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

Primera Parte

Docentes
Jaime Fili
Nicolás Kreiff

Contenidos

Interfaces de Usuario – Primera Parte 3

UnityGUI 3

Creando Controles con UnityGUI 4

Anatomía de un Control 6

Controls: Tipos de Controles 9

Interfaces de Usuario – Primera Parte

UnityGUI

Unity3D cuenta con un amplio API (Application Programming Interface http://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones) que permite crear una gran variedad de *controles* para nuestras interfaces de usuario. Además, los controles estándares que ofrece son muy sencillos de utilizar y muy fáciles de personalizar a través de *Skins*. Pero como no todo lo que brilla es oro, hay que aclarar que ésta es una de las áreas más débiles del engine y una de las más grandes deudas que la gente de Unity tiene para con sus desarrolladores.

En efecto, el API es extenso, sencillo de utilizar, fácilmente personalizable pero tiene severos problemas de performance si la interfaz que se quiere lograr es muy compleja. En estos casos, el diseño del API produce un gran acoplamiento entre el código dedicado a producir la interfaz y el código dedicado a manejar los distintos eventos que en ella se producen, es decir, entre la *vista* y el *controlador* (ver Wiki del Patrón MVC).

El API de Unity3D para GUI tiene un diseño completamente procedural; muy distinto al que poseen otros APIs en lenguajes de alto nivel, como Swing en Java. Cada control de la interfaz de usuario se crea mediante la invocación a un método estático de la clase GUI. En el siguiente ejemplo se ilustra esto, creando un botón con el texto “OK” en la esquina superior izquierda de la pantalla.

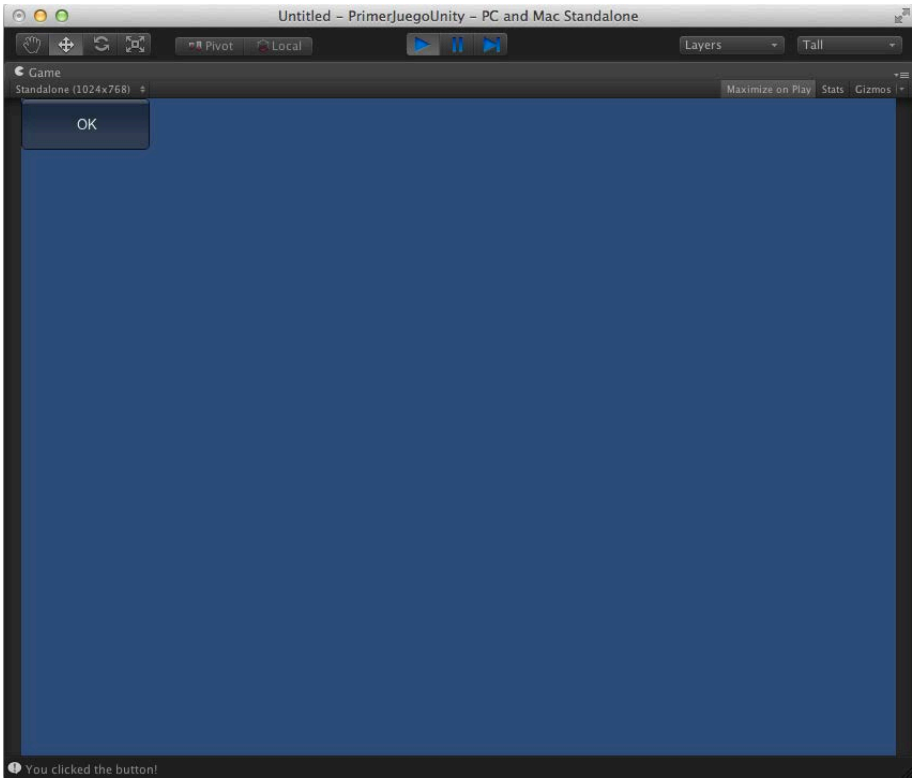
```
01  using UnityEngine;
02  using System.Collections;
03
04  public class GUITest : MonoBehaviour {
05
06              void OnGUI () {
07                      if (GUI.Button( new Rect( 0,0,120,48 ), "OK")) {
08                          Debug.Log( "You clicked the button!" );
09                          }
10              }
11
12  }
```

Toda la acción ocurre en la línea 07. En ella, Unity crea el botón solicitado pero además verifica si el usuario a efectuado un *click* en el área del botón y retorna un valor *booleano* para indicar el resultado de esta verificación. Si el usuario hubiese hecho *click* se imprimirá en la consola el texto “You clicked the button!”.

Seguramente, lo primero que viene a la mente del lector es “qué fácil!” pero si enseguida imagina una pantalla con varios botones y otros componentes, quizás con alguna animación o activación de sonidos respondiendo a la interacción del usuario con la pantalla, el panorama cambia rápidamente.

