UNIVERSIDAD DE DISEÑO, INNOVACIÓN Y TECNOLOGÍA

Unity

UI Toolkit

Diseño de la interfaz de usuario en Unity

Curso Académico 2024/2025

Prof. Luis Rubio

Introducción

Para distribuir los elementos de la interfaz de usuario, UI Toolkit utiliza hojas de estilo en cascada, concretamente, utiliza una implementación de Flexbox Yoga.

Veamos algunas ideas para ordenar la relación entre estos conceptos.

Acerca de CSS

CSS significa Cascading Style Sheets (Hojas de Estilo en Cascada). Se trata de un lenguaje que permite describir la presentación de un documento escrito en HTML o XML (incluyendo dialectos XML como SVG, MathML o XHTML). CSS describe cómo deben ser renderizados los elementos estructurados en la pantalla o en otros medios. Por ejemplo, CSS cubre fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, etc. La principal ventaja de CSS es que permite separar el contenido del diseño, facilitando su mantenimiento, la reutilización de estilos, y la adaptación de la presentación a diferentes tipos de dispositivos.

Acerca de Flexbox

Flexbox, o Modelo de Caja Flexible, es un módulo de diseño en CSS que permite crear de manera racional un diseño flexible y adaptable. La idea es dar al contenedor la capacidad de alterar la ancha, altura, y orden de sus elementos para ocupar el espacio disponible y adaptarse a cualquier dispositivo. Flexbox es ideal para disposiciones de componentes de una sola dimensión, por ejemplo, una barra lateral de navegación, barras de herramientas, o incluso espaciado entre ítems de una lista.

¿Qué es Yoga?

Yoga es un motor de diseño de código abierto que permite definir el layout de una interfaz de usuario usando Flexbox pero en un entorno no-web. Originalmente

desarrollado por Facebook, Yoga puede usarse en plataformas como iOS, Android, y Windows. Los layouts se definen utilizando la misma sintaxis de Flexbox, pero estas definiciones se pueden aplicar a elementos nativos de la interfaz de usuario en lugar de elementos web. Esto es especialmente útil para el desarrollo de aplicaciones móviles y de escritorio, donde se quiere aprovechar la flexibilidad de Flexbox sin depender de un navegador web.

Relación entre CSS, Flexbox, y Yoga

La relación conceptual entre CSS, Flexbox, y Yoga se puede entender a través de la idea de diseño adaptable y flexible. CSS es el lenguaje base que permite la creación de estilos y layouts para la web; Flexbox es un módulo dentro de CSS que provee una manera más eficiente y menos restrictiva de diseñar layouts adaptativos, especialmente cuando se trata de ajustar el tamaño y la disposición de los elementos de manera dinámica. Yoga extiende esta flexibilidad más allá de la web, permitiendo que los principios de diseño de Flexbox se apliquen a aplicaciones nativas, proporcionando así una herramienta de alto rendimiento para el desarrollo de interfaces de usuario consistentes y adaptativas en múltiples plataformas, como por ejemplo, la implementación de una interfaz de usuario en Unity.

Fuentes:

https://www.w3.org/TR/CSS/#css

https://developer.mozilla.org/es/docs/Learn/CSS/CSS_layout/Flexbox

https://www.yogalayout.dev/

UIToolkit

Se trata de un sistema para la creación de interfaces de usuario (UI) dentro del entorno de desarrollo de Unity. Permite crear UIs para juegos y aplicaciones. A continuación, se detallan sus elementos constituyentes, componentes, herramientas, flujo de trabajo y características funcionales.

Elementos Constituyentes

Assets

Panel Settings

- Permite renderizar la UI con parámetros generales

.uXML (Documento)

- Tiene la estructura de la UI
- Sirve como plantilla de la UI a desarrollar
- Para cada UI diferente en el juego, se debe tener su correspondiente archivo uXML

.uSS

- Hoja de estilos en cascada
- Permite establecer el estilo visual de los elementos y es reutilizable

.cs

- Archivo de C# que tiene la lógica de la UI

Componente

UI Document (Componente)

- Permite mostrar la UI en la escena
- Hay que pasarle como parámetros el Panel Settings y el documento uXML

Herramienta

UI Builder

- Una herramienta de emulación visual que facilita el maquetado de la UI.
- Genera los archivos uXML y uSS de la UI.

