UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNLVIRTUAL

Manipulación de objetos en 3D

Unidad 4: Interfaces de Usuario

Segunda Parte

Docentes
Jaime Fili
Nicolás Kreiff

Contenidos

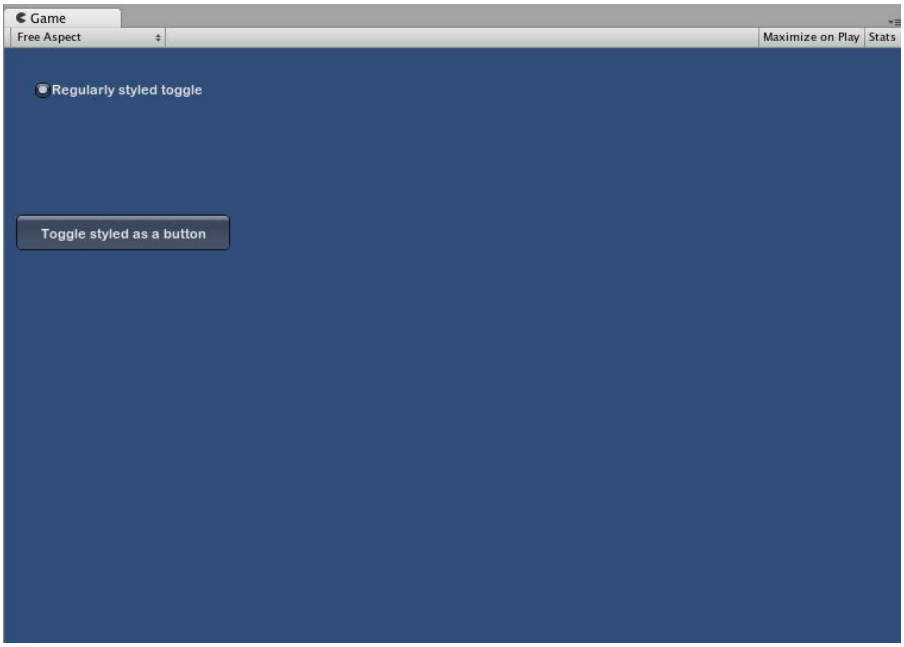
Interfaces de Usuario – Segunda Parte

Personalización de la Interfaz de Usuario

Como se ha visto, Unity3D brinda una amplia variedad de controles con los que se puede crear la Interfaz de Usuario de los juegos pero no se ha mencionado nada acerca de la apariencia de los mismos. Afortunadamente, el engine provee un mecanismo para personalizar los controles de la interfaz del juego: en eso consisten los Estilos.

En efecto, mientras los controles definen el contenido y el comportamiento, los estilos (GUIStyle) definen la apariencia. Esto permite crear combinaciones muy interesantes como, por ejemplo, *Toggles* que luzcan como botones normales.

(http://docs.unity3d.com/Documentation/ScriptReference/GUI.Toggle.html)



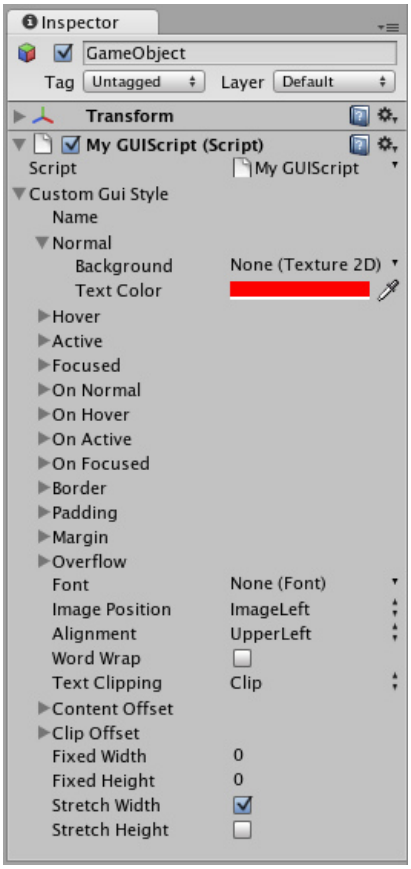
Los estilos están diseñados para imitar el comportamiento de los CSS (Cascading Style Sheets) de los navegadores web.

