The TWINE® Grimoire

G.C. "GRIM" BACCARIS

An introduction to using CSS and HTML to customize projects in Twine 2.0.

Volume One

TABLE OF CONTENTS

! Click on the arrows ➥ or the underlined text above to jump directly to a specific section of the document.

INTRODUCTION

In this section:

- **What to Expect**
- **About Twine**
- **About the Author**

Thanks for cracking open the **Twine Grimoire**! This arcane tome contains a series of tutorials created to help Twine users learn how to customize the appearance and behavior of their projects in the Twine 2.0 editor.

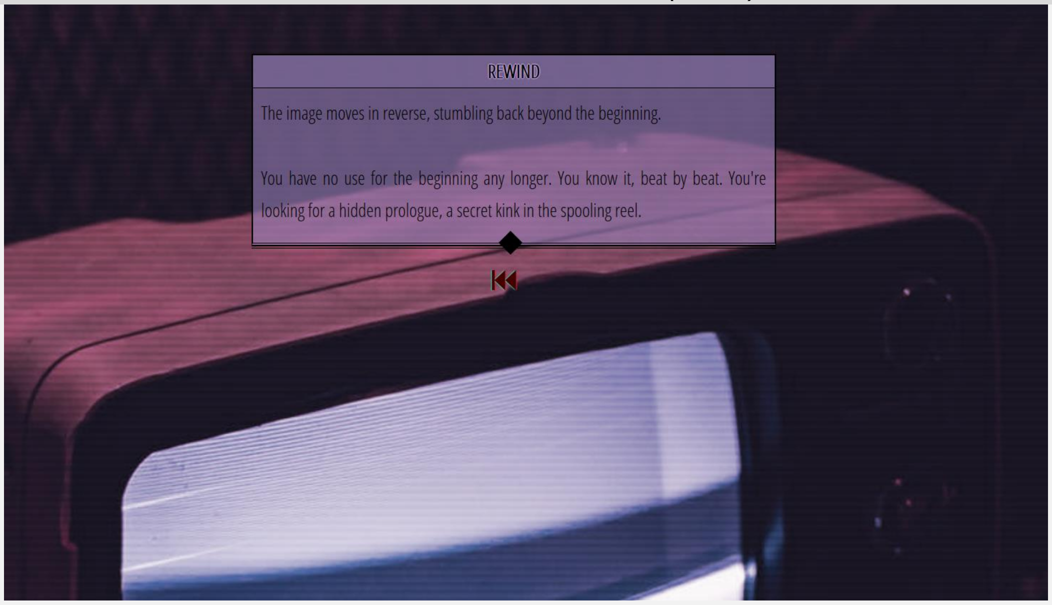
The aim is to demonstrate how **CSS and HTML** can be used in a variety of Twine 2.0 Story Formats as a method of fine-tuning a game's visuals. The information in these tutorials is not compatible with previous versions of Twine such as Twine 1.4.2. These tutorials are suitable for beginner to intermediate users, and will be geared toward the most common basic customizations and features that many first-time users are interested in working into their projects.

Twine® is an "an open-source tool for telling interactive, non-linear stories" originally created by Chris Kilmas in 2009; it is now maintained by "a whole bunch of people at several different repositories," per Twinery.org. Twine is also a registered trademark of the Interactive Fiction Technology Foundation, and an extremely powerful, versatile, accessible tool at your disposal!

I am **not** a developer of Twine itself, and am in no way affiliated with its development process, but I am a developer who primarily works with Twine. I'm a writer, narrative designer, and occasional game jam organizer; my work appears in Sub-Q Magazine and elsewhere, and I've created over 10 games of varying lengths and genres using Twine. I'm passionate about its utility to developers of all backgrounds, skill sets, and skill levels.

On the following two pages, I've provided few samples of **screenshots** from playable games I've created in Twine. These games were customized entirely using CSS, HTML, and skills that will be detailed in the tutorials to follow:

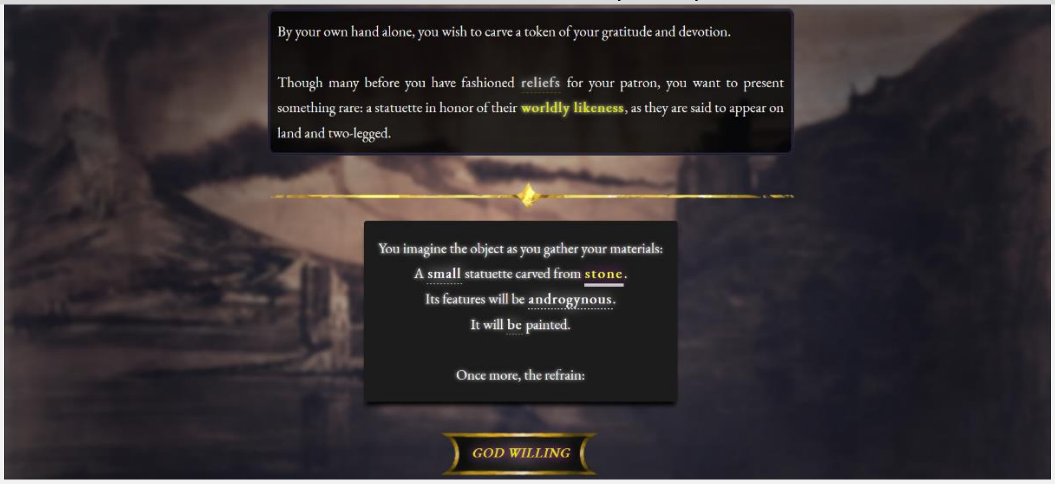
UNMAKING, UNMADE (2018)


REWIND

The image moves in reverse, stumbling back beyond the beginning.

You have no use for the beginning any longer. You know it, beat by beat. You're looking for a hidden prologue, a secret kink in the spooling reel.

DEVOTIONALIA (2018)

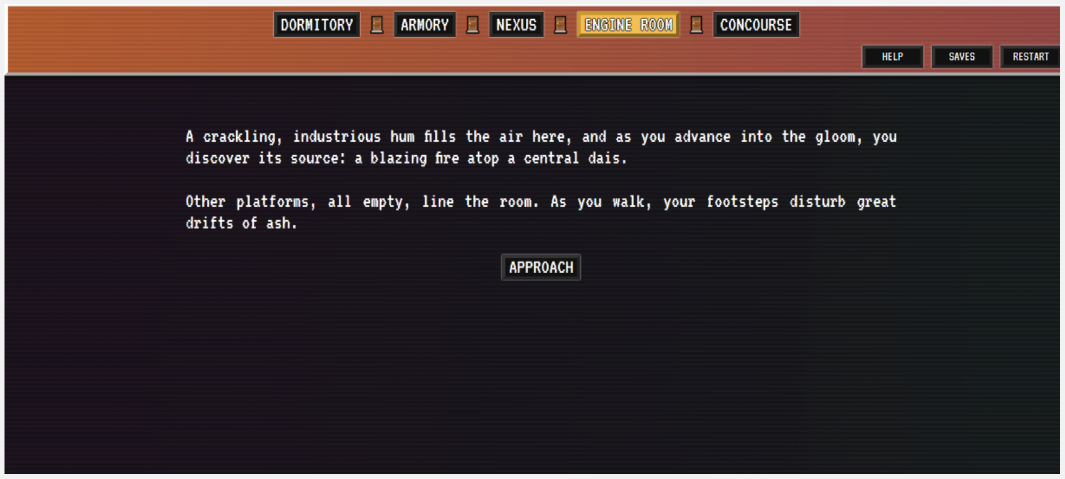

By your own hand alone, you wish to carve a token of your gratitude and devotion.

Though many before you have fashioned reliefs for your patron, you want to present something rare: a statuette in honor of their worldly likeness, as they are said to appear on land and two-legged.

You imagine the object as you gather your materials:
A small statuette carved from stone.
Its features will be androgynous.
It will be painted.

Once more, the refrain:

HABEAS CORPUS (2019)



A crackling, industrious hum fills the air here, and as you advance into the gloom, you discover its source: a blazing fire atop a central dais.

Other platforms, all empty, line the room. As you walk, your footsteps disturb great drifts of ash.

This guide will give you the tools to begin creating your own unique UI using accessible explanations of the processes and code involved. It also contains optional **exercises** you can use for practice or inspiration.

The syntax and methods for these features can **differ between versions and formats of Twine**, so this document contains demonstrations and exercises geared toward multiple formats. We'll be using **Twine 2 exclusively** and working with the most recent **SugarCube** and **Harlowe** formats primarily, but we will also mention previous versions from time to time for comparison. At the end of each tutorial, I'll provide some exercises you can do on your own.

In the following section, I'll be pointing out things like **where to input your custom CSS**, some **CSS basics**, how to see or change which **Story Format** you're using, and other features of the engine that you'll need to know while working on your project.

GETTING STARTED

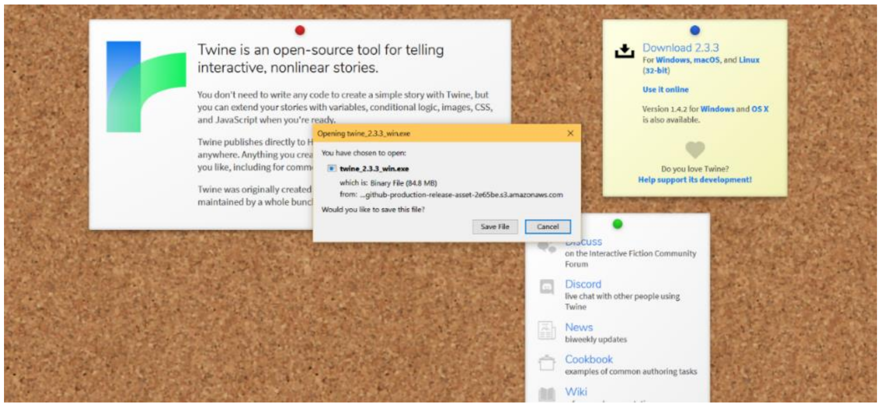
In this section:

➥ Installation

If you haven’t downloaded Twine yet, here’s a quick note on setup.

If you'd like to just check Twine out online before downloading it later on, you can do that here, but I **strongly recommend** downloading the program instead of working online in general. It's easier to keep games organized and saved that way, especially if you plan to work on them extensively or you have a project that involves embedded files that need to be accessible.

The rest of this document will assume that you have Twine **installed on your machine**.



Twine is an open-source tool for telling interactive, nonlinear stories.

You don't need to write any code to create a simple story with Twine, but you can extend your stories with variables, conditional logic, images, CSS, and JavaScript when you're ready.

Twine publishes directly to H
anywhere. Anything you crea
you like, including for comm

Twine was originally created
maintained by a whole bunc

Opening twine_2.3.3_win.exe

You have chosen to open:

twine_2.3.3_win.exe

which is: Binary File (84.8 MB)

from: ...github-production-release-asset-2e65be.s3.amazonaws.com

Would you like to save this file?

Download 2.3.3
For Windows, macOS, and Linux (32-bit)

Use it online

Version 1.4.2 for Windows and OS X is also available.

Do you love Twine?
Help support its development!

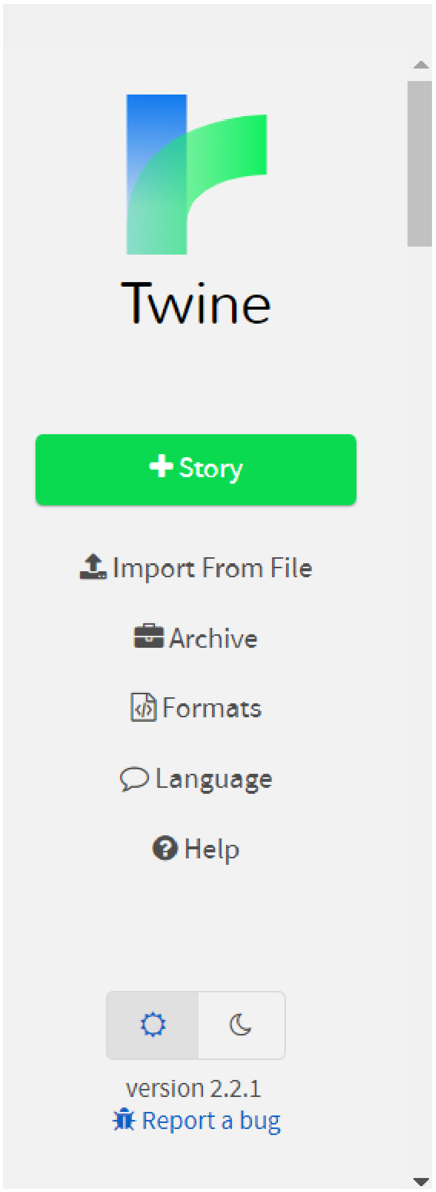
Discuss
on the Interactive Fiction Community Forum

Discord
live chat with other people using Twine

News
biweekly updates

Cookbook
examples of common authoring tasks

Wiki

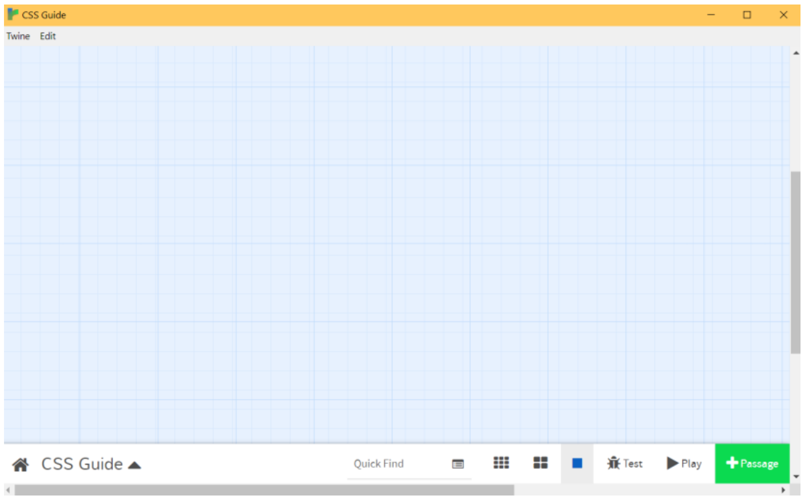


TO INSTALL TWINE:

- Head to http://twinery.org/
- Download the latest version for your operating system
- Run the binary file you just downloaded
- Follow the installation instructions
- Open Twine

Once you've downloaded and installed Twine, open it up and click "**+ Story**."