Workflow

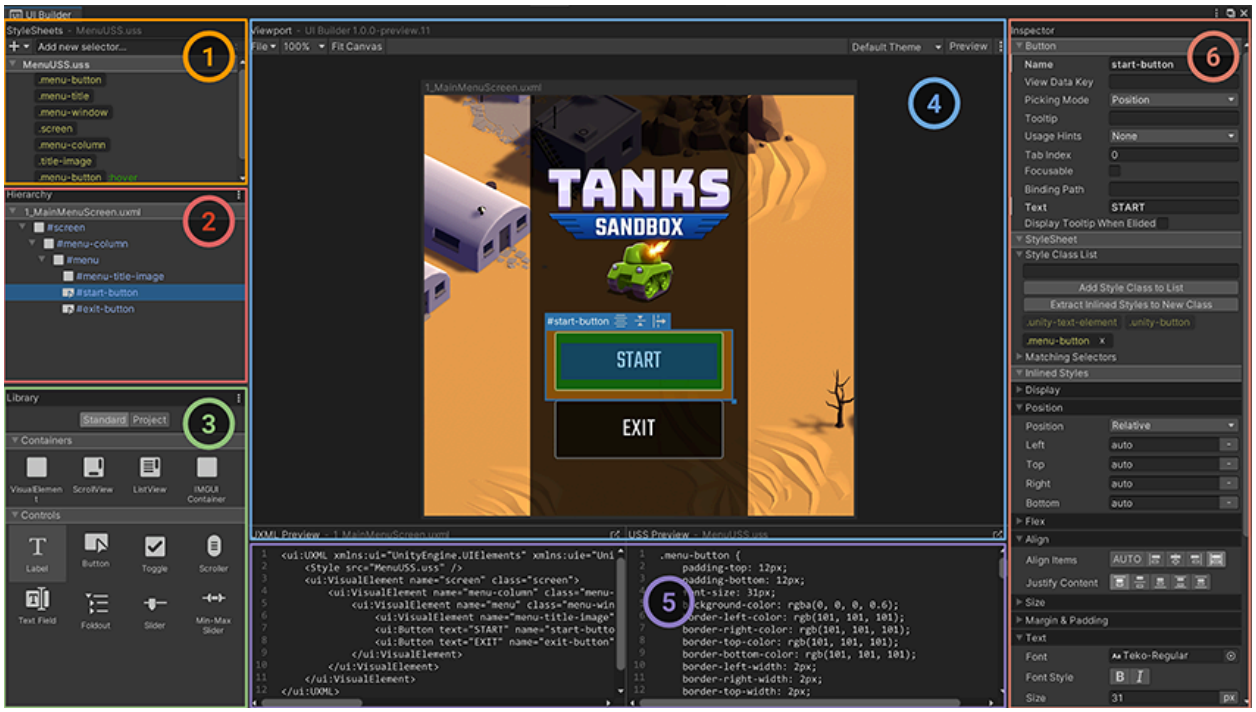
El flujo de trabajo para la creación de una UI con UIToolkit implica:

- Panel Settings
  - Creación a través del menú "Create -> UI Toolkit -> Panel Settings Asset".
- uXML
  - Generación mediante "Create -> UI Toolkit -> UI Document".
- uSS
  - Creación a través de "Create -> UI Toolkit -> Style Sheet".
- .cs
  - Creación de scripts de C# mediante "Create -> C# Script".

El uso del UI Builder mejora significativamente este flujo, permitiendo una visualización inmediata y edición interactiva de la UI.

