ADDING IMAGES AND MUSIC TO YOUR TWINE GAME

The following handout explains how to add images and music to your Twine game, and also how to design a folder structure and use relative links. Before beginning, make sure that your Twine game is set up for the SugarCube format. To do so, click on the name of your story in its main "story map" view. Select "Change Story Format" and check the box next to "SugarCube." Note: this handout is for my students, and assumes some familiarity with HTML and CSS.

Adding a photo or video from the web

Adding media to your Twine game is as easy as adding HTML tags. Let's say that you want to add a picture of a spooky hallway to the first passage of your game about being stuck in a hallway. All you would need to do is search for "hallway" on Google, refine to Images, find an image you like, and then click on "View Image." Now the image will be displayed in your browser, and its URL will be in the address bar. All you need to do is grab its URL by using "Cut."

Now you have all of the information you need to insert this image into your Twine game. Let's add an HTML `img` tag right above the existing content in the passage:

```
<img src="http://vignette1.wikia.nocookie.net/thestanleyparable/images/6/60/
Two_Doors_Room.jpg" alt="A photo of a hallway and two doors" />
You are in a hallway. You see two doors: [[the door on the left]] and the [[the
door on the right]].
<<set $hasKey to false>>
```

Good coding practice requires that you enter an `alt=` tag to describe the content of your image in words. Otherwise, this is all very straightforward. There you have it: there's an image in your game.

You could just as easily insert a video from YouTube into your game. Just find the video you want to insert, click on "Share" under the video, select "Embed," and copy the HTML code into your game. There you have it: there's a video in your game.

Building a folder structure

Okay, that works — but it's not ideal. First of all, if you worked this way, you'd probably need to rely on images made by other people — things already on the web, rather than your original content designed just for the game. Second, let's say you release your brilliant game, and then the link to your image goes dead. Now you've got an unsightly missing image in your game. Worse yet, imagine that it turns out the image you grabbed off the web is under copyright, and you get an angry letter from the copyright holder's lawyer.

The way to get **total creative control** over your game is to design it in a **folder structure** you've created yourself. Let's say, for example, that I make a folder called `hallwaygame` somewhere on my computer's hard drive. This is going to be where my game and all of its media assets (images, audio tracks, etc.) are going to live. In the main `hallwaygame` folder, I'm going to put the game itself, which I'll plan on calling `hallwaygame.html` (remember, Twine games publish as HTML files).

Now within the `hallwaygame` folder, I'm going to make a subfolder called `images`, which is where I'll put all the images for my game. Let's say I've made my own original image of a hallway with two doors at the end. I'll save that image as `hallway.jpg` and put it in the `images` subfolder I've just created.

Now, rather than using full URL to a file on the web, I'm going to use what is called a **relative link**. Rather that specifying where the image is on the web, I'm going to specify where the image is *in relation to the main Twine file* (`hallwaygame.html`). I know exactly where `hallway.jpg` is relative to `hallwaygame.html`, since I've placed it into my very own folder structure: it's in a subfolder called `images`. The way of representing that as a relative path in HTML is as follows: `images/hallway.jpg`

So let's put that relative path into my previous passage as the `src` of my `img` tag.

```
<img src="images/hallway.jpg" alt="A photo of a hallway and two doors" />
You are in a hallway. You see two doors: [[the door on the left]] and the [[the
door on the right]].
<<set $hasKey to false>>
```

If you press the Play button in Twine, you'll see — uh oh! — that the image doesn't appear. That's because when you press Play in Twine, you're just seeing a kind of preview of your game. *Where* this "preview" exists is a tricky question. It's sort of in a **netherworld**. It's not on the web; and while it's *somewhere* on your computer, it's not in a place that you can usefully access. So let's bring the Twine game out of this netherworld and put it somewhere concrete. We do this by clicking on the story name from the main "story map" view and selecting "Publish to File." Excellent: now let's save our game as `hallwaygame.html` in the folder we created for it.

