# Documentación de los cambios realizados

Para comenzar, hay partes del código que siguen exactamente igual ya que son como en el tutorial y son necesarias.  Estas son el loop y el initialize, aunque en este último es interesante conservar el this.setupInput(); ya que lo usaremos más adelante.

Por tanto la cosa quedaría así :

```
this.initialize = function(canvasElementId,sprite_data,callback) {

  this.canvas = document.getElementById(canvasElementId);
  this.width = this.canvas.width;
  this.height= this.canvas.height;

  this.ctx = this.canvas.getContext && this.canvas.getContext('2d');
  if(!this.ctx) { return alert("Please upgrade your browser to play"); }

  this.setupInput();
  this.loop(boxSize,n);
  SpriteSheet.load(sprite_data,callback);
};

this.loop = function() {

  for(var i=0, len = boards.length;i<len;i++) {
    if(boards[i]) {
      //Ficha que se mueve
      if (i==2){
        boards[i].step(boards[i]);
      }
      boards[i].draw(Game.ctx);
    }
  }
  setTimeout(Game.loop,30);
};
```

Para lo demás es útil empezar definiendo las estructuras de datos que vamos a usar en nuestro código para que sean lo mas parecidas posibles a lo que nos va a pasar y vamos a pasar a lógica.

Rotaciones y coordenadas posibles de la ficha:

```
getCoords : function(Game)
        /*
         *      Returns the coords (arry with 4 entries)
         *              rot0: [],                                       array of "info"s
         *              rot1: [],
         *              rot2: [],
         *              rot3: [],
         *
         *      With "info" set as :
         *              · coord
         *              · dummyPos                                      [ n, nw, w, sw, s, se, e, ne, c ]
         *                                                                      all false except the plausible dummy
         *                                                                  positions in the tile
         */
```

Informacion de la colocación de la ficha por el usuario:

```
setPos : function (Game, x, y, rot)
        /*
         *  Set the position where the tile is placed on the board and its rotation
         */
```

Una vez definido esto, tenemos que crear las estructuras de datos:

Array de rotaciones es simple y directo:

```
var rot0 = [];
var rot1 = [];
var rot2 = [];
var rot3 = [];
var posicionesValidas = [rot0,rot1,rot2,rot3];
```

Pero  a la hora de crearnos algo que dar a lógica, nos encontramos con que es mas útil llevar un objeto lo mas parecido al que tenemos que devolverles, ya que realmente no tenemos que devolver un objeto en si, sino unos valores en una función:

```
function ficha(num, coordx, coordy, rot, token) {
        this.num = num;
        this.coord = [coordx,coordy];
        this.rot = rot;
        this.token=token;
};
```

Donde num será el valor del sprite.

Una vez hecho esto, tenemos que pasar a crear los tableros que vamos a necesitar en el orden en el que deberían ser pintados por nuestro loop:

El primer tablero sin duda seria el fondo:

-Fondo: Este es el tablero mas sencillo, ya que solo hay que poner una imagen con unas coordenadas propias. ¿Cómo podríamos hacer esto mas sencillo? Convirtiendo el

fondo en una ficha de la que solo nos interesa el sprite y utilizar el mismo draw que usamos para las fichas solo que hacemos que ocupe todo el canvas:

```
function tableroFondo(fondo){
  this.draw = function(ctx){

    SpriteSheet.draw(ctx,fondo.num,fondo.coord[0],
        fondo.coord[1]);
  }
}
```

Despues de este irían el de fichas fijadas y el de rotaciones posibles en cualquier orden, ya que no van a solaparse, aunque en caso de que a lógica se le cuele un recuadro azul en la posición de una ficha fijada, no nos interesa que el jugador lo vea, por tanto Rotaciones primero:

-Rotaciones:

```
function tableroRotaciones(posicionesValidas){

  this.draw = function(ctx){
    for (i=0; i<posicionesValidas.length; i++){
      //Solo pintamos las de la rotacion que nos interesa
      if(i == fichaActual.rot ){
        for (j=0; j<posicionesValidas[i].length; j++){

          SpriteSheet.draw(ctx,fichAzul,posicionesValidas[i][j].coord[0]*altoficha,
                           posicionesValidas[i][j].coord[1]*anchoficha);
        }
      }
    }
  }
}
```

Donde fichAzul es el sprite designado para las casillas posibles.

Fijemonos ahora en el draw, podemos ver que es diferente y que faltan parámetros, esto se debe a que queremos dibujar todas las fichas iguales en tamaño por tanto el nuevo draw quedaría así haciendo excepción para el fondo:

```
this.draw = function(ctx,sprite,x,y) {
  var s = this.map[sprite];

   if (sprite == fichFondo){
    ctx.drawImage(this.image,s.sx ,s.sy,s.w,
                  s.h,x,y,canvas.width,canvas.height);
   }else{
    ctx.drawImage(this.image,s.sx ,s.sy,s.w,
                  s.h,x,y,anchoficha,altoficha);
   }

};
```

El siguiente paso es el tablero con las fichas fijadas

-FichasFijadas:

Este tablero tiene que dibujar las fichas fijadas, no cambia mucho mas allá de usar las estructuras de datos y sacar fuera el array de fichas, ya veremos que es útil tenerlo como variable global mas adelante.

-FichaActiva:

Aquí es donde esta la complicación,  esta es la pieza que tiene que irse dibujando a medida que movemos el raton, por eso, al ser dinámica, es la única que tiene un step.