Name your project, and you'll be taken to your nodemap, which looks like this:



➥ Story Formats & Your First Passages

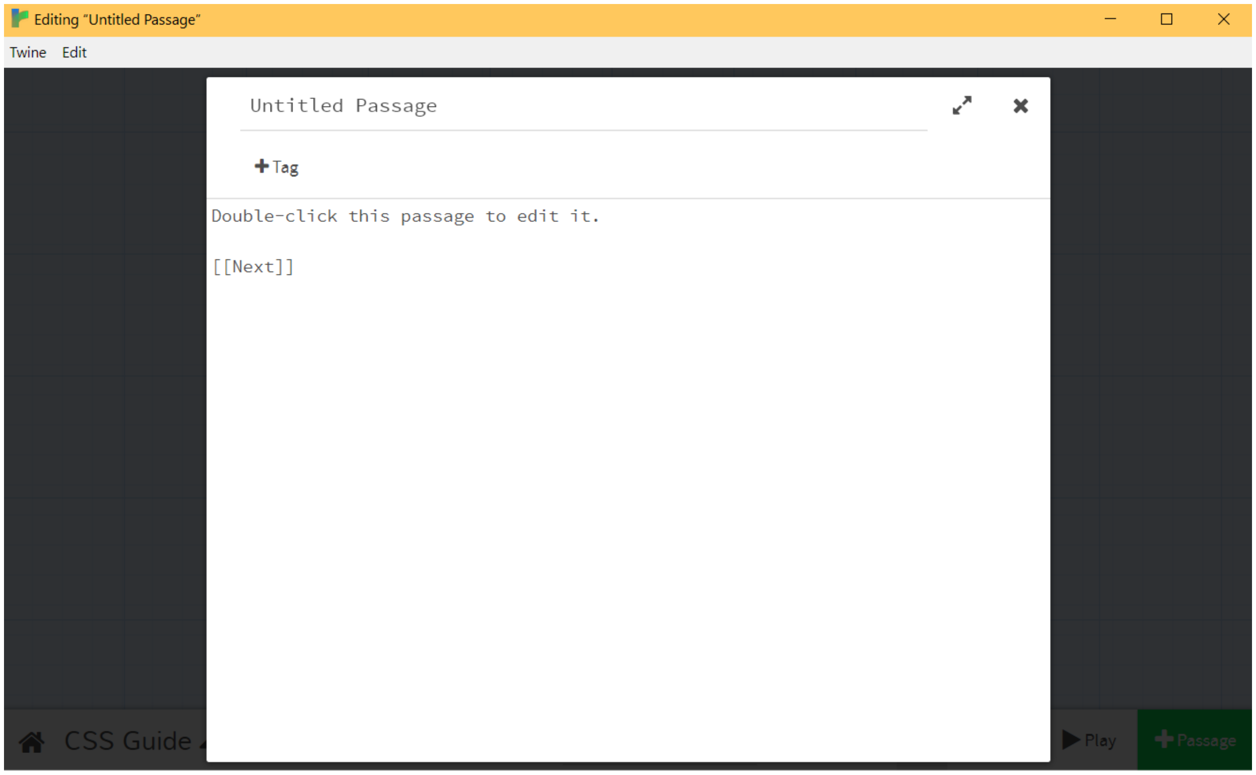
A **story format** "controls the appearance and behavior of your story during play," per Twine's format menu.

Depending on the story format in use, your project **may use different methods and syntax to achieve certain effects**. For instance, customizing the color of your links works differently in SugarCube than it does in Harlowe.

The default story format in Twine 2 is currently **Harlowe 3.1.0**, and this is the format your project will start in.

For now, try checking out how each format looks. Before taking a look, open your first passage by double clicking on it.

Create a simple link by adding "[[Next]]" to the end of the passage, then click out to return to the main screen.



```
Double-click this passage to edit it.

[[Next]]
```

Click **“Test”** or **“Play”** in the lower right of your window to get a load of Harlowe.

This format has large text and a dark color scheme, and if you click on your link, you’ll notice that Harlowe also has an Undo/Redo feature that allows you to return to the previous passage, in addition to displaying "Turns" and a Debug Mode while testing.

Changing Your Story Format

To check your story format, click on the title of your project displayed in the lower right. Select “**Change Story Format.”**

A menu will come up, showing the format you’re currently using; for now, **select SugarCube 2.21.0** and click out of the menu to close it.

Now that you’ve changed formats, click **“Test”** or **“Play”** again.

SugarCube 2.21.0 has a large sidebar menu and a dark color scheme. (It also starts in Debug Mode, which will make your links and other custom styling look weird at first; you can turn this off with the button in the sidebar.)

Not into it? There are other formats as well, all of which are usable, but most of which are no longer updated, such as SugarCube 1.0.35 and Harlowe 1.x. These will be discussed a bit later on, but:

! For the purposes of this guide, it’s recommended that you choose the most recent version of either SugarCube or Harlowe.

If you’re confident with your knowledge of CSS and HTML already, however, you can try using **Snowman**. This is a minimal format; you’ll notice the absence of any menus, undo options, links, or text colors to begin with. This guide will not be covering Snowman, but you can learn more about the format via Dan Cox, as well as the Snowman 2.0 documentation and this github repository.

Although you will eventually be able to customize just about every element you’ll see in these formats, taking a look at each one now can help you decide what kind of UI elements you'd like to include in your own project.

➥ Twine Resources

It’s important to note that each story format has **documentation** that you can open within Twine from the **format menu**, or documentation and resources that you can find online provided by its creators and users. Several external links to other useful sites have been provided below.

Twine:

- Twine Wiki
- Twine Cookbook
- Tutorial by Allison Parrish
- Dan Cox's Twine 2 Videos
- Twine 2 Guide

There are also many, many other Twine games and creators that can be found on itch.io, Twitter, and elsewhere if you're looking for inspiration.

SugarCube:

- 1.x Official documentation
- 2.x Official documentation
- Notepad++ syntax mode
- Source code repository and bug tracker

Harlowe:

- Official documentation
- Source repository and bug tracker
- How to change your story's appearance
- Notepad++ syntax mode
- Harlowe reference

Snowman:

- Official documentation
- Snowman (Twine Cookbook)

There are numerous other resources that you can find with a little digging as well. The Twinery Forums (now read-only), Twine Q&A, and Intfiction.org's dedicated Twine category all contain threads where people are looking for (and usually receiving) help with all kinds of questions, many of which pertain to CSS and HTML usage.

- Twinery Forums (Read-Only)
- Twine Q&A
- Intfiction.org Twine Category

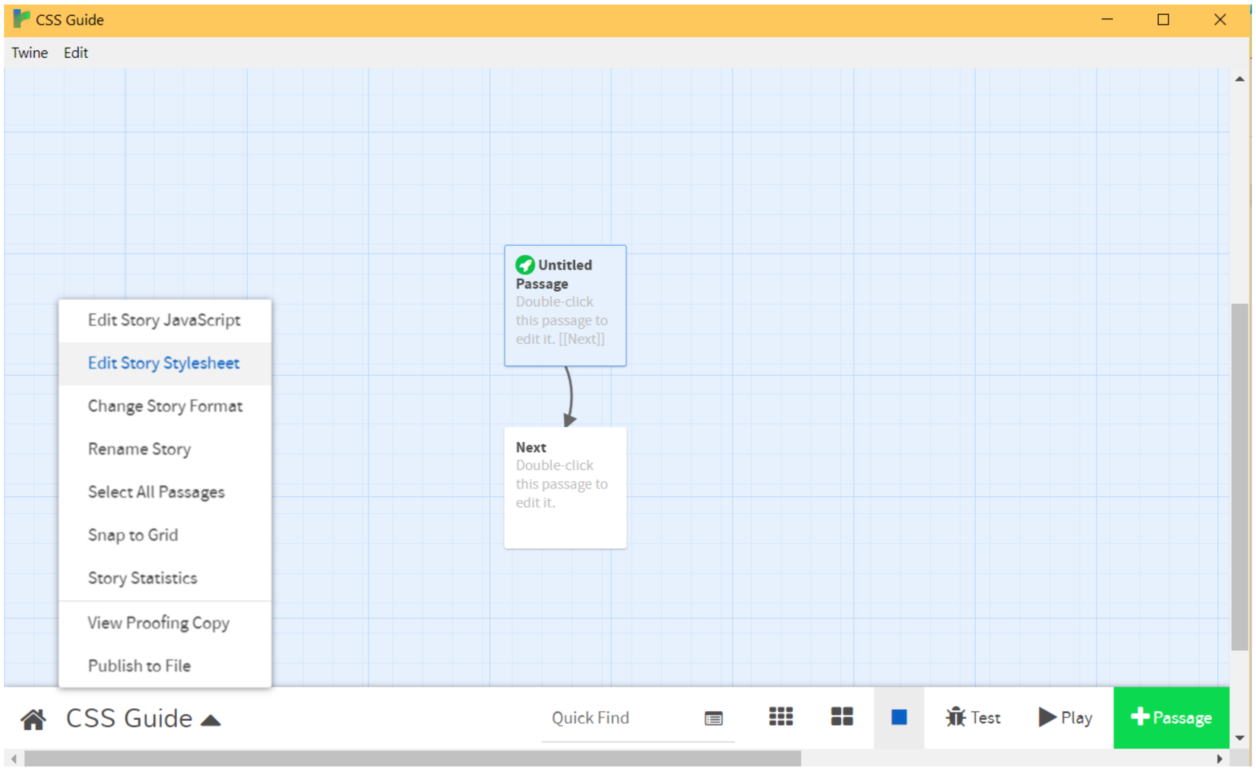
There's also a Twine subreddit, Twitter, and Discord server.

- Twine Subreddit
- Twine on Twitter
- Twine Discord

➥ CSS Basics

CSS stands for **Cascading Style Sheets**; CSS is a language used to define how elements are displayed in **HTML** (HyperText Markup Language).

To customize your CSS in Twine, click on your story title in the **lower left**, then select "**Edit Story Stylesheet**."



A new project starts with **no custom CSS**. We're not going to create anything new yet, but here are some **basics** to keep in mind:

When you add to your stylesheet, you can either create a new element, or override an existing element. Defaults will not be immediately visible in the stylesheet, which will be blank when first opened. When overriding an existing element, the way Twine formats name their elements may be **different** from standard CSS.

Much of what you will be doing in these tutorials will involve taking specific **selectors** — deciding which content we want to style — and **declaring** what we'll be changing about them using **properties and values**. (Certain selectors will require you to use HTML within your passages — but that's also something that will be expanded on later.)

Here’s a diagram of what many of your customizations will boil down to:

```
selector {

property: value;

}
```

This is a **declaration block** applied to a selector**.** This will be discussed in more detail in the next tutorial on **text styling.**

➥ CSS Resources

There are numerous resources for learning about CSS online, among them Mozilla and W3Schools. When it comes to working with Twine, I highly recommend the following entries in the Twine Cookbook:

- Defining CSS
- Reviewing CSS Selectors

These are excellent introductions to the basics of CSS, and will not bog you down with extra information that may not be relevant to Twine.

It's always worth noting that there's absolutely no shame in **using a search engine** to help solve your CSS and other programming problems. In fact, it's something you should be doing regularly. Being able to seek out the answers to your bugs and questions is always a huge advantage, and it's very likely to help you become more familiar and comfortable with Twine.

Even so, it can be difficult to find documentation for certain things, and some may find existing documentation difficult to understand or begin using. The Twine Cookbook advises that:

> "Overwriting existing CSS rules is an *advanced* technique. It has the potential to significantly change the presentation of content."

This is correct; CSS errors can lead to some very unpalatable bugs, but they can be fixed, and these tutorials have been created to help users become comfortable and confident in exploring this advanced technique.

Throughout this guide, I'll be doing my best to present things in a concise and accessible way. Without further ado, here are five more tutorials that cover how to achieve commonly desired visual features in Twine!

