Programación Avanzada 2021-2
Tutorial 2. Objetos y clases javascript

PhD. Carlos Cares

En este tutorial se ofrece un conjunto de pasos que el estudiante debe seguir para poner en práctica el uso de objetos y clases en javascript. Lo que haremos en este tutorial es una implementación sencilla de lo que se conoce como el "juego de la vida". El juego de la vida es un ambiente de células vivas y muertas con reglas de cómo dichas células se reproducen o mueren de un ciclo de tiempo a otro. Debido a que es una concepción del matemático Conway, también es conocido como el juego de Conway.

La forma de entregar los resultados es PEGANDO las imágenes solicitadas en el canal personal de slack en el curso. Las imágenes de pantalla deben incluir la fecha y la hora del sistema operativo y, además, debe incluir parte del fondo de su escritorio. Volveremos a revisar algunos conceptos del tutorial anterior y se explicarán sucintamente los conceptos usados.

PARTE-1 (30%). En este ejemplo revisaremos algunas variantes de estilo y la posibilidad de tener inclusiones de estilo que no estén en la cabecera. En términos del problema del "Juego de la Vida" implementaremos una pequeña grilla, usando una tabla HTML, con estilos definidos para una célula viva (usaremos color verde) y una célula muerta (color gris).

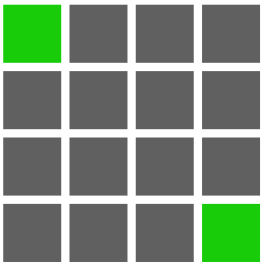
Parte 1 - paso 1. Cree el siguiente archivo de estilo y llámelo conway.css. Lo que hacemos en este archivo de estilo es darle una dimensión específica de alto y ancho como un elemento común llamado `celula` y un color diferente en los estilos `viva` o `muerta`.

```css
.viva {
    color: rgb(27, 202, 11);
    background-color: #18cc09;
    border-color: #126006;
}
.muerta {
    color: #707070;
    background-color: #606060;
    border-color: #404040;
}
.celula {
    padding-top: 7px;
    padding-left: 7px;
    padding-right: 7px;
    padding-bottom: 7px;
}
.ambiente {
    table-layout: fixed;
    border-spacing: 2px;
        padding-top: 7px;
    padding-left: 7px;
    padding-right: 7px;
    padding-bottom: 7px;
}
```

Parte 1 - paso 2. Cree el siguiente archivo html.

```html
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Tutorial 2 OOJavaScript</title>
</head>
<body>
    <link rel="stylesheet" type="text/css" href="./conway.css">
    <table class="ambiente">
        <tr>
            <td class="celula viva"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
        </tr>
        <tr>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
        </tr>
        <tr>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
        </tr>
        <tr>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula muerta"></td>
            <td class="celula viva"></td>
        </tr>
    </table>
</body>
</html>
```

Parte 1 - paso 3. Comprueba que la salida en el navegador es la siguiente.



Parte 1 - paso 4. Cambie el contenido del archivo html para que la grilla tenga 5 filas y 6 columnas. En este ejemplo SOLO la primera y última celda de la primera fila deben aparecer como células vivas (ninguna otra).

Note que en este problema la etiqueta <link> se encuentra en el cuerpo del archivo html y funciona igualmente.

Igualmente note que una forma de enfrentar un problema es resolver algunos elementos antes que otros probando técnicamente su factibilidad. En este caso, probamos primero el despliegue gráfico.

Debe reportar con las siguientes etiquetas:

Tu2-P1-A Foto del código donde se configura que la tabla tiene 5 filas y 6 columnas pedido en el paso 4.

Tu2-P1-B Salida en el navegador del ambiente de Conway.

PARTE-2 (30%). En este caso comenzamos a mejorar la estructura de datos de la solución y creamos el ambiente como una clase javascript. Crearemos esta clase en un archivo de código javascript. La recomendación es que los archivos de clases comiencen con la palabra class, pero esto no es un estándar.

Una buena forma de identificar los elementos necesarios de una clase es usándola, antes que sea creada. Una clase javascript, al igual que en Java, usa sus propiedades y métodos después del nombre del objeto separado por un punto.

