<u>Unit and Integration Tests</u>

This project contains many core features that interact with each other during the game. Each of these needed unit testing to ensure they behave correctly on their own, before being tested together in integration tests. There are approx 40 tests written to test our game. All unit tests are written in UnitTests.java and all integration tests are written in IntegrationTests.java

We made changes to almost all classes to add helper functions (getters and setters) to make testing easier.

1. Door mechanics

What it is: Doors can open and close when the button is pressed.

Why it needs testing: To ensure that doors behave correctly when toggled and respond to button interactions.

Tests written:

- UnitTests.java
    - testDoorInitialState() → verifies doors start closed
    - testDoorToggle() → verifies opening and closing of door works

2. Button mechanics

What it is: Buttons can be pressed or released, affecting connected doors and lasers.

Why it needs testing: Ensures button state updates correctly and affects other objects correctly.

Tests written:

- UnitTests.java
    - testButtonState() → checks initial, pressed, and released states
    - testLaserButtonInitialState → should start unpressed
    - testLaserButtonToggle() → verifies laser on and off works
    - testButtonCollision() → verifies collision changes when button is pressed

3. Laser mechanics

What it is: Lasers are a type of punishment that can be on or off when button is pressed.

Why it needs testing: To ensure that laser behave correctly when toggled and respond to button interactions.

Tests written:

- UnitTests.java
    - testLaserInitialState() → verifies that laser is on initially

4. Player singleton

What it is: Player is implemented as a singleton to ensure only one instance exists.

Why it needs testing: It prevents multiple instances of player that could cause inconsistent game state like multiple players and would cause the game to crash. It is a design decision that must behave correctly, hence it is tested.

Tests written:

- UnitTests.java
    - testPlayerSingleton() → confirms only one Player instance exists

5. Cystal collection

What it is: Crystals are collectible items (rewards) and are required to open teleporters.

Why it needs testing: Crystals should be collected by player and the collected state should update correctly.

Tests written:

- UnitTests.java

    - testCrystalCollection() → verifies that is starts as “not collected” and collection works

6. Teleporter mechanics

What it is: Teleporter is the final destination of the player for each game and opens when all crystals are collected and time remains.

Why it needs testing: To ensure that teleporter only opens if conditions are met.

Tests written:

- UnitTests.java
    - testTeleporterState() → checks initial state is closed
    - testTeleporterCrystalsRemaining() → stays closed if crystals remain
    - testTeleporterTimeOver() → stays closed if time is over
    - testTeleporterConditionsMet() → opens only when all conditions are met

7. Turret mechanics and vision

What it is: Turrets are stationary enemies that have a range of vision and can also rotate. When they rotate, their vision rotates as well. If player comes in this range of vision, they are caught and dead (level restarts).

Why it needs testing: To ensure turrets properly detect the player and vision updates correctly.

Tests written:

- UnitTests.java
    - testVisionAttached() → turret should have a vision attached
    - testVisionSyncUpdatesRays() → if vision (line of sight rays) updates with rotation
    - testPlayerDetectedInLOSTurret() → if player is detected in vision range
    - testPlayerNotInLOSTurret() → player not detected when out of vision range

8. Alien mechanics and vision

What it is: Aliens are moving enemies that follow a certain path. The player dies if they collide with the aliens.

Why it needs testing: To ensure that aliens detect the player correctly and collisions behave as expected.

Tests written:

- UnitTests.java
    - testAlienVisionAttached() → vision for alien attached
    - testPlayerDetectedInLOSAlien() → if player is deteched in vision range
    - testPlayerNotInLOSAlien() → player not detected when out of vision range
    - testCanCollideWithPlayer() → alien collision with player

9. Radiation, Slime and Lasers

What they are: These are punishments that affect the player in negative ways.

Why they need testing: To ensure they affect player correctly and trigger changes in GameLogic.

Tests written:

- UnitTests.java
    - testRadiationCountdownStartsOnCollision() → ensures that countdown begins when player is in radiation zone
    - testRadiationCountdownStopsWhenLeaving() → countdown should end when player exits the radiation zone
    - testRadiationCountdownDoesNotGoNegative() → count down should never go below 0. Level restarts when count down is over
    - testSlimeCollision() → ensures that player slows down when they collide with slime
    - testLaserCollision() → ensures that the level restarts when player collides with laser

10. Timestop, Jetpack, Alien charm, invisibility

What they are: These are powerups that affect the player in positive ways.

Why they need testing: To ensure they affect player correctly and trigger changes in GameLogic.

Test written:

- UnitTests.java
  - testPlayerBecomesInvisible() → ensures that invisibility power up works and player doesn't get caught by the enemy and then resets
  - testJetpack() → ensures that the speed boost works and then resets
  - testTimestop() → ensures that time stops and then resets
  - testAlienCharm() → ensures that player doesn't die (level doesn't restart) when collision with alien happens. Allows one free pass and then resets.

11. Interactions tested – door and button, laser and button

What they are: Door and laser work along with button.

Why it needs testing: To ensure the connection works and disables collision

Tests written:

- IntegrationTests.java
  - testDoorButtonOpensDoor() → pressing the button opens the door and turns off collision, releasing closes it
  - testLaserButtonTurnsOffLaser() → pressing the button disables the laser and turns collision off, releasing it updates the button state

12. Crystal collection interaction

What it is: Collecting crystals updates the remaining crystal count in the game logic

Why it needs testing: collecting crystals should remove them from the environment, reduce the remainingCrystals counter and affects condition of opening teleporter.

Tests written:

- IntegrationTests.java

    - testCrystalCollectionUpdatesGameLogic()

13. Power ups and Aliens

What it is: Some power ups also affect aliens, along with player.

Why it needs testing: To make sure the powerups affect all relevant game entities as expected, such as player, enemies and timer.

Tests written:

- IntegrationTests.java
    - testAlienCharmPreventsPlayerCapture()
    - testTimestopFreezesAllEnemies()
    - testInvisibility()

Excluded tests

- Turret rotation: This was not tested because the rotation logic was dependant on real time behaviour which is complex and unreliable for testing. The rotation uses “Game.loop().getDeltaTime()” and updates depending on the frame rate. These values keep changing. As a result, the output angle for each update would also change between test runs, making it unreliable for testing. Additionally, the turret rotation is synced with turret vision and triggers line of sight ray recalculation. To test all these things, we would have to create helper functions that take random discrete values which wouldn’t be an accurate test, since our game doesn’t work that way.
- Things that were dependent on LITIengine such as spawning. Spawning is handled by LITIengine’s map loader and is not explicitly coded by us, so we focus unit tests on logic we wrote by ourselves.

Findings

Writing and running the unit and integration tests helped us understand the game’s systems much better. We discovered some minor inconsistencies and missing checks in the production code. For example, some power-ups did not always reset properly after use, and certain collisions were not handled as expected in edge cases. Writing the tests allowed us to see and fix these issues, which improved the stability and reliability of the game.

Test Coverage