Simple Interaction with JavaScript

CS 418: Interactive Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Eric Shaffer

HTML Events

An HTML event can be
- something the browser does
- something a user does

Examples
- an HTML web page has finished loading
- an HTML input field was changed
- an HTML button was clicked
- user presses a keyboard key
- user clicks a mouse button.

JavaScript lets you execute code when events are detected

on-event Handlers

- Legacy event handling
- Supported in virtually all browsers
- Simple…usually sufficient
- Specify event handlers aka JavaScript functions
  - In HTML
  - Or using DOM document in JavaScript
    - …this is better as you can change event handlers dynamically
- Naming convention is **onX** where X is the event

```
function startup() {
canvas =
document.getElementById
            ("myGLCanvas");
document.onkeydown =
handleKeyDown;
document.onkeyup =
handleKeyUp;
}
```

Example

```
//----------------------------------------------------------------
//Code to handle user interaction
var currentlyPressedKeys = {};

function handleKeyDown(event) {
        console.log("Key up ", event.key, " code ", event.code);
        if (event.key == "ArrowDown" || event.key == "ArrowUp")
            event.preventDefault();
        currentlyPressedKeys[event.key] = true;
}

function handleKeyUp(event) {
        console.log("Key up ", event.key, " code ", event.code);
        currentlyPressedKeys[event.key] = false;
}

//----------------------------------------------------------------
```

```
/**
 * Startup function called from html code to start program.
 */
function startup() {
 canvas = document.getElementById("myGLCanvas");
 gl = createGLContext(canvas);
 setupShaders();
 setupMesh("cow.obj");
 gl.clearColor(0.0, 0.0, 0.0, 1.0);
 gl.enable(gl.DEPTH_TEST);
 document.onkeydown = handleKeyDown;
 document.onkeyup = handleKeyUp;
 tick();
}
```

```
//----------------------------------------------------------------
/**
  * Update any model transformations
  */
function animate() {
        if (currentlyPressedKeys["a"])
            eulerY-= 1;
        if (currentlyPressedKeys["d"])
            eulerY+= 1;
        if (currentlyPressedKeys["ArrowUp"])
            eyePt[2]+= 0.01;
        if (currentlyPressedKeys["ArrowDown"])
            eyePt[2]-= 0.01;
        document.getElementById("eY").value=eulerY;
        document.getElementById("eZ").value=eyePt[2];
}
```

Key Events

- **keydown**
  Physical key is pressed down
- **keyup**
  Physical key has popped back up
- **event.key**
  The character generated by the key event
- **event.code**
  Name corresponding to location of key on keyboard (e.g. KeyQ). This property returns a value which isn't altered by keyboard layout or the state of the modifier keys.

https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/code

Stopping Default Actions

The `Event` interface's **`preventDefault()`** method tells the user agent that if the event does not get explicitly handled, its default action should not be taken as it normally would be. The event continues to propagate as usual, unless one of its event listeners calls `stopPropagation()` or `stopImmediatePropagation()`, either of which terminates propagation at once.

This can be used to stop page scrolling in response to the arrow keys

---

Syntax

```
Event.preventDefault();
```

Parameters

None.

Return value

`undefined`.

DOM Level 2 Event Handling

Uses `addEventListener`

- `target.addEventListener(type, listener[, options]);`
  - Example: `document.addEventListener('keydown', handleKeyDown, false);`

`type`

A case-sensitive string representing the event type to listen for.

`listener`

The object which receives a notification (an object that implements the `Event` interface) when an event of the specified type occurs. This must be an object implementing the `EventListener` interface, or a JavaScript function.

`options` Optional

An options object that specifies characteristics about the event listener. The available options are:

- `capture`: A `Boolean` indicating that events of this type will be dispatched to the registered `listener` before being dispatched to any `EventTarget` beneath it in the DOM tree.
- `once`: A `Boolean` indicating that the `listener` should be invoked at most once after being added. If `true`, the `listener` would be automatically removed when invoked.

https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Handling and Listening

Terminology

The term **event handler** may be used to refer to:

- any function or object registered to be notified of events,
- or, more specifically, to the mechanism of registering event listeners via `on...` attributes in HTML or properties in web APIs, such as `<button onclick="alert(this)">` or `window.onload = function() { /* ... */ }`.

When discussing the various methods of listening to events,

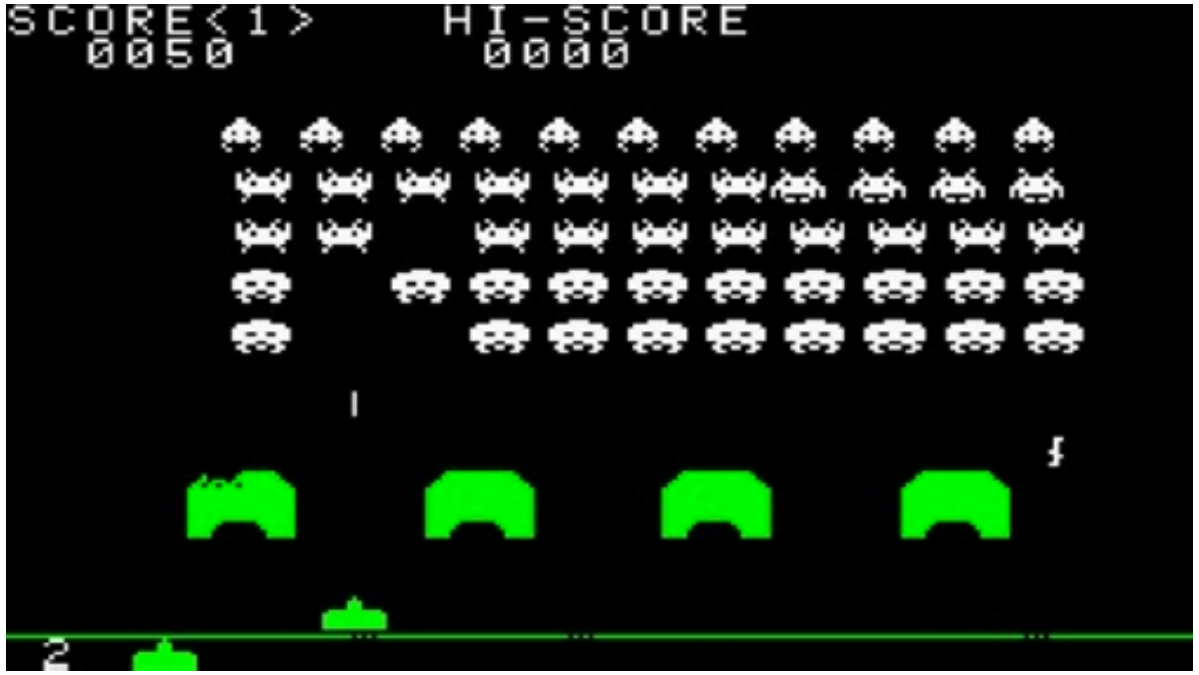
- **event listener** refers to a function or object registered via `EventTarget.addEventListener()`,
- whereas **event handler** refers to a function registered via `on...` attributes or properties.

jQuery Event Handling

The jQuery library has it’s own event handling capabilities

- Useful…it was more cross-platform than DOM Event Handling
- However, most modern browsers handle events pretty uniformly now
- So DOM event handling code you write should run everywhere
- You can use jQuery if you wish to…

You Now Know Enough to Make This....



- Developed by Tomohiro Nishikado and released in 1978