En casi todos los juegos que se desarrollen en esta Asignatura es muy probable que se utilicen gran cantidad de estilos diferentes. Para estos casos Unity provee un segundo mecanismo, los Skins (GUISkin), que no son más que una colección de GUIStyles. Los Skins permiten agrupar varios estilos pero no son necesarios si se utilizará sólo un GUIStyle.

El ejemplo que sigue muestra cómo utilizar un GUIStyle.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class MyGUIScript : MonoBehaviour {
05
06      public GUIStyle customGuiStyle;
07
08      void OnGUI() {
```

```
09          GUI.Button( new Rect( 10, 10, 150, 20 ), "Custom Button", customGuiStyle
10      );
11        }
12
13  }
```



Propiedades de los estilos

Name	El nombre del estilo, que puede ser utilizado para hacer referencia a él cuando forma parte de un Skin.
Normal	Imagen de fondo y color del texto del control en el estado por defecto.
Hover	Imagen de fondo y color del texto del control cuando el mouse está posicionado sobre el control.
Active	Imagen de fondo y color del texto cuando el mouse se encuentra presionando el control.
Focused	Imagen de fondo y color del texto del control cuando tiene el foco del teclado.
On Normal	Imagen de fondo y color del texto del control cuando se encuentra en el estado “Enabled”. A este estado sólo lo utilizan los Toggles cuando se encuentran activos.
On Hover	Imagen de fondo y color del texto del control cuando el mouse está posicionado sobre un control en estado Enabled (únicamente para Toggles).
On Active	Imagen de fondo y color del texto cuando el mouse se encuentra presionando sobre un control en estado Enabled (únicamente para Toggles).

On Focused	Imagen de fondo y color del texto de un control en estado Enabled (únicamente para Toggles) cuando tiene el foco del teclado.
Border	Número de pixels en cada lado de la imagen de fondo que no son afectados por la escala del control.
Padding	Espacio en pixels desde los bordes del control hasta el comienzo de su contenido.
Margin	Espacio en pixels desde los bordes del control hacia afuera. Permite establecer una distancia entre componentes.
Font	Fuente utilizada por todos los textos del estilo. Nótese que no es posible establecer una fuente distinta para cada estado del control en un único estilo.
Image Position	Establece la forma en la que la imagen de fondo y el texto serán combinados.
Alignment	Opciones de alineación de texto.
Word Wrap	Si se encuentra habilitado, el texto que alcance los límites del control continuará en la siguiente línea.
Text Clipping	Establece como se manejará el texto que supere los límites del control.
Fixed Width	Número de pixels del ancho del control. Este valor sobreescribirá cualquier otro valor utilizado al momento de definir el control.
Fixed Height	Número de pixels del alto del control. Este valor sobreescribirá cualquier otro valor utilizado al momento de definir el control.
Stretch Width	Si se encuentra habilitado, los controles que utilicen este estilo pueden ser escalados horizontalmente para un mejor alineamiento.
Stretch Height	Si se encuentra habilitado, los controles que utilicen este estilo pueden ser escalados verticalmente para un mejor alineamiento.

Cambiando los elementos de los estilos

Cuando se ha declarado un GUIStyle, este es modificable desde el Inspector. Como se puede observar en la imagen anterior, hay un gran número de estados que se pueden definir y aplicar a cualquier tipo de control.

Los estilos son modificados por script y por GameObject. Hay que tener en cuenta que cualquier ControlState tiene que tener asignado un color de fondo antes de que se le aplique el color del texto.

Trabajo con Skins. Para sistemas de interfaz de usuario más complicadas tiene sentido mantener una colección de estilos en un único lugar. Como antes se mencionaba, esto es lo que hace GUISkin; contiene diferentes estilos esencialmente para proveer cambios de apariencia de todos los GUI Controls.

Creación de un nuevo GUISkin. Para crearlo se debe seleccionar la opción *Assets>Create>GUI Skin* en la barra del menú. Esto creará un nuevo GUI Skin en la carpeta del proyecto. Al seleccionarlo se pueden ver todos los GUIStyles definidos por el Skin en el Inspector.

Aplicación del skin a la GUI. Para usar un skin propio se debe asignar a GUI.skin en la función OnGUI(), como se muestra a continuación.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      public GUISkin mySkin;
07
```

```
08          void OnGUI () {
09              // Asigna el skin
10              GUI.skin = mySkin;
11
12              //Crea un botón que tendrá el estilo por defecto del skin asignado.
13              GUI.Button (new Rect (10,10,150,20), "Skinned Button");
14          }
15
16      }
```

Se pueden cambiar de skins todas las veces que se quiera en una única llamada al OnGUI().

