Documentación Técnica: UI Toolkit Test Framework

Fuente: Unite 2025 - "Testing UI Toolkit: New Framework & Best Practices"

Este documento resume las funcionalidades, la arquitectura y los ejemplos de código del nuevo framework oficial de Unity para el testeo automatizado de interfaces basadas en UI Toolkit.

1. Conceptos Fundamentales

El nuevo framework busca resolver el problema de los "Flaky Tests" (tests inestables) mediante una simulación de eventos desacoplada de la tasa de frames y una API fluida para interactuar con el VisualTreeAsset.

1.1. Clases Base de Testeo

Dependiendo del contexto del test, se debe heredar de una de estas dos clases:

Clase	Contexto	Uso Principal
UITestFixture	**Edit Mode**	Testeo de ventanas del Editor, inspectores y lógica de diseño.
RuntimeUITestFixture	**Play Mode**	Testeo de la UI del juego en ejecución, flujos de usuario y game HUDs.

2. API de Simulación de Entrada (Low-Level)

El framework introduce objetos Pointer y Keyboard que inyectan eventos directamente en el sistema de despacho de la UI, sin necesidad de mover el ratón físico del sistema operativo.

2.1. Simulación de Puntero (Mouse/Touch)

La clase Pointer permite simular clics, arrastres y movimientos precisos.

```
using Unity.UI.TestFramework;
using UnityEngine.UIElements;
using System.Collections;
```

```
// Ejemplo de interacción básica
public sealed class MyUITests : RuntimeUITestFixture
{
   public IEnumerator TestButtonClick()
   {
      // 1. Localizar el elemento mediante Query
      VisualElement button = m_RootElement.Q<Button>("confirm-button");

      // 2. Usar el bloque 'using' para asegurar la limpieza del puntero
      using (Pointer pointer = new Pointer(PointerId.mouseDevice))
      {
         // Simula el movimiento, el presionado y el soltado
         pointer.Click(button);
      }

      // 3. Esperar un frame para que los callbacks se ejecuten
      yield return Yield();

      // 4. Verificación
      // Assert.IsTrue(logicState.confirmed);
   }
}
```

2.2. Simulación de Teclado

Permite inyectar texto y teclas especiales (como Enter o Escape) respetando el foco del sistema.

```
using (Keyboard keyboard = new Keyboard())
{
   keyboard.Type("Nombre de Usuario");
   keyboard.Press(KeyCode.Return);
}
```

3. Gestión de la Asincronía y el "Yielding"

Uno de los puntos clave del vídeo es cómo evitar que el test continúe antes de que la UI se actualice. El framework introduce Yield() para sincronizar el estado.

- **yield return Yield();**: Espera un frame de renderizado y procesamiento de eventos.
- **yield return Yield(int frames);**: Espera una cantidad específica de ciclos.

- **Uso en Async:** Si usas Task, puedes usar await Yield();.

4. Ejemplo de Flujo Completo: Navegación de A a B

Este ejemplo (basado en el mostrado en la Unite) ilustra cómo testear una transición entre pantallas.

```
[Test]
public IEnumerator NavigateFromMenuToSettings()
{
   // Cargar la escena o el UIDocument
   yield return LoadScene("MainMenu");

   // Buscar botón de ajustes
   VisualElement settingsBtn = m_RootElement.Q<Button>("settings-btn");

   // Ejecutar click
   using (Pointer pointer = new Pointer())
   {
      pointer.Click(settingsBtn);
   }

   // Esperar a que la animación de transición termine
   // El framework permite esperar a que un elemento aparezca
   yield return Yield();

   // Validar que el panel de ajustes está visible
   VisualElement settingsPanel = m_RootElement.Q<VisualElement>("settings-panel");
   // Assert.AreEqual(DisplayStyle.Flex, settingsPanel.resolvedStyle.display);
}
```

5. Funcionalidades Avanzadas Presentadas

5.1. Búsqueda Semántica (Q<T>)

El framework recomienda el uso de UQuery para localizar elementos por nombre o clase, asegurando que el test no se rompa si cambia la estructura visual pero se mantienen los nombres técnicos.

5.2. Picking y Coordenadas

A diferencia de los tests antiguos, el sistema calcula automáticamente las coordenadas del centro del VisualElement basándose en su worldBound, ignorando si el elemento está

escalado o rotado.

5.3. Testeo de "Focus"

El framework permite verificar si la navegación por teclado (Tab) es correcta:

```
VisualElement firstField = m_RootElement.Q<TextField>("user-field");
VisualElement secondField = m_RootElement.Q<TextField>("pass-field");

using (Keyboard keyboard = new Keyboard())
{
   keyboard.Press(KeyCode.Tab);
}

// Assert.AreEqual(secondField, m_RootElement.focusController.focusedElement);
```

6. Conclusiones y Recomendaciones

1. **Desacoplamiento:** El framework permite testear lógica de UI sin necesidad de cargar toda la escena del juego.
2. **Modularidad:** Se recomienda crear extensiones sobre UITestFixture para acciones comunes (ej. MyLoginHelper.Login(pointer, user, pass)).
3. **Integración con CI/CD:** Al no depender de la resolución de pantalla o del ratón real, estos tests son ideales para ejecutarse en servidores de integración continua sin cabeza (Headless).

7. Enlaces y Documentación Adicional

- Manual Oficial de UI Toolkit Test Framework (v1.0)