Game Tests Report

VALIDATION DOCUMENT

Simeon Markov| Class: EN08| 23/09/2025

Validation

This document outlines the test report based on the functional requirements specified in the User Requirements Specifications document.

Functional Requirements Test Cases

Feature	Brief Description	Passed
Multiple-Choice Questions	User selects one correct answer from four options.	Yes
Level Difficulty	Questions are tailored based on user-selected difficulty (easy, medium, master).	Yes(with Limitations on the question)
Category	Questions are tailored based on the chosen category.	Yes(with Limitations on the question)
Progress Tracking	The game tracks user's progress as questions are answered.	Yes
Post-Game/Feedback	Overall feedback is provided to the user after game completion.	Yes
Scoring System	One point is awarded for each correctly answered question.	Yes

UNIT TESTS

For the writing and executing tests, Unity Test Framework was used (NUnit) and Unity AI Assistant.

Setup: The tests script is located under the directory *Assets/Scripts/Tests*.

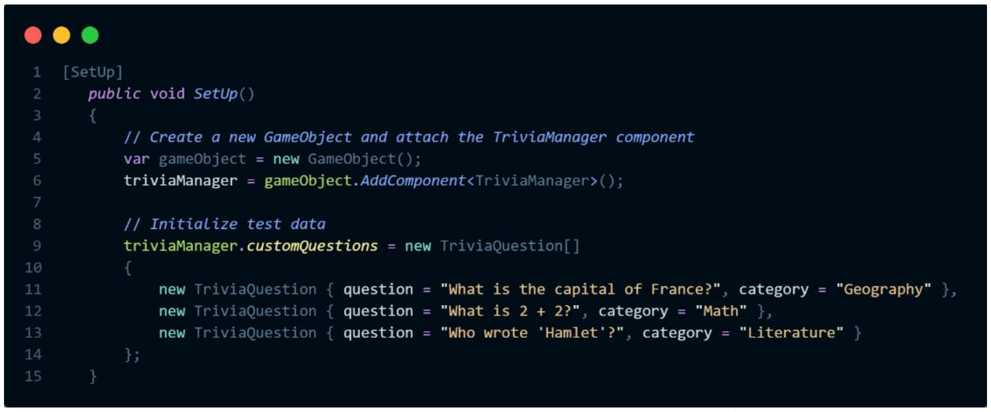
Annotations:

- **[Test]:** Marks a method as a test case (unit test).

- **[SetUp]:** Runs before each test to prepare the environment.
- **[TearDown]:** Runs after each test to clean up.

Assertions: Used for validating conditions.

Running tests: Test Runner window in Unity (*Window > General > Test Runner*). After running the tests (all at once) the results are displayed in Test Runner.

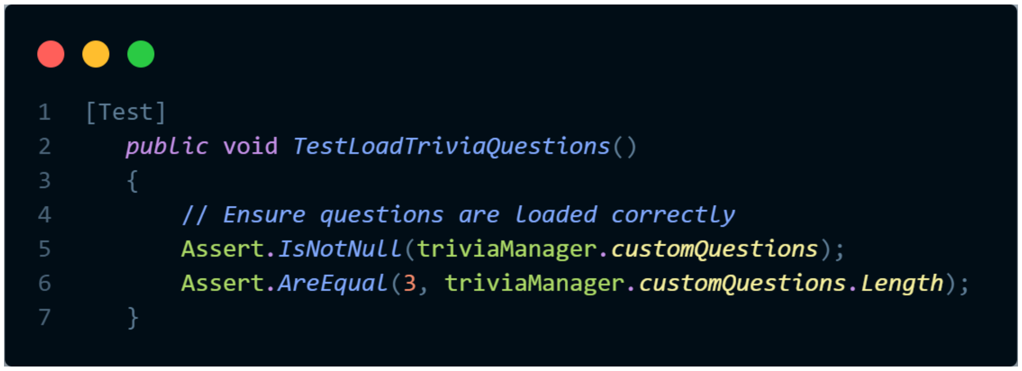


```
[SetUp]
    public void SetUp()
    {
        // Create a new GameObject and attach the TriviaManager component
        var gameObject = new GameObject();
        triviaManager = gameObject.AddComponent<TriviaManager>();

        // Initialize test data
        triviaManager.customQuestions = new TriviaQuestion[]
        {
            new TriviaQuestion { question = "What is the capital of France?", category = "Geography" },
            new TriviaQuestion { question = "What is 2 + 2?", category = "Math" },
            new TriviaQuestion { question = "Who wrote 'Hamlet'?", category = "Literature" }
        };
    }
```