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06          public GUISkin mySkin;
07          private bool toggle = true;
08
09          void OnGUI () {
10              // Assign the skin to be the one currently used.
11              GUI.skin = mySkin;
12
13              // Make a toggle. This will get the "button" style from the skin
14  assigned to mySkin.
15              toggle = GUI.Toggle (new Rect (10,10,150,20), toggle, "Skinned
16  Button", "button");
17
18              // Assign the currently skin to be Unity's default.
19              GUI.skin = null;
20
21              // Make a button. This will get the default "button" style from the
22  built-in skin.
23              GUI.Button (new Rect (10,35,150,20), "Built-in Button");
24          }
25
26  }
```

Modos de *Layout*

Fixed Layout vs Automatic Layout

Existen dos modos diferentes de organizar las GUIs: fijo y automático. Hasta ahora, todos los ejemplos de UnityGUI que se han visto han usado el *Fixed Layout*. Para usar el *Automatic Layout* se debe escribir GUILayout -en lugar de GUI- cuando se llama a las funciones de los controles. Sin embargo, no es necesario utilizar sólo un modo de layout; ambos modos se pueden utilizar simultáneamente en la misma función OnGUI().

El *Fixed Layout* tiene sentido cuando se tiene una interfaz prediseñada con la que trabajar. En cambio, el *Automatic Layout* es útil cuando no se sabe previamente cuántos elementos se necesitarán o cuando no se quiere posicionar manualmente cada Control. Por ejemplo, si se está creando un número diferente de botones basados en archivos de Save Game, se desconoce exactamente cuántos botones se dibujarán. En este caso, el *Automatic Layout* puede resultar más adecuado.

En síntesis, la utilización de un modo u otro de organización de las GUIs depende estrictamente de cómo se diseñe el juego con respecto a la presentación de la interface. Hay dos diferencias clave cuando se utiliza el modo *Automatic Layout*: se usa GUILayout en lugar de GUI y no es necesaria la función Rect() para los controles.

Esto puede verse con un sencillo ejemplo:

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      void OnGUI () {
07          // Fixed Layout
08          GUI.Button (new Rect (25,25,100,30), "I am a Fixed Layout Button");
09
10          // Automatic Layout
11          GUILayout.Button ("I am an Automatic Layout Button");
12      }
13
14  }
15
```

Organizando Controles

Según qué método de layout se esté utilizando, hay diferentes maneras de controlar dónde se posicionan los controles y cómo se agrupan. En el *Fixed Layout* se pueden colocar diferentes controles dentro de **grupos** y en el *Automatic Layout* dentro de **Areas, Horizontal Groups y Vertical Groups.**

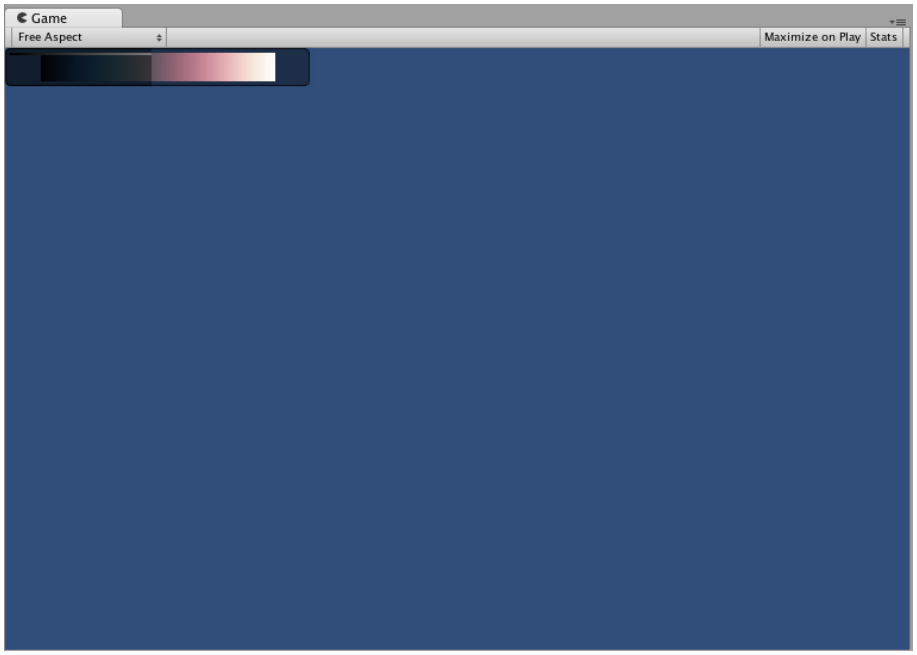
Fixed Layout – Groups. Los grupos son una convención disponible en el *Fixed Layout Mode* y permiten definir áreas de la pantalla que contendrán múltiples controles. Para definir qué controles se encuentran dentro del grupo se utilizan las funciones GUI.BeginGroup() y el GUI.EndGroup(). Todos los controles dentro de un grupo serán posicionados en base a la ubicación de la esquina superior izquierda de éste, en lugar de la esquina superior izquierda de la pantalla. De esta manera, si se reposiciona el grupo en tiempo de ejecución, la posición relativa de los componentes del grupo se mantendrá. En el siguiente ejemplo se puede apreciar el centrado de múltiples controles, ubicados en el medio de la pantalla mediante la utilización de grupos:

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      void OnGUI () {
07          // Creamos un grupo en el centro de la pantalla
08          GUI.BeginGroup (new Rect (Screen.width / 2 - 50, Screen.height / 2 -
09  50, 100, 100));
10          // Todos los Rect están ajustados al centro de la pantalla. (0,0) es la
11  esquina superior izquierda del grupo.
12
13          // Creamos una caja para poder ver dónde está el grupo en la pantalla.
14          GUI.Box (new Rect (0,0,100,100), "Group is here");
15          GUI.Button (new Rect (10,40,80,30), "Click me");
16
17          // Siempre recordar cerrar el grupo. MUY IMPORTANTE!!!
18          GUI.EndGroup ();
19      }
20
21  }
22
```

El ejemplo anterior centra los controles independientemente de la resolución de pantalla. También se pueden anidar múltiples grupos, uno dentro de otro. Cuando se hace esto, cada grupo tiene su contenido unido al espacio del padre.

Esto se refleja en código de la siguiente manera:

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      // imagen de fondo de 256 x 32
07      public Texture2D bgImage;
08
09      // imagen para el frente de 256 x 32
10      public Texture2D fgImage;
11
12      // un float entre 0.0 and 1.0
13      public float playerEnergy = 1.0f;
14
15      void OnGUI () {
16          // Crea un grupo que contiene las dos imágenes
17          // Ajusta las coordenadas para posicionarlo en algún lugar de la
18  pantalla
19          GUI.BeginGroup (new Rect (0,0,256,32));
20
21          // Dibuja la imagen de fono.
22          GUI.Box (new Rect (0,0,256,32), bgImage);
23
24              // Crea un segundo grupo
25              // Se desea clipear la imagen, no escalarlarla. Por eso se crea el
26  segundo grupo.
27              GUI.BeginGroup (new Rect (0,0,playerEnergy * 256, 32));
28
29              // Se dibuja la imagen de frente
30              GUI.Box (new Rect (0,0,256,32), fgImage);
31
32              // Se cierran ambos grupos
33              GUI.EndGroup ();
34
35          GUI.EndGroup ();
36      }
37
38  }
```



Automatic Layout – Areas. Las áreas se utilizan solamente en el modo *Automatic Layout*. Son similares en funcionalidad a los grupos del *Fixed Layout* ya que definen una porción finita de la pantalla para contener GUILayout Controls. Debido a la naturaleza del *Automatic Layout*, generalmente el uso de áreas es lo recomendado.

