MASON Library*, 19th ed., George Mason University, Jun 2015. [Online]. Available: https://cs.gmu.edu/%7Eeclab/projects/mason/manual.pdf [Accessed: Mar. 28, 2018]

[2] K. Alden, J. Timmis, P. Andrews, H. Veiga-Fernandes, and M. Coles, "Pairing experimentation and computational modeling to understand the role of tissue inducer cells in the development of lymphoid organs," *Frontiers in Immunology*, vol. 3, p. 172, 2012. [Online]. Available: https://www.frontiersin.org/article/10.3389/fimmu.2012.00172

[3] K. Kapelonis, "Don't Test Blindly: The Right Methods for Unit Testing Your Java Apps," Jan 2013. [Online]. Available: https://zeroturnaround.com/rebellabs/dont-test-blindly-the-right-methods-for-unit-testing-your-java-apps/ [Accessed: Mar. 29, 2018]

6 Appendix

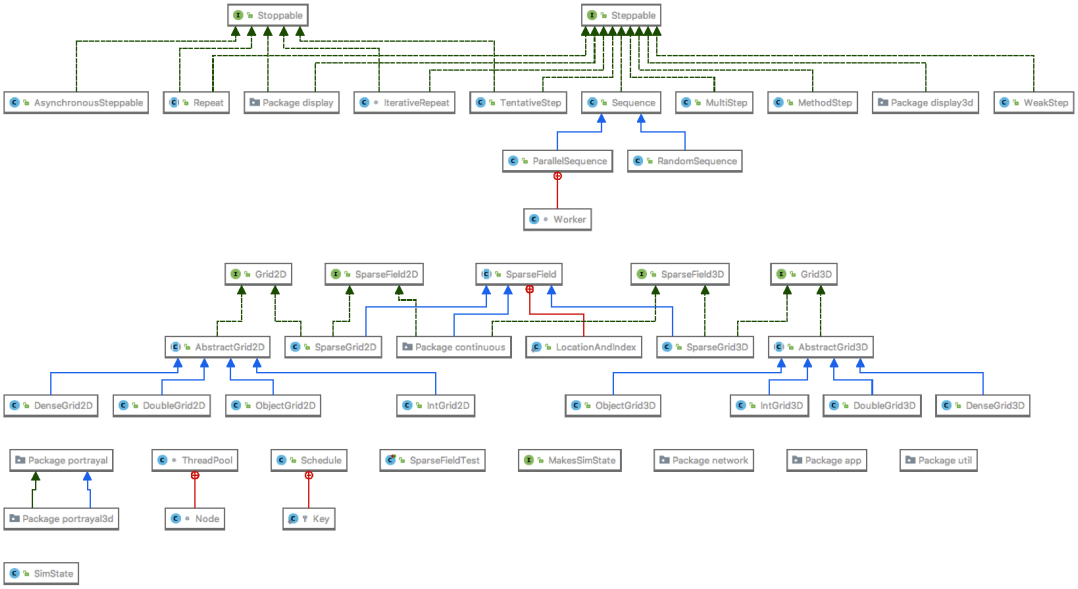

Figure 1: Reverse Engineered UML diagram of Class Hierarchy