Multimedia – FWD 142

Chapter 2: HTML5 Multimedia

Applied College

Chapter goals

- Understanding media, as non-plain-text files.
- Learn how to add HTML5 audio and video elements to a web page.
- Learn how to use media elements in HTML5 games.

Content

- Background on HTML media
- HTML5 media elements
- Accessing media elements with javascript
- Audio in HTML5 games
- Practice: Event and Background sounds in a game

Background on HTML media

What is media?

- Media refers to any non-plain-text file.
- Examples include images, videos, and audios.
- A first step to use media on a Web page, is to store the media files on a web server and insert links to them into the HTML files. The browser will know how to open most common forms of media.
- When web developers talk about embedding video and audio files in webpages, they generally refer to a file's format to distinguish one kind of file from another.
- What distinguishes one kind of media file from another is a complex mix of technical parameters, which can be broken down into *file type*, *codec*, and *format*:

What is media?

- **File type:** Also called a file container, this is the way that the video or audio data is packaged into a file that a computer can read.
  - Anything with a file extension is a file type, like .MOV, .MP4, etc.
  - Each of these file types can contain video or audio data stored in one of several different ways (or even other kinds of data, like image data, subtitles, etc.).
  - These file types are often called "formats," even though the video in one .MP4 file, for example, might be of a very different type from the video in another .MP4 file.
- **Codec:** Recording video or audio data directly from a camera or a microphone produces truly vast amounts of digital data. To make this data more manageable, it's almost always compressed for storage, then decompressed when it's needed for editing or playback.
  - The media industry uses numerous standards for this process; these are known as codecs short for "compressor/decompressor." .
  - You can think of the codec as the "language" in which the video or audio data is stored. H.264 is a very common video codec, and MP3 is an audio codec.
- **Format:** This refers to the internal structure of the media data.
  - For video, this includes frame size, framerate, and pixel aspect ratio.
  - Features of audio formats include the number of channels and configuration of those channels (whether stereo, multiple- mono, 5.1surround-sound, etc.).

Embedding media files

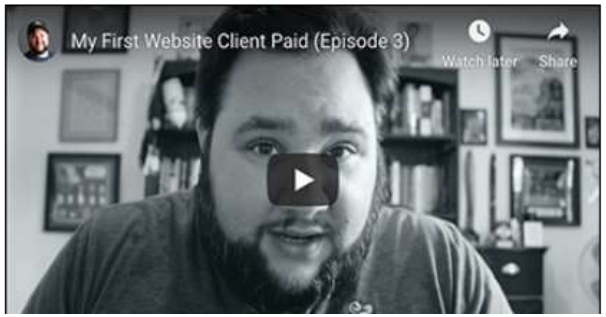
- Embedding media consists in display it directly on the page, without having to open separate windows or applications. The media can be viewed, watched, or listened to without the user leaving the page.
- When we embed video or audio in the webpage, take care to choose files in formats that the most common browsers support.
  - The best solution for video is usually to choose MP4, because all browsers support it, with some exceptions. Other supported formats include WebM and Ogg.
    - Embedding audio in the webpage: we need to be mindful of the formats that are supported. WAV is supported by all browsers except Internet Explorer. The MP3 format is supported by all browsers.

Embedding video

- We can embed a video on the webpage with the <video> element.
- There are a few attributes to know as well: width, height, and controls. The "controls" attribute has no value, but adding it tells the browser to include the play, pause, and volume buttons.
- Between the opening and closing <video> tags, we need to include the video source. Multiple video sources (which can use different formats) can be added, and the browser will pick the best one.

Storing media files

- Multimedia—audio and video—will use considerable resources and bandwidth that the web server is likely not specialized for. This can lead to the website crashing or quickly running out of storage and bandwidth.
- Solution: use of specialized services to host the media, and then embed links to those services into the website using the embed code provided by the service.



A YouTube video embedded on a webpage

The YouTube share dialog

HTML5 media elements

HTML5 media elements

- The media elements, audio and video, are a way of embedding playable media files directly into a web page without having to use Flash or a plug-in. The elements can be styled with CSS, integrated with SVG and Canvas, and controlled with JavaScript.

- Audio files are containers wrapping one type of media data, the audio stream, but video files typically wrap two different media streams: the video and the audio data streams.

- The *audio* and *video* elements both support a common syntax and subgroup of attributes. The only difference between the two elements is the content they manage, and a small group of additional attributes for the video element.

