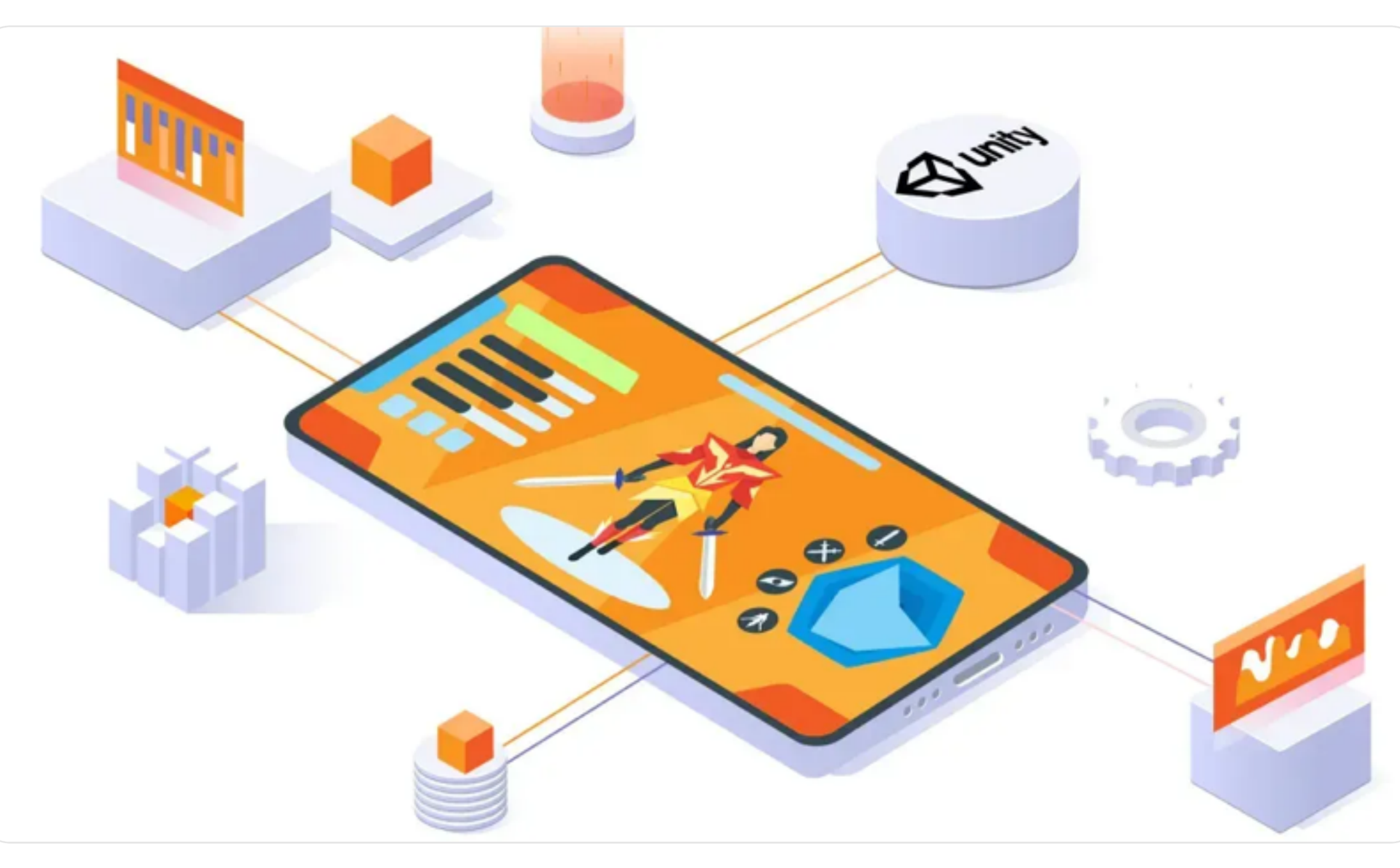

A Comprehensive Guide to Automated Testing in Game Development with the Unity Test Framework

May 29, 2024 by Rohan Singh

TEST AUTOMATION

Introduction

Are you tired of the endless cycles of manual testing in game development? The repetitive nature of these tasks can often lead to errors creeping into your code. Automating your testing process can save valuable time for more creative game development tasks.

Unity offers a solution with its Unity Test Framework, allowing you to create, manage, and run automated tests for your games. This ensures your project remains stable and functional even as you add, remove, or modify code. Let's explore how automated app testing can streamline your development workflow and enhance the quality of your games.

