

10. Un Matamarcianos

Si has completado con éxito el reto del tema anterior ya estas preparado para realizar tu primer videojuego jugable.

Gracias a la generación de números aleatorios de Benu, su sistema de detección de colisiones entre procesos y unas breves nociones sobre destrucción (Formalmente muerte) de procesos, dotaremos a nuestro videojuego de todo lo necesario para ser un matamarcianos de los años 80.

10.1 Introducción a la generación de números aleatorios en Benu

La generación de números aleatorios es un aspecto básico en cualquier videojuego para garantizar que el resultado de la ejecución no sea siempre el mismo.

En nuestro caso vamos utilizar los números aleatorios para crear un proceso enemigo capaz de moverse aleatoriamente en los ejes x, y de manera que su movimiento sea siempre impredecible.

La sintaxis de la generación de un número aleatorio es la siguiente:

```
rand (<valor_mínimo> , <valor_máximo> );
```

Donde `valor_mínimo` y `valor_máximo` son los valores numéricos del intervalo dentro del que será generado el número aleatorio. Unos ejemplos de aplicación de los números aleatorios serían los siguientes:

<pre>Rand (1 , 100);</pre>	Devuelve un valor aleatorio entre 1 y 100. Es el ejemplo más sencillo, pero utilizada por sí sola esta instrucción no tiene ningún efecto.
<pre>x = x + rand (-4 , 4);</pre>	El proceso modifica su posición en el eje x una cantidad aleatoria de píxeles comprendida entre -4 y 4.
<pre>IF (rand (0 , 1) == 0) x = x + 4; END</pre>	El proceso ejecuta la instrucción <code>x = x + 4;</code> una de cada dos veces. Es como tirar una moneda al aire, y si sale cara, realizar una acción.

Con estas nociones crea un nuevo proceso llamado enemigo, con lo que has aprendido hasta ahora puedes echarle imaginación o bien puedes consultar el enemigo en el videojuego de ejemplo asociado a este tema.

10.2 Introducción a la muerte de procesos en Benuu

Existen varias formas de matar a un proceso en Benuu, aunque vamos a ver la más sencilla de todas.

Generalmente nos interesará que un proceso muera cuando se cumpla una determinada condición. Dado que nuestros enemigos son capaces de moverse libre y aleatoriamente por la pantalla, vamos a proponer un ejemplo de muerte muy sencillo:

- Si uno de los enemigos sale fuera de los límites de pantalla, entonces morirá.

Con esto evitaremos que los enemigos salgan de pantalla y se vayan hasta el infinito, ya que si se diese ese caso estarían consumiendo memoria y recursos del PC innecesariamente.

Para matar a los enemigos haremos que utilicen la instrucción **BREAK**:

La instrucción **BREAK**; no mata directamente al proceso, pero hace que éste abandone su bloque **LOOP-END**, y como después del **LOOP-END** no hay más instrucciones el proceso deja de ejecutar instrucciones y muere. Es una forma relativamente sencilla y limpia de matar procesos.

El código a añadir en nuestro enemigo, dentro de su bloque **LOOP-END**, sería el siguiente:

```
IF ( x < 0 )           //Si sale de la pantalla por la izquierda
    BREAK;             //Abandona el LOOP y muere
ELSIF ( x > 800 )      //Si sale de la pantalla por la derecha
    BREAK;             //Abandona el LOOP y muere
ELSIF ( y < 0 )        //Si sale de la pantalla por arriba
    BREAK;             //Abandona el LOOP y muere
ELSIF ( y > 600 )      //Si sale de la pantalla por abajo
    BREAK;             //Abandona el LOOP y muere
END                   //Fin de las condiciones
```

Otra forma que ofrece Benuu para conseguir el mismo resultado sería la siguiente, haciendo uso de los operadores lógicos, en este caso **OR**, que sirve para activar una condición cuando se cumpla **una cualquiera** de sus "subcondiciones":

```
IF ( ( x < 0 ) OR ( x > 800 ) OR ( y < 0 ) OR ( y > 600 ) )
    BREAK;
END
```

De la misma forma que **OR**, podemos usar el operador lógico **AND**, que serviría para activar una condición cuando se cumplan **todas** sus subcondiciones. Aunque es algo que ya iremos viendo a medida que avance el temario.

10.3 Detección de colisiones entre procesos

El funcionamiento de nuestro matamarcianos será así de sencillo:

- Si un enemigo colisiona con nuestro protagonista, entonces nuestro protagonista morirá.
- Si un enemigo colisiona con un disparo, entonces el enemigo morirá.

Para ello haremos uso de la función `collision()`; cuya sintaxis es la siguiente:

```
collision ( type <nombre_del_proceso> )
```

La expresión anterior activará un condicional si el proceso que la ejecuta está colisionando en pantalla con cualquier proceso del tipo dado por `<nombre_del_proceso>`.

Por ejemplo, para que los enemigos mueran al colisionar con un proceso de tipo disparo deberían ejecutar el siguiente código dentro de su bloque LOOP-END:

```
IF ( collision ( type disparo ) )  
    BREAK;  
END
```

10.4 Generación aleatoria de enemigos

Hemos definido el proceso enemigo, pero no lo hemos invocado. De momento haremos que sea el protagonista quien genere aleatoriamente enemigos en el escenario, no es la mejor solución pero sí la más sencilla. Para ello el protagonista debería ejecutar las siguientes instrucciones:

```
IF ( rand ( 0 , 10 ) == 1 )           //Si un dado de 10 caras saca como resultado 1  
    enemigo ( ) ;                   //Invocamos un enemigo  
END                                   //Fin del condicional
```

Y para que los enemigos aparezcan aleatoriamente en las esquinas de la pantalla podemos añadir estas instrucciones iniciales, antes de su bloque LOOP-END:

```
x = rand ( 0 , 1 ) * 800;           //Su posición x podrá valer o bien 0 o bien 800  
y = rand ( 0 , 1 ) * 600;           //Su posición y podrá valer o bien 0 o bien 600
```

Nota que el carácter '*' (Asterisco) es el signo de multiplicación en Benu.