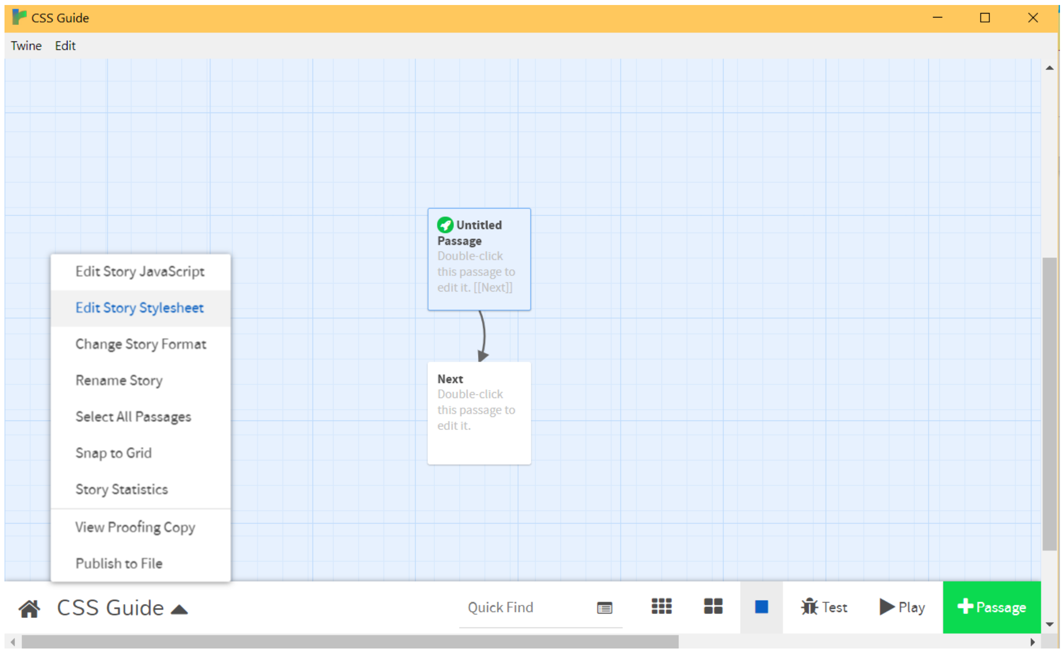
TEXT STYLING

In this section:

- **Text Properties**
- **Choosing & Importing Fonts**
- **Font Colors & Sizes**
- **Custom Classes**
- **Other Properties: Text Shadows, Alignment, etc.**
- **Implementing Custom Classes Using HTML**
- **Selection Properties**
- **Exercises**

➥ Getting Started

To begin, open the Story Stylesheet.



If you're using SugarCube or Snowman, start with this:

```
body {

   /* Your CSS Here */

}
```

! <u>Delete the **entire** " */* Your CSS Here */* " portion</u>! /* */ is used to comment that line out so it doesn't interfere with your stylesheet.

You must have both **{braces}** ; all of your properties will go {between them}.

The "**body**" selector will affect **all of your game's text** and is a good step to set your default font and keep your text at a consistent size.

However, the code above **won't work in Harlowe**; for Harlowe, you'll use:

```
tw-story {

   /* Your CSS Here */

}
```

Neither of these selectors should begin with a period.

As before, you'll need both braces with any selector you're ever working with. (I've accidentally deleted an end bracket and briefly broken an entire game before. Having to go back and hunt for where you either deleted or forgot to add one is unpleasant!)

➥ Choosing and Importing Fonts

There are a lot of ways to style text, but a good place to start is picking your project's **default font**.

Before setting a font, there are a few considerations to make: readability, style, and whether or not you'll need to **import** the font.

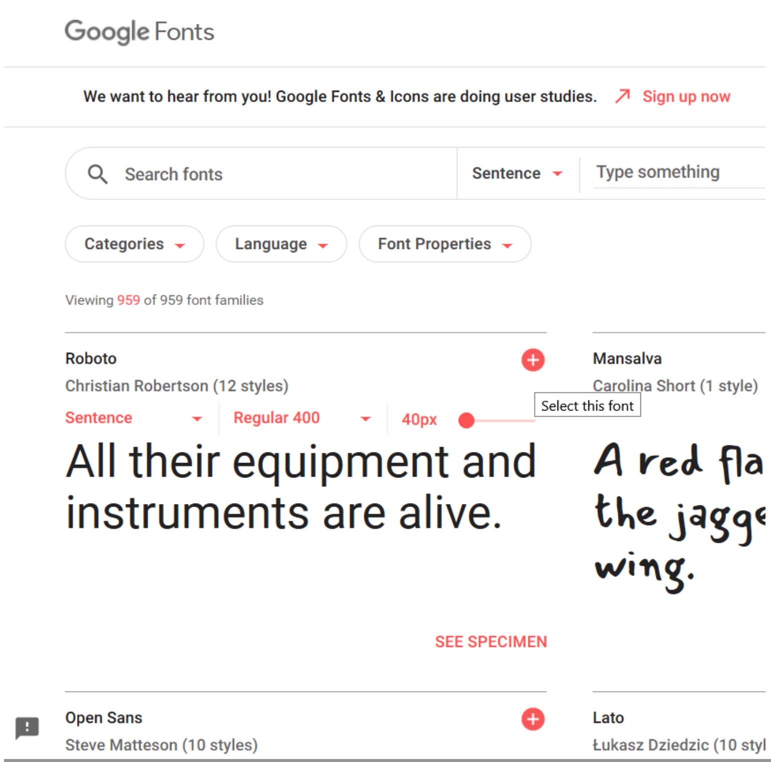
I'm by no means a font designer, so I just think of fonts in a few basic categories: Serif, Sans Serif, Monospace, and Handwritten/Embellished/Other.

I'm partial to Serif fonts (like Georgia, Times New Roman), but Monospaced (Courier, Inconsolata) and Sans Serif (Arial, Helvetica) fonts are also nice; they tend to have a sleeker appearance, are visually accessible, and can look good in a variety of genres.

Other fonts, whether handwritten, heavily embellished, very stylized, or similar, should be used sparingly; these are fun for adding some visual interest or immersion to small portions of text or to a single passage, but having your entire project in cursive or another irregular font is not recommended. It impacts readability negatively in almost every case.

Many of the fonts I mentioned above are fonts that you can easily set in Twine without worrying about whether they will display properly to your players; just about everyone will have Arial or Georgia on their operating system and a browser that is capable of recognizing and displaying the font.

But if no common fonts feel right for your project and you're looking for something a little more unique, you can use Google Fonts to choose and embed a different one. These fonts must be embedded in order to display correctly.



We want to hear from you! Google Fonts & Icons are doing user studies.

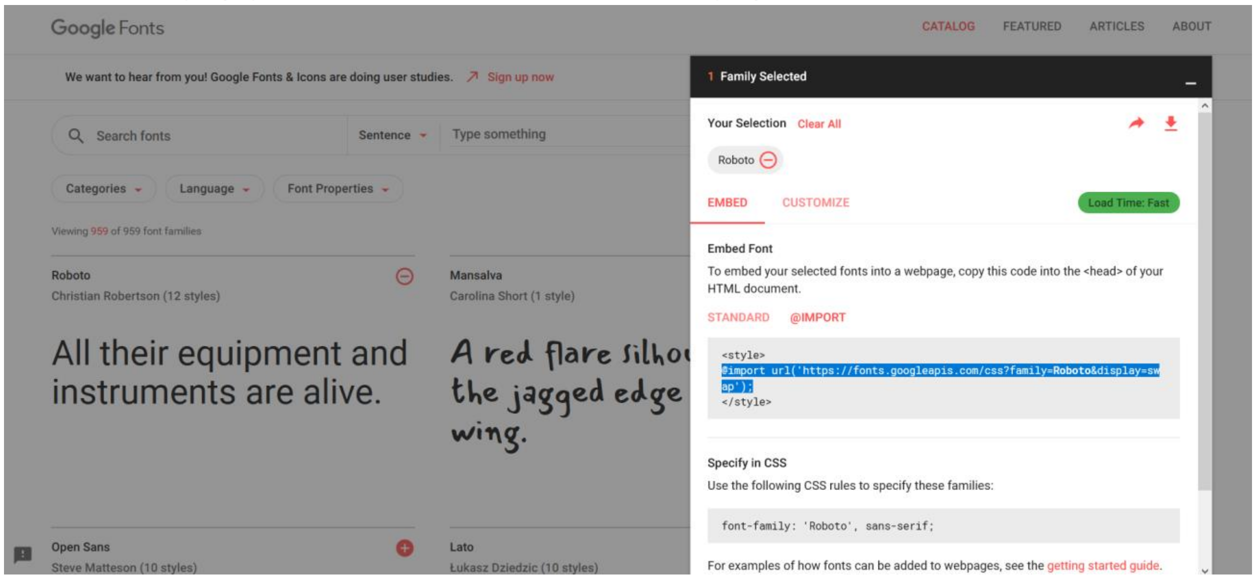
Viewing 959 of 959 font families

All their equipment and instruments are alive.

A red fla
the jagge
wing.

Spend however long you want locating your ideal font, and then be sure to **click the red + button**.

From there, click on the "**1 Family Selected**" bar to open a window with more info about your chosen font; from here, click **@IMPORT** instead of STANDARD.



We want to hear from you! Google Fonts & Icons are doing user studies.

Viewing 959 of 959 font families

All their equipment and instruments are alive.

A red flare silhou
the jagged edge
wing.

Embed Font

To embed your selected fonts into a webpage, copy this code into the <head> of your HTML document.

```
<style>
@import url('https://fonts.googleapis.com/css?family=Roboto&display=sw
ap');
</style>
```

Specify in CSS

Use the following CSS rules to specify these families:

```
font-family: 'Roboto', sans-serif;
```

For examples of how fonts can be added to webpages, see the getting started guide.

Copy and paste **only the selected line** (you don't need the <style></style> tags for Twine) into the top of your stylesheet, before your other selectors. Then copy and paste the **CSS specification** into your **body** or **html** properties, between the braces.

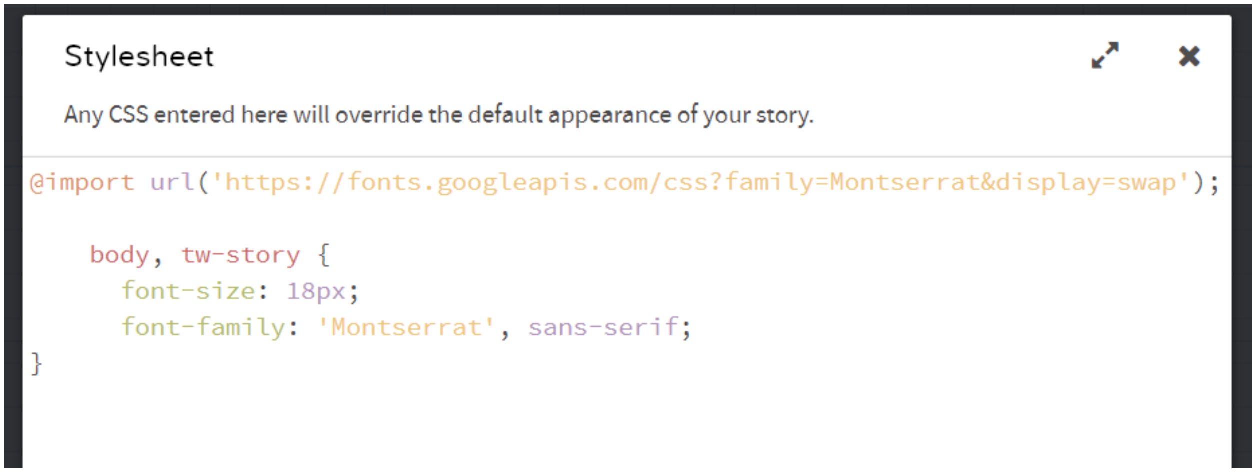
➥ Font Colors & Font Sizes

When it comes to size, I **strongly recommend** that you use a font **no smaller than 14px** for visibility; any smaller, and it can become very difficult for players to read or focus on.

To set your font size, add the following (though feel free to use a larger size if you want to) to your **body** or **tw-story** properties:

```
        font-size: 16px;
```

Your stylesheet should now look something like this:


Stylesheet
Any CSS entered here will override the default appearance of your story.

```
@import url('https://fonts.googleapis.com/css?family=Montserrat&display=swap');

   body, tw-story {
     font-size: 18px;
     font-family: 'Montserrat', sans-serif;
}
```

(If you're still deciding on a story format, use both body and tw-story separated with a comma so that you can easily compare between formats, like I'm doing for the purpose of the tutorial.

Once you've decided on your format, though, **remove what you aren't using**. If you're using Harlowe, you don't need "body," and if you're not, you don't need "tw-story." Keep everything else.)

What about **color**? The default, depending on your story format and version, will be either black (#000000) or white (#ffffff).

If you'd like to change to another color, find its hex code (https://www.color-hex.com/) and add this to what you have so far, replacing the *s with the 6-digit hex:

```
            color: #******;
```

I generally recommend sticking to black or white early on, unless you have a specific color scheme in mind where your background color(s) will complement your font color(s) and remain easily visible.

Backgrounds?