Parte 2 - paso 1. Cambie el archivo principal HTML al siguiente código- (igualmente puede "guardar como" para conservar la solución anterior.

```
1  <html>
2  <head>
3      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4      <title>Tutorial 2 OOJavaScript</title>
5      <script type="text/javascript" src="./class.AmbienteConway.js"></script>
6  </head>
7  <body>
8      <link rel="stylesheet" type="text/css" href="./conway.css">
9      <div id="grilla"></div>
10     <script>
11         var x = new AmbienteConway(6,6);
12         x.activa(0,0);
13         x.activa(5,5);
14         x.inyectaAmbiente("grilla");
15     </script>
16 </body>
17 </html>
```

Note que en la línea 5 se incluye el archivo que contendrá la clase. El link de la línea 8 lo hemos mantenido en este paso. Hemos agregado un vacío, con la intención que sea este elemento donde inyectaremos la grilla del ambiente de Conway.

Con todo esto, nos queda claro que necesitaremos un constructor del objeto de la clase `AmbienteConway` que recibirá la dimensión (alto,ancho). Asumiremos que todas las células creadas serán células muertas. Además, si es así, necesitamos un método que llamaremos `activa`, de modo que activaremos una célula transformándola de muerta a viva. Finalmente usamos en la línea 14 el supuesto método `inyectaAmbiente` que dejará el ambiente como contenido del elemento HTML que tenga este identificador.

Parte 2 - paso 2. Con todo lo anterior tenemos la siguiente propuesta para la clase AmbienteConway

```
1  class AmbienteConway {
2      constructor(alto,ancho) {
3          this.alto = alto;
4          this.ancho = ancho;
5          this.celula = [];
6          this.creaCelulas();
7      }
8
9      creaCelulas() {
10         var i,j;
11         for(i=0;i<this.alto;i++) {
12             this.celula[i] = [];
13             for(j=0;j<this.ancho;j++) {
14                 this.celula[i][j]=false;
15             }
16         }
17     }
18
19     activa(i,j) {
20         if(this.posicionOK(i,j)) {
21             this.celula[i][j]=true;
22         }
23     }
24
25     estaViva(i,j) {
26         if(this.posicionOK(i,j))
27             return this.celula[i][j];
28         else
29             return false;
30     }
```

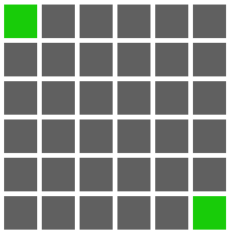
```
32     posicionOK(i,j) {
33         return (i>=0 && i<this.alto && j>=0 && j<this.ancho);
34     }
35
36     inyectaAmbiente(idDiv) {
37             var tabla,fila,celd,i,j;
38             tabla = document.createElement("TABLE");
39             tabla.setAttribute("class","ambiente");
40             for(i=0; i<this.alto ; i++) {
41                 fila = document.createElement("TR");
42                 for(j=0; j<this.ancho ; j++) {
43                     celd = document.createElement("TD");
44                     if(this.estaViva(i,j)) {
45                         celd.setAttribute("class","celula viva");
46                     }
47                     else {
48                         celd.setAttribute("class","celula muerta");
49                     }
50                     fila.appendChild(celd);
51                 }
52                 tabla.appendChild(fila);
53             }
54             document.getElementById(idDiv).appendChild(tabla);
55         }
56 }
```

El constructor (líneas 2 a 7) asigna los valores a los elementos internos. Note que como javascript no es fuertemente tipeado (strongly typed) no necesita definir los tipos de sus elementos internos.

Las células del ambiente se implementan como un arreglo (matriz) de valores booleanos. Tal como en Java, el tipo matrix no existe, y es necesario decir que se trata de un arreglo unidimensional que en cada valor, contiene otro arreglo unidimensional, formado de ese modo la matriz (en la línea 5 se crea el arreglo principal y en la línea 12 se crean los arreglos dentro del arreglo principal.

Se ha generado un método interno llamado `posicionOK` para verificar si una posición cualquiera está o no dentro de los valores admitidos para la grilla, esto es para evitar direccionar (usar) un elemento inexistente de la matriz que, como resultado, nos reporte un error.

Parte 2 - paso 3. Verifique que la salida en el navegador es la siguiente:



Debe reportar con los cambios indicados y con las siguientes etiquetas:

Tu2-P2-A Altere el código SÓLO en el programa HTML para que el ambiente que aparezca sea de 8x8 y tenga como células vivas las cuatro esquinas de la grilla. Muestre esta parte del código en una foto.

