Taller de Programación 2 / 2021-2
OO-Javascript y control DOM

Ph.D. Carlos Cares
carlos.cares@ceisufro.cl

Requerimientos:

- Haber realizado el Tutorial 2 de Programación Avanzada.
- Un editor de texto, idealmente un editor de desarrollo de software.
- Un navegador de Internet.
- Este tutorial se puede seguir en cualquier plataforma, Linux, Windows, o Mac.

La entrega de los resultados debe ser en el canal privado de Slack del curso, **en el momento en que obtiene las pantallas correctas** de su resultado. Las preguntas y respuestas deben quedar en el canal público. Recuerden que hay puntos porcentuales adicionales por preguntar y por responder las preguntas en el canal público. En cada problema se piden secciones de código y de resultados en el navegador de Internet, cada foto de pantalla (screenshot) debe hacerse con las siguientes características:

Características de los screenshots:

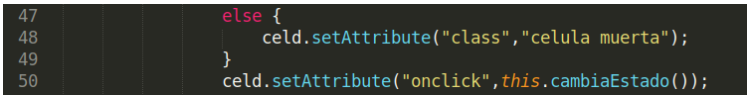
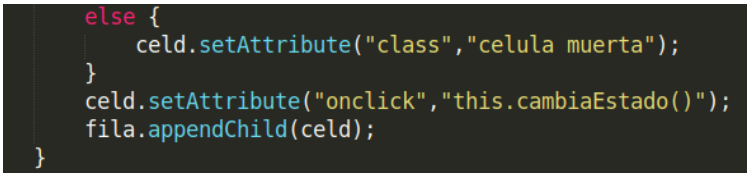
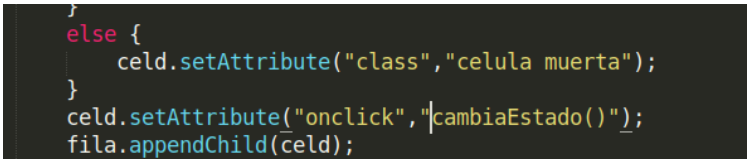
- Deben ser legibles
- Deben incluir una porción del fondo del escritorio
- Deben incluir la parte del escritorio que muestra fecha y hora

Las instrucciones revisadas en el tutorial 2 **NO son suficientes** para realizar los problemas de este Taller de Programación, por lo que en el problema 1 de este taller se ofrecen explicaciones adicionales que deberá seguir con atención.

Problema 1. (20%) Invocación de métodos de una clase javascript en tiempo de generación DOM. Explicación y Resultado.

En este Taller implementaremos, entre otras, la funcionalidad de reproducción en el ambiente de Conway del tutorial anterior, esto es habilitar que cada celda del ambiente pueda recibir un clic y cambiar el estado de la célula que representa (viva o muerta). Así que lo primero es decidir dónde estará este evento. Los elementos `TD` (celda de tabla) aceptan el evento `onclick` así que probaremos agregando en la gráfica del ambiente, la generación de este evento.

Bien, podemos probar varias opciones, y las alternativas serían las siguientes.

1. Invocación del método de la clase.	 Esto tiene un problema, y es que el método se invoca cuando se ejecuta la línea, y no cuando se ejecuta el clic, luego **no es útil** para la funcionalidad deseada.
2. Generación de la llamada del método de la clase usando comillas	 Pero, si lo piensa, cuando se presione el clic, será en el documento principal, el "this" ya no representará el objeto de la clase actual, luego **esto provocará un error.**
3. Generación de la llamada usando un método externo	 Esto, funciona de seguro, pero debemos crear el método `cambiaEstado` fuera de la clase `AmbienteConway`, **perdiendo así la orientación a objetos**.

Lo que se acaba de presentar es un problema porque la generación DOM no conserva el scope (ambiente de las variables) del momento de la generación y se pierde la referencia de qué objetos del programa generaron el documento HTML.

La forma que he ocupado como profesional del área para este problema, es más bien un "truco", y se basa en el siguiente supuesto conceptual: cuando un objeto javascript se usa para generar parte del documento HTML, podemos decir que esos objetos javascript están activos y presentes en esa parte del documento, y por lo tanto, los eventos que esos objetos generan deberían invocar a esos objetos activos. Por lo tanto, la solución que les presento es de crear

objetivos “activos” de cada clase javascript que genera elementos usando DOM. Esto lo haremos de la siguiente manera.

