UD02 - Transiciones CSS

Angel Berlanas

November 3, 2019

Contents

1 Transiciones CSS

1.1 ¿Qué son las transiciones CSS?

La manera más sencilla de animar elementos del DOM es utilizando Transiciones CSS. En esta unidad veremos como funcionan estas transiciones CSS y como utilizarlas en nuestras páginas web.

1.2 Propiedades

A continuación vamos a ver las diferentes propiedades que puede tener una transición CSS, tener en cuenta que muchos de los conceptos que se vean aquí se **re-aprovecharán** en las **Animaciones CSS**.

1.2.1 transition-property

Se trata de la propiedad que vamos a transicionar, o que va a cambiar en la transición, ya sea el `background-color` o `width`.

```
.transicionable{
     transition-property: background-color;
}

.transicionado{
     background-color: #FABADA;
}
```

Cuando a un elemento del DOM que tenga la clase `transicionable` se le aplique la clase `transicionado` el color de fondo del elemento cambiará.

Los siguientes atributos se encargan de modificar *cómo* se realizará ese cambio de las propiedades que *transicionan*.

1.2.2 transition-duration

Esta propiedad indica el número de segundos (`s`) que dura la transición. Se trata de un parámetro **requerido** si usamos la propiedad combinada (ver más adelante).

```
.transicionable{
     transition-property: background-color;
     transition-duration: 2s;
}

.transicionado{
     background-color: #FABADA;
}
```

1.2.3 transition-timing-function

Uno de los conceptos más complicados y que merecen más explicación, no es sencillo de explicar, pero intentaremos acercarnos lo más posible.

```
.transicionable{

     background-color: blue;

     transition-property: background-color;
     transition-duration: 2s;
     transition-timing-function: linear;
}

.transicionable:hover{
     background-color: #FABADA;
}
```

En este caso, lo que ocurre es que cambia el color de manera *constante* a lo largo de los dos segundos que dura la transición. Si lo reproducís en vuestro navegador, previo paso de crear los ficheros necesarios, vereis como al poner encima el cursor del elemento, va cambiando de color de manera constante hasta llegar al color `#FABADA`.

Esto en los cambios de color no se aprecia tanto, pero vamos a ver ahora una transición un pelín diferente, un cambio de posición respecto a la posición en **X**.

La modificación de la *velocidad* de transición nos permite dar efectos con más presencia en la naturaleza, lo que hace que la experiencia del usuario sea más confortable al ser más *cercana a la realidad*. Un ejemplo de esto es que si tenemos que desplazar un elemento desde una posición a otra en una ventana, el usuario verá más *natural* que el movimiento comience lento, acelere y luego frene al llegar a su destino, de la misma manera que lo hacemos nosotros y podemos observarlo en el mundo real.

Para much@s esta diferencia es muy sutil, pero puede ser la diferencia entre un diseño muy bueno y otro excelente.

Para abordar con éxito el ejemplo presentaremos dos conceptos que ayudarán a comprenderlo mejor.

- transition: transform;
- transform: translateX(Npx);

Figure 1: El diseño según Dilbert

1.2.4 transform

Es una propiedad `css` que nos permite modificar el espacio de coordenadas del modelo de formato visual CSS. Usándola, los elementos pueden ser trasladados, rotados, escalados o sesgados de acuerdo a los valores establecidos

Se trata de una de las propiedades más usadas en las las transiciones y animaciones, ya que nos permite mover los elementos a diferentes puntos de la ventana desde el CSS.

`transform` presenta muchas posibilidades, pero para este ejemplo nos centraremos en una muy sencilla: `translateX`.

1.2.5 translateX

Esta transformación *reposiciona* un elemento horizontalmente en un plano 2D. Veremos algunos ejemplos de esto más adelante.

```
/* <length-percentage> values */
transform: translateX(200px);
transform: translateX(50%);
```

Esto desplazará el objeto `200px` hacia la derecha y en el caso de la segunda opción lo trasladará la mitad de la pantalla.

Una vez explicadas estas dos propiedades, continuamos con la explicación de las diferentes formas de *animar* una transición. Volvamos a **transition-timing-function**.

1.2.6 Diferentes valores para transition-timing-function

1. ease-out

Representa el efecto de ir deteniéndose paulatinamente antes de llegar al destino, como si se acabara el impulso.

2. ease-in

   En este caso la animación *acelerará* hasta llegar a una velocidad de crucero y acabará la animación a esa velocidad.

3. ease-in-out

   Una combinación de ambas, representa el inicio lento, acelerar, y luego frenar cuando esté cerca del final.

4. cubic-bezier

   Si las combinaciones anteriores no acaban de ajustarse al efecto que necesitamos, o sencillamente se nos pide que sea de otra manera, por ejemplo : *rápido-lento-rápido*, deberemos `programar` nuestra propia animación haciendo uso de las `curvas-bezier`.

   Esto lo veremos más adelante en esta unidad, pero ahora vamos a realizar unos ejercicios para afianzar lo aprendido.

