Listas

¿Qué es una lista?

Una lista es un tipo especial que, en lugar de ser una única variable, es una lista de elementos de un mismo tipo. En definitiva, es una lista de variables.

En videojuegos se utilizan principalmente para decidir un aleatorio de entre varios (por ejemplo, el pokémon salvaje que saldrá de la hierba alta, o que a veces suene un sonido de puñetazo y a veces otro, para que no se vuelvan repetidos).

Además de para decidir elementos aleatorios de entre un conjunto, también se utilizan, en definitiva, para cualquier acción que necesite hacerse sobre varios elementos.

Por ejemplo:

- Recorrer los enemigos que entran dentro de un área para bajarles la vida.
- Recorrer las unidades seleccionadas en un juego de estrategia para darles a todas la misma orden.
- Representar los objetos de un inventario hasta un máximo de capacidad.

Cómo decirle al código que vamos a utilizar listas

Para poder utilizar las listas, tenemos que incluir, al principio del archivo, el código:

```
using System.Collections.Generic;
```

Este código ya nos lo incluye Unity automáticamente al crear un script.

Declarar una lista

En definitiva, una lista es un tipo de variable más, como las clases de Unity `Rigidbody`, `GameObject`, etc. Solo que al declarar la variable de tipo lista, tenemos que especificarle también de qué tipo van a ser los elementos de dentro de la lista.

```
List<int> miListaDeEnteros = new List<int>();
```

Acceder a un elemento del lista

Para acceder a un elemento de un lista basta con utilizar la notación `miLista[i]`, siendo `i` el elemento del lista al que queremos acceder.

Pero cuidado, porque en los listas se empieza a contar desde el 0, por lo tanto el elemento `miLista[0]` es el primer elemento de la lista. Suponiendo hubiéramos añadido 5 elementos a la lista, el elemento `miLista[4]` sería el quinto y último. Si intentásemos acceder al elemento `miLista[5]` el programa daría error porque nos estaríamos saliendo del lista, ya que no existe un sexto elemento.

Por ejemplo, con el código `miLista[2]` estamos accediendo al número almacenado en la tercera posición del lista.

Una vez accedemos a un elemento del lista, podemos tratarlo como si fuera una variable cualquiera, tanto para modificarlo, como para utilizarlo, ya sea mostrándolo por pantalla, comparándolo con otro número dentro de un if, etc.

```
// Declaración
List<int> miLista= new List<int>();

// Para añadir elementos a la lista:
miLista.Add(7);
miLista.Add(11);
miLista.Add(55);
miLista.Add(117);
miLista.Add(13);

// Modificar el 4º elemento del lista:
miLista[3] = 99;
// Ahora el 4º elemento del lista es 99
// en lugar de 117

// Podemos pedir el primer elemento del lista
// para usarlo en comparaciones
if (miLista[1] == 0)
{
 Console.WriteLine("El segundo elemento es igual a 0");
}

// También podemos escribirlo por pantalla:
Console.WriteLine("El último elemento es " + miLista[4]);
```

Recorrer un lista mediante un bucle For

Podemos hacer un bucle for que vaya desde 0 hasta el tamaño del lista, y utilizar la variable `contador` para acceder a los distintos elementos del lista.

```
// Supongamos que hemos añadido 5 elementos a la lista.
// Tenemos los elementos miLista[0], miLista[1]... así hasta miLista[4]
// Este bucle pasa por 0, 1, 2, 3, 4...
// y a la de 5 se sale porque ya no cumple la condición
for (int contador = 0; contador < 5; contador++)
{
 Console.WriteLine(miLista[contador]);
}
```

La variable contador se suele llamar `i`, que viene de *index* o índice.

```
for (int i = 0; i < 5; i++)
{
 Console.WriteLine(miLista[i]);
}
```

También puedes aprovechar la variable contador para que los mensajes al usuario sean más claros:

```
for (int i = 0; i < 5; i++)
{
 Console.WriteLine("El elemento nº " + i + " es " + miLista[i]);
}
```

Recorrer lista hasta su tamaño, sea cual sea

Podemos acceder más adelante a la longitud del lista sin necesidad de recordar su tamaño con el código `miLista.Count`. Por ejemplo, tenemos la lista del ejemplo del principio:

```
// Declaración
List<int> miLista= new List<int>();

// Para añadir elementos a la lista:
miLista.Add(7);
miLista.Add(11);
miLista.Add(55);
miLista.Add(117);
miLista.Add(13);
```

En el caso caso, `miLista.Count` devuelve 5. A ojos del programa es exactamente lo mismo escribir `miLista.Count` que escribir `5` tal cual, ya que internamente lo reemplaza por un 5, pero... ¿Y si en el código que estamos escribiendo no sabemos el tamaño que tiene la lista porque nos viene creada de antemano? Este caso ocurre a menudo en Unity, por lo que siempre es mejor usar `miLista.Count`.

```
for (int i = 0; i < miLista.Count; i++)
{
 // Your code here
}
```

Recorrer lista del revés

```
for (int i = miLista.Count - 1; i >= 0; i--)
{
 // Your code here
}
```

Inicializar listas: rellenar un lista con tantos elementos como indique el usuario

También podemos permitir al usuario definir el tamaño del lista.