El código para realizar todas estas acciones se mezcla con el código para dibujar la propia GUI resultando en algo muy complejo de seguir y de mantener a futuro (si bien es cierto que la prolijidad depende del desarrollador, el API de Unity no ayuda para nada).

La salida del código anterior se puede observar en la siguiente imagen.

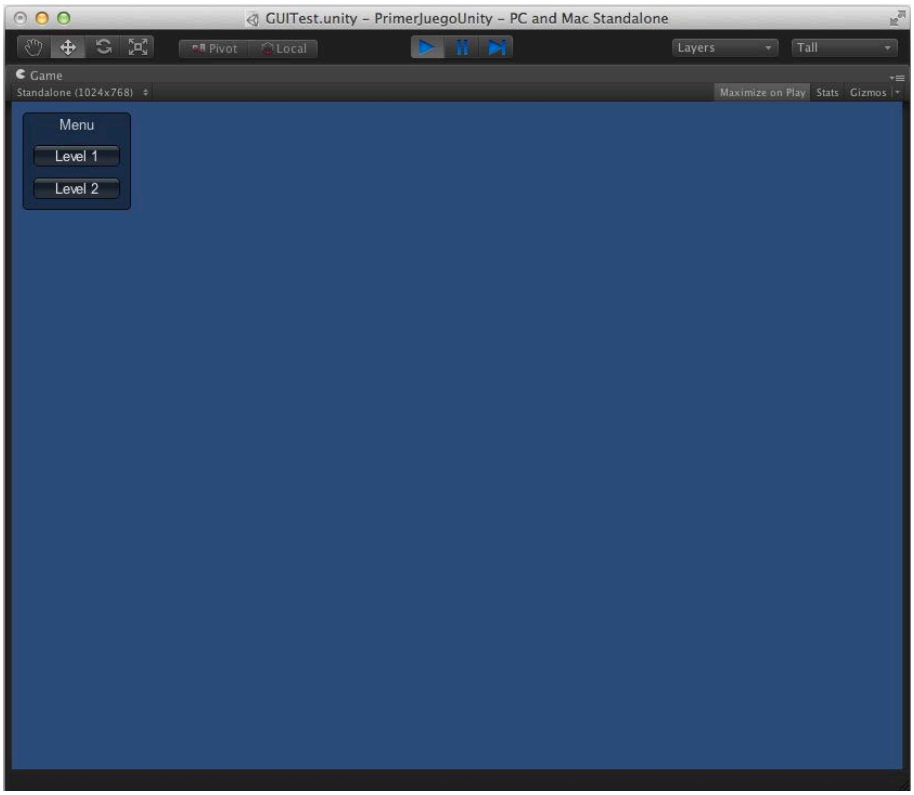


Creando Controles con UnityGUI

Los controles de UnityGUI hacen uso de una función especial llamada OnGUI(). Esta función es llamada en cada cuadro mientras el script que la contiene esté habilitado –tal como la función Update().

Los GUI Controls son de estructura simple, como puede verse en el siguiente ejemplo.

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07              void OnGUI () {
08                    // Crea una caja de fondo
09                   GUI.Box(new Rect(10,10,100,90), "Menú");
10
11
12                  // Primer Botón
13                 if(GUI.Button(newRect(20,40,80,20), "Level 1")) {
14                          Application.LoadLevel(1);
15                  }
16
17                 // Segundo Botón
18                 if(GUI.Button(new Rect(20,70,80,20), "Level 2")) {
19                          Application.LoadLevel(2);
20
21
22                }
23        }
24  }
25
```



Para comenzar a ver en detalle el código del ejemplo anterior, cabe observar la primera línea de la GUI, GUI.Box (Rect (10,10,100,90), "Menú"), que muestra un Box Control con un texto en el encabezado que dice “Menú”. Sigue la forma tradicional de declaración del esquema de controles GUI que se verá en breve.

La siguiente línea es una declaración de Button Control. Nótese que es levemente diferente de la declaración del Box Control. Específicamente, la declaración completa del Button Control está dentro del “if”. Cuando la aplicación está corriendo y el botón es presionado, la sentencia “if” devuelve “true” y cualquier código dentro del bloque es ejecutado.

Como el código que se encuentra dentro del método OnGUI() se ejecuta en cada cuadro, no es necesario crear o eliminar GUI Controls explícitamente. La línea que declara el control es la misma que lo crea. Si es necesario mostrar controles en momentos específicos, se puede usar cualquier tipo de lógica de scripting para hacerlo.

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07          void OnGUI () {
08                  if (Time.time % 2 < 1) {
09                          if (GUI.Button (new Rect (10,10,200,20), "FLASH!")) {
10                                  print ("You clicked me!");
11
12
13                          }
14                  }
15          }
16  }
```

Aquí, GUI.Button() sólo se llama segundo de por medio, por lo que el botón aparecerá y desaparecerá.