Share this

Table of Contents

Essential Unity Game Testing Practices

- **Start Early and Test Often:** Begin testing in the early development stages to catch issues before they escalate. Regular testing throughout the cycle keeps your game on track.
- **Automate When Possible:** Automating repetitive tests can save significant time. Utilize Unity's Test Runner to handle many testing processes, allowing you to concentrate on more complex tasks.
- **Involve Real Players:** Automated tests are crucial, but feedback from real players during beta testing can reveal unexpected insights. Player input is vital for enhancing usability and engagement.
- **Monitor and Optimize:** Use Unity's Profiler and other monitoring tools to assess your game's performance. Optimization should be a continuous effort, not just a one-time task.
- **Document and Track:** Maintain detailed records of tests, results, and fixes to ensure clarity and organization. This helps track progress and identify recurring issues throughout development.

What are the Two Ways to Test in the Unity Test Framework

Unity Test Framework (UTF) offers two primary ways to test your project code: Edit mode and Play mode. You can also target test code for platforms like standalone, iOS, or Android.

Integrating UTF into your project is accomplished by incorporating it via the Package Manager. It integrates with NUnit, a well-known open-source testing library for .NET languages.

- **Edit Mode Tests:** These run in the Unity Editor and can access both the Editor and game code. This allows you to test custom Editor extensions or modify settings in the Editor and enter Play mode, which helps adjust Inspector values and run automated tests with different settings.
- **Play Mode Tests:** These let you test your game code at runtime. Tests are typically run as coroutines using the [UnityTest] attribute, enabling you to test code across multiple frames. Play mode tests run in the Editor by default but can also be run in a standalone player build for various platforms.

> ***Read:** A Comprehensive Guide to Unit Testing*

A Detailed Comparison of Edit Mode and Play Mode Tests in Unity Test Framework

Edit Mode Tests:

- **Functionality:** Test custom Editor extensions using the UnityTest attribute. These tests operate within the EditorApplication.update callback loop.
- **Control:** Manage entering and exiting Play Mode from Edit Mode tests, allowing modifications before entering Play Mode.
- **Requirements:** Tests need an assembly definition that references nunit.framework.dll and targets only the Editor platform.

Play Mode Tests:

- **Execution:** These tests can be run standalone in a Player or within the Editor, enabling runtime game code testing. They run as coroutines if marked with the UnityTest attribute.
- **Requirements:** Tests should have an assembly definition referencing nunit.framework.dll, and be placed in a folder with the. asmdef file, and reference the necessary code assembly.

Recommendations:

- **Attributes:** Prefer the NUnit Test attribute over UnityTest unless special instructions (e.g., yielding in Edit Mode or waiting in Play Mode) are needed.
- **References:** Test Assemblies can reference tools in UnityEngine.TestRunner and UnityEditor.TestRunner, with the latter being available only in Edit Mode. Specify these references in the Assembly Definition References.

> ***Also Read:** A Comprehensive Guide to Efficiently Writing and Implementing iOS Unit Tests*

How to Test with Unity Test Framework: A Step-by-step Guide

Testing your Unity project using the Unity Test Framework is crucial to ensuring its stability and reliability. Below is a comprehensive guide to help you set up and run automated tests seamlessly.

Setting Up the Unity Test Framework

1. Install the Unity Test Framework Package:

- Launch Unity and access the Package Manager via Window > Package Manager.
- Search for "Test Framework" within the Package Manager and install the Unity Test Framework package.

2. Create Test Assemblies:

- Within your Unity project, establish a dedicated folder named "Tests."
- Right-click within the Tests folder and select Create > Testing > Test Assembly Folder to generate the necessary assembly definition files for your tests automatically.

Writing Tests

1. Create Test Scripts:

- Within the Tests folder, craft C# scripts to define your tests, such as "MyTests.cs."
- Organize your test scripts into appropriate folders based on their type: Editor for Editor tests or Runtime for Playmode tests.

2. Leverage NUnit Framework:

- The Unity Test Framework operates on the NUnit framework, granting you access to NUnit attributes and methods for your tests.
- Utilize attributes like [Test] to designate test methods and Assert class for making assertions.

```
C#
// Example of an NUnit test
using NUnit.Framework;
public class MyTests {
  [Test]
public void MyTest() {
   // Test logic
 Assert.AreEqual(2, 1 + 1);
}
}
```

3. Craft Playmode Tests:

- Playmode tests are executed within the Unity Editor's Playmode, offering a robust platform for assessing scenes, GameObjects, and gameplay mechanics.

```
C#
 // Example of a Playmode test
using UnityEngine.TestTools;
using NUnit.Framework;
using System.Collections;
public class PlaymodeTests {
[UnityTest]
   public IEnumerator MyPlaymodeTest() {
// Test logic
 yield return new WaitForSeconds(0.1f);
 Assert.IsNotNull(GameObject.FindObjectOfType<Rigidbody> ());
 }
}
```

- Use the [UnityTest] attribute to identify coroutine methods as Playmode tests, enabling comprehensive runtime evaluations.