1. En cada clase que genera elementos usando DOM se generará una variable global que será el nombre del objeto activo de cada clase, uno por clase. Esto evita que se genere un error por nombres repetidos. En nuestro caso sería así.

   Al comienzo de la clase se necesita definir la variable global y asignarle valor inicial dentro de la clase (línea 1 y línea 4).

```
1  var objActivoAmbienteConway;
2  class AmbienteConway {
3      constructor(alto,ancho) {
4          objActivoAmbienteConway = this;
5          this.alto = alto;
6          this.ancho = ancho;
7          this.celula = [];
```

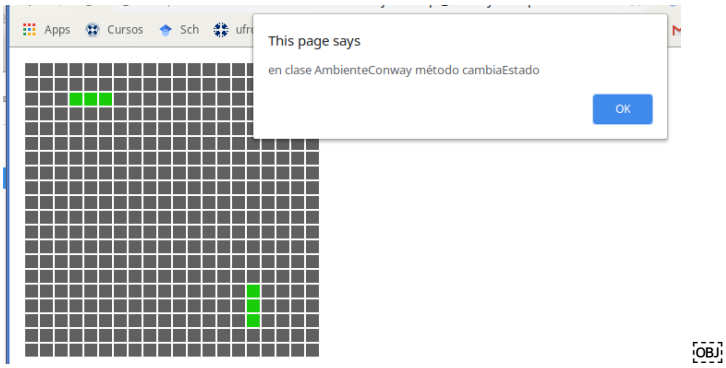
2. En la generación entonces, actualizamos el objetivo activo y usamos, en lugar de `this`, el nombre del objeto activo, en nuestro caso quedaría así:

```
51                 celd.setAttribute("class","celula muerta");
52             }
53             objActivoAmbienteConway = this;
54             celd.setAttribute("onclick","objActivoAmbienteConway.cambiaEstado()");
55             fila.appendChild(celd);
```

3. Para poder probar que funciona debemos crear una versión inicial del método DENTRO de la clase, esto podría ser así:

```
62     cambiaEstado() {
63         alert("en clase AmbienteConway método cambiaEstado");
64     }
65
66     proximoTurno() {
```

4. Con todo esto, la salida es capaz ahora de responder a un clic en una célula cualquiera. Se vería así:



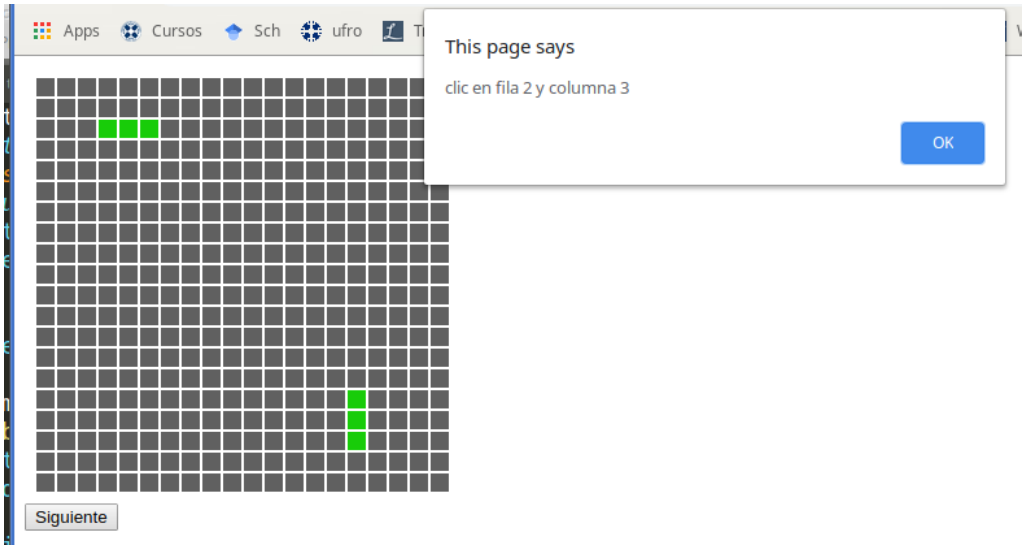
Bien, por último, necesitamos recuperar la coordenada específica de la célula para cambiar esa célula de estado, necesitamos pasar los valores de i y j. Lo hacemos así:

```
54              var st ="objActivoAmbienteConway.cambiaEstado("+i+","+j+")";
55              celd.setAttribute("onclick",st);
56              fila.appendChild(celd);
57          }
58          tabla.appendChild(fila);
```