En síntesis, se puede utilizar cualquier lógica deseada para controlar cuando los GUI Controls se muestran y son funcionales.

Anatomía de un Control

Hay tres piezas claves de información requeridas cuando se declara un GUI Control:

- Type (Position, Content). Esta estructura es una función con dos argumentos. Type hace referencia al tipo de control y es declarado llamando a la función en la clase GUI de Unity o la clase GUILayout, que se verá en detalle en la sección de Modos de Layout de este apunte. Por ejemplo, GUI.Label() crea una etiqueta no interactiva.

- Position. La posición es el primer argumento de cualquier función de un GUI Control. El argumento en sí mismo es provisto con una función Rect(), que define cuatro propiedades: *left-most position*, *top-most position*, *total width* y *total height*. Estos valores son enteros y se corresponden con valores en pixels. Todos los controles de UnityGUI funcionan en Screen Space, que es la resolución del *player* publicado en pixels.

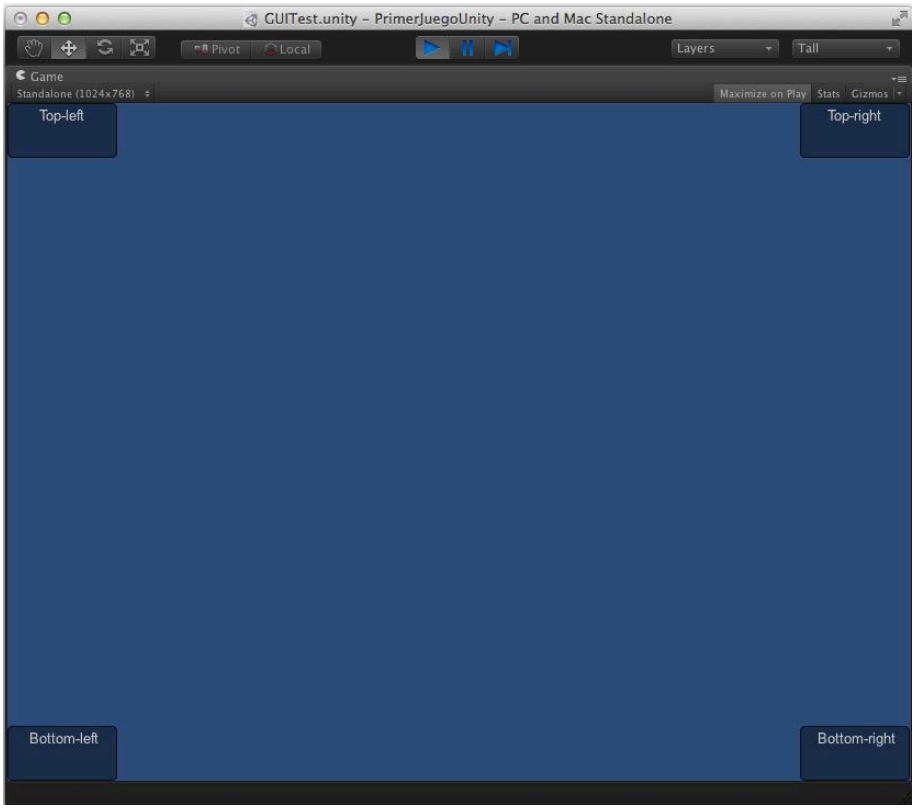
El sistema de coordenadas en el que está basado es top-left. Por ejemplo, Rect(10,20,300,100) define un *Rectangle* que comienza en las coordenadas 10,20 y termina en las coordenadas 310,120. El segundo par de valores indica el ancho total y el alto total y NO las coordenadas donde termina el control.

Se pueden utilizar las propiedades Screen.width y Screen.height para obtener el tamaño total disponible de pantalla en el player.

A continuación se muestra un ejemplo para aclarar esto:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07
08              void OnGUI(){
09                      GUI.Box (new Rect (0,0,100,50), "Top-left");
10                      GUI.Box (new Rect (Screen.width - 100,0,100,50), "Top-
11  right");
12                      GUI.Box (new Rect (0,Screen.height - 50,100,50), "Bottom-
13  left");
14                      GUI.Box (new Rect (Screen.width - 100,Screen.height -
15  50,100,50), "Bottom-right");
16              }
17
18
19  }
```

En la siguiente imagen se pueden observar las cuatro cajas posicionadas según el código del ejemplo:



- Content. El segundo argumento para un GUI Control es el contenido que se mostrará con dicho control. La mayoría de las veces, se querrá mostrar algún texto o imagen en el mismo.

Estos son algunos ejemplos:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07          void OnGUI () {
08                  GUI.Label (new Rect (0,0,100,50), "Esta es una cadena
09  de texto para un Label Control");
10          }
11
12
13  }
```