1.2.7 Ejercicio 10

Utilizando el código fuente suministrado, crear o realizar los cambios correspondientes en los ficheros :

- base.css
- script.js

Para que los diferentes Cthulhus transicionen con los diferentes valores vistos hasta ahora al pulsar el botón.

1. SubMisión 01:

   El código asociado a las clases `css`: ease, ease-in, ease-out, ease-in-out, tan solo se compone de 1 línea.

2. SubMisión 02:

   El código JS es tan solo una línea de código, sin `;`. Para ello, se puede utilizar las funciones lambda.

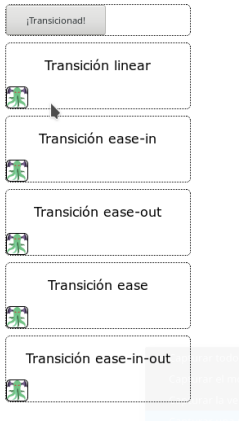

Figure 2: Ejercicio 10 : Inicio

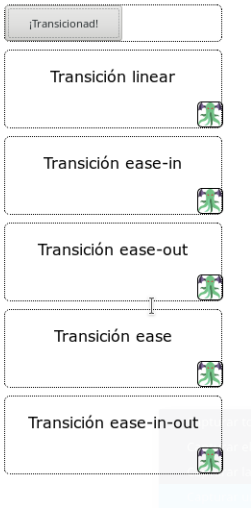

Figure 3: Ejercicio 10 : Fin

1.2.8 cubic-bezier

Las curvas bezier, es una notación que se utilizan en multitud de programas de diseño 3D y 2D como el AutoCAD o Gimp que indican como se deben modificar unas curvas para adaptarse a una determinada función.

No es *relevante* que seamos capaces de crear nosotros los números que forman la función *bezier* de manera manual, para ello contamos las *herramientas de developers* en los navegadores que nos permiten crear nuestras

curvas bezier de forma gráfica.

https://developer.mozilla.org/es/docs/Web/CSS/Herramientas/Cubic_Bezier_Generator

Esto nos permite crear animaciones mucho más personalizadas.

1.2.9 Ejercicio 11

Utilizando el código fuente del ejercicio anterior, sustituye las palabras clave de la transición: `ease,ease-in,ease-out,ease-in-out y linear` por una llamada a la función `cubic-bezier` manteniendo la misma **UX** (***User Experience***).

1.2.10 Transicionando 2 o más propriedades

Podemos *transicionar* 2 (o más) propiedades `CSS` separandolas mediante una coma (˜,˜), en la propiedad `transition` o `transition-property`.

```
.selector {
  transition: background-color 1s ease-out,
      color 1s ease-out;

  /* O tambien */
  transition-property: background, color;
  transition-duration: 1s;
  transition-timing-function: ease-out;
}
```

Podemos vernos tentandos de hacer lo siguiente:

```
/* NUNCA HAGAIS ESTO */
.selector {
  transition-property: all
}

/* ESTO ESTA BIEN */
.selector {
  transition-property: background-color, color, transform;
}
```

Esto es debido a que tendrá un impacto *muy negativo* en el rendimiento de las transiciones, al igual que pasa con las excepciones de `try-catch` en Java, cuanto más cerca estés del *problema*, más fácilmente el motor genera el código adecuado y es capaz de trabajar con la transición (*o la excepción,...*).