Además el método debe recibir los parámetros, el método cambia así:

```
63      cambiaEstado(fila,columna) {
64          alert("clic en fila "+fila+" y columna "+columna);
65      }
66
```

La salida ahora debería ser de este modo después de hacer un clic en la primera célula viva.



Se piden las siguientes evidencias del trabajo con las siguientes etiquetas:

TP2-P1-A Fuente javascript del código de la clase incluyendo el objetivo activo

TP2-P1-B Secuencia de dos fotos en el navegador que muestre la ventana de alerta después del clic en dos celdas diferentes.

Problema 2 (30%). Cambiando interactivamente el Ambiente de Conway.

Usando la solución anterior provoque que la célula elegida para el clic cambie de viva a muerta o de muerta a viva. El cambio debe expresarse en el mismo ambiente desplegado que se le hizo el clic.

Ayuda: Una buena idea sería que, en el último despliegue del ambiente, el objeto ambiente guarde cuál fue el identificador del objeto HTML donde inyectó el ambiente, para que los cambios de dichas células, cambien el ambiente actual y no otro.

Se piden las siguientes evidencias del trabajo con las siguientes etiquetas:

TP2-P2-A Código fuente javascript con comentarios en las líneas correspondientes explicando cómo se produce el cambio (viva-muerta) en el ambiente.

TP2-P2-B Usando la funcionalidad dibuje una cruz en el ambiente y suba esa foto.

Problema 3 (25%). Agregando patrones.

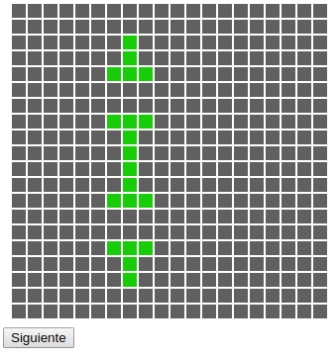
A manera de ejemplo considere el siguiente método en la clase `AmbienteConway`.

```
101     agregaPatron(fila,columna,nombrePatron) {
102         var y=fila;
103         var x=columna;
104         if(nombrePatron=="pentaDecatlon") {
105             console.log("lllega");
106             for(var i=x; i<(x+8); i++)
107                 for(var j=y; j<(y+3);j++) {
108                     console.log("activando "+i+" , "+j);
109                     this.activa(i,j);
110                 }
111             this.celula[x+1][y+1]=false;
112             this.celula[x+6][y+1]=false;
113         }
114     }
```

Este método agrega un patrón del juego de la vida conocido como “pentaDecatlon” en la posición fila columna del ambiente de Conway. En particular el patrón puede invocarse al iniciar el ambiente. Por ejemplo, en el HTML podemos tener:

```
<body>
    <link rel="stylesheet" type="text/css" href="./conway.css">
    <div id="grilla"></div>
    <script>
        var x = new AmbienteConway(20,20);
        x.agregaPatron(6,6,"pentaDecatlon");
        x.inyectaAmbiente("grilla");
    </script>
    <button onclick="proximoPaso()">Siguiente</button>
</body>
```

Y la salida, después de algunos pasos será la siguiente:



En este problema se espera que, en base a la información pública de otros patrones del juego de la vida (por ejemplo la de **wikipedia**) agregue, DOS PATRONES MÁS, y los ponga a prueba (al mismo tiempo y en el mismo ambiente) en una cuadrícula de 30x30. Use los nombres de los patrones que usa la página indicada.

Se piden las siguientes evidencias del trabajo con las siguientes etiquetas:

TP2-P3-A Foto del código fuente del método `agregaPatron` extendido a los dos patrones adicionales pedidos y la invocación de estos para la generación de ejemplo de los patrones en el ambiente.

