Testing

Cohort 1 Group 2

Yusuf Almahmeed
Jane Boag
Charmaine Chamapiwa
Charlie Coyne
Anupam Cunden
Bertie Kerry
Leo Xu-Latti

This page does not count toward the total page number

Unit Testing of Core Game Logic
Automated unit tests were written using JUnit5 to validate behaviour of core game entities. Testing focuses on logic-heavy components such as player movement and dean (enemy) path-following. Rendering, input handling, and visual feedback systems weren't tested using unit tests, as these all depend on user interaction and graphics, which cannot be reliably verified using automated testing.

This approach is appropriate for this project because the game architecture separates entity logic from rendering logic, allowing for behaviour change detection.

Headless Testing
To enable testing of LibGDX-dependent code, the HeadLessTest superclass was implemented. This class initialises a new HeadlessApplication, which provides the required libGDX lifecycle without opening any windows. All test classes extend HeadLessTest.

This approach is appropriate because the project's architecture separates game logic from rendering, allowing logic to be tested in isolation without opening any windows or relying on a display, improving reliability and automation.

Mockito
Mockito was used to mock the CollisionSystem, so during testing, collision checks are forced to succeed. This allows movement logic to be tested without relying on the full collision detection implementation.

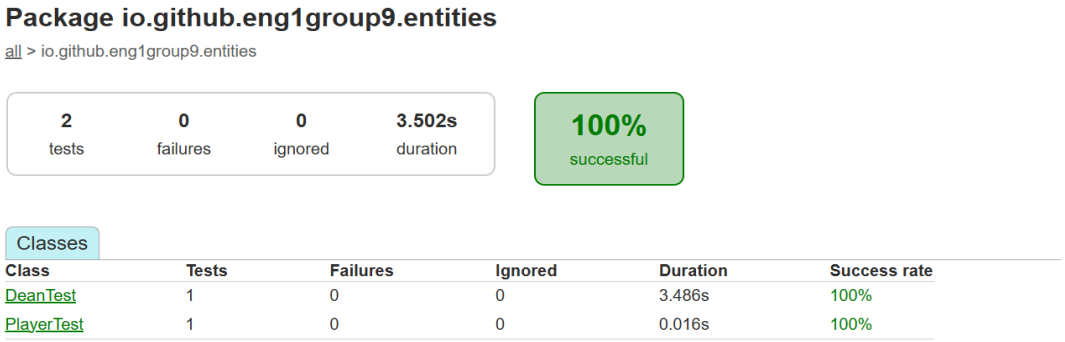
This approach is appropriate as it ensures tests remain deterministic, as mocked methods are predictable and consistent across all test runs. It also ensures tests focus on a single responsibility, as movement logic is tested independently of collision handling. Finally, it ensures tests are independent of any untested systems.

Isolating tests
Each test is isolated using @BeforeEach, ensuring no shared states between tests. This improves repeatability and reliability.

Overview of Automated Tests

Automated testing was implemented to verify the behaviour of key gameplay logic within the maze game. All tests were executed using Gradle's test task, which runs using JUnit 5, and produces both test results and a code coverage report. The tests consist of three test classes: HeadLessTest, PlayerTest, and DeanTest.

In total, 2 Tests were executed, run in a headless LibGDX environment to ensure compatibility with automated testing. These tests focus on validating core gameplay logic within the game, specifically player movement and enemy path-following behaviour, rather than rendering or user interaction, as this logic can be tested reliably in isolation. Both tests passed, as shown in the image below.



Package io.github.eng1group9.entities

all > io.github.eng1group9.entities

2	0	0	3.502s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
DeanTest	1	0	0	3.486s	100%
PlayerTest	1	0	0	0.016s	100%

Tests:

PlayerTest

This class checks that the player entity responds correctly to movement commands. Each test initialises the player at a known starting position, and applies a single directional input. To prevent interference from other systems, the CollisionSystem class is mocked to ensure collision checks always succeed, and that the test focuses exclusively on movement logic instead of collision detection. This was done using Mockito.

assertEquals and assertTrue were used to verify the player's position in the x and y axis respectively. The test checks one direction specifically, testing whether the x axis is affected by the input for the up directional movement, which confirms that movement logic behaves as expected when collisions aren't present.

The player test passed, taking 0.016s, which indicates player movement logic is correct under the headless controlled conditions.

DeanTest

The DeanTest class focuses on testing the movement behaviour of the dean(s) following its predefined path and ensuring it is valid. The collision system is again mocked to isolate the logic behind the dean's path-following, as it is provided with a fixed movement sequence, so it removes any external dependencies that could affect the test in any way

The test ensures that invoking the deans movement results in a change in its position. This demonstrates correct execution of the path-following.

The dean test also passed, taking 3.486s, which indicates the dean movement logic is working and well done.

HeadLessTest

All tests were executed successfully in a headless LibGDX environment. This ensures LibGDX-dependent code is executed separate from rendering, which means it can be tested reliably without the need for graphical context using Gradle. The successful execution of all tests confirms that core game logic is sufficiently separated from rendering code.

Evaluation and Limitations

The automated tests passed under the current implementation, but while the tests do provide confidence in the correctness of core movement and entity behaviour, it does not aim to provide full coverage of the whole system. For example, rendering, input handling, and visual feedback were validated through manual gameplay testing and questionnaires instead of the automated tests.