En este modo no se define el área de la pantalla donde el control se dibuja a nivel de Control sino que éste es ubicado automáticamente en el punto superior izquierdo de su área contenedora, que puede ser la pantalla o áreas posicionadas que se hayan creado manualmente. Los GUILayour Controls dentro de un área son ubicados en la esquina superior izquierda de esa área.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      void OnGUI () {
07          GUILayout.Button ("I am not inside an Area");
08          GUILayout.BeginArea (new Rect (Screen.width/2, Screen.height/2,
09  300, 300));
10          GUILayout.Button ("I am completely inside an Area");
11          GUILayout.EndArea ();
12      }
13
14  }
```

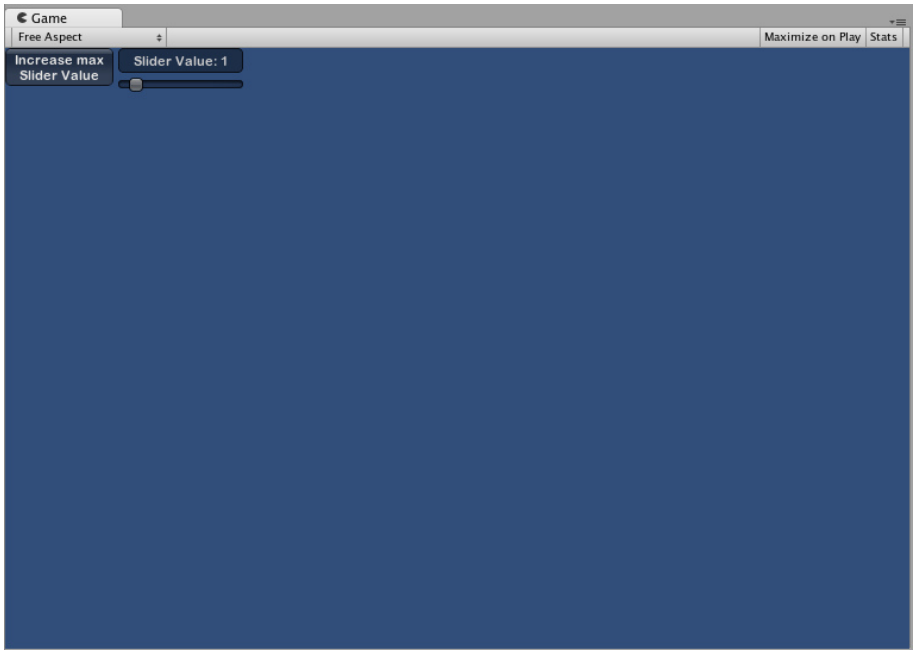
Nótese que, dentro de un área, los controles con elementos visibles como los Buttons y los Boxes se estiran horizontalmente hasta el ancho total del área.

Automatic Layout - Horizontal and Vertical Groups. Cuando se utiliza el *Automatic Layout*, los controles por defecto aparecen uno seguido del otro, de arriba hacia abajo. En muchas ocasiones se desea tener un mejor manejo de la ubicación de los Controls; las opciones para estos casos son los grupos horizontales o verticales.

Al igual que otros controles de *layout*, se deben llamar funciones separadas para comenzar y finalizar estos grupos. Las funciones especificas son GUILayout.BeginHoriztontal(), GUILayout.EndHorizontal(), GUILayout.BeginVertical(), y GUILayout.EndVertical().

Cualquier control dentro de un grupo horizontal siempre se dispone horizontalmente y cualquier control dentro de un grupo vertical siempre se dispone verticalmente. Esto suena simple hasta que se comienza a anidar grupos dentro de grupos, lo cual permite que se organicen los controles –cualquiera sea su cantidad- en toda configuración imaginable.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      private float sliderValue = 1.0f;
07      private float maxSliderValue = 10.0f;
08
09      void OnGUI()
10      {
11          // Wrap everything in the designated GUI Area
12          GUILayout.BeginArea (new Rect (0,0,200,60));
13
14          // Begin the singular Horizontal Group
15          GUILayout.BeginHorizontal();
16
17          // Place a Button normally
18          if (GUILayout.RepeatButton ("Increase max\nSlider Value"))
19          {
20              maxSliderValue += 3.0f * Time.deltaTime;
21          }
22
23          // Arrange two more Controls vertically beside the Button
24          GUILayout.BeginVertical();
25          GUILayout.Box("Slider Value: " + Mathf.Round(sliderValue));
26          sliderValue = GUILayout.HorizontalSlider (sliderValue, 0.0f,
27  maxSliderValue);
28
29          // End the Groups and Area
30          GUILayout.EndVertical();
31          GUILayout.EndHorizontal();
32          GUILayout.EndArea();
33      }
34
35  }
```



Usando GUILayoutOptions para definir controles

Se puede utilizar GUILayoutOptions para redefinir alguno de los parámetros del *Layout Automático*. Esto se puede hacer pasando las opciones como el parámetro final del GUILayout Control.