Tu2-P2-B Muestre el código de la clase `AmbienteConway`,

Tu2-P2-C Salida en el navegador del ambiente de 8x8.

Parte - 3 (30%).

En esta parte implementaremos los pasos que permiten la reproducción o muerte de las células. Implementaremos el algoritmo básico de Conway que son las siguientes reglas textuales desde Wikipedia:

> - Una célula muerta con exactamente 3 células vecinas vivas "nace" (es decir, al turno siguiente estará viva).
> - Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere (por "soledad" o "superpoblación").

Primero que nada, note que dice “turno”, es decir que el estado actual tenemos un presente y luego tenemos un futuro. Si alteramos el “turno” actual de inmediato, alteraremos el resultado pues el hacer “vivir” una célula en el momento, significará que alteramos el estado actual como si fuese el futuro, con lo cual nuestra matriz tendrá al mismo tiempo valores actuales y valores futuros lo que nos provocará un error. Para evitar esto el algoritmo del siguiente paso usa un buffer, esto es, una memoria adicional, pero temporal, para el ambiente.

Parte 3 - paso 1. Implementación del algoritmo básico de conway como método.
En este paso se pide que copie el algoritmo creado como un método dentro de la clase `AmbienteConway`.

```
57      proximoTurno() {
58          var celu = [];
59          for(var i=0;i<this.alto;i++) {
60              celu[i] = [];
61              for(var j=0;j<this.ancho;j++) {
62                  var v = this.vecinasVivas(i,j);
63                  if(this.estaMuerta(i,j) && v==3)
64                      celu[i][j] = true;
65                  else if (this.estaViva(i,j) && (v==2 || v==3))
66                      celu[i][j] = true;
67                  else
68                      celu[i][j] = false;
69              }
70          }
71          this.celula = celu;
72      }
```

Note que celu, es una estructura local que se construye de la misma forma que la estructura de células de la clase, es decir, como un arreglo de arreglos de contenidos booleanos. El contenido de esta nueva estructura es el cálculo de Conway, ve cómo las líneas 62 a 68, implementan las reglas anteriormente mostradas. Así mismo, usamos métodos que no han sido creados aún, como `estaMuerta` y `vecinasVivas`, pero los creamos después que los necesitamos. Note que una vez calculado todo el nuevo turno, este simplemente asignado al componente `celula` del objeto actual. En teoría, el algoritmo interno de recolección de basura (garbage collector) tomará la memoria dejada disponible de la matriz inicial y se la devolverá al navegador para que pueda volver a usarla.

Parte 3 - paso 2. Los otros dos métodos nuevos quedan como sigue y se pide que los implemente tal cual.

```
74      estaMuerta() {
75          return !this.estaViva();
76      }
77
78      vecinasVivas(i,j) {
79          var total=0;
80          if(this.estaViva(i-1,j-1)) total++;
81          if(this.estaViva(i,j-1)) total++;
82          if(this.estaViva(i+1,j-1)) total++;
83          if(this.estaViva(i-1,j)) total++;
84          if(this.estaViva(i+1,j)) total++;
85          if(this.estaViva(i-1,j+1)) total++;
86          if(this.estaViva(i,j+1)) total++;
87          if(this.estaViva(i+1,j+1)) total++;
88          return total;
89      }
```

Note que el método `vecinasVivas` no pregunta si las posiciones pertenecen o no a la grilla, esto se puede hacer así porque `estaViva` hace este trabajo.

Parte 3 - paso 3. En este paso cambiamos la invocación del HTML para que, contenga un estado inicial diferente de células vivas, y para tener un botón que avance con los turnos de Conway. Esta es la versión del nuevo HTML.