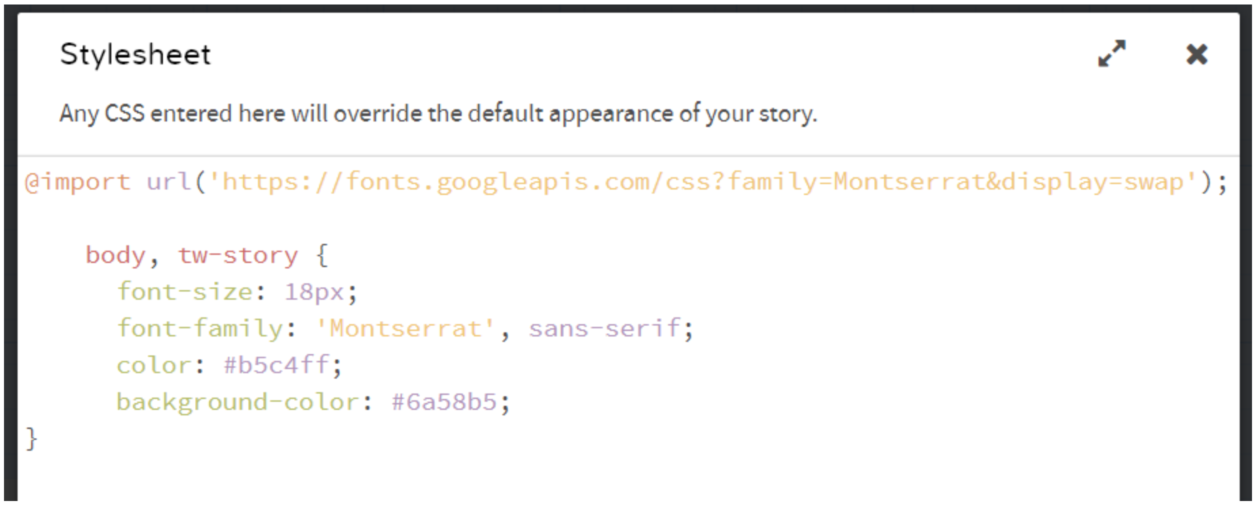
A potential issue for some of you right away: you may have chosen a format with a black backgrounds by default, but would prefer white BGs and black text and don't want to change to a new format. Or you could be having the exact opposite problem, or maybe you want to use a different background color altogether.

Other tutorials in this guide will provide detailed instructions on changing background elements, but for you to be able to stick with your preferred format, here's a simple fix. If needed, you can change the default background color right now by adding:

```
            background-color: #******;
```

This will affect the background of your entire project.

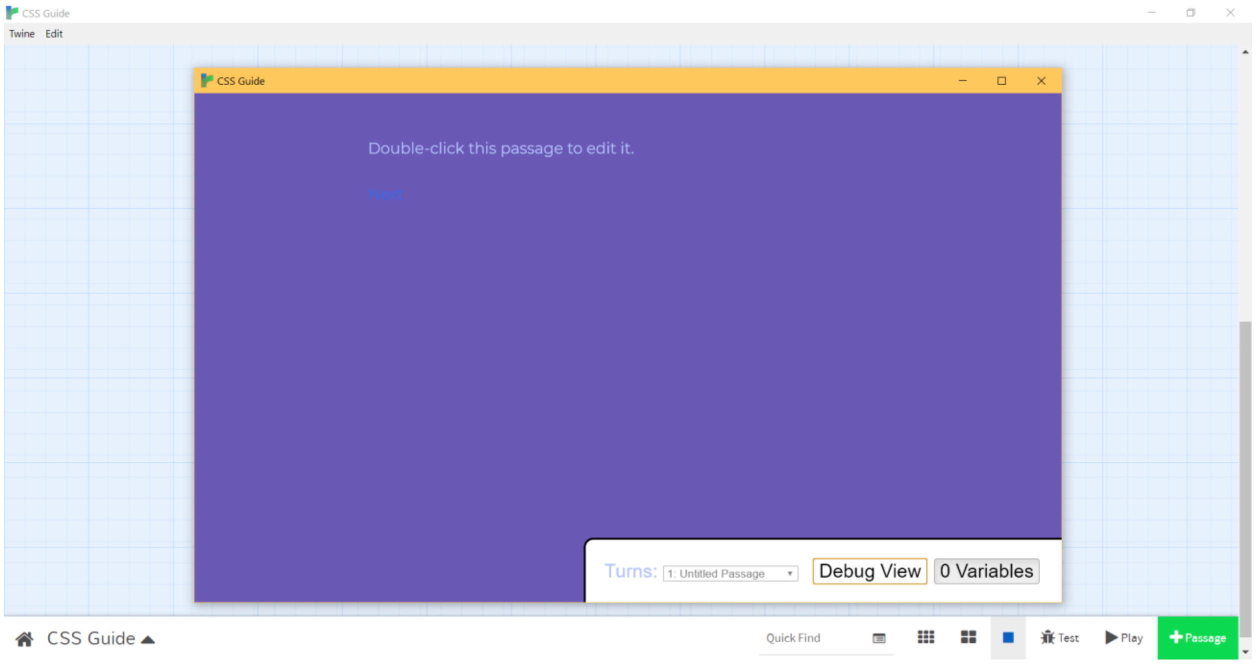
Now your stylesheet should look like this:



Any CSS entered here will override the default appearance of your story.

```
@import url('https://fonts.googleapis.com/css?family=Montserrat&display=swap');

    body, tw-story {
      font-size: 18px;
      font-family: 'Montserrat', sans-serif;
      color: #b5c4ff;
      background-color: #6a58b5;
}
```

And your project is starting to take shape!



I like this background color, but now the links are hard to see. If your color scheme has a similar problem, don’t worry. **We’ll fix this in the next tutorial.**

➥ Other Properties

There are **other properties** you can use to style text, like shadows, alignment, transformation, spacing, and more, but these are things you wouldn't typically apply as a default — so we're going to create a class you can use for **emphasizing text**.

This idea can be used for a variety of effects, and you don't have to use it in your final project, but it's useful to know how to do.

Copy and paste the following into your stylesheet, after your previous selector's final brace:

```
.emphasis {

   /* Your CSS Here */

}
```

This must have a **period** preceding it; there should be **no space** after the period.

You can use this selector to experiment with other effects. Try adding the following within the braces:

```
color: #f3cb88;
text-shadow: 0px 0px 10px #fff;
text-transform: uppercase;
font-weight: 800;
letter-spacing: 1px;
```

Experiment with the **text shadow** to change its color or increase its size, location, and diffusion; above is what I usually start with.

Transforming your text isn't always necessary; you can always just control the case yourself, but it can be useful for things like links or special emphasis where you know you want that kind of text to be rendered in either all caps, lowercase, or with an Emphasis Capital each time without having to worry about typing it yourself.

The “heavier” the **font weight**, the bolder your text will appear. Font weight does not use px; just a numeric value will work.

Letter spacing can give your text an interesting effect as well. There’s also a “word-spacing” property that works similarly.

Alignment is something we haven't added yet (since our embellishments are currently inline), but the "text-align" property can justify your text, or align it to the center, left, or right. If you like the look of justified text (I do), you could also apply it as a default by adding "text-align: justify;" to your body or tw-story selector; similarly, if you're really adventurous, you could make all of the text in your game center- or right-aligned with "text-align: center;" or "text-align: right;".

Text decoration is another way to mess with your text; it can be in the form of an underline, overline, strikethrough, or multiple at once (like an underline paired with an overline).

You don’t have to use a strikethrough in your project, but let’s create one just to demo the property.

Add this below your emphasis selector in the stylesheet:

```
.strike {
   text-decoration: line-through dashed;
   text-decoration-color: #ff0000;
   color: #000;
}
```

("Dashed" can also be replaced with "dotted" or "solid.")

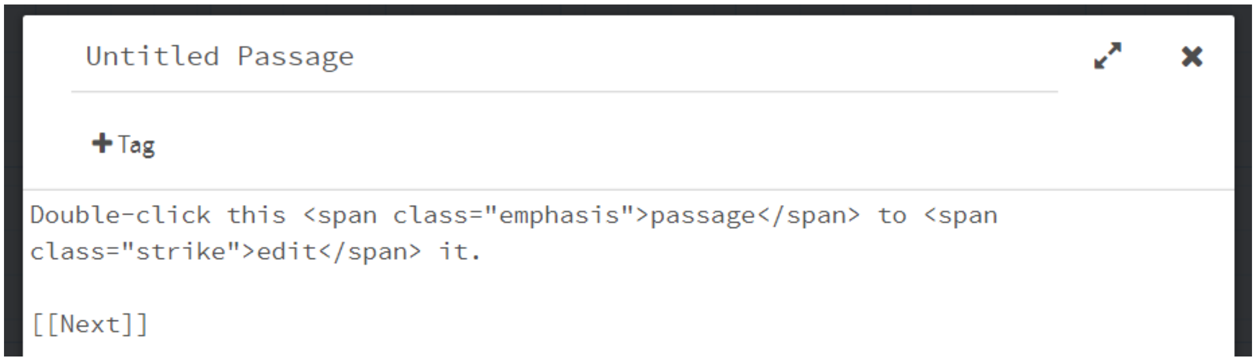
➥ Using HTML

To implement your emphasis and strike classes, you'll need to use HyperText Markup Language, or **HTML**.

Open up your first passage; its default text is fine for now. To get the emphasis and strikethrough properties working, the specific words or phrases being affected must be enclosed within the correct **HTML tags**.

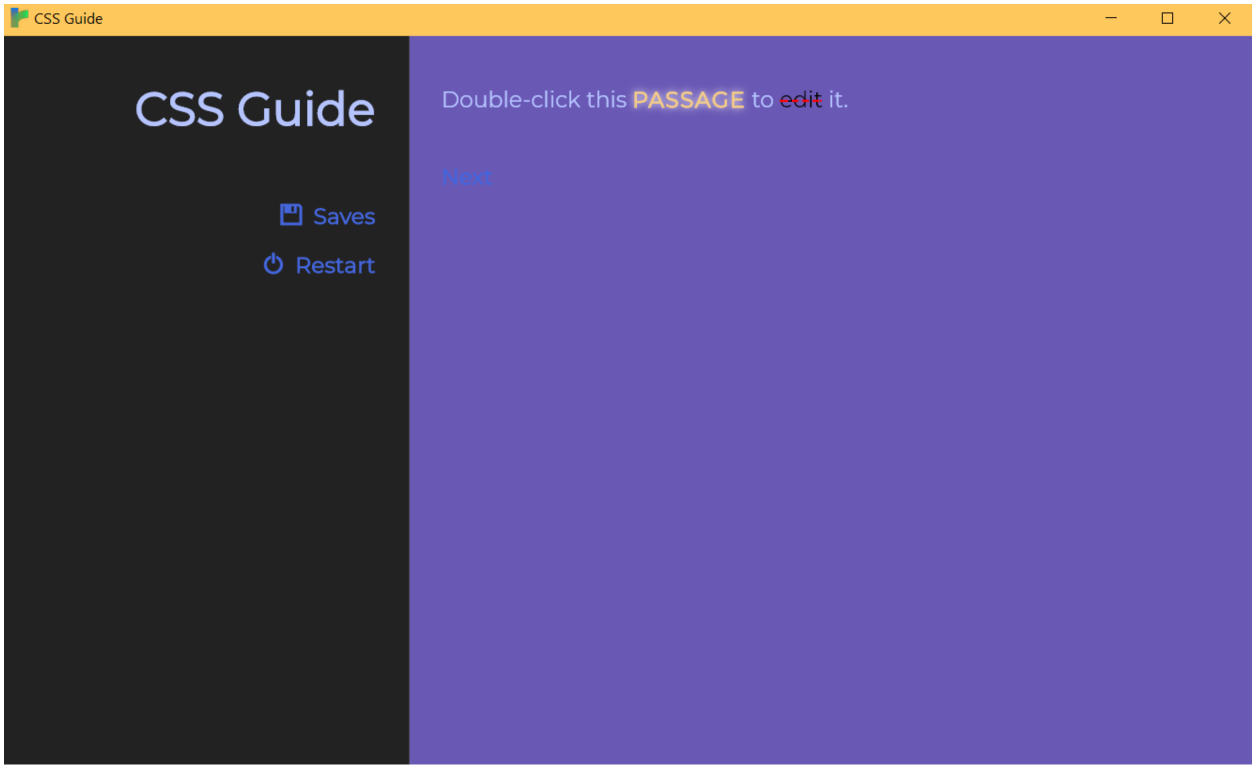
For these types of effects, use **<span class=""></span>** — the name of the selector ("emphasis" or "strike") goes in quotations, and the word or phrase being embellished goes between the tags. The "**span class**" tag keeps your text inline, as opposed to a **div class**, which will not. A div class will typically create the appearance of a block or line breaks (unless you set it to display inline, which you don't need to go to the trouble of doing for this).

Here's how this will look in the editor:



```
Double-click this <span class="emphasis">passage</span> to <span
class="strike">edit</span> it.

[[Next]]
```

And here’s how that will look when you test it:



Looks kind of goofy so far, but that's okay for now.

