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

INTRODUCTION / EVENT MANAGEMENT IN JAVASCRIPT

HTML5 events

There is no input or output in JavaScript. We treat events made by the user as an input, and we manipulate the DOM structure as output. Usually in games, we will change state variables of moving objects like position or speed of an alien ship, and the animation loop will take care of these variables to move the objects.

In any case, the events are called DOM events, and we use the DOM APIs to create event handlers.

How to listen to events

There are three ways to manage events in the DOM structure. You can attach an event inline in your HTML code like this:

Method 1: declare event handlers in the HTML code

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

This method is very easy to use, but it is not the recommended way to handle events. Indeed, It works now, but it's deprecated and will probably be abandoned in the future. Mixing 'visual layer' (HTML) and 'logic layer' (JavaScript) in one place is really bad practice and causes a range of problems during development.

Method 2: declare an event handler to an HTML element in JavaScript

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

This method is fine, but you will not be able to attach multiple listener functions. If you need to do this, the preferred version is shown below.

Method 3: register a callback to the event listener with the addEventListener method

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

Note that the third parameter describes whether the callback has to be called during the captured phase. This is not important for now, just set it to false.

What to do with the DOM event that is passed to the event listener function?

When you create an event listener and attach it to an element, it will provide an `event` object as a parameter to your callback, just like this:

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

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

In the following lessons, we will remind you now how to deal with the keyboard and the mouse (this has been studied in HTML5 Part 1) in the context of a game engine (in particular, how to manage multiple events at the same time), and also show how you can use a game pad using the new Gamepad API.