Para hacer todo lo que esta ficha tiene que hacer, lo mas interesante es declarar la ficha como variable global, ya que siempre vamos a tener una ficha activa y hay problemas luego si no esta definida asi. Por tanto el tablero quedaría asi:

```
function tableroActiva(fichaActiva){
  this.draw = function(ctx){
    SpriteSheet.draw(ctx,fichaActiva.num,fichaActiva.coord[0],
        fichaActiva.coord[1]);
  }
  this.step = function(x,y) {
    function getMousePos(canvas, evt) {
        return {
          x: evt.clientX - rect.left,
          y: evt.clientY - rect.top
        };
    }
    document.addEventListener('mousemove',actPosition ,false);
    function actPosition(evt) {
        var pos = getMousePos(canvas, evt);
        fichaActiva.coord[0]=pos.x - anchoficha/2;
        fichaActiva.coord[1]=pos.y- altoficha/2;
    }
  }

}
```

Los únicos cambios realizados son el cambiar el event listener para que ocupe un poco menos de recursos (Aunque de todas maneras sigue estando mal ya que nuestra aplicación no debería ocupar 250M de ram en algunos navegadores). Y el sacar los event listeners fuera para ponerlos todos juntos en la función declaratoria como en el tutorial, lo que miraremos ahora.

-SetupInput:

Aquí es donde van declarados los event listeners generales del canvas. Lo primero es volver a poner lo que hemos quitado de dentro del último tablero aquí.

-Colocar Fichas

Ahora al hacer click mira a ver si esa posición y rotación es válida dentro de las guardadas. Ahora vemos por qué es interesante sacar el array de posiciones activas y ponerlo fuera, ya que podemos añadir fichas desde cualquier parte del código

```
function checkValida(ficha){
var valida=false
for (i=0; i<posicionesValidas.length; i++){
  for (j=0; j<posicionesValidas[i].length; j++){

    if(posicionesValidas[i][j].coord[0]*anchoficha==ficha.coord[0]
      &&posicionesValidas[i][j].coord[1]*altoficha==ficha.coord[1]
      &&ficha.rot==i){
      valida=true;
    }

  }
}
return valida;



canvas.addEventListener("click", getPosition, false);

function getPosition(evt)
{
  if(!click){
  var pos = getMousePos(canvas, evt);
  fichaActiva.coord[0]=Math.floor(pos.x/altoficha)*altoficha;
  nuevax=fichaActiva.coord[0]
  fichaActiva.coord[1]=Math.floor(pos.y/anchoficha)*anchoficha;
  nuevay=fichaActiva.coord[1]

  var nuevaFicha = new ficha(fichaActiva.num,nuevax,nuevay,fichaActiva.rot,0);
    if (checkValida(nuevaFicha)){
      console.log("ES VALIDA")
      boards[2].add(nuevaFicha);
      /**************************************
      AQUI TENEMOS INFO PARA  LOGICA DESPUES DE SELECCIONAR O NO UN MONIGOTE

      **************************************/
    }else{
        alert("Ficha no valida!");
    }
  }

}
```

Una vez cubierto lo del click y añadir la ficha y ponerla en el tablero, nuestra siguiente parada sería rotar las fichas.

-RotarFichas:

Para rotar las fichas, se puede hacer de dos maneras, con codigo javascript que la rote directamente o creando sprites directamente rotados. Yo me decidí por la segunda opción, ya que la otra me estaba dando bastantes problemas tanto al rotar, como a la hora de guardar y dibujar.

Independientemente de cual se use, la rotación se lleva en un valor de cada una de las fichas, y es el que se usa para interactuar en las comprobaciones de fichas posibles y en las funciones que cambian su rotación.

Una vez dicho esto, digo como están implementadas.

Como podréis ver, Hay declarados 4x 24 sprites diferentes, asi como el fondo y el spite azul, cada uno de ellos es una pieza con una de sus posibles rotaciones, así los que empiezan por 0 no están rotados, los que empiezan por 1 están rotados una posición, los de 2 dos y los de 3, 3.


Lo cual es bastante útil a la hora de girar ya que para pasar de uno a otro solo hay que sumar  o restar 100 y cambiar la rotación de la ficha.

Esto nos lleva a ver como se hacen las rotaciones:

Para no meternos en codigo de meteor aun, ya que también seria posible rotarlas con botones, creamos event listeners para el teclado. Tambien necesitamos, para que la tecla no se quede presionada, un evento que nos diga cuando se han soltado las teclas, asi como cancelar el comportamiento por default de las teclas usadas y solo de esas para que no nos de comportamientos extraños:

```
window.addEventListener('keydown',check ,false);
  function check(e) {

    var code = e.keyCode;
     if(code ==37) {
       e.preventDefault();
       console.log("izquierda pressionada");
       if (fichaActual.rot!=0){
         fichaActual.rot=fichaActual.rot -1;
         fichaActual.num=fichaActual.num -100;
         pressed=true;
       }

     }else if(code==39) {
       e.preventDefault();
       console.log("derecha pressionada");
      if (fichaActual.rot!=3){
         fichaActual.rot=fichaActual.rot +1;
         fichaActual.num=fichaActual.num +100;
         pressed=true;
       }
     }else {
     pressed=false;
     }
}

window.addEventListener('keyup',function(e) {
  pressed=false;
},false);
```

Todo esto se puede hacer gracias a que la ficha actual esta declarada como variable global y se va cambiando desde ahí.

Otros añadidos:

-Modificado para que el tablero acepte un numero de piezas variable y que las piezas se ajusten a esos tamaños:

```
//Valores iniciales de tamaño de fichas y fondo
var cantfichas=150;
var anchoficha;
var altoficha;


anchoficha= canvas.width/cantfichas;
    console.log(anchoficha)
altoficha= canvas.height/cantfichas;
    console.log(altoficha)
```

-Añadidas piezas de base para mostrar rotaciones. De momento se usan como piezas, pero en el momento que tengamos el array de rotaciones el uso no va a cambiar. Ya que esta pensado para que solo tenga en cuenta los campos del array de rotaciones.