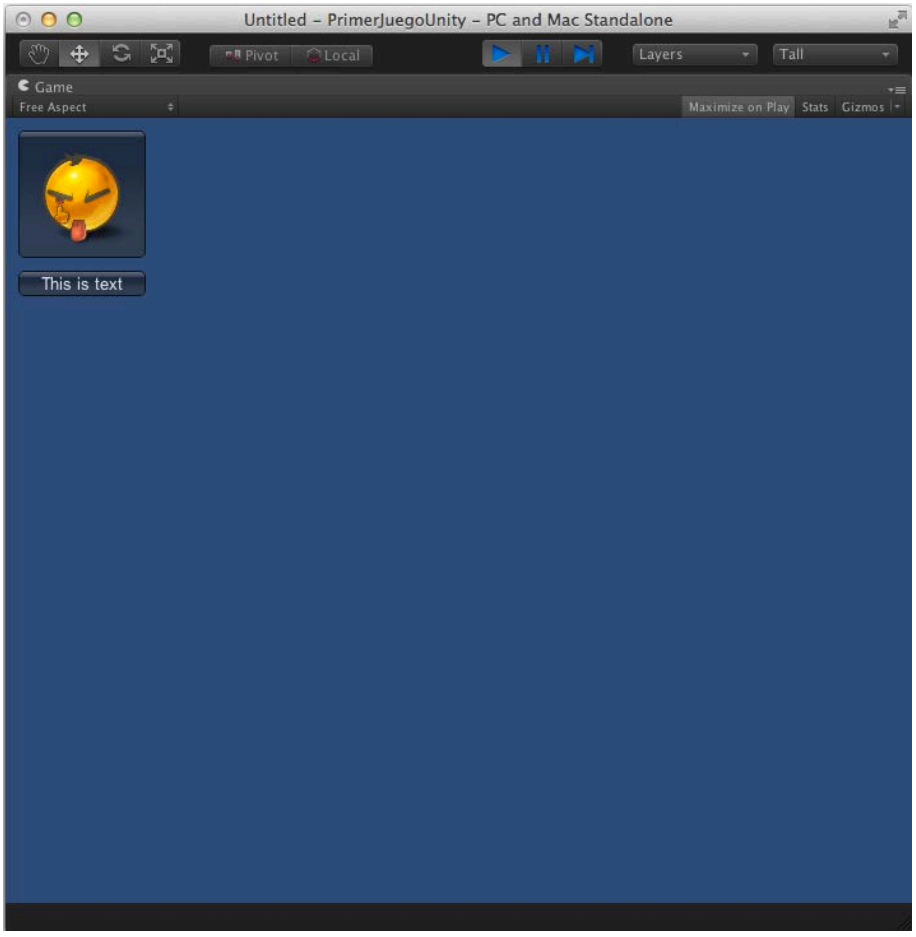
Como se puede apreciar, el código anterior permite declarar una etiqueta con el texto “Esta es una cadena de texto para un Label Control”. Si se quisiera utilizar una imagen, se debe declarar una variable pública del tipo Texture2D y pasar el nombre de la variable como contenido del argumento, tal como se muestra en el siguiente fragmento de código:

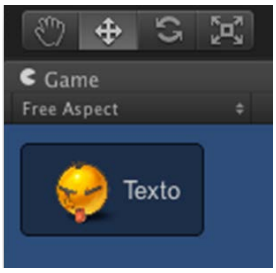
```
01  public Texture2D controlTexture;
02    ...
03
04
05  void OnGUI () {
06          GUI.Label (new Rect (0,0,100,50), controlTexture);
07  }
```

El que sigue es un ejemplo un poco más elaborado:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07              public Texture2D icon;
08
09
10              void OnGUI () {
11                      if (GUI.Button (new Rect (10,10, 100, 100), icon)) {
12                              print ("click en el icono");
13                      }
14
15
16                      if (GUI.Button (new Rect (10,120, 100, 20), "This is text")) {
17                              print ("click en el botón con texto");
18                      }
19              }
20
21
22  }
```

Aquí se muestran dos botones, uno debajo del otro. El primero tiene asignado como contenido una textura mientras que el segundo simplemente muestra un texto; en la imagen siguiente se puede observar el resultado de esto. Se debe tener en cuenta que, como la variable *icon* es pública, el objeto actual se asignó directamente desde el inspector.





Existe una tercera opción que permite mostrar imágenes y texto juntos en un GUI Control. Se puede proveer un objeto GUIContent como el argumento del Content y definir la cadena y la imagen que se mostrará dentro del GUIContent.

El siguiente ejemplo muestra cómo lograr esto.

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07              public Texture2D icon;
08
09              void OnGUI () {
10                      GUI.Box (new Rect (10,10,100,50), new GUIContent("Texto",
11  icon));
12              }
13
14
15  }
```

En la página de la referencia de GUIContent se ofrecen varios ejemplos más: http://unity3d.com/support/documentation/ScriptReference/GUIContent.GUIContent.html

Controls: Tipos de Controles

Existen varios tipos de GUI Controls que se pueden utilizar. A continuación se listan todos los disponibles:

Label. La etiqueta es un componente **no interactivo**, es decir, sólo se utiliza para mostrar contenido y no puede ser clickeado o movido.