Now, let's leave Twine for a moment. Let's go to the folder where we saved `hallwaygame.html` and double click on it. It will open up in a web browser — and if you've entered your `img` tag correctly, your image will display.

Now collect a bunch more images for your game, and save them all in the `images` subfolder. Wherever you want them to appear in your game, use `img` tags with relative paths pointing to `images/`. Now your Twine game is all in one place — all contained within the `hallwaygame` folder. You could now upload this folder onto a webserver and it would display just as well as it does on your own computer. (Since relative paths are, well, *relative*, they don't care whether they're on your computer or on the web — all they care about is how to get from one place to another, and whether you're on the web or on your own computer, the way of getting from `hallwaygame.html` to `hallway.jpg` is the same: look into the subfolder `images`, and it will be there.) You could put the whole folder on Dropbox or Google Drive, make the folder public, and share the link with your friends — instant web hosting! You could also make a .zip file out of the `hallwaygame` folder and email your whole Twine game, with all its images, to anyone you choose.

Adding music

In theory, there is nothing special about adding music. You could just use the standard HTML tags for adding music. In practice, though, SugarCube has a much simpler way to add music — a set of so-called "macros" that make working with music a breeze.

The first thing we need to do locate some music (.mp3 format works best; and don't break any laws if you're planning on hosting your game publicly — remember the lawyers!). Next, as with images, we need to figure out where we'll store the music. Like we did with our images, let's put all of our music into its own subfolder of `hallwaygame`. Let's call this subfolder `music`.

SugarCube's audio macros work in two steps. First, you need to load the songs and give them unique names (much like a variable name). Next, when you're ready to actually play the song, or pause the song, or fade the song out, you insert a different macro in the particular passage of your story where you want this to happen.

To load the songs, we're going to create a new passage called `StoryInit`. This is a special passage that SugarCube treats in a special way: it executes all the commands in this passage before showing the player the first passage of the game. `StoryInit` isn't linked to any other passages, because it isn't part of the story. It's just a place for doing mundane business related to the story. It's a great place to initialize variables, and it's an ideal place to load songs.

Okay, so let's make a passage called `StoryInit` and write in the following code:

```
<<cacheaudio "mainsong" "music/dauphin.mp3">>
<<cacheaudio "happysong" "music/henry.mp3">>
```

The first line creates an "audio asset" named `mainsong` (though you can call it whatever you want) out of an audio file called `dauphin.mp3` that is located in the `music` subfolder relative to `hallwaygame.html`. The second line creates a second audio asset called `happysong` out of a file called `henry.mp3` that we've put in this same folder.

Now that our songs are loaded, let's do something with them! Let's say we want `mainsong` to start playing right away as soon as our player starts the game. All we need to do is add the following code into the first passage:

```
<<audio mainsong play>>
```

This line invokes SugarCube's audio macro, then specifies the code-name of the song we want to play (`mainsong`, created in `StoryInit`), and then specifies an action (`play` starts playing the song).

If you hit "Play" within Twine, you'll notice that this isn't working yet. Well, of course it isn't working! We need to bring our Twine game out of the netherworld and use the "Publish to File" feature to save it as a concrete HTML file in the `hallwaygame` folder — its true home, where all its media assets are located! — before it will start working. If you do that, it you'll hear your song.

Now let's say that when your player gets to the "win" screen, you want to reward them with a happy song. Placing the following lines of code into that "winning" passage will fade out `mainsong` and fade in `happysong`, the other song we created in `StoryInit`

```
<<audio mainsong fadeout>>
<<audio happysong fadein>>
```

One you again publish your game to the `hallwaygame` folder, you'll find that you now have a very advanced Twine game, complete with a soundtrack that adapts to the action!

For a full list of SugarCube audio commands (it can do a lot more than just play, fade in, and fade out!), visit http://www.motoslave.net/sugarcube/1/docs/macros.html#macros-cacheaudio