Se muestra aquí un ejemplo:

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      void OnGUI () {
07          GUILayout.BeginArea (new Rect (100, 50, Screen.width-200,
08  Screen.height-100));
09          GUILayout.Button ("I am a regular Automatic Layout Button");
10          GUILayout.Button ("My width has been overridden",
11  GUILayout.Width (95));
12          GUILayout.EndArea ();
13      }
14
15  }
```

Extendiendo UnityGUI

Existen diferentes formas de equilibrar y extender las utilidades de UnityGUI de manera de alcanzar las necesidades del desarrollo.

Controles Compuestos

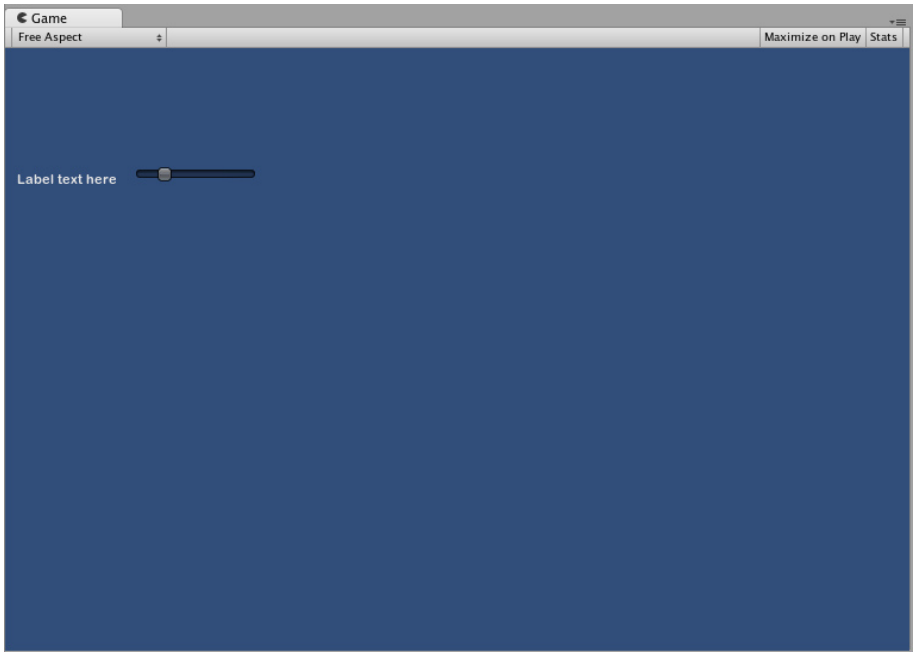
En algunos casos, puede suceder que dos tipos de controles aparezcan siempre juntos. Por ejemplo, si se está armando una pantalla de creación de personajes con varios sliders horizontales, todos estos sliders necesitan una etiqueta para identificarlos, de modo que el jugador sepa qué está ajustando. En este caso, se puede unir cada llamada a GUI.Label() con una llamada a GUI.HorizontalSlider(), o bien se puede crear un control compuesto que incluya ambos componentes .

Esto se ve en el siguiente ejemplo:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07      private float mySlider = 1.0f;
08
09      void OnGUI () {
10          mySlider = LabelSlider (new Rect (10, 100, 100, 20), mySlider, 5.0f,
11  "Label text here");
12      }
13
14      float LabelSlider (Rect screenRect, float sliderValue, float
15  sliderMaxValue, string labelText) {
16          GUI.Label (screenRect, labelText);
17
18          // &lt;- Push the Slider to the end of the Label
19          screenRect.x += screenRect.width;
20
21          sliderValue = GUI.HorizontalSlider (screenRect, sliderValue, 0.0f,
22  sliderMaxValue);
            return sliderValue;
```

```
23          }
24
25  }
```

En este ejemplo, llamando a LabelSlider() y pasando los argumentos necesarios se provee un Label agrupado con un Horizontal Slider. Cuando se escriben controles compuestos se debe recordar el retorno al valor correcto al final de la función para hacerlo interactivo.



