Gizmoball: JUnit Testing Strategy

Preliminary Release Documents

Group: MW1

Testing Summary

- Make use of the JUnit Framework
- Follow standard procedures and naming conventions within the JUnit framework.
- Make use of the Test-Driven Development paradigm in order to build a robust and fully-documented system. This will keep code and tests separate and will not interfere with the structured nature of the Model View Controller system.
- Make use of the regression testing strategy to ensure new code doesn't break old code.
- Handle exceptions appropriately

How you are going to identify JUnit test cases?

Our testing strategy will put specific focus on methods and classes which rely heavily on complex calculations, for example our collisions methods. Our tests will test the methods based on different inputs from a range of different data sets including normal, abnormal, extreme and invalid data in order to pressure test our project as a whole, building a robust system. In the case that an extreme/invalid input is given, the class under test is expected to react in a controlled way in order to continue the flow of the program. This should be handled through appropriate exceptions being raised, or hard coded error messages being delivered in order to aid debugging.

Our strategy will have slightly less focus on the testing of constructors, setters and getters, but will test them nonetheless to ensure they function as intended. In the event they fail, they will be expected to deliver feedback as to why they couldn't complete their specified operation through raising exceptions or returning error messages for debugging. The tests for this section will involve attempting to retrieve valid data, with invalid data being handled by the method.

How you are going to approach JUnit testing as a group?

As a group, we intend to follow the test-driven development process. This involves each group member establishing a solid plan of what the class under development is expected to do and what type of data should be going through it. From this we can write JUnit tests based on our

robust specification to ensure the class will behave as it should. Then, only after we have a full JUnit test written out will we begin writing/generating the code for the class.

After initial JUnit tests are created, we plan to make use of a technique known as regression testing. This involves repeating our previously executed JUnit tests throughout the entire development process, especially after large implementation changes, in order to ensure the changes have not had an adverse effect on previously working code.

As a result of our choice to follow a test-driven development cycle, our project will experience the benefits associated with using the cycle. One major advantage will be making sure any code we write will be entirely inline with the project specification. This will ensure our development stays on track and that the focus will always be on implementing the key elements of our project in a clean, easy to read and uncomplicated manner. Another bonus is that our JUnit test classes will serve as living documentation for the project. All group members will be able to access the documentation and understand a class written by another member by reading through the JUnit associated, this will include all the behaviours said class will adhere to.

All members of the group will revise each other's tests from time to time to ensure that nothing has been (obviously) missed, and subsequently we will modify tests based upon group discussion and feedback. We will also make use of code coverage at all times to make sure that the vast majority of our code is covered by test, this should be eased by our choice to follow the TDD paradigm.