Paneles Funcionales del UI builder

UI Builder ofrece varios paneles funcionales para facilitar el desarrollo de UIs



1 - Style Sheets

- Se puede crear una nueva hoja de estilo o agregar una existente
- Contiene un listado de clases de estilo
- Permite establecer un estilo unificado y reutilizable a los controles de la UI

2 - Hierarchy

- Muestra la estructura del documento uXML (Árbol visual)
- Permite nombrar los elementos del árbol para determinar el tipo y las clases que están anexadas a los elementos

3 - Library

- Permite definir el tipo de interface:
    - -Tree View, para interfaces en el juego
    - - Editor Extension Authoring, para crear herramientas en el editor
- Se divide en dos secciones:
    - Standard, para elementos predefinidos que vienen con Unity
    - Proyecto, para reutilizar elementos de otros o del mismo proyecto
- Se agrupan en:
    - Contenedores (VisualElement, ScrollView, ListView...)
    - Controles (Label, Button, Slider...)
- Cada vez que se añade un contenedor o control a la UI, se añade una entrada en la ventana hierarchy

4 - Viewport

- Visor que permite ver en tiempo real el aspecto la la UI que se está creando

5 - Preview

- Vista previa del código uXML y uSS

6 - Inspector

- Permite ver las propiedades del elemEnto seleccionado

Flexbox y Yoga

Como ya se ha comentado en la introducción de este documento, UIToolkit utiliza Flexbox, un modelo de diseño CSS, para una distribución flexible y eficiente de los elementos en la UI, permitiendo que se adapten dinámicamente al espacio disponible.

El sistema de layout subyacente, Yoga, se encarga de calcular el tamaño y la posición de los elementos sin realizar ningún dibujo por sí mismo.

Características de un Contenedor en Flexbox Yoga

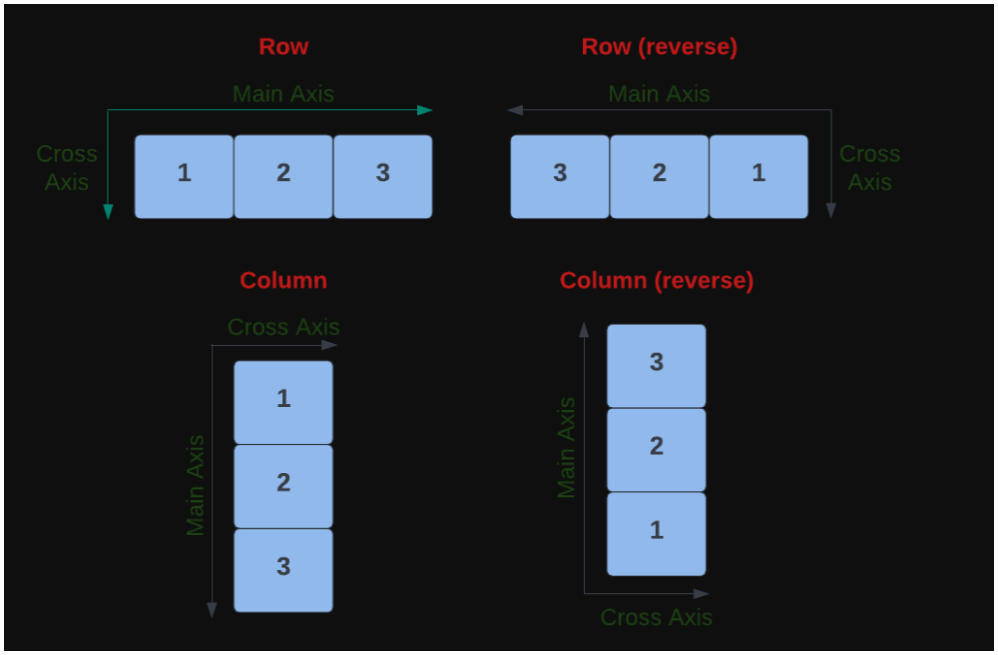
Contenedor

Un contenedor es un elemento que puede contener otros elementos (llamados hijos). El contenedor es fundamental para el diseño de la UI porque define cómo se distribuyen y alinean estos elementos hijos dentro de él.