Ejemplo: GUI.Label (new Rect (25, 25, 100, 30), "Label")

Button. El botón es el típico componente interactivo. Responde una sola vez cuando se le hace click y no importa cuánto tiempo se mantenga presionado; la respuesta ocurre apenas el botón es soltado.

En UnityGUI, los botones devuelven “true” cuando son presionados. Para ejecutar el código cuando un botón es presionado se debe encapsular la función GUI.Button dentro de una sentencia “if”. En ella está el código que será ejecutado cuando el botón sea presionado.

Ejemplo: if (GUI.Button (new Rect (25, 25, 100, 30), "Button")) {
// This code is executed when the Button is clicked
}

RepeatButton. Este componente es una variación del botón normal. La diferencia es que el RepeatButton responde en cada cuadro en que el botón permanece presionado. Esto permite crear eventos del tipo *click-and-hold*.

En UnityGUI, los RepeatButtons devuelven "true" por cada cuadro en que permanecen presionados. Igual que en el anterior, para ejecutar el código mientras el botón está presionado se debe encapsular la función GUI.RepeatButton en una sentencia "if"; en ella está el código que será ejecutado mientras el RepeatButton permanezca presionado.

```
Ejemplo: if (GUI.RepeatButton (new Rect (25, 25, 100, 30), "RepeatButton")) {
            // This code is executed every frame that the RepeatButton remains clicked
}
```

TextField. Es un control interactivo, editable en un campo de una sola línea que contiene una cadena de texto. El TextField siempre muestra una cadena de texto, que para ello debe ser provista en el control. Cuando se edita el contenido de la cadena, la función TextField devuelve la cadena modificada.

```
Ejemplo:
private string textFieldString = "text field";

void OnGUI () { textFieldString = GUI.TextField (new Rect (25, 25, 100, 30),
            textFieldString); }
```

TextArea. Es un control interactivo, editable, y un área multilínea que contiene una cadena de texto. El TextArea siempre muestra una cadena, que debe ser provista para que aparezca en el control. Cuando se edita el contenido del TextArea, la función devuelve el contenido modificado de la cadena.

```
Ejemplo:
private string textAreaString = "text area";

void OnGUI () {
            textAreaString = GUI.TextArea (new Rect (25, 25, 100, 30), textAreaString);
}
```

Toggle. Este control crea un *checkbox* con estados on/off persistentes. El usuario puede cambiar el estado simplemente seleccionándolo. Los estados on/off del control Toggle están representados por valores *booleanos true/false*. Se debe proveer el valor booleano como parámetro para lograr que el control Toggle muestre su estado actual. La función Toggle devuelve un nuevo valor booleano si el control es presionado. Para capturar esto interactivamente, se debe asignar el valor booleano como valor de retorno de la función Toggle.

```
Ejemplo:
private bool toggleBool = true;

void OnGUI () {
toggleBool = GUI.Toggle (new Rect (25, 25, 100, 30), toggleBool, "Toggle");
}
```

Toolbar. Este control es esencialmente una fila de Buttons. Sólo uno de los botones de la Toolbar puede estar activo en un momento dado y se mantiene activo hasta que un botón diferente es seleccionado. Este comportamiento emula el de una típica barra de herramientas y, en ella, se puede definir un número arbitrario de botones.

El botón activo en la Toolbar se *trackea* a través de un entero, que debe ser provisto como argumento de la función. Para hacer que la Toolbar sea interactiva, se debe asignar el entero al valor de retorno de la función. El número de elementos en el arreglo de contenido determina la cantidad de Buttons que se mostrará en la Toolbar.

Ejemplo:
```
private int toolbarInt = 0;
private string[] toolbarStrings = {"Toolbar1", "Toolbar2", "Toolbar3"};

void OnGUI () {
            toolbarInt = GUI.Toolbar (new Rect (25, 25, 250, 30), toolbarInt,
toolbarStrings);
}
```

SelectionGrid. Este control SelectionGrid es una Toolbar de múltiples filas y se debe determinar el número de columnas y filas de la grilla que contendrá. Sólo un botón puede estar activo en un instante determinado.

El botón activo en la SelectionGrid se *trackea* a través de un entero que debe ser provisto como argumento en la función. Para interactuar con el SelectionGrid, se debe asignar el entero al valor de retorno de la función. El número de elementos en el arreglo de contenido determina la cantidad de Buttons que se mostrará en el SelectionGrid. También se puede indicar el número de columnas dentro de los argumentos de la función.