Figure 1: Snippet of the SetUp method (Setting test environment)

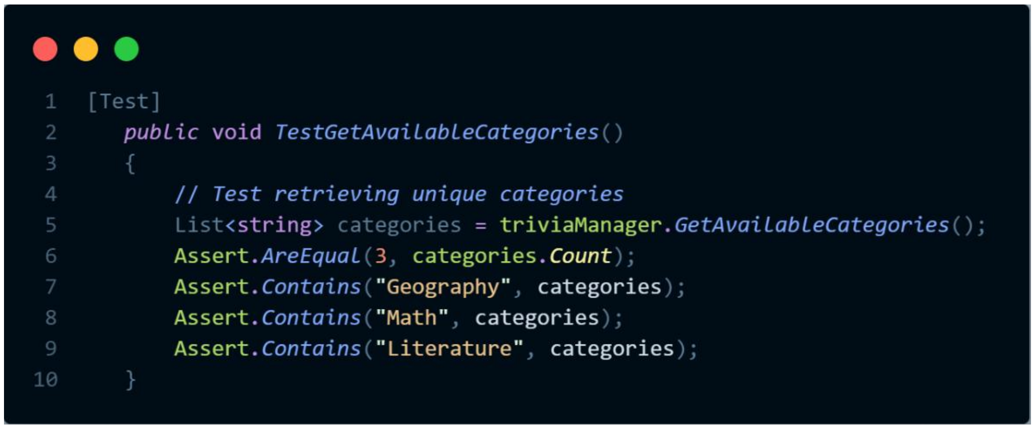
Explanation: It creates a new **GameObject**, attaches the **TriviaManager** (Parent class) component, and initializes it with sample questions.



```
[Test]
    public void TestLoadTriviaQuestions()
    {
        // Ensure questions are loaded correctly
        Assert.IsNotNull(triviaManager.customQuestions);
        Assert.AreEqual(3, triviaManager.customQuestions.Length);
    }
```

Figure 2: Test case for loading question

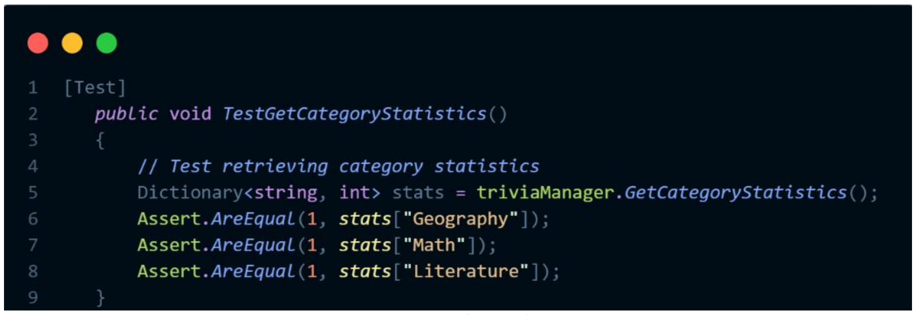
Explanation: Checks if questions are loaded and their count is correct. The assert statement is used for confirming conditions (NotNull, AreEqual).



```
[Test]
    public void TestGetAvailableCategories()
    {
        // Test retrieving unique categories
        List<string> categories = triviaManager.GetAvailableCategories();
        Assert.AreEqual(3, categories.Count);
        Assert.Contains("Geography", categories);
        Assert.Contains("Math", categories);
        Assert.Contains("Literature", categories);
    }
```