Direction

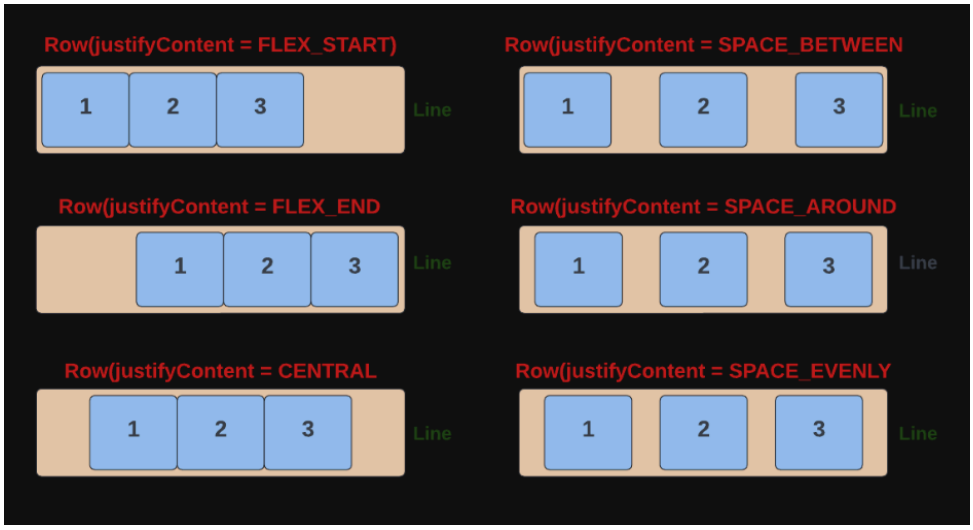
Determina cuál es el eje principal (main), y cuál es el eje secundario o transversal (cross). Es fundamental para determinar cómo se van a distribuir los hijos.

Puede ser horizontal o vertical, y ambos pueden estar invertidos.



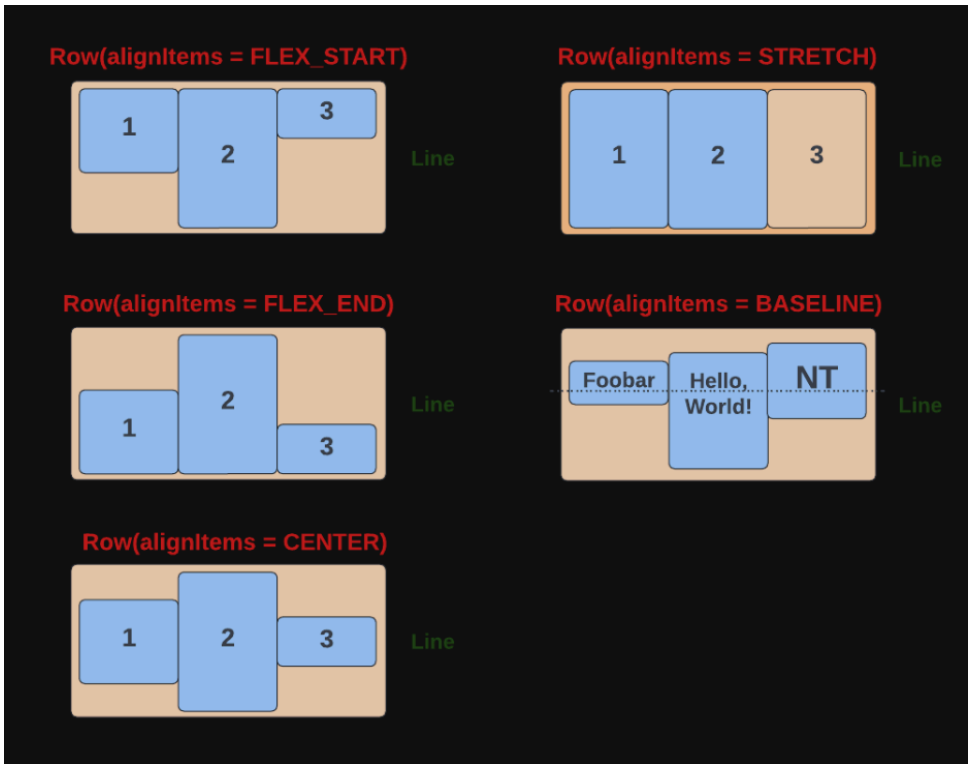
Justify

Determina cómo se distribuyen los hijos en el eje principal (main)



Align

Determina cómo se distribuyen los hijos en el eje secundario o transversal (cross).