Antes de asignar la variable miLista recién declarada, le preguntaremos que cuantos elementos quiere, y después inicializaremos el lista hasta los elementos que haya indicado el usuario.

```
List<int> miLista = new List<int>();
int listSize;

Console.WriteLine("¿De cuánto tamaño quieres la lista?");
listSize = Convert.ToInt32(Console.ReadLine());

for (int i = 0; i < listSize; i++)
{
 Console.WriteLine("Inserta el elemento Nº" + i);
 miLista.Add(Convert.ToInt32(Console.ReadLine()));
}
```

Comprobar si un lista contiene un elemento

Por ejemplo, imaginemos un programa al que le introducimos 10 números, y nos tiene que decir si entre ellos hemos introducido el número 555.

```
List<int> miLista = new List<int>()

// El usuario introduce 10 números
for (int i = 0; i < 10; i++)
{
 Console.WriteLine("Introduzca el elemento " + i);
 miLista.Add(Convert.ToInt32(Console.ReadLine()));
}

// ¿Ha introducido el usuario el número 666?
// Si el usuario ha introducido el número 666 está poseído
bool usuarioPoseido = false;
for (int i = 0; i < miLista.Count; i++)
{
 if (miLista[i] == 666)
 {
  usuarioPoseido = true;
 }
}

// Le contamos al usuario si está podeído o no
if (usuarioPoseido == true)
{
 Console.WriteLine("Usuario, estás poseído. Siento ser yo quien te de la noticia");
}
else
{
 Console.WriteLine("¡Enhorabuena! ¡Ningún demonio mora en tu interior!");
}
```

Comprobar si una lista contiene un elemento más fácilmente

Cuando necesitemos hacer cosas con las listas, como ordenarla, eliminar elementos repetidos, vaciarla, etc; podemos comprobar en la documentación oficial de C# para ver si el tipo List<> tiene ese método ya predefinido:

Documentación oficial List<> C#

En este caso lo tiene, el método Contains(), que nos devuelve `true` si la lista contiene el elemento o `false` en caso contrario.

A continuación, vamos a repetir el ejemplo anterior del usuario poseído, pero utilizando el método Contains().

```
// ...
// ¿Ha introducido el usuario el número 666?
// Si el usuario ha introducido el número 666 está poseído
bool usuarioPoseido = false;
if (miLista.Contains(666))
{
 usuarioPoseido = true;
}
// ...
```

Ejercicios de Listas

1. Declara una lista de 5 enteros que represente tus números favoritos, añádele tus 5 números favoritos, e imprime por pantalla esos 5 números (no hace falta que uses un bucle).

2. En base al ejercicio anterior, esta vez usa un bucle For para imprimir los números favoritos.

3. En base al ejercicio anterior, permite que sea el usuario el que rellena esos números, primero, y que después los muestre por pantalla.

4. Crea una lista de nombres de un tamaño definido por el usuario y permite que el usuario la rellene. Después muéstrala.

5. Crea un programa que permita al usuario rellenar una lista de nombres que represente los concursantes de un sorteo. Cuando el usuario haya rellenado todos los nombres, haz el sorteo y elige a una persona aleatoria de la lista.

6. El usuario que utiliza el programa es el organizador de un concurso. Primero, pregúntale como se llama. Después, permítele rellenar una lista de 10 participantes, para después hacer un sorteo y elegir a uno aleatoriamente. Antes de hacer el sorteo, comprueba que el organizador no haya hecho trampa y se haya metido a sí mismo.

Anexo: generar número aleatorio en C# (fuera de Unity)

Haciendo el Flappy Bird, vimos que para generar un número aleatorio se hacía mediante la clase `Random` y su método `Range`, tal que así: `Random.Range(min, max)`.

Sin embargo, esto solo podemos utilizarlo cuando estemos escribiendo código dentro de Unity. Fuera de Unity, tenemos que crear primero una semilla aleatoria y después extraer el número aleatorio de ella, tal que

así:

```
// Creo la semilla aleatoria...
Random miRandomSeed = new Random();

// Extraigo un número aleatorio de la semilla:
// entre 0 y 99
int numeroAleatorio = miRandomSeed.Next(0, 99);
```