Minimal element syntax

- In the HTML, an *audio* element is used to embed an audio file encoded as Ogg Vorbis into the web page.
- Ogg Vorbis is a fully open, non-proprietary, patent-and-royalty-free, general-purpose compressed audio format for mid to high quality
- The URL for the audio file is given in the audio element's src attribute.

```
1  <head>
2    <title>Audio</title>
3    <meta charset="utf-8" />
4  </head>
5  <body>
6    <audio src="audiofile.ogg"> </audio>
7  </body>
```

Minimal element syntax

- Unlike the audio element, the video element has a play area that should show as long as there's no error loading the video, and the video element isn't deliberately hidden.
- To see the audio file in the page, we need to add the boolean *controls* attribute.

```
<audio src="meadow.ogg" controls> </audio>
```

- Both the video and audio elements support the *"controls"* attribute for adding a default control UI (user interface) for the media resource.

Alternative media

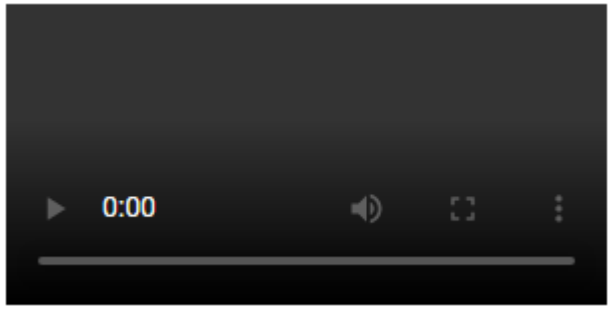
- Both the video and audio elements can contain zero or more source elements.
  - These child elements define a way to specify more than one audio or video file in different formats.
  - If a browser doesn't support one format, it will find a format it supports in another source element.
- **Example 1:** The web page contains a *video* element, but rather than provide the location of the video file in the element's *src* attribute, three different video files are defined in three different *source* child elements.

```
1  <head>
2    <title>Video</title>
3    <meta charset="utf-8" />
4  </head>
5  <body>
6    <video id="meadow" controls>
7    <source src="videofile.mp4" />
8    <source src="videofile.ogv" />
9    <source src="videofile.webm" />
10   </video>
11 </body>
```

Alternative media

- **Example 2:** A web page containing
  - an embedded video element with three separate video types and
  - fallback content for browsers that don't support HTML5 video

```
<body>
 <video controls>
      <source src="videofile.mp4" />
      <source src="videofile.ogv" />
      <source src="videofile.webm" />
      <iframe width="640" height="390"
         src="http://www.youtube.com/embed/YE7VzlLtp-4">
      </iframe>
 </video>
</body>
```



Media attributes

Attribute	Description
Accesskey	A unique, ordered, and space separated (as well as case sensitive) set of tokens that enables specifically named keyboard key access to the media element.
class	A set of space separated tokens specifying the various classes the element belongs to.
contenteditable	If true, content can be edited; if false, content cannot be edited.
Contextmenu	The id of the context menu associated with the element.
dir	The directionality of the element's text.
draggable	Whether the media element can be dragged. If true, the element can be dragged; if false, the element cannot be dragged.
dropzone	What happens when an item is dropped on the element.

Media attributes

Attribute	Description
hidden	A boolean attribute that determines if the element is “relevant”. Elements that are hidden are not rendered.
id	A unique identifier for the element.
lang	Specifies the primary language of the element’s contents.
Tabindex	Determines if media element is focusable, and the element’s order is in the tabbing sequence.
title	Advisory information, such as a tooltip.

Media-Specific Attributes

Attribute	Description
preload	The preload attribute provides hints to the user agent about preloading the media content.
autoplay	The autoplay attribute is a boolean attribute whose presence signals the user agent to begin playing the media file as soon as it has loaded enough of the media file so that it can play through it without stopping.
loop	The loop attribute resets the media file back to the beginning when finished, and continues the play.
muted	If the muted attribute is present, the media plays, but without sound.

Example of attributes usage

- The combination of *loop* and *autoplay* can be used to create a background sound for when a page is loaded, and continues to play until the user leaves the page.
- These attributes can be used with an audio element that doesn't have a controls attribute, and is also hidden using CSS.
- **Example** of repeating auto-started audio in two formats to ensure browser coverage:

```
<head>
    <title>Repeating Audio</title>
    <meta charset="utf-8" />
    <style>
    #background
    {
       display: none;
    }
    </style>
</head>
<body>
    <audio id="background" autoplay loop>
        <source src="audiofile.mp3" type="audio/mpeg" />
        <source src="audiofile.ogg" type="audio/ogg" />
    </audio>
</body>
```

Accessing media with javascript

Audio and video objects

- The audio and video elements are represented in JavaScript by the HTML *Audio* and *Video* element objects, respectively. Both of these are derived from HTML *Media* element.
- To access a specific audio or video element using JavaScript, we need to set its *id* attribute.

