Test Report

Team 6

Sanjel Sadikaj
Adam Hagan
Tudor Ciobanu
Na Tang
Armintas Zadeika
Liu Zhang

Summary (a):

Our test plan is primarily driven by our requirements - tests were written to cover all of them, in addition to a handful of tests that are not explicitly linked to a requirement.

While not all aspects of the engine could be mocked, we were able to mock some parts to create a more consistent testing environment. All our test classes inherit from TestBase, which (in addition to providing some common testing functions), mocks the engine's timer and temporarily clears the preferences file, to ensure the testing environment is as consistent as possible.

Our test classes are organised into two packages - UITests is a set of unit tests for the UI, which primarily covers pressing buttons and ensuring the correct changes to state are made. GameTests contains both unit and interface tests, and covers objects on the river.

Challenges

There were two major issues that restricted our ability to write automated tests:

The first problem was that it was not possible to use the engine's rendering libraries in the testing environment, so rendering related functions couldn't be tested. Some extensive refactoring would have enabled us to mock the libraries and run the functions, but our ability to write meaningful tests on the functions would have been limited, as mocking the libraries removes most of the functionality worth testing. We decided this was not worth pursuing with the time constraints.
For this reason, all tests for rendering related functions are done manually.

The second problem was that it was impossible to test the Game class, as it was too closely linked to the game engine and could not be instantiated without errors. The relevant parts of the engine could not be mocked to avoid this, as the Game class relies on complex behaviours of the engine and a suitable mock couldn't be written within the time constraints. For this reason, no automated system tests could be written, and all tests for the Game class are done manually.

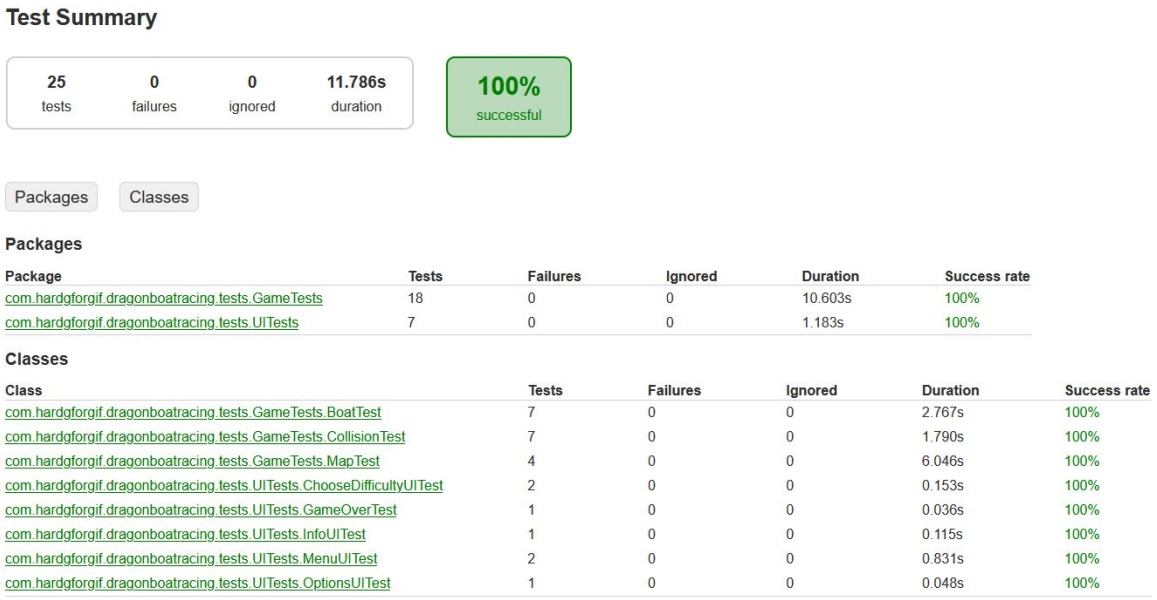
To mitigate the effects of being unable to write automated system tests, we created Interface tests for as many aspects of the system as we could. This does not fully replace system testing (in particular, all interactions with the Game class aren't covered), but it ensures that as many sub-system interactions as possible are covered.