Align-self

Permite redefinir cómo se alinea un elemento particular en el eje transversal.

https://developer.mozilla.org/en-US/docs/Web/CSS/align-self

Véase también:

https://docs.unity3d.com/Manual/UIE-USS-Properties-Reference.html

Wrap

Determina si los hijos pueden distribuirse más allá de los límites del padre

https://fblitho.com/docs/mainconcepts/flexbox-yoga/

Hijos

- Grow: Define la capacidad de un elemento para crecer si es necesario para adaptarse al espacio disponible.

- Shrink: Establece si un elemento se contraerá para adaptarse al espacio disponible, en caso de que el conjunto de los hijos no quepa en el contenedor.
- Position: (Top, left, right, bottom) funcionan como un offset para desplazar al hijo dentro de los límites del padre (modo relative) o dentro de los límites del documento raíz (modo absolute)

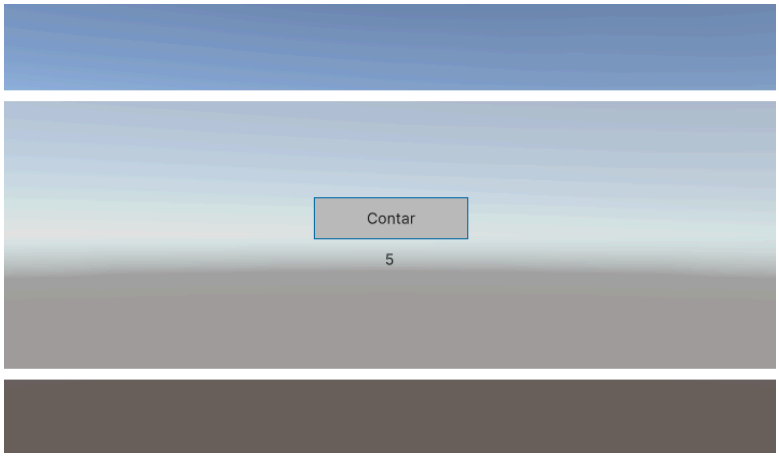
UIToolkit de Unity Technologies ofrece un sistema completo para el desarrollo de interfaces de usuario, que integra un conjunto de herramientas y componentes. Sistemas como Flexbox y Yoga permiten diseñar interfaces que se adaptan a distintos tamaños y resoluciones de pantalla en una amplia gama de dispositivos.

Conexión con el código

- Para conectar los elementos de la UI con el código del juego, el script .cs debe tener variables que permitan referenciar al documento y a los controles que posee.
- Hay que incluir al principio, una referencia a la librería (*UnityEngine.UIElements*)
- El documento es de tipo *UIDocument*, mientras que los controles deben ser del tipo empleado, por ejemplo *Button*, *Label*, etc
- El documento, si es público o serializado, se puede enlazar a través del editor de Unity, si es privado se puede obtener mediante *GetComponent<UIDocument>()*
- Los controles se enlazan a través del root del documento, buscándolos por el nombre.
- Los eventos de los controles, se registran mediante RegisterCallback.

He aquí un ejemplo:

La interfaz de usuario cuenta con un botón y un texto simple:



El código que controla su comportamiento:

```
Assembly-CSharp                                   Test_04_UI_Controller
1   using UnityEngine;
2   using UnityEngine.UIElements;
3
    Script de Unity | 0 referencias
4   public class Test_04_UI_Controller : MonoBehaviour
5   {
6       private UIDocument _document;
7
8       private Button _counterButton;
9       private Label _counterLabel;
10
11      // Start is called before the first frame update
        Mensaje de Unity | 0 referencias
12      void Start()
13      {
14          _document = GetComponent<UIDocument>();
15
16          _counterButton = _document.rootVisualElement.Q<Button>( name: "myCounterButton");
17          _counterLabel = _document.rootVisualElement.Q<Label>( name: "myCounterLabel");
18
19          _counterButton.RegisterCallback<ClickEvent>(Count);
20
21      }
22
        1 referencia
23      public void Count(ClickEvent click)
24      {
25          if (click.button == 0)
26          {
27              int count = int.Parse(_counterLabel.text);
28              count++;
29              _counterLabel.text = count.ToString();
30          }
31      }
32  }
```

PropertyDrawer

Es la clase base de la que derivar representaciones personalizadas de propiedades. Esta clase se utiliza para crear representaciones a medida de clases serializadas personalizadas, o bien, para variables de un script que posean atributos de propiedad personalizados (*PropertyAttributes*).