Audio attributes in javascript

- Attributes representation in a HTML5 webpage:
  - `<video width="320" height="240" autoplay muted>`
  - `<video width="320" height="240" autoplay>`

- In javascript, the current state of the audio or video is available through properties such as duration, currentTime, and volume.

- Since the audio element will be referenced in numerous places, we can declare a **variable** to store it. This will avoid the need to search through the DOM every time it is used.

```
var audio = document.getElementById("audio");
```

Controlling media

- The ***togglePlay()*** method is called when the user clicks the "Play" button. If the current state of the audio element is "paused" or "ended", it calls the *play()* predefined function. Otherwise, it calls the *pause()* method.

- The ***updatePlayPause()*** method is registered on the audio element for both the *play* and *pause* events. It sets the label of the "Play" button to reflect the state of the audio element. If the audio is currently playing, the text is changed to "Pause" since that will be the result if the button is clicked. Otherwise, the text is set to "Play."
- The ***endAudio()*** function is registered with the audio element in response to the "ended" event, which is raised when the audio has finished playing.

```
function togglePlay() {
    if (audio.paused || audio.ended) {
        audio.play();
    }
    else {
        audio.pause();
    }
}
```

```
function updatePlayPause() {
    var play = document.getElementById("play");
    if (audio.paused || audio.ended) {
       play.value = "Play";
    }
    else {
       play.value = "Pause";
    }
}
```

```
function endAudio() {
    document.getElementById("play").value = "Play";
    document.getElementById("audioSeek").value = 0;
    document.getElementById("duration").innerHTML = "0/" +
    Math.round(audio.duration);
}
```

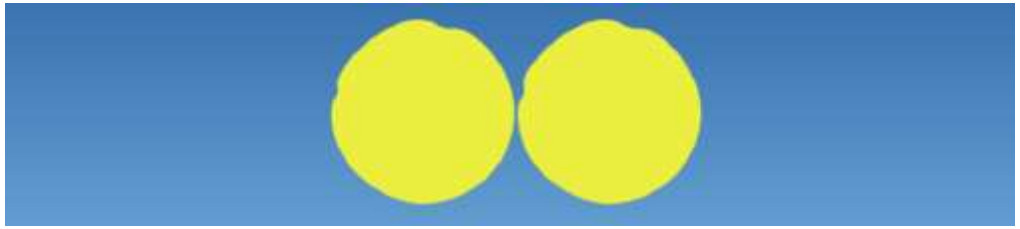
Audio in HTML5 Games

Audio in HTML5 games

- Sound is a very important (and often neglected) aspect of game creation. Sound is a powerful and efficient tool for establishing mood and feeling, for building tension and excitement, and for controlling the pacing of a piece.
- Audio in games is not optional.
- Audio is such a very powerful mechanism for conveying action and meaning that even simple sound effects will dramatically improve the feel of your game.

Scenario of sound effect

- Imagine two identical circles—one on the left side of the screen, one on the right, and both at the same height. The two circles move horizontally, crossing at the center of the screen. When they reach the other side, they move back again until they reach their start positions.
- Something interesting happens when we introduce a simple sound. If, at the very moment the two circles intersect, we make a “plop” sound (like a tennis racket hitting a ball), suddenly there’s a strong illusion that the two circles, actually, collide and rebound!



Sound control with javascript

- The plop example demonstrates the simplest way to get some sound to play via JavaScript.
  - Create a new ***Audio*** element and assign the “source”, in the same way we’d create an Image element.
  - This is also equivalent to embedding an ***<audio />*** tag in the main web page, but created on demand when required throughout our game.

```
const plop = new Audio();
plop.src = "./res/sounds/plop.mp3";
```

- When the “tennis balls” collide (their x position offsets are < 5 pixels apart) we call ***play*** on the Audio element:

```
if (Math.abs(x1 - x2) < 5) {
    plop.play();
}
```

Sound control with javascript

- **Repeating Sounds:** By default, all the sounds are one-shot. They play through from start to end, then stop. That's perfect for most effects (like power-up twinkles and collision explosions).
- Other times, we'll want sounds to loop over and over. To facilitate this, add the ***loop*** property as one of our possible options when creating a **Sound instance**. If looping is set to *true*, we'll apply it to the Audio element (via its corresponding loop property).

```
class Sound {

    constructor(src, options = {}) {
        this.src = src;
        this.options = Object.assign({ volume: 1 }, options);
        // Configure Audio element
        ...
    }

    play(options) {}
    stop() {}
}
```