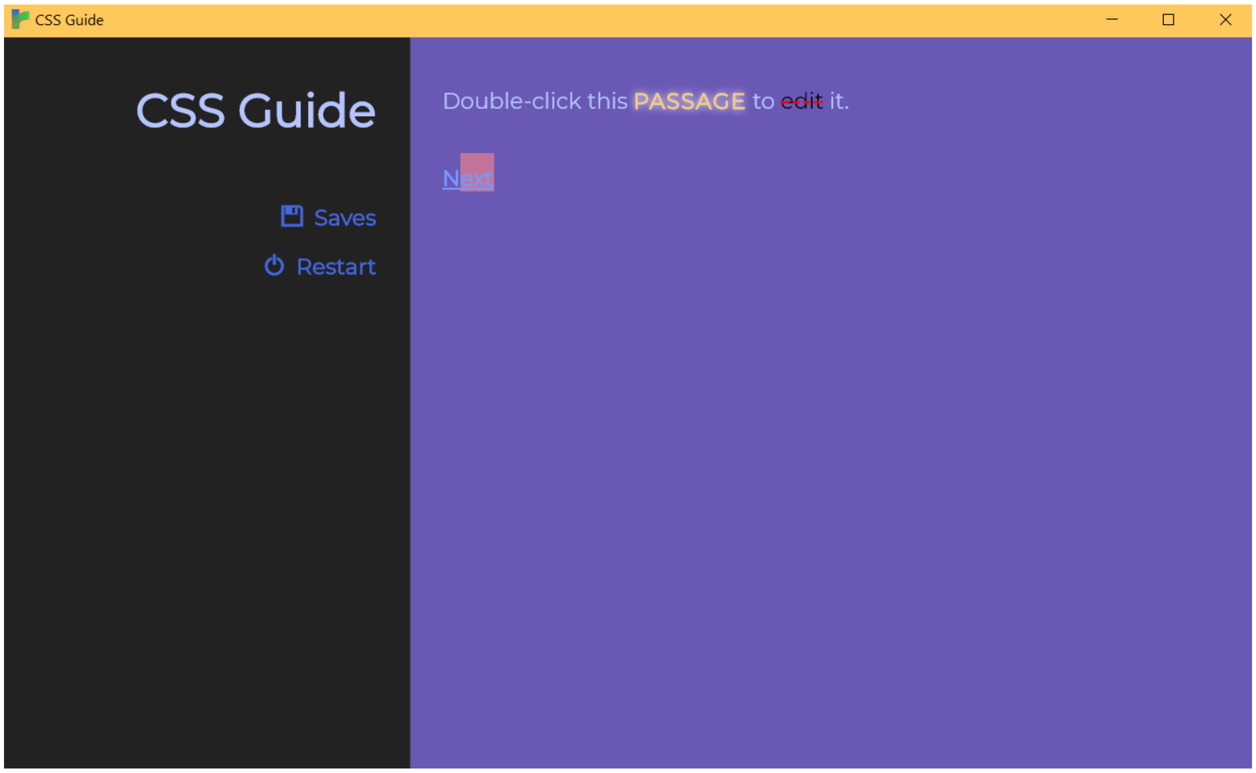
➥ Highlighted Text

There's one more thing you may be interested in adding to your stylesheet: if you want your text to have a specific color or background color when highlighted, we can override the default text selection properties using CSS like so:

```
::selection {
   background: #ffb7b7;
}

::-moz-selection {
   background: #ffb7b7;
}
```

The different selectors are to ensure that different types of browsers cooperate; to my knowledge, these have to stay separated and you cannot combine them with a comma.

Some browsers may still ignore your selection override, but as far as I'm aware, Firefox, Chrome, and Safari will display it. It's still a nice touch if you're interested in making sure your color scheme really comes together! You can check how it looks it by testing, then highlighting your text.



The whole look here is kind of chaotic and weird so far for the purpose of being very visible as a demo, but we'll be adjusting and polishing that kind of thing as we go. It can take a while for a full stylesheet to come together; you may go through several changes in color scheme, fonts, and other properties.

➥ Exercises

EXERCISE 1:

- **Replace the default text** with **your own opening lines** and **at least one [[link]]** leading to another passage. Anywhere between two and five sentences is effective; this can be sample text or text from a first draft.

EXERCISE 2:

- Change your project's **default font** either by selecting a **common font** or **importing one** from Google Fonts. You may also want to adjust the size and color of your default text; **14px or larger** is a good size.

EXERCISE 3:

- **Design and implement** your own **custom class**. This will be an entirely customized span class that you will create using CSS and implement using inline HTML within a passage.

LINK STYLING

In this section:

➥ Types of Links

There are four types of **link selectors** that you can use. The main three states are:

- **Default** — a link that isn't being hovered over and has not been clicked on previously.
- **Hover** — a link over which the mouse cursor is positioned.
- **Visited** — a link that has been clicked on previously.

A lesser-used state:

- **Visited (Hover)** — a visited link with a hover pseudoclass different from the default hover pseudoclass.

➥ Pseudoclasses

Per w3schools, a **pseudoclass** "is used to define a special state of an element." Click here for more information, but be aware that it might be more info than you need right now. Our focus will essentially be on one specific pseudoclass.

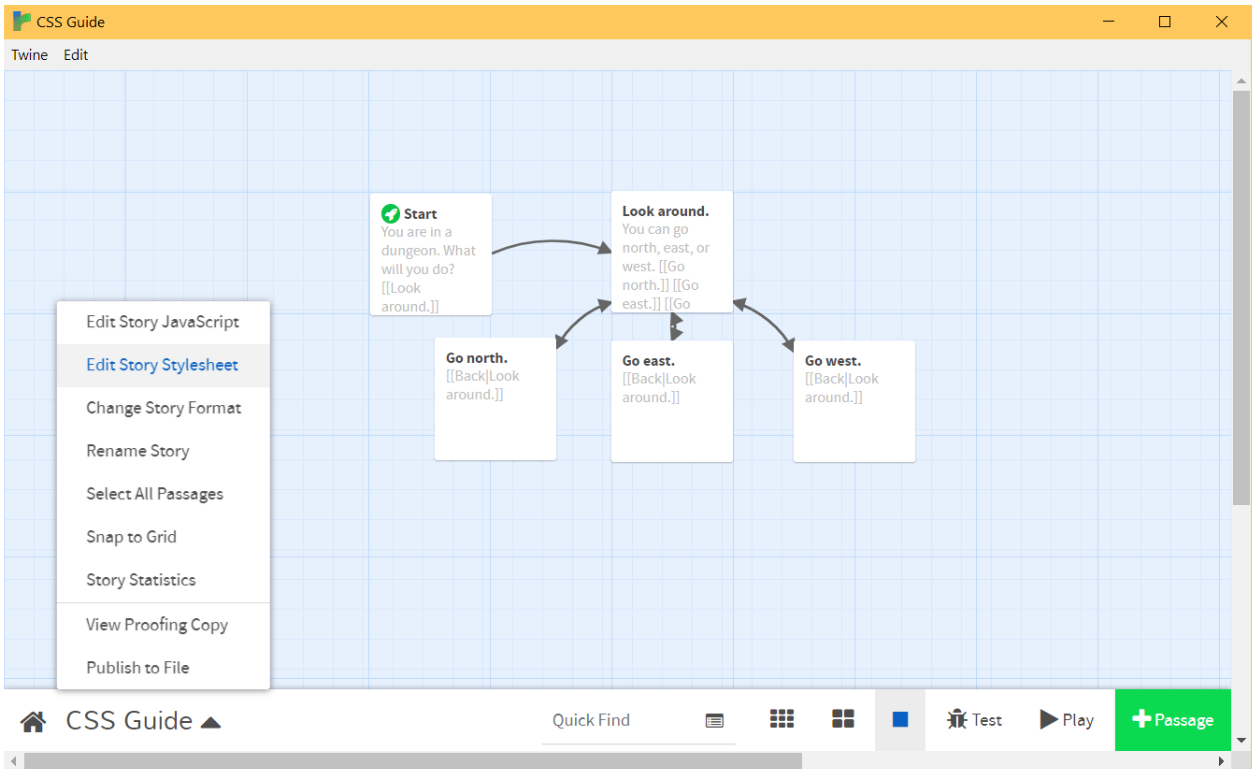
For our purposes, the **special state** we'll focus on is the **hovered** state or **:hover pseudoclass**, which appears when the player is mousing over a link.

! While :visited is an extant pseudoclass, it is not compatible with Twine! We'll be discussing alternative approaches to visited links.

➥ CSS Implementation

In just about every tutorial, the way you'll input your CSS will vary between Harlowe and other formats. In SugarCube, you'll use more conventional selectors than the special ones you'll need for certain elements in Harlowe, but the following will demonstrate how Harlowe also simplifies certain things, like visited links.

To start working with your CSS again, click on your project's title in the lower-left, then select "Edit Story Stylesheet."



SugarCube 2.x

Default links*:

```
a {
   /* Your CSS Here */
}
```

***** The font size and family you set here will be used as a **default** for **all links**; if you don't set one, the global default you set in the last tutorial will take over.

Hover links:

```
a:hover {
   /* Your CSS Here */
}
```

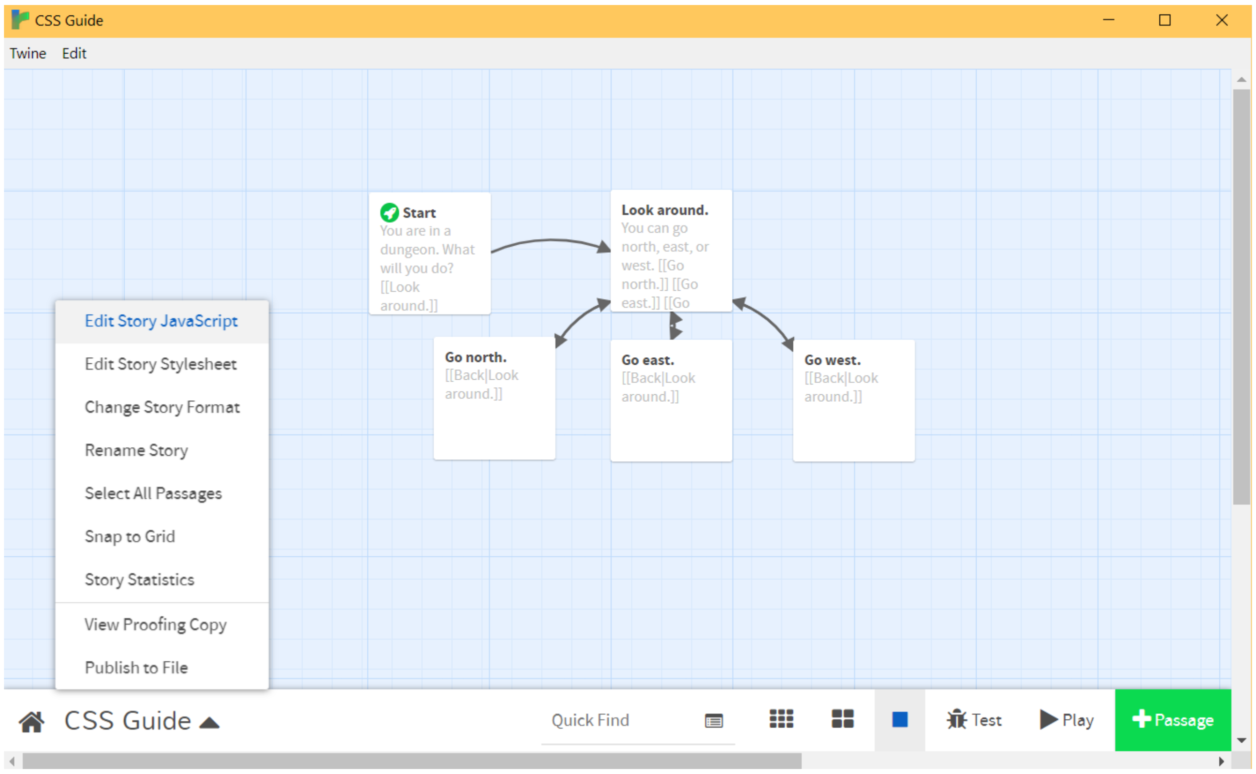
(There should be no spaces in this selector.)

Visited links?

