For testing our system, we have employed a variety of techniques. The idea for using a variety of techniques instead of a singular technique is that some techniques are more effective in certain cases. Overall, we used:

|  |  |
| --- | --- |
| Technique | When this was used |
| Black-box JUnit Testing | Used when making new functions. The JUnit test is created before the function and the function is designed to fit the test. Best for finding errors within the function. |
| White-box JUnit Testing | Rarely used. Only used if the initial JUnit testing is deemed to have not been fully effective. |
| Black-box Manual Testing | Used more as the system becomes more developed. Used to find errors that may not have been anticipated by the programmer of a function. |
| White-box Manual Testing | Usually used as the programmer creates a function. Helps the programmer ensure that the function interacts with the rest of the system as expected. Best for finding system architecture errors. |

As the game runs in a Game->Screen->Stages->Actors hierarchy with Game at the top and Actors at the bottom, we have found the best way to run the JUnit tests is to have a special version of the Screen class called JUnitTester which calls the JUnit tests after initialising appropriate variables. The reason for doing this as opposed to the more traditional approach is that libGDX requires some initialisation steps in order for the functions to work in the intended way, so the tests are only truly representative if they are called from inside the architecture.

A table of the major JUnit tests is below. This is by no means a comprehensive list of all the tests we have conducted but merely a list of the formal tests that have been used.

|  |  |  |
| --- | --- | --- |
| Test Code | Class Tested | Testing File |
| Test\_Image | ImageActor | ImageActorTests |
| Test\_Animated | AnimatedActor | AnimatedActorTests |
| Test\_Moving | MovingActor | MovingActorTests |
| Test\_Entity | Entity | EntityTests |