Figure 3: Test case for checking unique categories

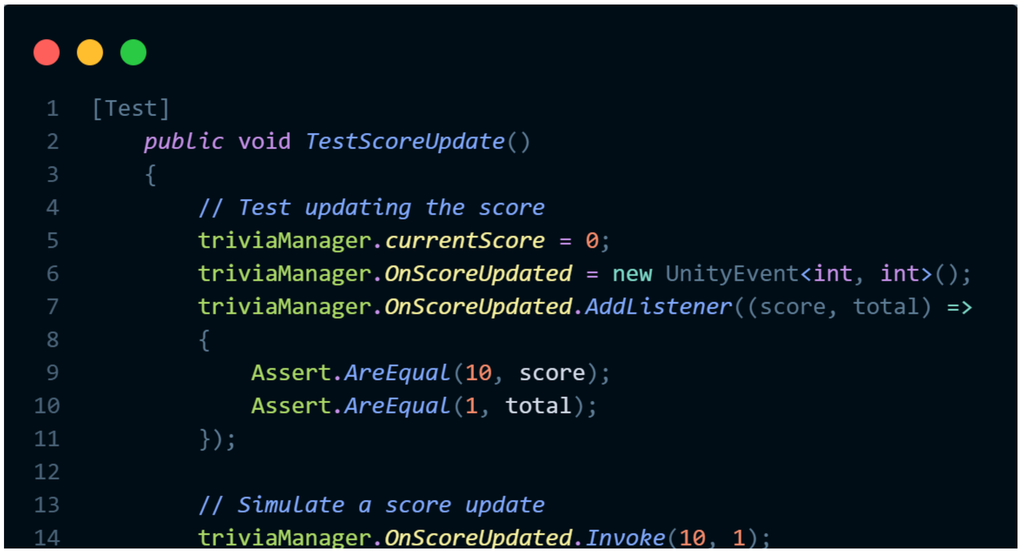
Explanation: Verifies that unique categories are being fetched.



```
[Test]
    public void TestGetCategoryStatistics()
    {
        // Test retrieving category statistics
        Dictionary<string, int> stats = triviaManager.GetCategoryStatistics();
        Assert.AreEqual(1, stats["Geography"]);
        Assert.AreEqual(1, stats["Math"]);
        Assert.AreEqual(1, stats["Literature"]);
    }
```

Figure 4: Test case for checking category statistics

Explanation: Checks if category statistics are correct.

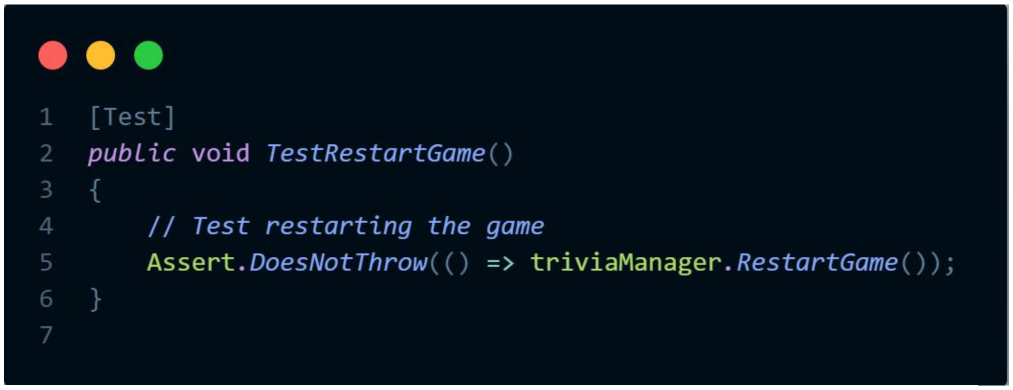


```
[Test]
    public void TestScoreUpdate()
    {
        // Test updating the score
        triviaManager.currentScore = 0;
        triviaManager.OnScoreUpdated = new UnityEvent<int, int>();
        triviaManager.OnScoreUpdated.AddListener((score, total) =>
        {
            Assert.AreEqual(10, score);
            Assert.AreEqual(1, total);
        });

        // Simulate a score update
        triviaManager.OnScoreUpdated.Invoke(10, 1);
```

Figure 5: Test case for the scoring component

Explanation: Checks if progress bar works properly (if advances per correct answer) by initializing a current score, adding test data (in this case 1 out of ten) and calls the update method to listen to changes in the state and then updates.