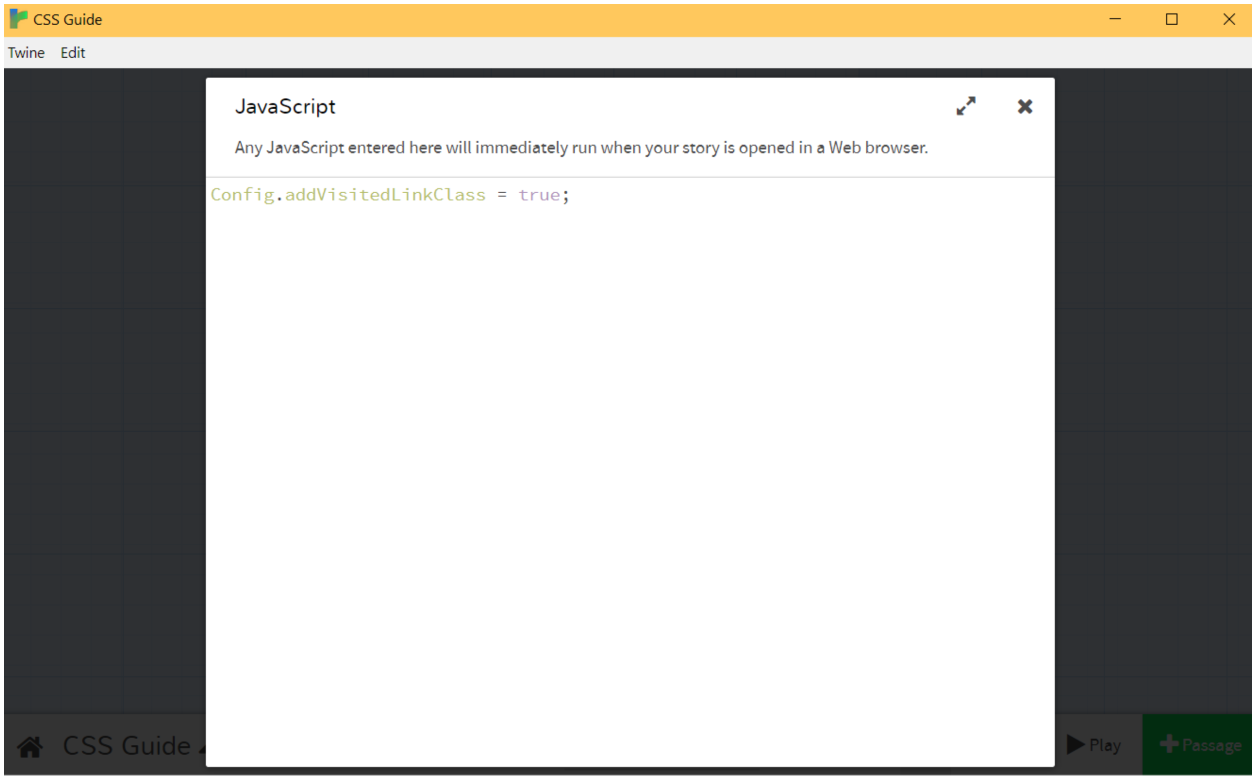
The :visited pseudoclass will not work! **If you want to use visited link states, your best bet currently is SugarCube 2.x or Harlowe.**

*In SugarCube 2.x**, this is how you will need to **configure visited links** if you want to use them:

1. Click "Edit Story JavaScript"



2. Paste "*config.addVisitedLinkClass = true;*" (without quotation marks) into your project's JavaScript



Any JavaScript entered here will immediately run when your story is opened in a Web browser.

```
Config.addVisitedLinkClass = true;
```

3. Paste the following into your Story Stylesheet:

```
a.link-visited {

    /* Your CSS Here */

}
```

4. Fill in your desired properties between the braces. Be sure to delete the entire */* Your CSS Here */* line.

Visited links (hover):

To style a different hover pseudoclass for your visited links:

```
a.link-visited:hover {

    /* Your CSS Here */

}
```

Harlowe

In Harlowe, you'll use the *tw-link* selector **instead of** *a*. There is also a *tw-enchantment* link class; for a quick explanation of what this is, click here.

Default links:

```
tw-link, tw-enchantment {

    /* Your CSS Here */

}
```

Hover links:

```
tw-link:hover, tw-enchantment:hover {

    /* Your CSS Here */

}
```

Visited links:

Styling visited links is super simple in Harlowe, since the ability is built in and needs no extra configuration!

```
tw-link.visited {

   /* Your CSS Here */

}
```

Visited links (hover):

```
tw-link.visited:hover {

   /* Your CSS Here */

}
```

➥ Macros & Named Hooks

! If you are using SugarCube, you can skip this section.

The Twine Cookbook notes that Harlowe strongly encourages authors to style stories using **macros** and **named hooks**.

Here is a quick overview of what that means, how to achieve it, and when and well as why you might want to try it.

Macros in Harlowe

From the Harlowe manual: "a **macro** is a piece of code that is **inserted into passage text**." This means that when you're setting up a macro, you do so directly within the passage editor alongside your existing text and links.

Macros' usage and syntax differ between story formats; Harlowe handles them very differently than SugarCube does, and has some built-in macros that SugarCube does not.

Typically, for a macro to take effect, it is attached to a **hook**.

Using Named Hooks

A hook is "a means of indicating that **a specific span** of passage prose is **special** in some way;" this is designated by enclosing text in **[**brackets**]**. To change this special text, you attach a **macro** to the hook; all text **within** the hook will be affected, and any text outside of the hook will not.

This is all true of a typical hook. **Named hooks** are slightly different. An author can create their own named hooks, but for now we'll just look at the built-in named hooks that are used in customization. These **special named hooks** can affect **more** than just a specific span of text.

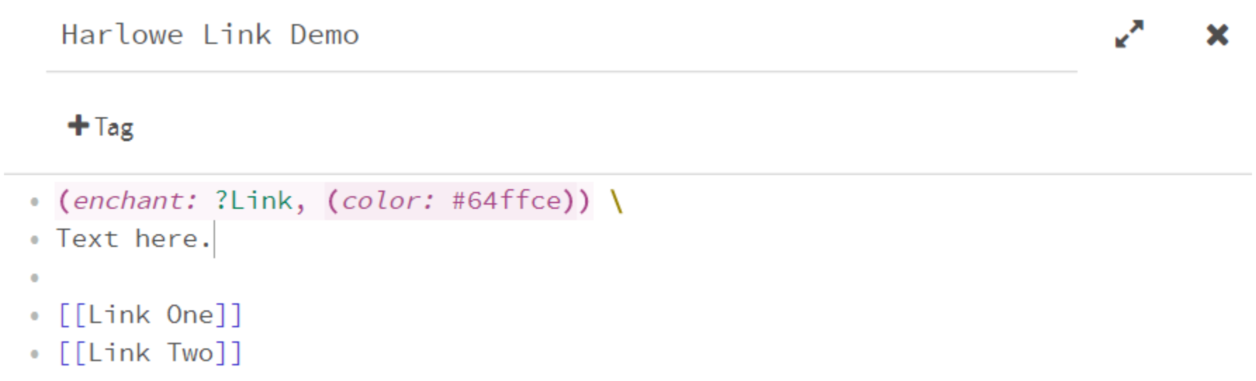
There are **four** hooks with special names that allow you to style specific elements of your Harlowe project **without opening** the CSS stylesheet editor. These hooks correspond to HTML elements, which correspond to specific CSS selectors:

- **?Page** — corresponds to the *<tw-story>* element
- **?Passage** — corresponds to *<tw-passage>*
- **?Sidebar** — corresponds to *<sidebar>*
- **?Link** — affects all passage links and (link:) macros in the passage

For now, this will just be a demonstration of usage of the **?Link** named hook.

Hook: **?Link**

Using the (enchant:) macro in conjunction with other styling macros, the appearance of your links can be altered like so:



```
(enchant: ?Link, (color: #64ffce)) \
Text here.

[[Link One]]
[[Link Two]]
```

When to Use Macros vs. When To Use the CSS Stylesheet

Since all of the above named hooks correspond to specific, existing selectors, you can continue customizing your Harlowe project via the CSS stylesheet if you'd prefer.

The macros & hooks method can be very convenient when you just want to alter the look of **a single passage**, but bear in mind that these changes will **only be applied to that passage.** When you click a link and progress to the next passage, the macros inserted into the previous passage will **not** carry over to the next passage; they only affect the contents of the passage they exist within.

When you want to set a **default** font, a default color, or a property that will affect **several passages** at once, I recommend heading back to the **CSS stylesheet** and working under the appropriate selectors as we've done before.

➥ Designing Your Links

This will become more of an informal design discussion now as I share my thoughts on ways to style different link states. As with a lot of aesthetic guidelines, nothing I'll talk about is a hard-and-fast rule that you aren't allowed to experiment with; these are just some text-game-geared design sensibilities that I think are useful to be aware of.

Since there has been a lot of discussion about text properties so far, I'll leave the color-choosing and font-picking and such to you (you can refer to the previous tutorial for more help), but I'll show a sample of my CSS at the end as a model.

Default:

A good default link is **very clearly a clickable element**, and having eye-catching — but not too distracting — default links is always helpful. It should stand out significantly enough from your default text, but should **avoid causing eye strain** against your background or clashing with other elements.

Last time we experimented with changing default text and background colors, but our default links ended up being eye strain city. Similarly, gray text or gray links on a white or black background is **almost never** comfortable to read. I find white-on-black easiest to focus on, and it seems popular, but bear in mind that this isn't the case for everybody.

Using a different font for your links to distinguish them further from your text can work well if your fonts look pleasant together. In most cases, it's best to use as few different fonts as possible, since having 3+ will usually give your project a less cohesive, coherent look. (That is to say, you probably shouldn't use a different font for each link state; it's usually best to keep all your links in the same font family.)

Hover:

You may not want to style your hovered links, or may want to do so only minimally. I think that even a very slight change can help **communicate interactivity** to the player, making it easier for them to identify what to click and when.

It doesn’t have to be an elaborate animation or a significant color shift; just a minute change in letter spacing or text-shadow could do the trick if you want to keep things subtle, but color changes are often helpful too.

Visited:

I find that you don’t always need to style visited links, but they can be very useful. Sometimes it **depends on the type of game** you’re making or on its mechanics.

For instance, in a more linear game or one with a lot of forward momentum where you'll rarely or never experience the same passage twice, the player will probably never see an already-visited link, and isn't likely to be confused by the lack of a visibly "visited" link.

But in games more geared toward exploring locations from a hub, examining a set of objects in a room, or similar mechanics, it can be extremely helpful to distinguish between what the player has already seen and what they have yet to click on. If you'd like to make this type of game, creating a visited link state is highly recommended.

➥ Transitions

An abrupt hovering effect can be neat in some situations, but the state change will look and feel both smoother and more attractive with a transitional effect.

There are A LOT of forms transitions can take, but my most simple go-to is the following:

```
-webkit-transition: all 1s ease;
-moz-transition: all 1s ease;
-ms-transition: all 1s ease;
-o-transition: all 1s ease;
transition: all 1s ease;
```

This looks like a lot, but the multiple prefixes here just allow for support on different browsers, ensuring that your transitions will be visible in any browser.

-webkit- = Safari & Chrome; *-moz-* = Mozilla Firefox; *-ms-* = Microsoft (Internet Explorer 10); *-o-* = Opera.

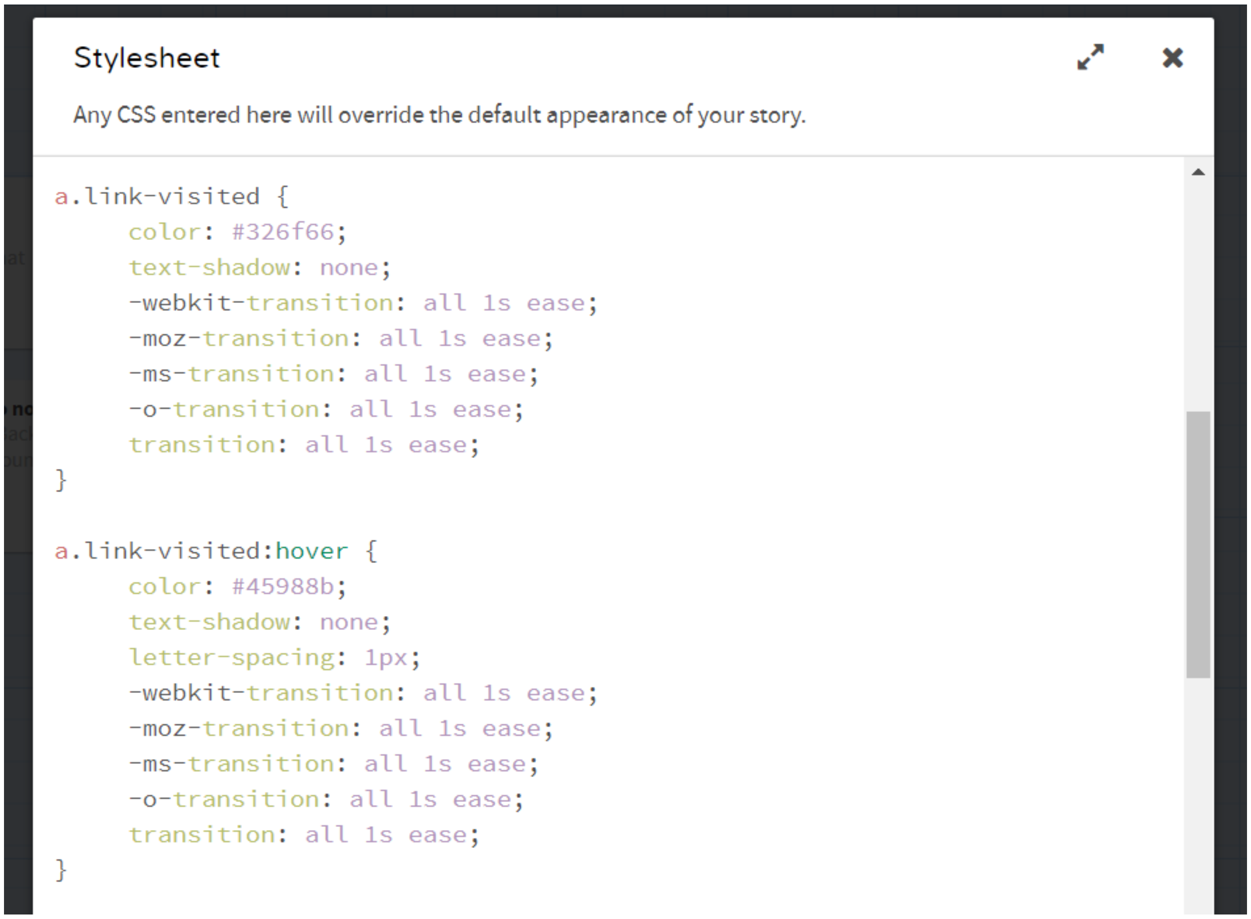
This is a simple transition that makes your state changes ease into each other in a way that's smooth, quick, and easy on the eyes. You'll want to add it to **both** your **default** (*a* or *tw-link*) and **hover** states so that it will work when hovering *and* when removing your mouse from the link.