Ejemplo:
```
private int selectionGridInt = 0;
private string[] selectionStrings = {"Grid 1", "Grid 2", "Grid 3", "Grid 4"};

void OnGUI () {
            selectionGridInt = GUI.SelectionGrid (new Rect (25, 25, 300, 60),
selectionGridInt, selectionStrings, 2);
}
```

HorizontalSlider / VerticalSlider. Estos son los típicos controles de desplazamiento que pueden ser arrastrado para cambiar un valor entre un mínimo y un máximo predeterminados. Como se puede observar, ambos componentes difieren únicamente en la orientación del mismo.

La posición de la "perilla" del slider (que es un botón) se almacena como *float*. Para mostrar la posición de la perilla se debe proveer el float como uno de los parámetros de la función y hay dos valores adicionales que sirven para determinar el mínimo y el máximo. Si se desea hacer que el botón del slider sea ajustable se debe asignar el valor float como valor de retorno de la función del Slider.

Ejemplo:
```
private float hSliderValue = 0.0f;
private float vSliderValue = 0.0f;

void OnGUI () {
          hSliderValue = GUI.HorizontalSlider (new Rect (25, 25, 100, 30), hSliderValue,
0.0f, 10.0f);
          vSliderValue = GUI.VerticalSlider (new Rect (25, 100, 100, 30), vSliderValue,
10.0f, 0.0f);
 }
```

HorizontalScrollbar / VerticalScrollbar. Estos controles representan barras de desplazamiento y se utilizan para navegar el ScrollView Control.

La implementación de estos controles es idéntica a la de los HorizontalSlider y VerticalSlider con una excepción: hay un parámetro adicional que controla el ancho o alto del botón de desplazamiento del control.

Ejemplo:

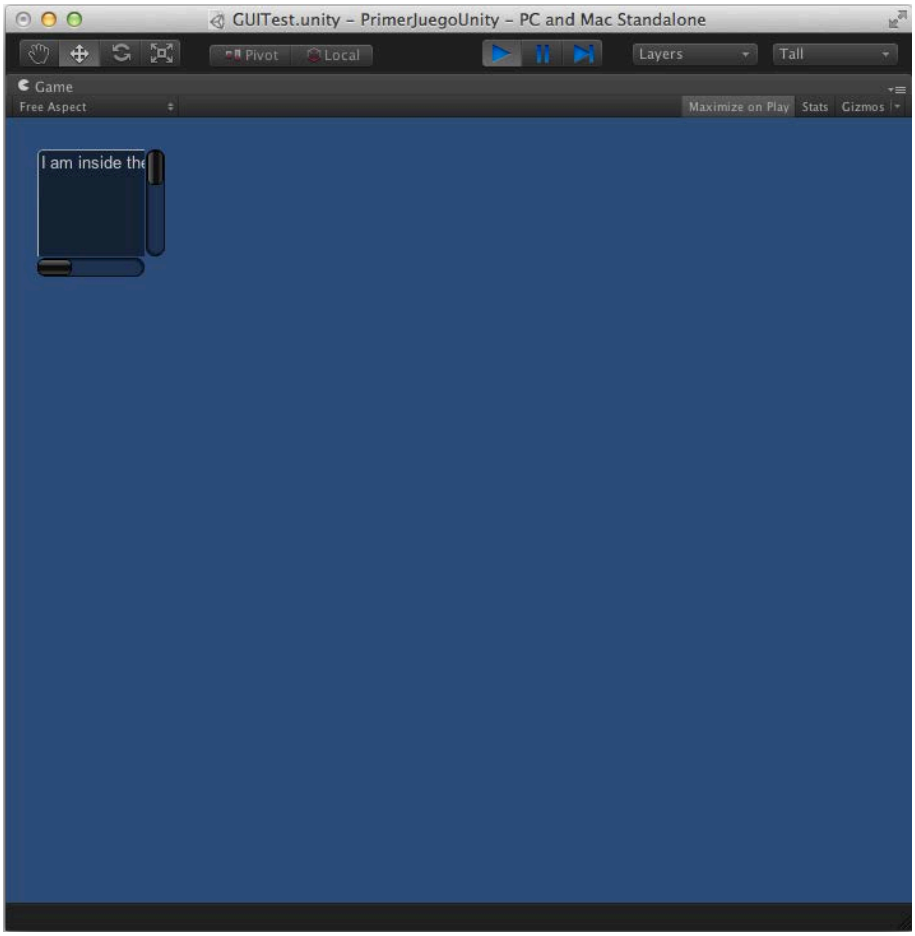
```
private float hScrollbarValue;
private float vScrollbarValue;

void OnGUI () {
        hScrollbarValue = GUI.HorizontalScrollbar (new Rect (25, 25, 100, 30),
hScrollbarValue, 1.0f, 0.0f, 10.0f);
vScrollbarValue = GUI. VerticalScrollbar (new Rect (25, 25, 100, 30), vScrollbarValue,
1.0f, 10.0f, 0.0f);
}
```

ScrollView. Son controles que muestran un área visible de un set mayor de controles.