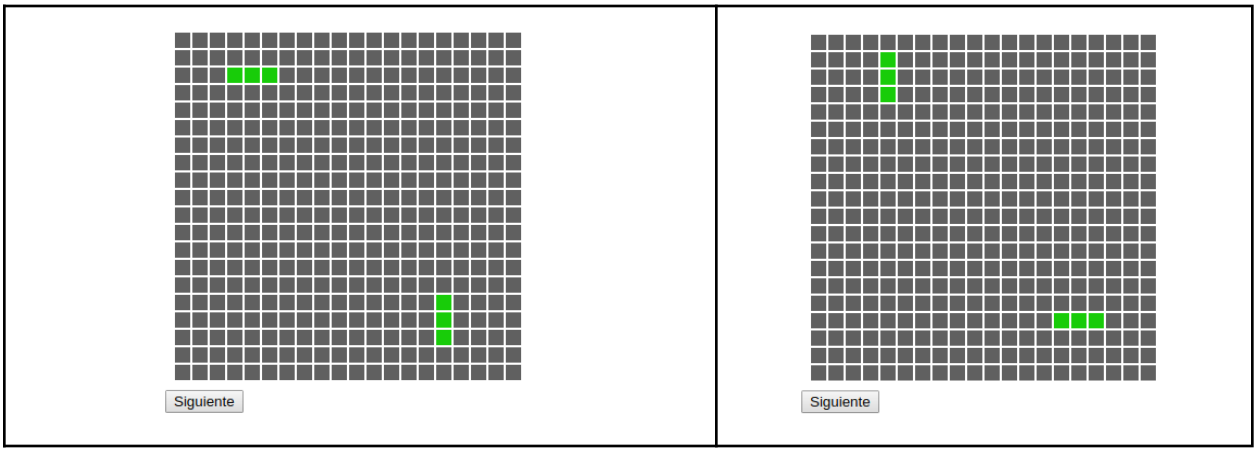
```
1  <html>
2  <head>
3      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4      <title>Tutorial 2 OOJavaScript</title>
5      <script type="text/javascript" src="./class.AmbienteConway.js"></script>
6      <script>
7          function proximoPaso() {
8              x.proximoTurno();
9              document.getElementById("grilla").innerHTML="";
10             x.inyectaAmbiente("grilla");
11         }
12     </script>
13 </head>
14 <body>
15     <link rel="stylesheet" type="text/css" href="./conway.css">
16     <div id="grilla"></div>
17     <script>
18         var x = new AmbienteConway(20,20);
19         x.activa(2,3);
20         x.activa(2,4);
21         x.activa(2,5);
22         x.activa(15,15);
23         x.activa(16,15);
24         x.activa(17,15);
25         x.inyectaAmbiente("grilla");
26     </script>
27     <button onclick="proximoPaso()">Siguiente</button>
28 </body>
29 </html>
```

Básicamente entre la línea 6 y 12 hemos agregado un manejador de evento click para el nuevo botón “Siguiente”. Note los detalles javascript. Por ejemplo, note que la función usa el objeto x, que, como no lo define, se entiende que es global. Pero el objeto x, cuando está en la línea 8 es la primera vez que aparece en el programa. Bien, note que esto no es un error, porque, como es un lenguaje interpretado, la variable global se busca en el momento en que se usa, es decir cuando se active el evento.

Esta función invoca el método `proximoTurno`, y luego deja un string vacío como contenido de este grilla. Esto se hace porque el método siguiente, inyectaAmbiente, lo que hace es que, justamente, inyecta el Ambiente en el div existente manteniendo lo anterior.

Lo otro que se ha agregado en esta versión es las activaciones de células de la línea 19 a 24, básicamente dos pequeños patrones, uno vertical y otro horizontal. En la línea 27 hemos agregado el botón para activar el cambio de Turno.

Parte 3 - paso 4. Ejecute el nuevo HTML y revise el comportamiento con el botón Siguiente. Debería obtener algo como lo que se indica:



Debe reportar con los cambios indicados y con las siguientes etiquetas:

Tu2-P3-A Fotos de los métodos nuevos: `proximoTurno` y `vecinasVivas`.

Tu2-P3-B Salida en el navegador antes y después del primer clic en el botón “Siguiente”.

Parte - 4 (10%).

Este último 10% se pide que haga un trabajo por usted mismo. No necesita conocimiento adicional ni instrucciones diferentes a las ya utilizadas. Se pide que el <link> NO ESTÉ en el HTML principal, porque es una dependencia, de este modo la clase, que utiliza el CSS sea también la que lo incluye. Entonces, lo que se pide es que lo saque del HTML y que sea la clase la que genere un elemento “LINK” (con createElement) y que asignando los atributos correspondientes, quede como parte del inyectado. Muestre su nuevo código en la clase y muestre que funciona igualmente.

Debe reportar con los cambios indicados y con las siguientes etiquetas:

Tu2-P4-A Foto del HTML sin la etiqueta <LINK>.

Tu2-P4-B Foto del código Javascript en la clase que inyecta el LINK en el HTML.

Tu2-P4-C Suba los 3 archivos de este tutorial separadamente para verificar.