

Dealing with key events



INTRODUCTION

This has been a sort of nightmare for years, as different browsers had different ways of handling key events and key codes (read this if you are fond of JavaScript archeology: <http://unixpapa.com/js/key.html>). Fortunately it's much better today, and we are able to rely on methods that should work on any browser.

When you listen to keyboard related events (`keydown`, `keyup` or `keypressed`), the `event` parameter passed to the listener function will contain the code of the key that fired the event. Then it is possible to test what key has been pressed or released, like this:

```
window.addEventListener('keydown', function(event) {  
  if (event.keyCode === 37) {  
    //left arrow was pressed  
  }  
}, false);
```

At line 2, the value "37" is the key code that corresponds to the left arrow. It might be difficult to know the correspondences between real keyboard keys and codes, so here are handy pointers:

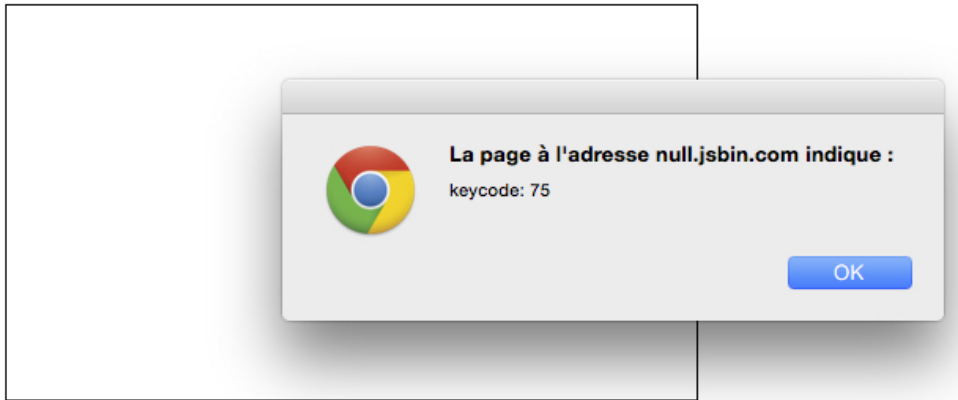
- Try key codes with this interactive example: <http://www.asquare.net/javascript/tests/KeyCode.html>
- And find a list of keyCodes (taken from: <http://css-tricks.com/snippets/javascript/javascript-keycodes/>) below:

Key	Code	Key	Code	Key	Code
backspace	8	e	69	numpad 8	104
tab	9	f	70	numpad 9	105
enter	13	g	71	multiply	106
shift	16	h	72	add	107
ctrl	17	i	73	subtract	109
alt	18	j	74	decimal point	110
pause/break	19	k	75	divide	111
caps lock	20	l	76	f1	112
escape	27	m	77	f2	113
(space)	32	n	78	f3	114
page up	33	o	79	f4	115
page down	34	p	80	f5	116
end	35	q	81	f6	117
home	36	r	82	f7	118
left arrow	37	s	83	f8	119
up arrow	38	t	84	f9	120
right arrow	39	u	85	f10	121
down arrow	40	v	86	f11	122
insert	45	w	87	f12	123
delete	46	x	88	num lock	144
0	48	y	89	scroll lock	145
1	49	z	90	semi-colon	186
2	50	left window key	91	equal sign	187
3	51	right window key	92	comma	188
4	52	select key	93	dash	189
5	53	numpad 0	96	period	190
6	54	numpad 1	97	forward slash	191
7	55	numpad 2	98	grave accent	192
8	56	numpad 3	99	open bracket	219
9	57	numpad 4	100	back slash	220
a	65	numpad 5	101	close bracket	221
b	66	numpad 6	102	single quote	222
c	67	numpad 7	103		
d	68				

EXAMPLE 1: ADDING A KEY LISTENER TO THE WINDOW OBJECT

A lot of people think that the canvas element is not able to get key events. Many examples on the Web handle key events on canvas by adding a listener to the window object directly, like that:

Online example: <http://jsbin.com/voviva/2/edit>



Extract from source code:

```
<canvas id="myCanvas" width="350" height="200">
</canvas>

<script>
function init() {
  // This will work when you press a key, anywhere on the document
  window.addEventListener('keydown', handleKeydown, false);
}

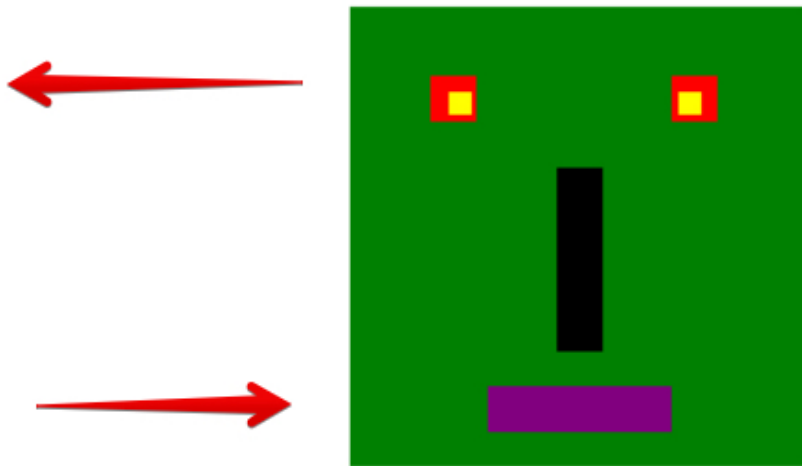
function handleKeydown(e){
  alert('keycode: '+e.keyCode);
  return false;
};
</script>
```

Indeed this solution works well if you write a game, and want to detect events wherever the mouse cursor is, and without worrying about what HTML element has the focus, etc...

Move the monster with the keyboard

Online example: <http://jsbin.com/qorupi/1/edit>

Move smoothly the monster by pressing / releasing the right and left arrows keys...



Notice that 1) we used a key listener on the \$ window object and 2) in the listener we just change a global variable, used by the animation loop (this is the secret to smoothness!)

Extract from source code:

```
<script>
var canvas, ctx;
var monsterX=100, monsterY=100, monsterAngle=0;
var incrementX = 0;
function init() {
    // This function is called after the page is loaded

    // 1 - Get the canvas
10. canvas = document.getElementById('myCanvas');

    // 2 - Get the context
    ctx=canvas.getContext('2d');

    // 3 add key listeners to the window element
    window.addEventListener('keydown', handleKeydown, false);
    window.addEventListener('keyup', handleKeyUp, false);
    // 4 - start the animation
```

```

20.   requestId = requestAnimationFrame(animationLoop);
      }
      function handleKeydown(evt) {
        if (event.keyCode === 37) {
          //left key
          incrementX = -1;
        } else if (event.keyCode === 39) {
          // right key
          incrementX = 1;
30.   }
      }
      function handleKeyup(evt) {
        incrementX = 0;
      }
      function animationLoop() {
        // 1 - Clear
        ctx.clearRect(0, 0, canvas.width, canvas.height);

40.   // 2 Draw
        drawMonster(monsterX, monsterY, monsterAngle, 'green', 'yellow');

        // 3 Move
        monsterX += incrementX;

        // call again mainloop after 16.6 ms (60 frames/s)
        requestId = requestAnimationFrame(animationLoop);
      }
    </script>

```

EXAMPLE 2: WHAT IF I WANT TO LISTEN TO KEY EVENTS ONLY IN MY CANVAS?

If you add a key listener to a canvas element, the problem is that it will get events only when it has the focus. And by default, it will never have the focus!

The `tabindex` attribute of the canvas element makes it focusable. Without it, it will never get the focus!