Los ScrollViews requieren dos *Rects* como argumentos. El primero define la ubicación y el tamaño del área visible del ScrollView en la pantalla. El segundo define el tamaño del espacio contenido dentro del área visible. Si es necesario aparecerán Scrollbars. También se deberá proveer y asignar un Vector 2D que almacene la posición del área visible que se está mostrando.

Esto puede verse en un ejemplo concreto:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07
08              private Vector2 scrollViewVector = Vector2.zero;
09              private string innerText = "I am inside the ScrollView";
10
11
12              void OnGUI () {
13                      // Begin the ScrollView
14                      scrollViewVector = GUI.BeginScrollView (new Rect (25, 25,
15  100, 100), scrollViewVector, new Rect (0, 0, 400, 400));
16
17
18                      // Put something inside the ScrollView
19                      innerText = GUI.TextArea (new Rect (0, 0, 400, 400), innerText);
20
21
22                      // End the ScrollView
23                      GUI.EndScrollView();
24              }
25
26
27  }
```



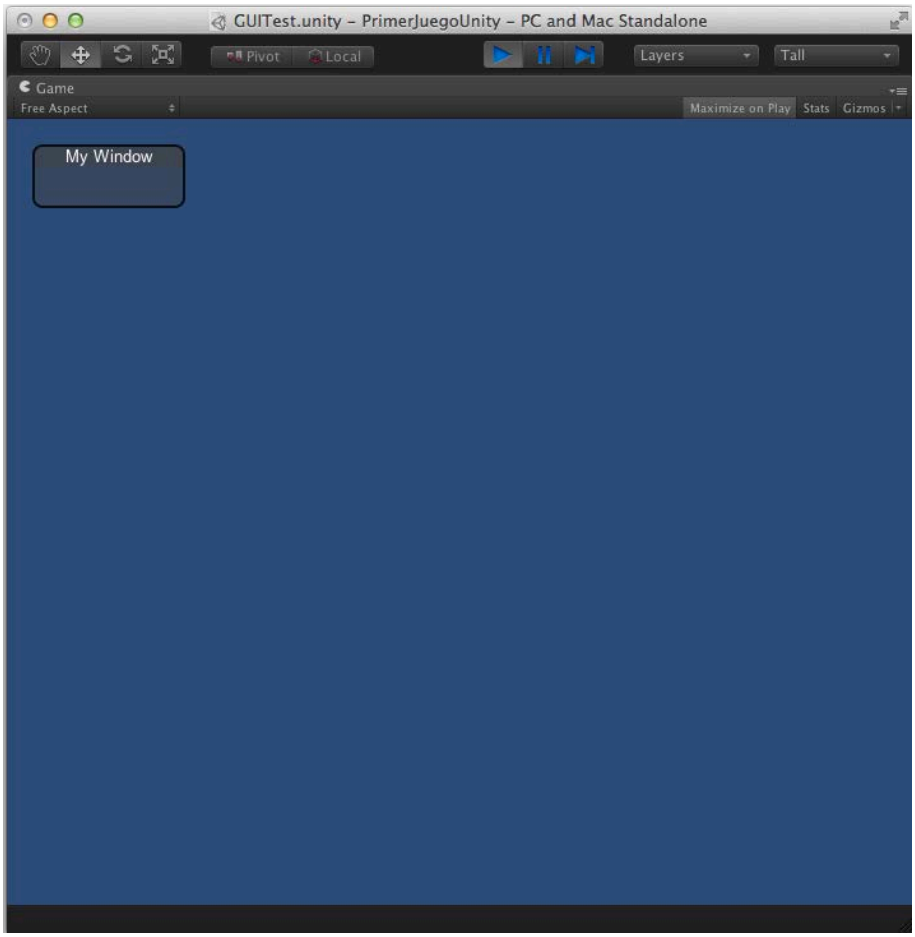
Window, Las ventanas son contenedores de controles que se pueden arrastrar. Pueden recibir y perder el foco cuando se hace click sobre ellas. Debido a esto, están implementadas de manera diferente del resto de los Controls. Cada Window tiene un número de "id" y sus contenidos están declarados dentro de una función separada que se llama cuando la ventana obtiene el foco.

Es el único tipo de Control que requiere una función adicional para ejecutarse adecuadamente. Se debe proveer un número de "id" y el nombre de la función a ser ejecutada por la ventana y, dentro de ella, se crean los comportamientos o controles contenidos.

Un ejemplo sería el siguiente:

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07          private Rect windowRect = new Rect (20, 20, 120, 50);
08
09
10          void OnGUI () {
11                  windowRect = GUI.Window (0, windowRect, WindowFunction,
12  "My Window");
13          }
14
15
```

```
16          void WindowFunction (int windowID) {
17                  // Draw any Controls inside the window here
18          }
19
20
21          }
```