Sound control with javascript

- Then we'll define it in the options object when making the game's loopable sound element:

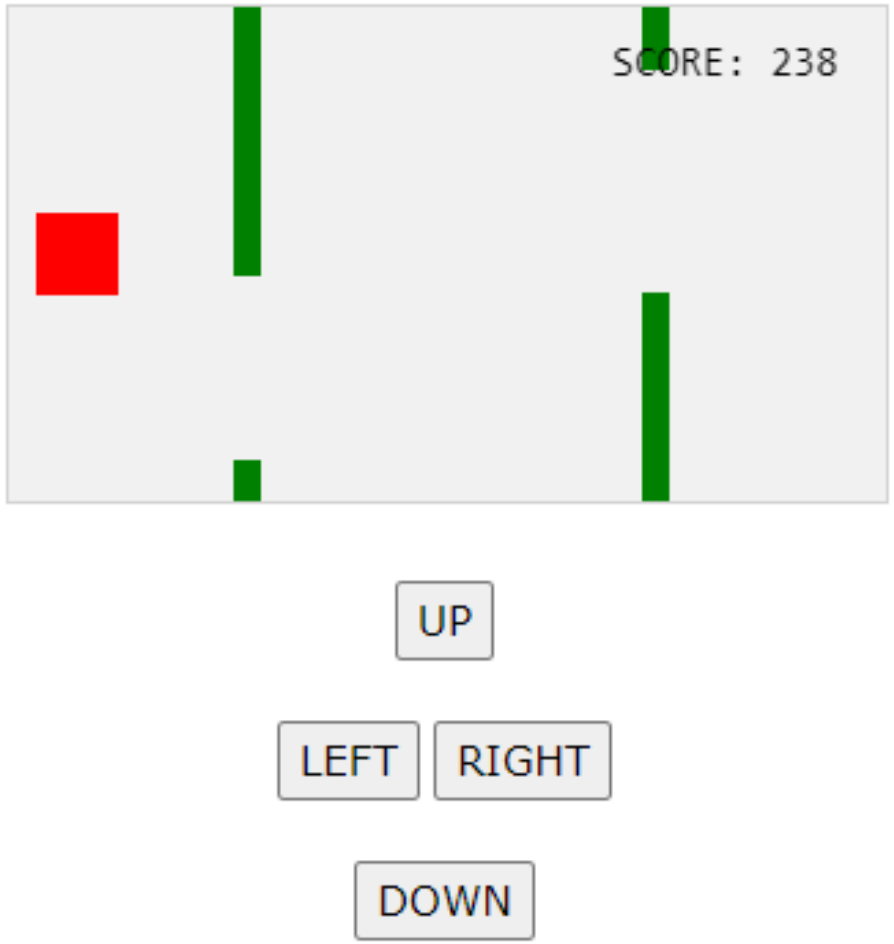
```
const sounds = {
    swoosh: new Sound("./res/sounds/swoosh.m4a", {volume: 0.2,
    loop: true})
};
```

- If a player is holding down the up button (***controls.y()*** is less than 0), the jetpack is applying its upward force and the relevant woosh sounds should play:

```
// Player pressed up (jet pack)
if (controls.y < 0) {
    sounds.swoosh.play();
    ...
}
```

