

Project Phase 3

All JUnit tests should go under the following directory: *src/test/java*

Unit Test Areas:

Unit Testing: Takes results from a single unit, such as a function call.

1. Test checkCollision()
 - a. Test whether enemies and player objects interact correctly when colliding
2. Player Movement testing
 - a. Check that input moves character accordingly
 - b. Check whether movement interacts with terrain correctly
3. Testing expected zombie movement
 - a. Check that enemies move as expected with different player positions
 - b. Check whether movement interacts with terrain correctly
4. Test objective behaviours
 - a. Check whether objectives are collected in the appropriate cases
 - b. Check whether doors open at the appropriate times
 - c. Check whether player is on or off door
5. Test whether resetLevel resets a level back to its default state
 - a. Check whether all positions and associated game mechanics are reset to their appropriate default states
6. Testing the MusicController
 - a. Check whether volume minimums and maximums are working correctly
 - b. Check whether music stops when supposed to
7. Testing Expected Components in Campaign Panels and Practice Panel.
 - a. Check for panel initialization
 - b. Check whether buttons correctly exist or not
 - c. Check whether background image properly loads
 - d. Check for proper drawing of text boxes and buttons
8. Testing for game over conditions
 - a. Checks whether game over works in practice mode
 - b. Checks whether game over happens appropriately during loss of lives
 - i. Checks whether if game over inappropriately happens at > 0 lives
 - ii. Checks for opposite condition (whether game over happens at 0 or < lives)
9. Testing for level completion conditions
 - a. Checks whether practice mode level completion works as intended
 - b. Checks whether after a level is completed, the next level is appropriately moved onto
 - c. Checks whether for campaign mode, if all levels are completed, the game ends properly
 - d. Checks for edge case, whether if selected game difficulty is valid or not

__REPORT__

Unit Testing Approach

We focused on implementing and testing key functionalities in our game, including collision detection, enemy movement, and level initialization. These components are critical to the gameplay and have the highest potential for bugs. To minimize issues and potential bugs, we debugged and refined the code to ensure smooth and reliable performance.

Tests










To ensure the game runs smoothly, we focused on testing how responsive our interface is. One of our tests involved verifying the functionality of the Campaign and Practice Panels. We checked that all buttons within each panel, including those for selecting levels 1 to 3 across different difficulty settings, functioned as intended. This testing ensured a consistent user experience. We also wanted to ensure a seamless loading up of our levels, so we tested for level initialization, which was where we counted the number of moving/stationary enemies, walls, objectives, etc., to make sure that they were all counted for as expected. We also wanted to make sure none of the names were mixed up so we tested the names so that they would match with their intended difficulty.

Within the game, we tested our ability to collect the objectives as intended. These tests consisted of printing lines whenever a reward was collected, ensuring that the score went up and that the door to the next level opened once all keys were collected. We also tested for more in-game functionalities such as player and enemy movement, ensuring that the enemies started off where they were intended and that the player was able to move up, down, left and right. We also checked for collisions with the wall such that the player could no longer move in that direction and for enemy collisions where if the player and enemy land in the same tile a life would be lost.

Challenges

One of the challenges we faced was our lack of experience in implementing and testing UI components, so setting up an effective testing environment was a challenge since we had to ensure the tests covered various scenarios. To overcome this, we took the time to learn and adapt as we progressed. While we didn't make any major modifications to the code, we did address minor logic errors that surfaced during testing. These issues, though subtle, were only noticeable thanks to our test cases, as the code appeared to function correctly at first glance.

Test and Code Coverage

Element ^	Class, %	Method,...	Line, %	Branch,...
▼  grave_escape	97% (34/...	83% (141/...	84% (107...	62% (152...
>  enemy	100% (3/3)	77% (7/9)	90% (19/...	100% (8/8)
>  game	100% (6/6)	84% (21/25)	47% (131...	27% (30/...
>  levels	100% (14...	100% (66/...	100% (59...	100% (90...
>  menu	100% (3/3)	54% (6/11)	84% (104...	0% (0/2)
>  modes	100% (2/2)	65% (15/23)	86% (148...	37% (3/8)
>  objectives	100% (5/5)	84% (21/25)	85% (54/...	72% (13/...
>  player	100% (1/1)	71% (5/7)	90% (19/...	100% (8/8)
 Main	0% (0/1)	0% (0/2)	0% (0/3)	100% (0/0)

Findings

This phase of project development was a learning experience for everybody involved. For example, we gained more experience in developing and implementing user-interface components as brought up in our “Challenges” section. Through JUnit testing, we also found ways to increase our code coverage. We also found a few bugs in this section of development. For example, we found that in edge cases, enemy movement was prohibited because of a few logic errors. We were able to catch this and fix it for release.