En definitiva *PropertyDrawers* tiene dos usos:

- Personaliza la GUI de cada instancia de una clase serializable.
- Personaliza la GUI de variables miembros de una clase que tengan *PropertyAttributes*.

Se puede ver en la siguiente clase:

```
using System;
using UnityEngine;

public enum IngredientUnit { Spoon, Cup, Bowl, Piece }

// Custom serializable class
[Serializable]
public class Ingredient
{
    public string name;
    public int amount = 1;
    public IngredientUnit unit;
}

public class Recipe : MonoBehaviour
{
    public Ingredient potionResult;
    public Ingredient[] potionIngredients;
}
```

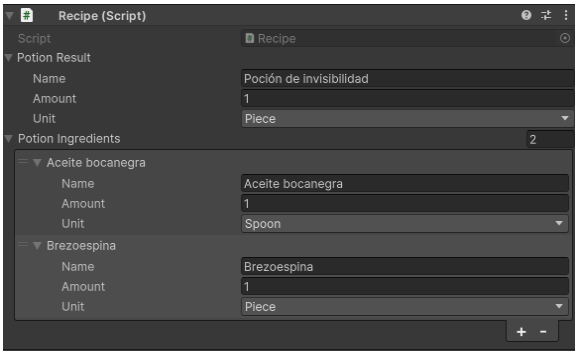
Si se utiliza un *PropertyDrawer*, se puede cambiar cada aparición de la clase ingrediente en el inspector del editor.

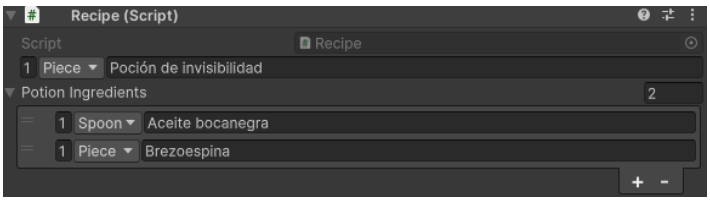
Se puede asociar un *PropertyDrawer* a una clase Serializable utilizando un atributo *CustomPropertyDrawer* attribute y pasándole el tipo de clase que será objetivo de una representación personalizada.

Para crear un *PropertyDrawer* personalizable utilizando UI Toolkit, se debe sobreescribir el método *CreatePropertyGUI* de la clase *PropertyDrawer*.

```
using UnityEditor;
using UnityEditor.UIElements;
using UnityEngine.UI;
using UnityEngine.UIElements;

[CustomPropertyDrawer(typeof(Ingredient))]
0 referencias
public class IngredientDrawerUIE : PropertyDrawer
{
    0 referencias
    public override VisualElement CreatePropertyGUI(SerializedProperty property)
    {
        var container = new VisualElement();

        var amountField = new PropertyField(property.FindPropertyRelative( relativePropertyPath: "amount"), label: "");
        var unitField = new PropertyField(property.FindPropertyRelative( relativePropertyPath: "unit"), label: "");
        var nameField = new PropertyField(property.FindPropertyRelative( relativePropertyPath: "name"), label: "");

        container.style.flexDirection = new StyleEnum<FlexDirection>(FlexDirection.Row);
        container.style.justifyContent = new StyleEnum<Justify>(Justify.SpaceBetween);

        container.Add(amountField);

        container.Add(unitField);

        nameField.style.flexGrow = 1;
        container.Add(nameField);

        return container;
    }
}
```

Se puede comparar el look de los Ingredientes en el Inspector sin, y con un *PropertyDrawer* personalizado:





Fuentes:

https://docs.unity3d.com/ScriptReference/PropertyDrawer.html