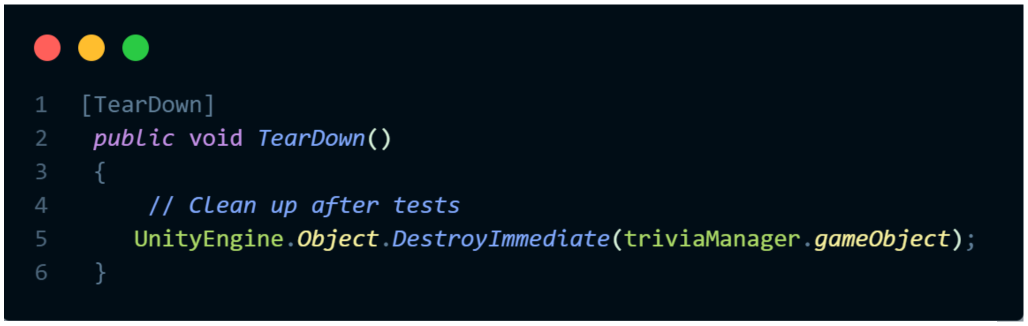


```
[Test]
public void TestRestartGame()
{
    // Test restarting the game
    Assert.DoesNotThrow(() => triviaManager.RestartGame());
}

```

Figure 6: Test case for the restart method

Explanation: Ensures that the restart functionality works.

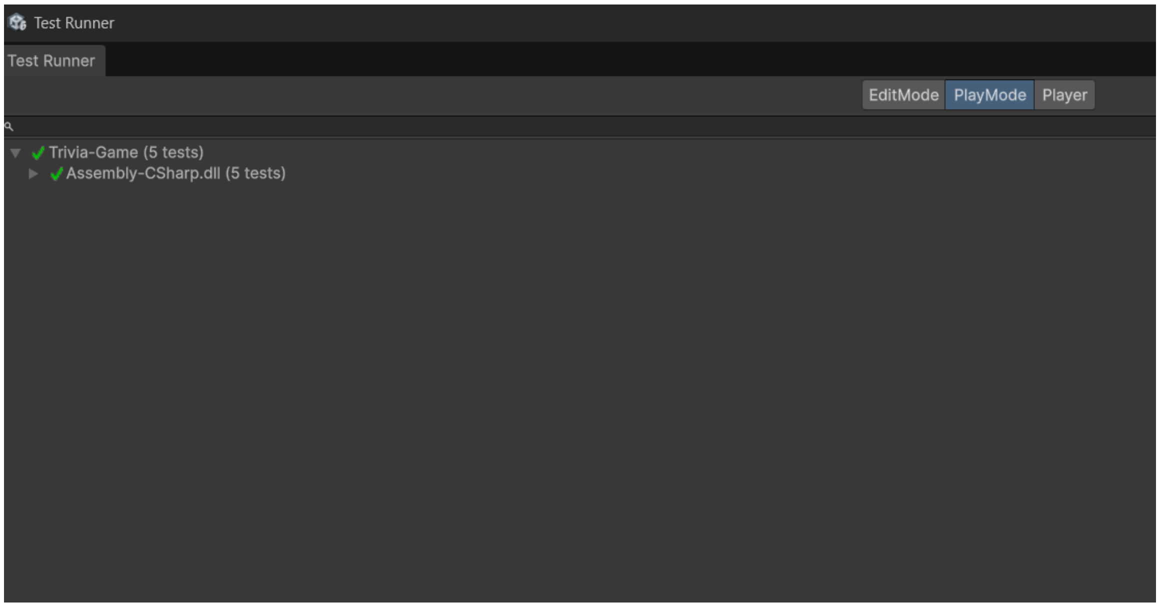


```
[TearDown]
 public void TearDown()
 {
     // Clean up after tests
    UnityEngine.Object.DestroyImmediate(triviaManager.gameObject);
 }
```

Figure 7: Dispose method

Explanation: Releases the used resource after completion of tests.

Results: All the tests were completed successfully as the output is displayed in the Test Runner console in Unity:



```
✓ Trivia-Game (5 tests)
  ✓ Assembly-CSharp.dll (5 tests)
```

Figure 8: Tests completion result