HW: Writing Edit Mode and Play Mode Tests in Unity Test Runner

Objective:

Write both Edit Mode and Play Mode tests using Unity Test Runner for your final project.

Introduction: Writing Effective Unit Tests in Unity

1. Understand the Code to be Tested:

- Clearly understand the functionality of the code you are testing.

- Recognize the edge cases and special conditions that need to be tested.

2. Setup Unity Test Framework:

- Separate your tests into `Edit Mode` and `Play Mode` tests, creating distinct folders for each under the 'Tests' folder.

3. Write Isolated Tests:

- Ensure that each test is independent and does not rely on the state of other tests.

- Use mock objects or stubs to isolate the unit of work. **//(if time permits to go over it in class)**

4. Keep Tests Simple and Readable:

- Use clear and descriptive names for test methods.

- Avoid complex logic within tests; tests should be straightforward.

5. Test Single Responsibility:

- Each test should verify one thing only.

- Avoid testing multiple behaviors in a single test.

6. Document Tests:

- Comment on what each test is verifying, and comment SetUp method.

Task Details:

1. Test Plan

- Write a brief high-level test plan outlining the functionality to be tested. This plan should include:

  - The classes and methods you will test in Edit Mode.

  - The MonoBehaviour scripts and game mechanics you will test in Play Mode.

  - The purpose of each test.

2. Edit Mode Tests

- Write at least 10 tests to validate the functionality of key non-MonoBehaviour classes or methods in your project. These could include data models, utility classes, or other non-runtime logic.

- Example: If you have a class managing player statistics, write tests to ensure statistics are calculated correctly.

3. Play Mode Tests

- Write at least 10 tests to validate the functionality of key MonoBehaviour scripts. This could include checking object interactions, and state transitions, or ensuring that specific game mechanics work as expected.

- Example: Test that player health decreases when taking damage, or that the score increments correctly when an enemy is defeated.

4. Test Documentation and Results

- Document each test in your **code comments**, explaining its purpose and how it verifies the functionality.

- Run your tests and take screenshots of the test results showing that all tests pass.

Submission Requirements:

- Include the brief, high-level test plan.

- Ensure all tests pass successfully before submission.

- Provide screenshots of the **tests and the run results.**

Marking Scheme:

Criteria	Description	Marks
Setup and preparation	Proper setup of Unity Test Runner and test folder structure	10%
Test Plan	Clear and concise test plan outlining the functionality to be tested	10%
Edit Mode Tests	Correct and meaningful tests for key non-MonoBehaviour classes or methods (10 tests)	30%
Play Mode Tests	Correct and meaningful tests for key non-MonoBehaviour classes or methods (10 tests)	30%
Test Documentation & Results	Clear documentation in code comments for each test and screenshots showing the results of the test runs	20%