Practice

In this section: Event and background sounds in games



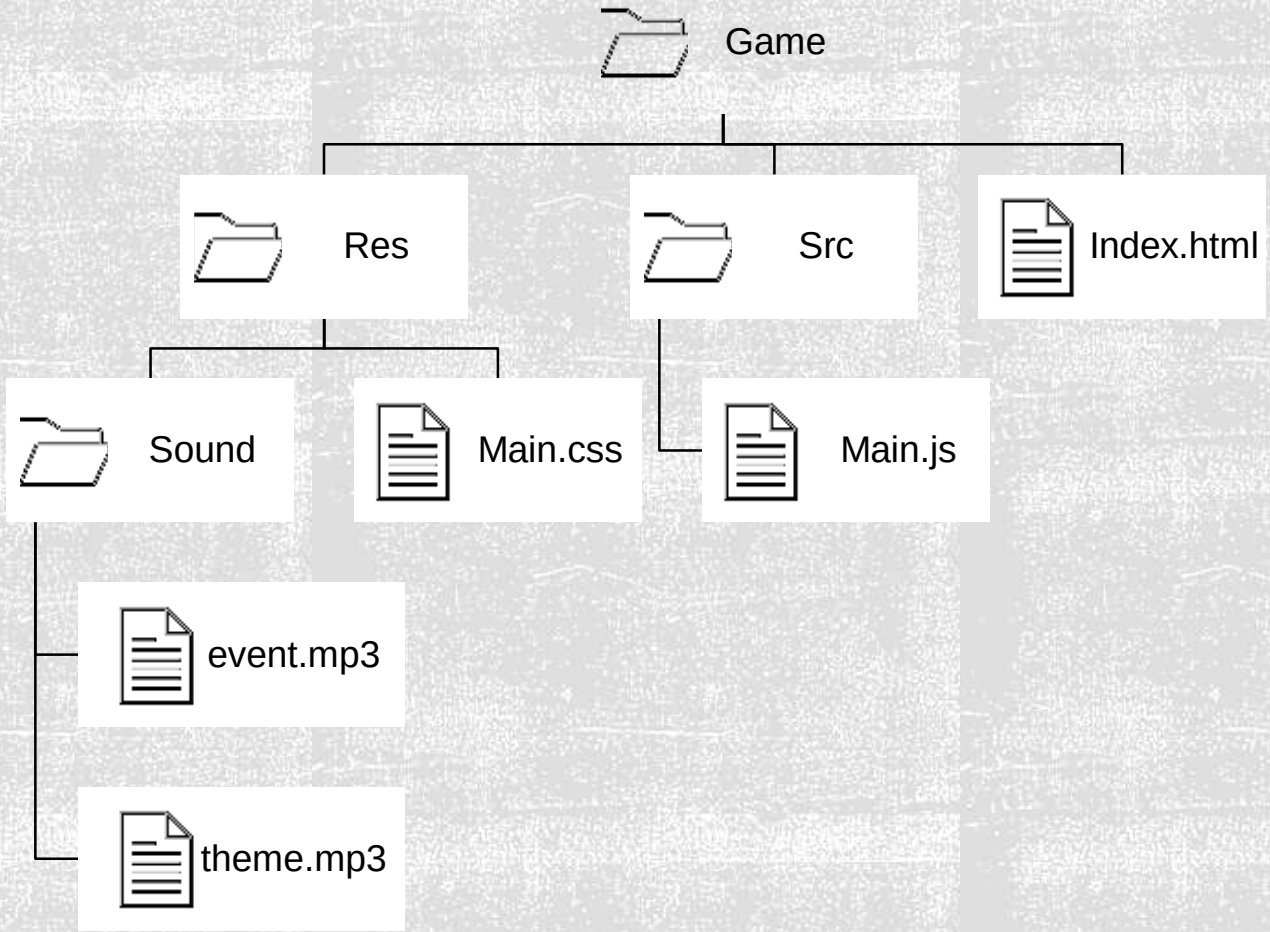
Example of game sounds

- **Game:** Push the buttons to move the red square.
- **Sound effects:**
  - Hearing a "dunk" when the red square hits an obstacle.
  - Background music.

- Source code available in:

https://www.w3schools.com/graphics/tryit.asp?filename=trygame_sound_music

Project structure



- **Game:** main folder.
- **Res:** folder containing the used resources:
  - css file,
  - audio files (event.mp3 and theme.mp3 sounds).
- **Src:** folder containing the game's main script (main.js).
- **Index.html:** main HTML5 webpage of the game

```
function sound(src) {
  this.sound = document.createElement("audio");
  this.sound.src = src;
  this.sound.setAttribute("preload", "auto");
  this.sound.setAttribute("controls", "none");
  this.sound.style.display = "none";
  document.body.appendChild(this.sound);
  this.play = function(){
    this.sound.play();
  }
  this.stop = function(){
    this.sound.pause();
  }
}
```

Adding sound

- Use the HTML5 *<audio>* element to add sound and music to the game.
- In this examples, we create a new object constructor to handle sound objects.

```
var myGamePiece;
var myObstacles = [];
var mySound;

function startGame() {
  myGamePiece = new component(30, 30, "red", 10, 120);
  mySound = new sound("bounce.mp3");
  myGameArea.start();
}

function updateGameArea() {
  var x, height, gap, minHeight, maxHeight, minGap, maxGap;
  for (i = 0; i < myObstacles.length; i += 1) {
    if (myGamePiece.crashWith(myObstacles[i])) {
      mySound.play();
      myGameArea.stop();
      return;
    }
}
```

Adding sound

- To create a new sound object use the sound *constructor*, and when the red square hits an obstacle, play the sound.

```
var myGamePiece;
var myObstacles = [];
var mySound;
var myMusic;

function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 120);
    mySound = new sound("bounce.mp3");
    myMusic = new sound("gametheme.mp3");
    myMusic.play();
    myGameArea.start();
}
```

Background Music

- To add background music to the game, add a new sound object, and start playing when starting the game.

Thank you!