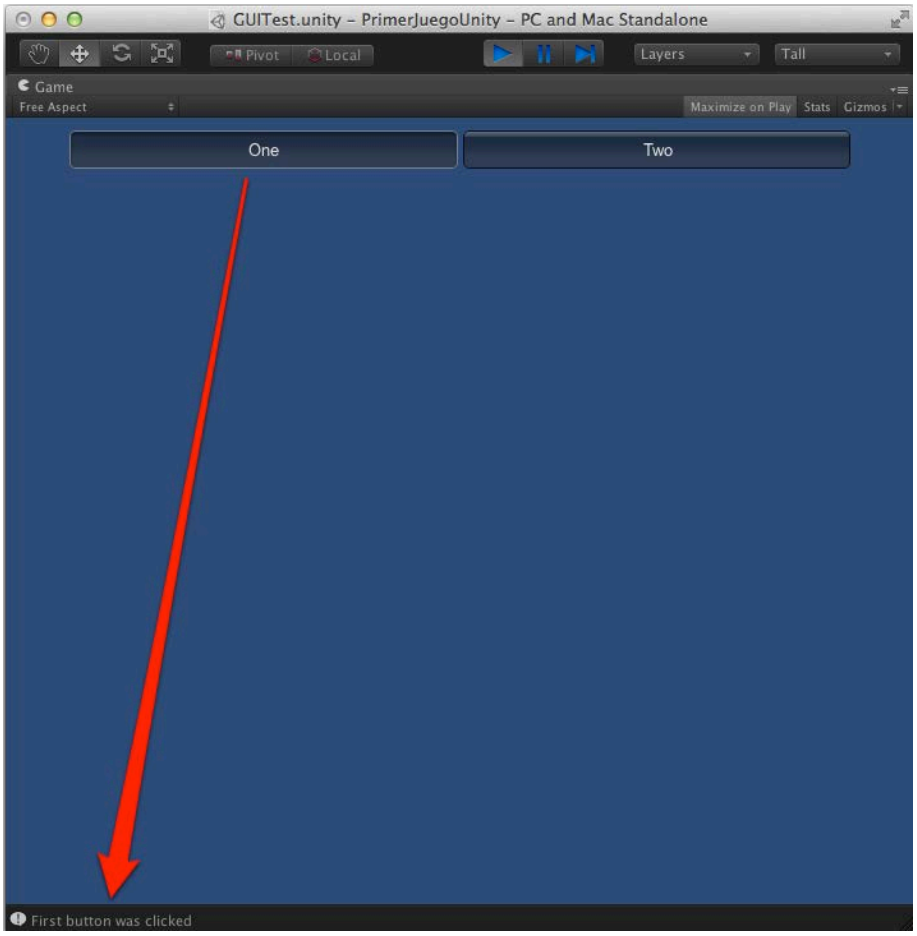


GUI.changed. Para detectar si el usuario realizó alguna acción en la GUI (si presionó un botón, arrastró un slider u otras), se puede leer el valor de GUI.changed del script. Este se *setea* en "true" cuando el usuario ha hecho algo, lo cual permite validar los inputs del usuario.

Un escenario común sería el de una Toolbar donde se quiere chequear un valor específico en base a qué botón en la Toolbar fue presionado. No se desea asignar el valor en cada llamada al método OnGUI(), sino sólo cuando uno de los Buttons ha sido presionado.

```
01  using UnityEngine;
02  using System.Collections;
03
04
05  public class GUITest : MonoBehaviour {
06
07
08          private int selectedToolbar = 0;
09          private string[] toolbarStrings = {"One", "Two"};
10
11          void OnGUI () {
```

```
12                      // Determine which button is active, whether it was clicked
13  this frame or not
14                      selectedToolbar = GUI.Toolbar (new Rect (50, 10,
15  Screen.width - 100, 30), selectedToolbar, toolbarStrings);
16
17                      // If the user clicked a new Toolbar button this frame, we'll
18  process their input
19                      if (GUI.changed)
20                      {
21                              Debug.Log("The toolbar was clicked");
22
23                              if (0 == selectedToolbar)
24                              {
25                                      Debug.Log("First button was clicked");
26                              }
27                              else
28                              {
29                                      Debug.Log("Second button was clicked");
30                              }
31                      }
32              }
33
34  }
35
36
37
38
39
```



GUI.changed devuelve “verdadero” si cualquier GUI Control ha sido manipulado por el usuario.