Introduction

Phase 3 focused on developing and applying a complete set of automated unit and integration tests for Red Corridor. The goal of this phase was to verify the correctness of core gameplay systems, including movement, interaction logic, world initialization, and maze loading. By testing these systems systematically, we identified issues that were not visible through manual play and improved the overall reliability of the codebase.

Test Environment Setup

All tests were written in Java 21 using the JUnit testing framework and executed through Apache Maven. Production code is located in src/main/java, while all test files are stored in src/test/java. Maze files and other test resources are contained in src/test/resources.

JUnit and JaCoCo were configured in the project's pom.xml so that anyone who clones the repository can run the tests without needing extra setup. Running tests through Maven ensured that builds were consistent across all development environments.

Test Organization and Feature Mapping

The test suite mirrors the project’s package structure so that related classes remain easy to find. Each major test class corresponds to a specific game system or entity. The following list summarizes the coverage for each test class:

PlayerTest verifies player initialization, movement, healing, and damage handling, fragment collection, and medkit behaviour.

DroneTest confirms correct drone initialization, movement along the dominant axis, and proper behaviour when the drone is inactive.

InputHandlerTest tests that key presses and releases correctly update internal state and that medkit use requests behave as a single action.

MazeLoaderTest checks maze dimensions, exit placement, handling of blank lines, and ensures that malformed maze files cause the correct exceptions.

GameEngineLogicTest evaluates engine-level interactions, including collision rules, exit availability based on fragment count, door interactions for fragments, medkits, and traps, and updates to the world map layers.

GameFactoryWorldIntegrationTest tests world creation and verifies player placement, exit tile existence, fragment requirements, timer initialization, and dynamic overlay of the player and drones on the visual map.

KeyFragmentTest, MedkitTest, TrapTest, and PositionTest validate entity behaviours such as collection, healing amounts, trap activation, and coordinate handling.

This organization ensures that each essential system in the game has direct and meaningful test coverage.

Unit Tests

Player and Entity Behaviour

Player related tests verify all core behaviours, including construction, movement, health updates, and inventory changes. These tests confirm that health remains within valid limits, that medkits behave correctly when collected and used, and that fragments are registered properly. Supplemental tests for traps, medkits, key fragments, and positions confirm that each entity handles state updates correctly.

Drone Behaviour

Drone tests confirm that drones initialize with the correct attributes, move predictably toward the player, and remain stationary once deactivated. These tests ensure that enemy movement remains consistent and prevents unexpected behaviour in gameplay.

Maze Loading

MazeLoaderTest verifies that maze files are loaded into correctly sized grids and contain at least one exit tile. Tests also confirm that blank lines are handled safely. Maze files containing extra rows or columns or files that do not exist cause the correct exceptions to be thrown. These tests helped identify and resolve issues in the maze validation logic.

Integration Tests

Game Engine Logic

GameEngineLogicTest evaluates how multiple systems interact when the player moves or interacts with objects in the world. By invoking core movement logic directly, the tests confirm that walls block movement, that the exit cannot be entered until the required number of fragments has been collected, and that doors containing fragments or medkits update the map layers and player inventory correctly. Trap behaviour is also verified to ensure that traps apply damage and update trap counters.

World Initialization and Dynamic Rendering

GameFactoryWorldIntegrationTest ensures that world creation is consistent across different game settings. The tests confirm that the player is placed correctly, that the maze contains an exit, that fragment requirements are valid, and that world timers are initialized. They also verify that the dynamic map overlay correctly places the player and all active drones before gameplay begins.

Test Quality and Coverage

The test suite provides strong coverage for core gameplay logic, including entity behaviour, movement, maze loading, and world initialization. Coverage is expectedly lower for user interface and JavaFX related classes due to the limitations of headless testing.

To maintain quality, all tests follow a clear Arrange Act Assert structure. Test names describe intended behaviour, and each test verifies both the resulting state and any associated side effects. Tests run independently by initializing new objects in each setup. Expected exceptions are used to verify correct error handling, especially in maze loading.

Coverage numbers from JaCoCo can be inserted here after generating the report.

Findings and Improvements

Testing revealed several issues, which we addressed during this phase. A map validation issue allowed mazes with extra columns to load successfully, which could lead to unexpected behaviour. This was corrected, and new tests were added to enforce the expected rules. Redundant exception handling in the map loader was removed once tests confirmed that the catch block could never be triggered. Integration tests confirmed that exit door behaviour follows fragment requirements consistently. Tests covering traps clarified expected damage and ensured that trap counters update correctly. Engine level integration tests also strengthened confidence in movement, item interactions, trap behaviour, and map rendering across different states.

Conclusion

Phase 3 significantly improved the foundation of Red Corridor. Through the combination of unit and integration tests, we verified essential game mechanics, identified hidden issues, and increased the consistency and reliability of the game logic. The completed test suite provides confidence in the stability of the core systems and supports future development in later phases of the project.