I **don't recommend** making simple transitions longer than a **second or two** at the most. They could even stand to be quicker. Generally, you won't want a visually elaborate or time-consuming animation on an element with which the player will be regularly interacting.

➥ CSS Samples & Screenshots

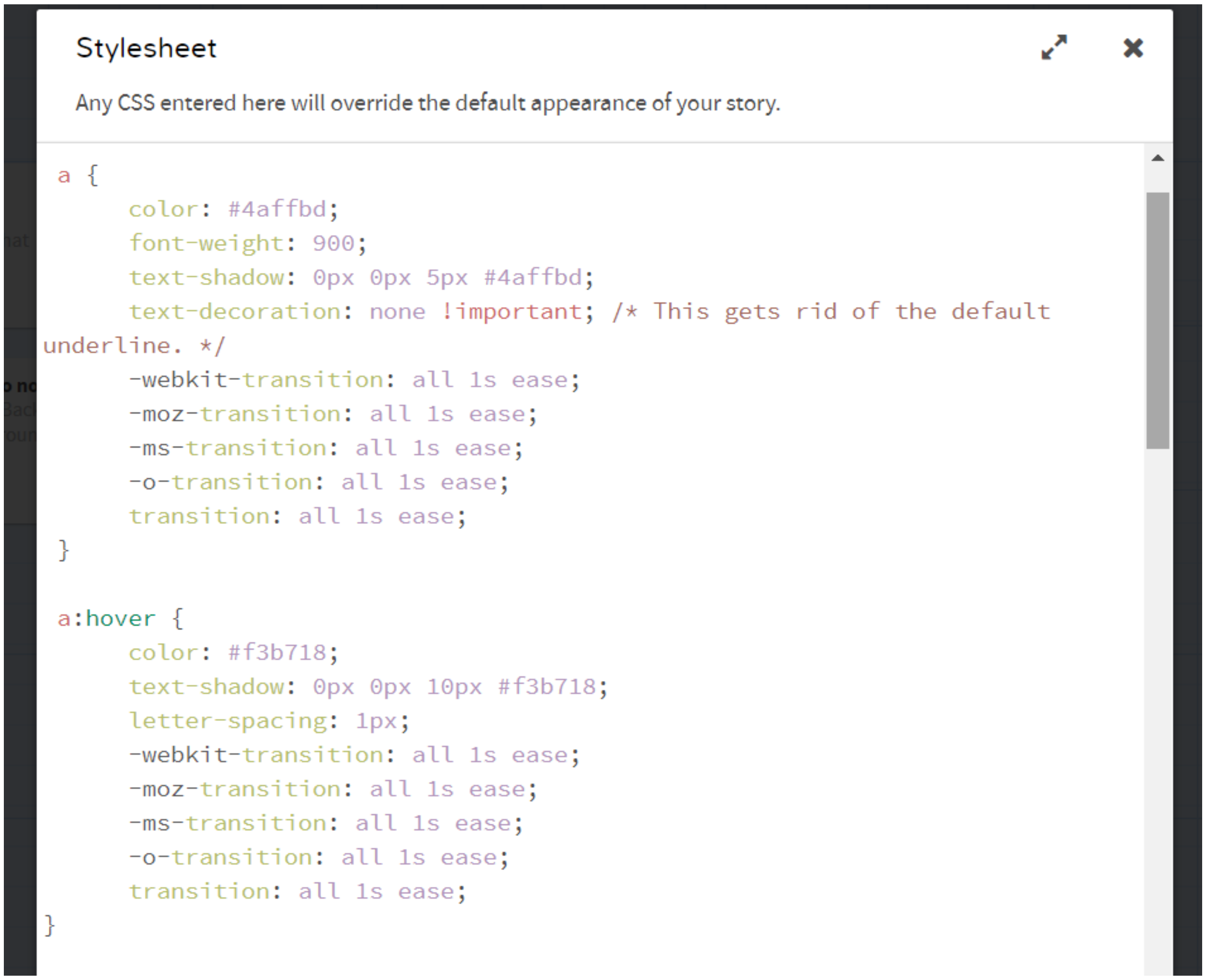
Here is some sample CSS for each format, along with how that CSS will look when testing it in Twine.

SugarCube 2.x*:



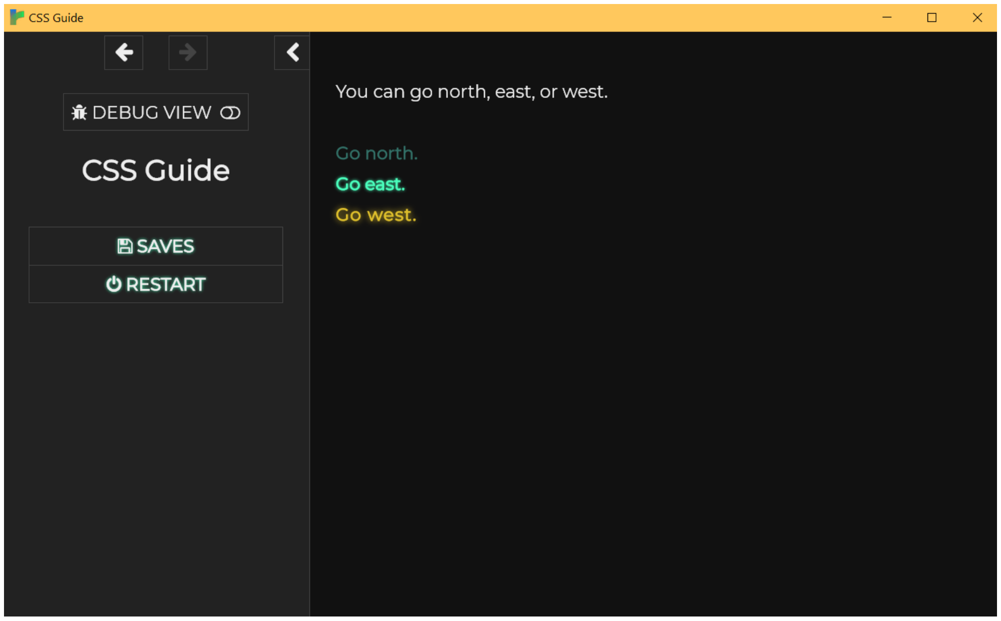
Any CSS entered here will override the default appearance of your story.

```css
a.link-visited {
     color: #326f66;
     text-shadow: none;
     -webkit-transition: all 1s ease;
     -moz-transition: all 1s ease;
     -ms-transition: all 1s ease;
     -o-transition: all 1s ease;
     transition: all 1s ease;
}

a.link-visited:hover {
     color: #45988b;
     text-shadow: none;
     letter-spacing: 1px;
     -webkit-transition: all 1s ease;
     -moz-transition: all 1s ease;
     -ms-transition: all 1s ease;
     -o-transition: all 1s ease;
     transition: all 1s ease;
}
```



Any CSS entered here will override the default appearance of your story.

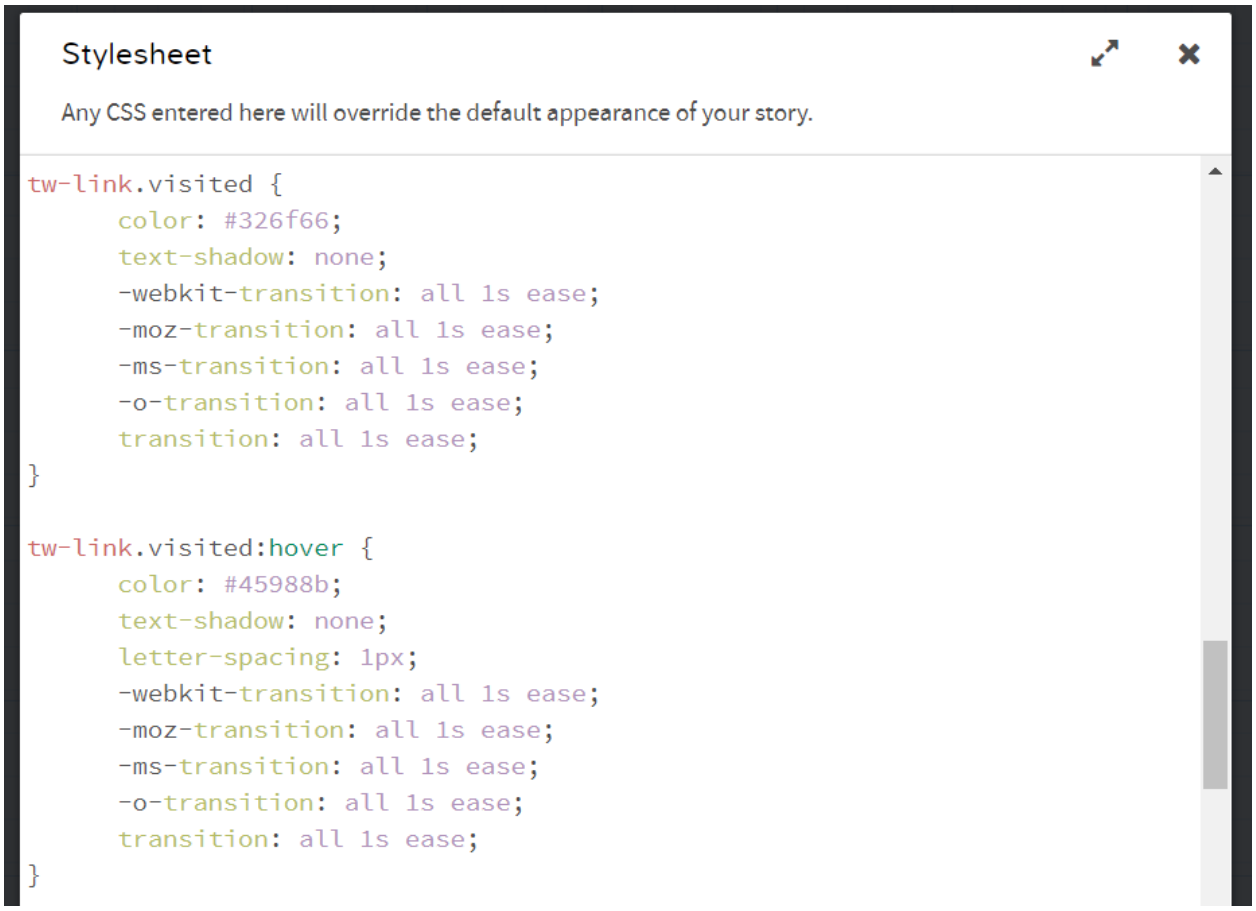
```
a {
      color: #4affbd;
      font-weight: 900;
      text-shadow: 0px 0px 5px #4affbd;
      text-decoration: none !important; /* This gets rid of the default
underline. */
      -webkit-transition: all 1s ease;
      -moz-transition: all 1s ease;
      -ms-transition: all 1s ease;
      -o-transition: all 1s ease;
      transition: all 1s ease;
 }

 a:hover {
      color: #f3b718;
      text-shadow: 0px 0px 10px #f3b718;
      letter-spacing: 1px;
      -webkit-transition: all 1s ease;
      -moz-transition: all 1s ease;
      -ms-transition: all 1s ease;
      -o-transition: all 1s ease;
      transition: all 1s ease;
}
```



You can go north, east, or west.

Go north.
Go east.
Go west.