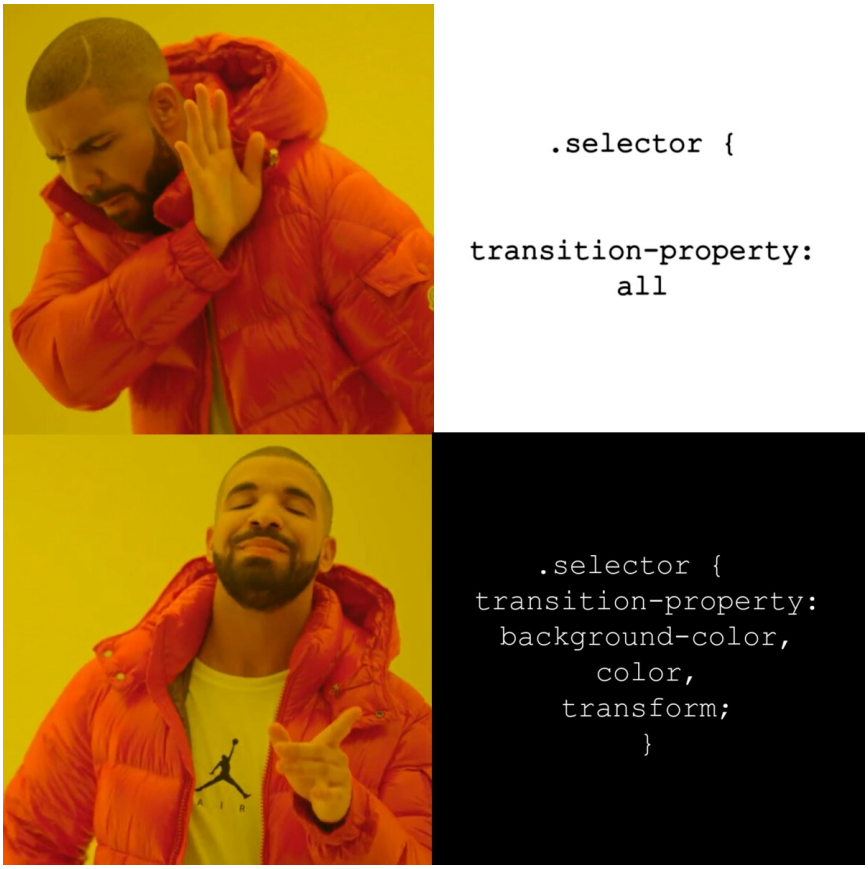

Figure 4: Como hacer las transiciones

1.2.11 transition-delay

Por último la propiedad `transition-delay` nos permite especificar un tiempo de espera para realizar la transición *después* de que el cambio haya sido realizado en el código.

```
.transicionable{
     transition-delay: 1s;
}
```

1.2.12 Ejercicio 12 : Restaurando pergaminos

En el antiguo Egipto se le rendia culto a *Sedefkar* un sacerdote que según cuenta la leyenda, era capaz de invocar a peligrosas criaturas que realizaban pequeñas tareas para mantener el inmenso poder de *Sedefkar*.

Figure 5: Ejercicio 12 - Restaurando el pergamino

En las últimas semanas se ha encontrado un pergamino que contiene una plantilla que puede usarse para volver a invocar a los *Seres de Sedefkar*.

Sin embargo el tiempo ha hecho mella en el pergamino y se han perdido fragmentos de los otros dos pergaminos a los que hace referencia ese pergamino, uno escrito en la lengua de los antiguos Dioses (`JS`) y otro perteneciente a la *Casta Sacerdotal de Sedefkar* (`CSS`).

1. Tareas:

   Se pide a los investigadores que recompongan los pergaminos, sabiendo lo debe pasar con los seres invocados.

   - Cuando se pulse el botón, debe aparecer una `caja`. Esta se encuentra en un estado latente, esperando.

- Si se pulsa sobre la `caja`, esta *despierta*, moviendose hacia la izquierda hasta situarse sobre otra caja (si la hubiera) y cambia su color.
- Si se vuelve a pulsar sobre esa caja, vuelve a la posición anterior y recupera su color anterior, pero ahora, está a punto de mostrar su verdadero poder.
- Si volvemos a pulsar sobre la `caja`, aparece un *Ser de Sedefkar* y queda a la espera, ahora si pasamos el ratón sobre la caja, esta cambia de tamaño, para indicar el poder latente que queda en el interior.

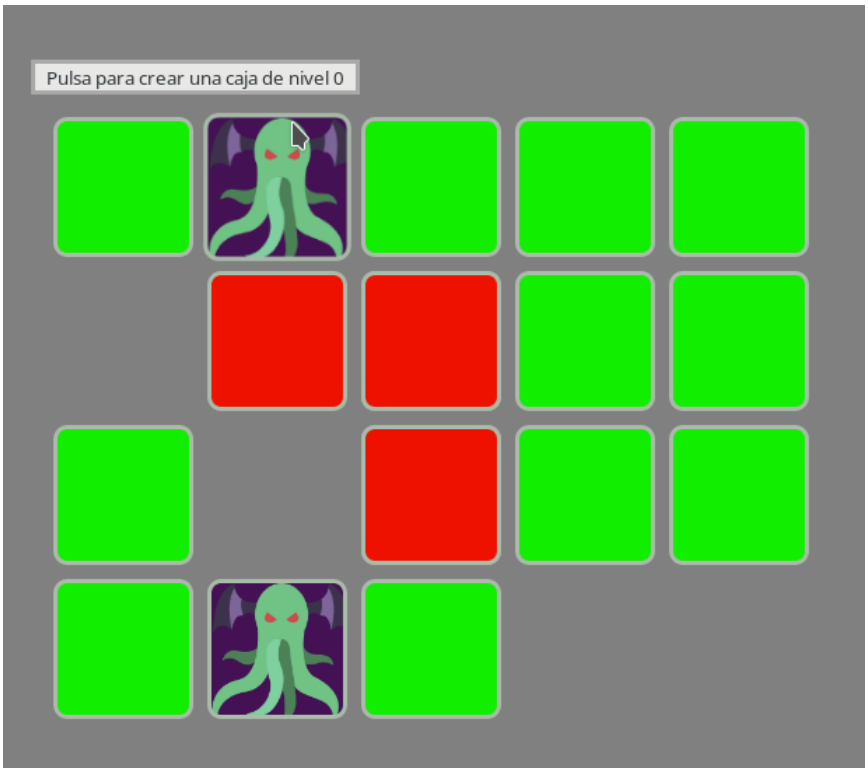

Figure 6: Ejercicio 12 - Restaurando el pergamino