Additionally, where appropriate and feasible we refactored logic out of the Game class so that it could be tested. The most significant change is that handling and creation of the box2d world was moved to Map, which allowed us to write tests for collision logic and certain race functions.

Report (b):

Results

We implemented all the tests in our testing plan, and all of them passed except for the manual test TEST_NFR_FAST_TRANSITION. A full list of our tests and evidence of their completion can be found on our website at: https://mrpoketes.github.io/mrpoketes.github.io-ENG1-Team6-Website2/docs-new/test2.html



Test Summary

25 tests	0 failures	0 ignored	11.786s duration

100% successful

Packages | Classes

Packages

Package	Tests	Failures	Ignored	Duration	Success rate
com.hardgforgif.dragonboatracing.tests.GameTests	18	0	0	10.603s	100%
com.hardgforgif.dragonboatracing.tests.UITests	7	0	0	1.183s	100%

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
com.hardgforgif.dragonboatracing.tests.GameTests.BoatTest	7	0	0	2.767s	100%
com.hardgforgif.dragonboatracing.tests.GameTests.CollisionTest	7	0	0	1.790s	100%
com.hardgforgif.dragonboatracing.tests.GameTests.MapTest	4	0	0	6.046s	100%
com.hardgforgif.dragonboatracing.tests.UITests.ChooseDifficultyUITest	2	0	0	0.153s	100%
com.hardgforgif.dragonboatracing.tests.UITests.GameOverTest	1	0	0	0.036s	100%
com.hardgforgif.dragonboatracing.tests.UITests.InfoUITest	1	0	0	0.115s	100%
com.hardgforgif.dragonboatracing.tests.UITests.MenuUITest	2	0	0	0.831s	100%
com.hardgforgif.dragonboatracing.tests.UITests.OptionsUITest	1	0	0	0.048s	100%

The manual test TEST_NFR_FAST_TRANSITION did not pass, as the transition time when entering a leg is ~1.5 seconds, which is not less than 0.5 seconds. In the original program, this delay was ~5 seconds, and we made an attempt to correct this by staggering the generation of the three maps (the cause of the delay). This was a significant improvement, but not enough to meet the requirement.

In order to pass this test, we would need to reduce the amount of time it takes to generate a map. The root cause of the delay is the creation of bodies in the box2d world from json files - this is an expensive operation (especially for the river banks) with no simple way to make it faster.

If the bodies were changed to be less complex (reducing the accuracy of the hitboxes), or significantly fewer bodies were generated (reducing the number of obstacles and powerups), this could allow the test to pass, though the first solution would be very difficult as we do not have the tools to modify the json files the bodies are loaded from, and the second solution would sacrifice most of the gameplay.

Completeness and Correctness

Our testing plan was not comprehensive by any stretch of the definition. Our tests achieved the following coverage statistics:

Element	Class Coverage	Method Coverage	Line Coverage	Branch Coverage
`core` package	100%	71%	70%	56%
`UI` package	75%	50%	46%	44%
`Game` class	0%	0%	0%	0%
Overall	83%	48%	47%	30%

Our tests don't cover our codebase completely or equally. The core package was covered most thoroughly, as it was both the easiest to test and most heavily changed, making it liable to contain errors. The UI package was covered less thoroughly, mostly due to the inability to run tests on rendering related code (which made up a larger portion of the package).

The gaps in our core and UI coverage are mostly due to the inability to run rendering code - almost all non-rendering methods are covered by our test suite. Some UI classes only have rendering functionality, so they were not covered at all by our automated tests.

The Game class was not covered by our automated testing, for reasons stated above. This is the largest hole in our test coverage, as Game not only makes up a large portion of our code (about a quarter of the codebase by lines), but it also contains almost all of the saving and loading logic, and is necessary to run black box tests.

We attempt to alleviate these issues with our manual tests, which are black box by nature and include tests for the Game class and rendering related code. However, manual tests are less ideal than automated ones, as they are harder to run and more likely to allow small bugs to go unnoticed. Enabling Game to be used in the testing environment is the most significant way we could improve the test suite.