Input & output: how events work in Web apps & games

INTRODUCTION - EVENT MANAGEMENT IN JAVASCRIPT

In JavaScript, we treat events made by users as an input, and we manipulate the DOM structure as an output. Most of the time in games/animations, we will change state variables of moving objects, such as position or speed of an alien ship, and the animation loop will take care of these variables to move the objects.

The events are called *DOM events*, and we use the *DOM JavaScript API* to create *event handlers*.

THERE ARE THREE WAYS TO MANAGE EVENTS IN THE DOM STRUCTURE.

First way: declare event handlers in the HTML code

You will often find this in examples on the Web:

```
<div id="someDiv"onclick="alert('clicked!');">
    content of the div
</div>
```

Note: this is not the recommended way to handle events, even if it's very easy to use. Mixing the 'visual layer' (HTML) and the 'logic layer' (JavaScript) in one place is ok for small examples (we have used this in some examples in this course) but is not the recommended way for full scale applications where a clean separation is best.

Second way: add an event handler to an HTML element in JavaScript

Here is an example:

```
document.getElementById('someDiv').onclick= function(evt) {
  alert('clicked!');
}
```

This method is fine, but you will not be able to attach several listener functions. If you need to do this, the preferred version is the next one.

Third way: register a callback to the event listener with the `addEventListener` method

This is how we do it:

```
document.getElementById('someDiv').addEventListener('click',function(evt) {
    alert('clicked!');
}, false);
```

The third parameter is not important for now, just set it to `false`, or simply do not add a third parameter.

THE DOM EVENT THAT IS PASSED TO THE EVENT LISTENER FUNCTION

When you create an EventListener and attach it to an element, an event object will be passed as a parameter to your callback, just like this:

```
element.addEventListener('click',function(event) {
   // now you can use the event object inside the callback
}, false);
```

Depending on the type of event you are listening to, we will use different properties from the event object in order to get useful information like: "what keys have been pressed down?", "what is the position of the mouse cursor?", "which mouse button is down?", etc.

We will now cover how to deal with the keyboard and the mouse. Some experimental APIs like the gamePad API are on their way and supported by some browsers (to be studied in the HTML5 Part-2 course to come at W3Cx).

SOURCE CODE FOR THE KNOWLEDGE CHECK 4.3.1

Online example on JS Bin

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <meta charset="utf-8">
 <title>Click on button</title>
</head>
<body>
   <button id="myButton">Click me!</button>
 <script>
   var button =document.getElementById('myButton');
   // Define a click listener on the button
   button.addEventListener('click',processClick);
   // callback
   function processClick(event) {
      console.log("Button clicked");
      // What is the event parameter?
   }
 </script>
</body>
</html>
```

KNOWLEDGE CHECK 4.3.1 (NOT GRADED)

When you create an event listener like in the code above, what is the `event` parameter (line 15) passed to the callback function useful for?

- It will hold relevant data about the interaction (element that fired the event, key code, mouse button and position of the mouse cursor, etc.)
- It's not useful, it's just here for debug purposes.