The trick is to declare the canvas like this:

```
<canvas id="myCanvas" width="350" tabindex="1" height="200">
```

```
</canvas>
```

And we force the canvas to get the focus with:

```
canvas=document.getElementById('myCanvas');  
...  
canvas.focus();
```

Now, if we try an example with the above canvas declaration, we show when an HTML element has the focus: a border is added to it, like in this example: <http://jsbin.com/cohide/2/edit>.

Note that the line that forces the focus to the canvas is commented by default. Try to click on the canvas, then press a key, then click out of the canvas, then press a key: this time nothing happens!

**The blue border appears
when the canvas has the focus**

**Key events are detected only
when the canvas has the focus**

Extract from the code:

```
var canvas;  
function init() {  
    canvas=document.getElementById('myCanvas');  
  
    // This will work only if the canvas has the focus  
    canvas.addEventListener('keydown', handleKeydown, false);  
    // We can set the focus on the canvas like that:  
    // canvas.focus();  
    // ... but it will stop working if we click somewhere else  
    // in the document
```

10.

```
}  
  
function handleKeydown(e){  
    alert('keycode: '+e.keyCode);  
    return false;  
20. };
```

Line 10 is useful to initially set the focus on the canvas, but this trick will not work if we click somewhere else in the HTML page.

EXAMPLE 3: A BETTER WAY: SET THE FOCUS WHEN THE MOUSE CURSOR ENTERS THE CANVAS

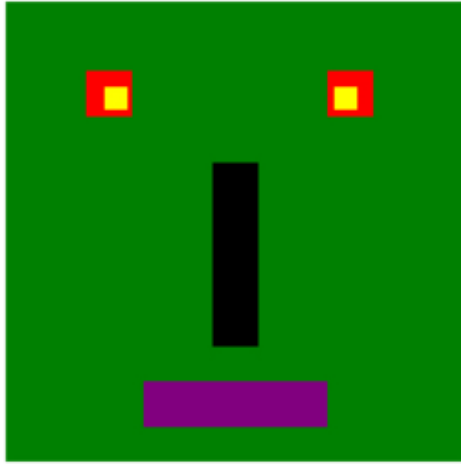
A better way to manage key events on a canvas is to set the focus when the mouse is over the canvas, un-focus it otherwise.

Here is a modified version of the "move monster example" seen earlier. This time, you move the monster with the left and right arrow only when the mouse cursor is over the canvas. We added two mouse event listeners on the canvas: one for the `mouseenter` event and the other for the `mouseout` event.

When the mouse enters the canvas we call `canvas.focus()` to set the focus to the canvas, and when the mouse cursor goes out of the canvas, we call `canvas.blur()` to unset the focus.

Online example: <http://jsbin.com/johaco/1/edit>

With this example, the canvas gets the focus when the mouse cursor enters it, and it loses the focus when the mouse exits the canvas...



Like that you can control the monster with the arrow keys only when the mouse cursor is over the canvas...

Extract from the code:

```
function init() {  
  // This function is called after the page is loaded  
  // 1 - Get the canvas  
  canvas = document.getElementById('myCanvas');  
  // 2 - Get the context  
  ctx=canvas.getContext('2d');  
  // 3 - Add key listeners to the window element  
  canvas.addEventListener('keydown', handleKeydown, false);  
  canvas.addEventListener('keyup', handleKeyup, false);  
12. canvas.addEventListener('mouseenter', setFocus, false);  
   canvas.addEventListener('mouseout', unsetFocus, false);  
  // 4 - Start the animation  
  requestId = requestAnimationFrame(animationLoop);  
}  
function setFocus(evt) {  
  canvas.focus();  
};
```


22.

```
function unsetFocus(evt) {  
    canvas.blur();  
};
```

The third parameter (false) of lines 12 and 13 means "we do not want to propagate the event to the ancestors of the canvas in the DOM."

KNOWLEDGE CHECK 4.3.2 (NOT GRADED)

We suppose we have defined a key event listener to a canvas. Should this canvas have the focus in order to fire key events?

- ☐
- ☐ Yes
- ☐ No