Controles Compuestos Estáticos

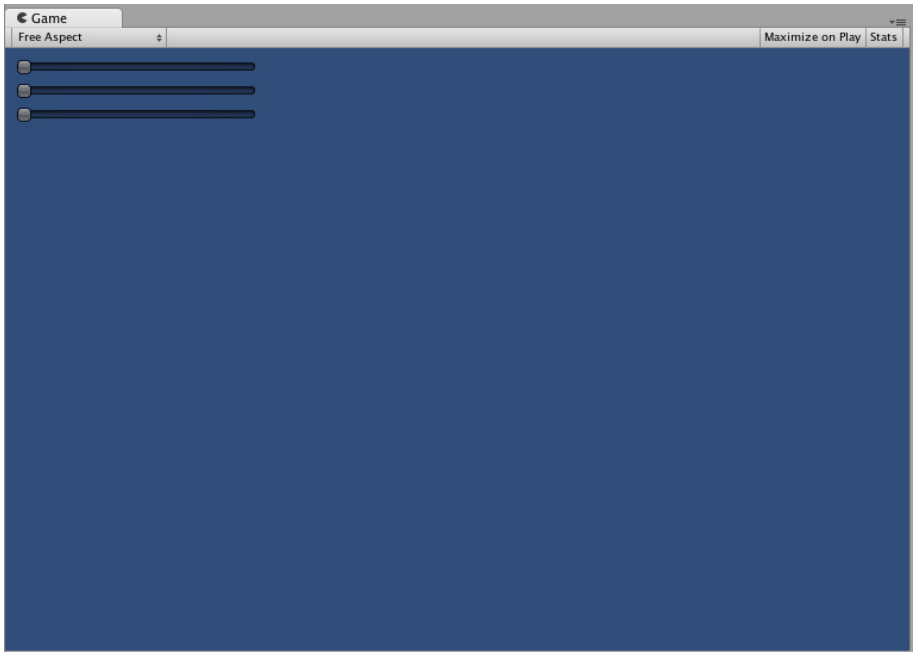
Usando funciones estáticas se pueden crear arreglos enteros de componentes propios que están autocontenidos. De esta manera, no se tendrá que declarar la función en el mismo script del que se desea usar sino simplemente invocarlos de donde se desee utilizarlos.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class CompoundControls : MonoBehaviour {
05
06          public static float LabelSlider (Rect screenRect, float sliderValue, float
07  sliderMaxValue, string labelText) {
08                  GUI.Label (screenRect, labelText);
09
10                  // &lt;- Push the Slider to the end of the Label
11                  screenRect.x += screenRect.width;
12
13                  sliderValue = GUI.HorizontalSlider (screenRect, sliderValue, 0.0f,
14  sliderMaxValue);
15                  return sliderValue;
16          }
17
18  }
```

Guardando el ejemplo anterior en un script, se puede llamar a la función LabelSlider() desde cualquier otros script simplemente tipeando ScriptName.LabelSlider() y pasando los parámetros necesarios.

En el siguiente ejemplo, se ve cómo elaborar un RGB Slider que puede ser reutilizado en cualquier otro proyecto.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      public Color myColor;
07
08      void OnGUI () {
09          myColor = RGBSlider (new Rect (10,10,200,10), myColor);
10      }
11
12      Color RGBSlider (Rect screenRect, Color rgb) {
13          rgb.r = GUI.HorizontalSlider (screenRect, rgb.r, 0.0f, 1.0f);
14
15          // &lt;- Move the next control down a bit to avoid overlapping
16          screenRect.y += 20;
17          rgb.g = GUI.HorizontalSlider (screenRect, rgb.g, 0.0f, 1.0f);
18
19          // &lt;- Move the next control down a bit to avoid overlapping
20          screenRect.y += 20;
21
22          rgb.b = GUI.HorizontalSlider (screenRect, rgb.b, 0.0f, 1.0f);
23          return rgb;
24      }
25  }
```



```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06      public Color myColor;
07
08      void OnGUI () {
09          myColor = RGBSlider (new Rect (10,10,200,30), myColor);
10      }
11
12      Color RGBSlider (Rect screenRect, Color rgb) {
13          rgb.r =  CompoundControls.LabelSlider (screenRect, rgb.r, 1.0f, "Red");
14
15          // &lt;- Move the next control down a bit to avoid overlapping
16          screenRect.y += 20;
17          rgb.g = CompoundControls.LabelSlider (screenRect, rgb.g, 1.0f,
18  "Green");
19
20          // &lt;- Move the next control down a bit to avoid overlapping
21          screenRect.y += 20;
22
23          rgb.b = CompoundControls.LabelSlider (screenRect, rgb.b, 1.0f,
24  "Blue");
25
26          return rgb;
27      }
28
29  }
30
```