TP2-P3-B Despliegue del ambiente de 30x30 con los dos patrones originales y otra foto 5 clics después.

Problema 4 (25%). Hilos en Javascript

En este problema mostraré primero cómo generar hilos de ejecución adicionales al principal de modo que, no importando que esté ocurriendo, estos se ejecuten.

Considere y pruebe el siguiente código principal

```
6     <script>
7         function proximoPaso() {
8             x.proximoTurno();
9             document.getElementById("grilla").innerHTML="";
10            x.inyectaAmbiente("grilla");
11        }
12        var ciclos = 10;
13        function proSigue(milisecs) {
14            if(ciclos>0) {
15                ciclos--;
16                proximoPaso();
17                setTimeout(proSigue,milisecs,milisecs);
18            }
19        }
20    </script>
```

`setTimeout` (en la línea 17), es una función especial de javascript que permite la invocación de una función diferida en el tiempo. Esta función requiere al menos 2 argumentos: el primer argumento es el nombre de una función, en este caso se llama la función `proSigue`, es decir, la función `proSigue`, provoca una invocación de sí misma. El segundo argumento es el tiempo en milisegundos que se esperará para ejecutar la función especificada. El resto de los argumentos corresponde a los argumentos de la función invocada. En este caso como proSigue utiliza un argumento de espera, entonces se necesita pasar el argumento milisecs dos veces, la primera como argumento de la función `setTimeout`, la segunda vez como argumento de `proSigue`.

Explicado como funciona `setTimeout`, se explicará cómo funciona `proSigue`. Esta función trabaja en referencia a la variable global `ciclos`, la cual comienza en 10. La variable funciona como un cronómetro que va disminuyendo a medida que pasa el tiempo. En el fondo, el ciclo se ejecutará automáticamente sólo 10 veces.

Use el siguiente código HMTL para probar esta transformación automática.

```
24      <div id="grilla"></div>
25      <script>
26          var x = new AmbienteConway(30,30);
27          //ejemplo agregado de patrones, mantenga los ya agregados
28          x.agregaPatron(6,6,"pentaDecatlon");
29          x.inyectaAmbiente("grilla");
30          //invocación de 10 llamadas automáticas cada 2 segundos
31          proSigue(2000);
32      </script>
33      <button onclick="proximoPaso()">Siguiente</button>
```

Los patrones están bien si se comienzan a transformar automáticamente. Note que, **entre** los dos segundos de espera, usted puede presionar el botón “Siguiente” y funciona igualmente. Es decir, **durante la espera**, es posible activar otro método o función.

En este problema se espera que use lo explicado para que la funcionalidad sea como se indica.

1. Habrá **dos** botones. Uno que dirá “Siguiente” y otro que dirá “Avance Automático”. Al inicio el ambiente estará quieto, sólo con los patrones iniciales. Al presionar “Siguiente” funcionará cómo hasta el momento, es decir, un paso de Conway. Al presionar el botón “Avance Automático”, comenzará el proceso de avanzar automáticamente, pero SIN LÍMITES, es decir, no se detendrá. Cuando comienza el proceso automático deben pasar dos cosas primero: (a) el botón “Siguiente” debe quedar inhabilitado, y el botón “Avance Automático” debe cambiar el texto (atributo value) a “Detener Avance”. Cuando se presione “Detener Avance”, el botón “Siguiente, volverá a estar habilitado, el texto del

botón presionado volverá a cambiar a “Avance Automático” y, por supuesto, el proceso de avance de transformación de patrones se detendrá.

Se piden las siguientes evidencias del trabajo con las siguientes etiquetas:

TP2-P4-A Foto del código fuente en la gestión de los botones nuevos.

TP2-P4-B Despliegue del ambiente de 30x30 con uno de los patrones y otro patrón no programado pero configurado con clics. Muestre fotos de la configuración al inicio, y al clic 5,10,15 y 20.

TP2-P4-C Archivo zip (único modo de compresión aceptado) con los 3 archivos que resuelven el problema.

Muestre el código del HTML que logra esto, y una secuencia de las figuras para mostrar cómo se transforman