Running Tests

1. Utilizing the Test Runner Window:

1. Open the Test Runner window by navigating to Window > General > Test Runner.
2. Select the desired test assemblies or individual scripts from the Test Runner interface and initiate the testing process with the "Run All" button.

2. Executing Tests from the Command Line:

1. Executing tests directly from the command line for streamlined integration into continuous integration (CI) pipelines.
2. Employ the command **unity -runTests -projectPath <path_to_project> -testResults <path_to_results>** to execute tests and capture the results.

Advanced Testing Techniques

1. Implementing Test Fixtures:

1. Employ the [TestFixture] attribute to establish setups for multiple tests, incorporating setup and teardown routines as needed.

```
C#
 // Example of a test fixture
[TestFixture]
public class MyTestFixture {
[SetUp]
public void SetUp() {
// Code to set up before each test
}
[TearDown]
public void TearDown() {
// Code to clean up after each test
}
    }
```

2. Harnessing Mocking Frameworks:

- Employ sophisticated mocking frameworks like Moq or NSubstitute to facilitate unit testing of intricate dependencies within your codebase.

3. Seamless Integration with Continuous Integration (CI):

- Seamlessly integrate Unity tests into popular CI tools such as Jenkins, GitHub Actions, or GitLab CI to automate test runs with each code commit, ensuring continuous quality assurance.

Streamlining Automation and CI Processes

- Streamline your development workflow by automating test runs following each build, enabling early detection of regressions and bugs.
- Optimize the test reporting process by separating build and run phases, facilitating efficient sharing and analysis of test results beyond the confines of the Unity Editor.

By following these steps and adopting advanced testing techniques, you can harness the Unity Test Framework's power to ensure your Unity projects' robustness and reliability.

> ***Check out:** A Complete Guide to JUnit Testing*

Elevate Your Unity Game Testing with HeadSpin

Once you've mastered the capabilities of Unity Test Frameworks, it's time to elevate your testing endeavors with HeadSpin. HeadSpin presents a transformative platform that furnishes actionable insights to enhance your game's performance across diverse devices and networks. Integrating HeadSpin into your Unity game testing regimen offers a myriad of benefits:

- **Real Device Testing:** Gain access to a global device infrastructure, enabling comprehensive testing on real devices and diverse networks across 90+ global locations. This ensures optimal performance regardless of your players' geographical locations.
- **AI-Driven Insights:** Harness the power of AI-driven analytics to pinpoint performance bottlenecks, user experience discrepancies, and other pivotal factors that could influence your game's success. Utilize these insights to continuously enhance and optimize your game through iterative processes.
- **Continuous Testing:** Seamlessly integrate HeadSpin into your testing pipeline to facilitate continuous testing. This integration empowers you to monitor and enhance your game's performance perpetually, even post-release, ensuring sustained quality and user satisfaction.

With HeadSpin's advanced capabilities complementing Unity's testing tools, you can fortify your game's testing strategy and embark on a journey toward unparalleled excellence and player engagement.

Conclusion

Delving into the intricacies of Unity game testing transcends mere bug-fixing; it's about refining your game to dazzle in a fiercely competitive market. The Unity Test Framework offers a versatile platform that can be extended in various ways to tailor testing workflows to your project's unique needs and seamlessly integrate with other packages. These extensions complement the robust features provided by NUnit.

Armed with the right tools, a comprehensive understanding of mobile app automated testing methodologies, and the backing of platforms like HeadSpin, you have the potential to elevate your game's performance to new heights. Crafting memorable experiences for your players enables you to establish a unique identity within the vast landscape of game development.

Connect Now

FAQs

Q1. What functionality does the Unity automated QA package offer?

Ans: The Unity automated QA package allows users to record and playback touch or drag interactions within the UI of a Unity Project. Additionally, it enables recordings to drive Unity Tests, both within the editor and on iOS or Android devices.

Q2. What are the steps to add a test device in Unity?

Ans: To add a test device:

1. Navigate to the Monetization dashboard and select Current Project > Testing from the secondary navigation menu.
2. Click "Add Test Device" to register a new Android or iOS test device.
3. Enter the Device Name and Advertising ID in the provided fields within the Add Test Device window.

Related blogs

Browse all blogs

October 10, 2024
Improving Mobile Game Performance and User Experience Through Effective Testing
Read More

October 10, 2024
Measuring Success: A Comprehensive Guide to Calculating Test Automation ROI
Read More

October 8, 2024
Adopting Cloud Computing for Banking and Financial Service Innovation
Read More