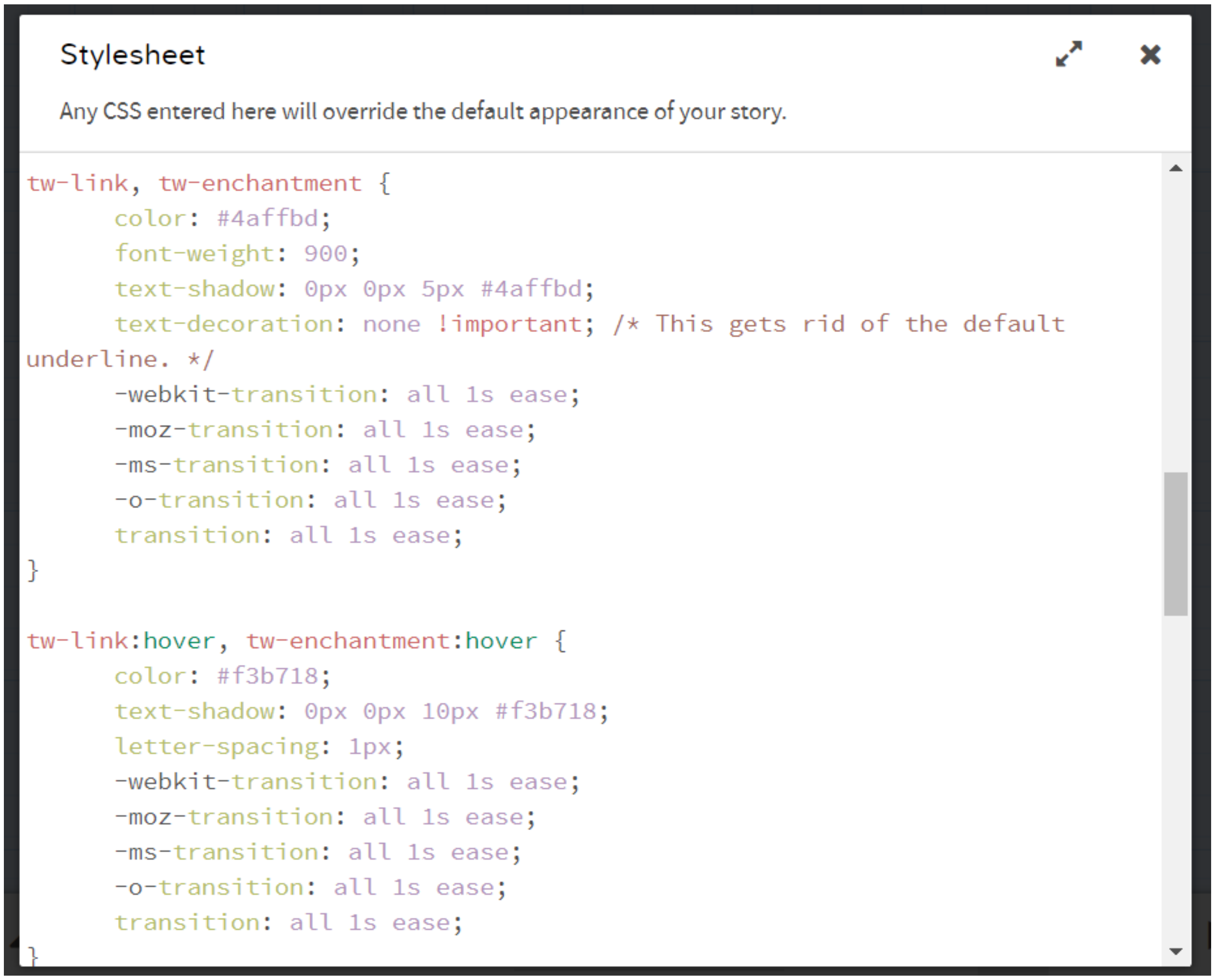
Harlowe:



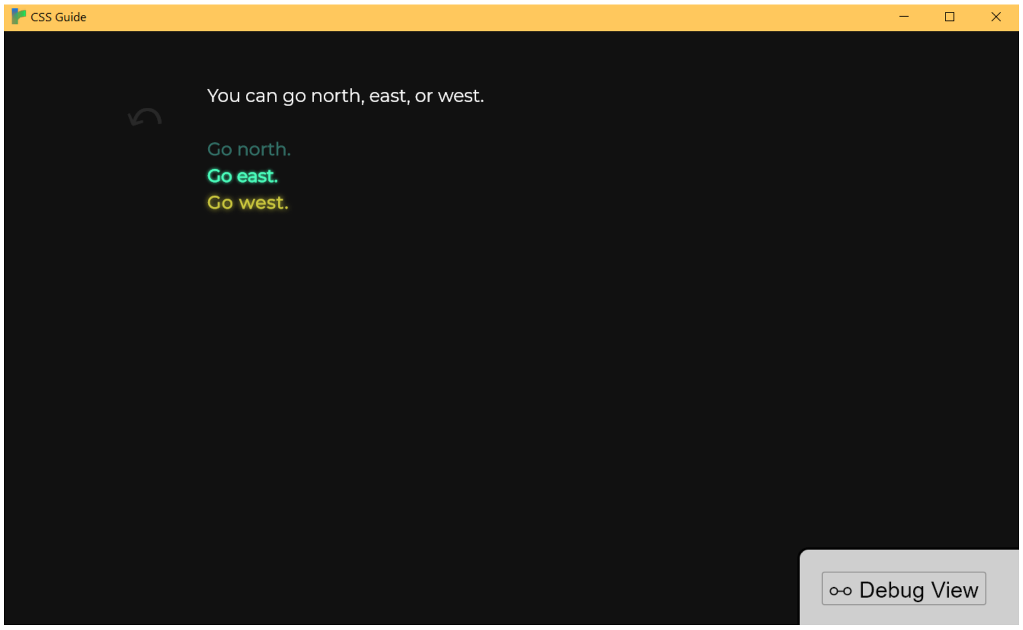
Stylesheet

Any CSS entered here will override the default appearance of your story.

```
tw-link.visited {
      color: #326f66;
      text-shadow: none;
      -webkit-transition: all 1s ease;
      -moz-transition: all 1s ease;
      -ms-transition: all 1s ease;
      -o-transition: all 1s ease;
      transition: all 1s ease;
}

tw-link.visited:hover {
      color: #45988b;
      text-shadow: none;
      letter-spacing: 1px;
      -webkit-transition: all 1s ease;
      -moz-transition: all 1s ease;
      -ms-transition: all 1s ease;
      -o-transition: all 1s ease;
      transition: all 1s ease;
}
```


Stylesheet
Any CSS entered here will override the default appearance of your story.

```
tw-link, tw-enchantment {
	color: #4affbd;
	font-weight: 900;
	text-shadow: 0px 0px 5px #4affbd;
	text-decoration: none !important; /* This gets rid of the default
underline. */
	-webkit-transition: all 1s ease;
	-moz-transition: all 1s ease;
	-ms-transition: all 1s ease;
	-o-transition: all 1s ease;
	transition: all 1s ease;
}

tw-link:hover, tw-enchantment:hover {
	color: #f3b718;
	text-shadow: 0px 0px 10px #f3b718;
	letter-spacing: 1px;
	-webkit-transition: all 1s ease;
	-moz-transition: all 1s ease;
	-ms-transition: all 1s ease;
	-o-transition: all 1s ease;
	transition: all 1s ease;
}
```



You can go north, east, or west.

Go north.
Go east.
Go west.

! *If at some point you decide to switch from SugarCube 2.x to 1.x, Snowman, or Harlowe AND you had configured visited links previously, be sure to **delete** your "*config.addVisitedLinkClass = true;*" JavaScript, or you'll run into errors.

➥ Exercises

EXERCISE 1:

- Design and implement your **default link state**. Identify which selector you'll need to change according to your story format, then customize it using CSS. Use the "Test" or "Play" button to check how it looks.

EXERCISE 2:

- Design and implement a **:hover pseudoclass** for your links. Include a **transition** between states by applying transition properties to both your default and hover states.

EXERCISE 3: (OPTIONAL)

- Design and implement a **visited link state**; include a :hover pseudoclass if you want one. Double check the **method** you'll need to use to do this according to your story format.

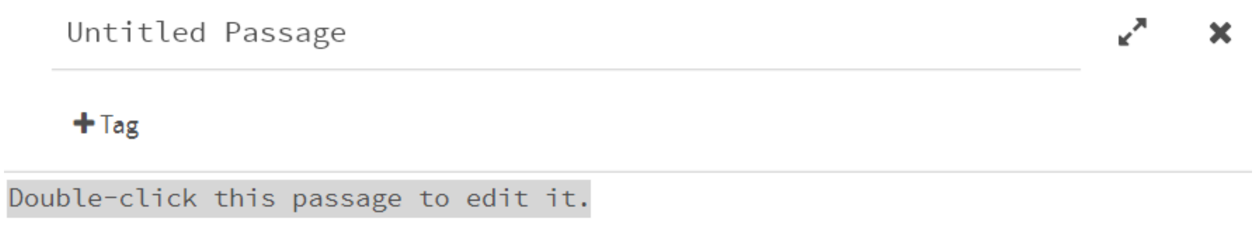
PASSAGE STYLING

In this section:

- **Parts of A Passage**
- **CSS Implementation**
- **Design Tips**
- **Using Tag-Based Styling**
- **Container Elements (CSS + Inline HTML)**
- **CSS Samples & Screenshots**
- **Exercises**

➥ Parts of a Passage

Passages are the basic building blocks of a Twine project, and they have a few key elements of their anatomy: the **title**, the **content**, and the **tags**.



```
Untitled Passage
```

```
Double-click this passage to edit it.
```

In most cases, your **passage titles** won't be seen by players unless they really look under the hood (like by importing your game's index file into Twine, which they can do if your game is downloadable, but which I don't think players do very often; even so, I try not to give my passages especially embarrassing names just in case).

The **content** is whatever you replace the default "*Double-click this passage to edit it.*" message with, and can include raw text as well as HTML, scripts, and macros.

Tags function as labels for your passages that you can use in a few ways; the essential use that will be examined here is **a method for styling elements of individual passages** using these tags.

But first: the relevant **CSS** for styling your passages in general, without complicating things with tags just yet.

➥ CSS Implementation

SugarCube 2.x & 1.x

```
.passage {

   /* Your CSS Here */

}
```

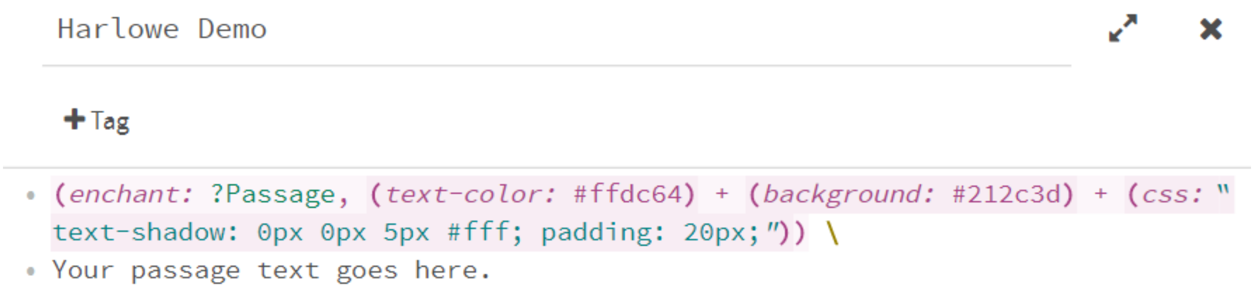
Harlowe

```
tw-passage {

   /* Your CSS Here */

}
```

These are small differences that are easy to confuse (particularly *#passage* vs *.passage*), but the tiny changes are completely necessary.

Macros & Named Hooks

As another demonstration of Harlowe's named hooks, you can change the (text-color:), the (background:) color, and apply some custom (css:) to the text of a single passage by using macros with the **?Passage** named hook like so:



```
(enchant: ?Passage, (text-color: #ffdc64) + (background: #212c3d) + (css: "
text-shadow: 0px 0px 5px #fff; padding: 20px;")) \
Your passage text goes here.
```

Note the usage of the (css:) macro — the **quotation marks** around the CSS itself are essential.

Also note that changes made via hooks and macros will **only apply to the current passage.** To change a property for the entire game, you must override it in the Story Stylesheet.

➥ Design Tips

These are just my miscellaneous, subjective thoughts on designing a passage, if you want some ideas before you start customizing.

Padding

I recommend using a little **padding** around your passage if you're going to be doing things like changing the background color, adding a border, etc. You can adjust the padding of all 4 sides of the passage element (properties: *padding-top, padding-bottom, padding-left, padding-right*), but I'd generally recommend keeping it uniform like so:

```
        padding: 15px;
```

Borders

If you want to add a **border**, check out this page and experiment with a design you like. The main elements of a border are the **style** (*solid/dotted/dashed/double/etc*), the **width** (a numerical value in px), and the **color**. All of these properties can be defined separately, but I usually prefer to address them all within the *border* property.

To add **a border around your entire passage**, try something like:

```
        border: solid 3px #fff;
```

With the color changed to whatever you want.

If you want a border **only on certain sides**, the relevant properties are *border-top, border-bottom, border-left*, and *border-right*. Again, you will probably want to keep this uniform, but experimenting can create some cool effects.

Border Radius

Maybe those harsh corners of your passage are bothering you and you'd like **a softer, rounder look**. For that, there's the ***border-radius*** property, which you'll input as a value in px.

To gently round off all of your corners:

```
            border-radius: 5px;
```

To completely sand down everything, use higher values:

```
            border-radius: 100px;
```

(However, if you want an element to be perfectly circular, it must have the same width and height.)

You can experiment with the intensity, and you might also want to experiment by changing the values of different corners, like so:

```
            border-radius: 50px 0px 50px 0px;
```

There's more detailed information on border-radius values here.

Backgrounds & Transparency

Depending on the background you're using for the main body of your project, it can also be neat to **experiment with transparency**. You can do this by assigning a background color using RGBA rather than a hex code! Here's **half-opaque black**:

```
            background-color: rgba(0,0,0,0.5);
```

Or **totally transparent**:

```
            background-color: transparent;
```

➥ Tag-Based Styling

Now that you're getting an idea of how you want your passages to look, maybe you're realizing that you'd like **certain passages to look totally different from other ones**. This is a perfect use for your **passage tags**.

As an example, let's say that you want passages **tagged "left"** to have left-aligned white text and a black background, and passages **tagged "right"** to have right-aligned black text and a white background.

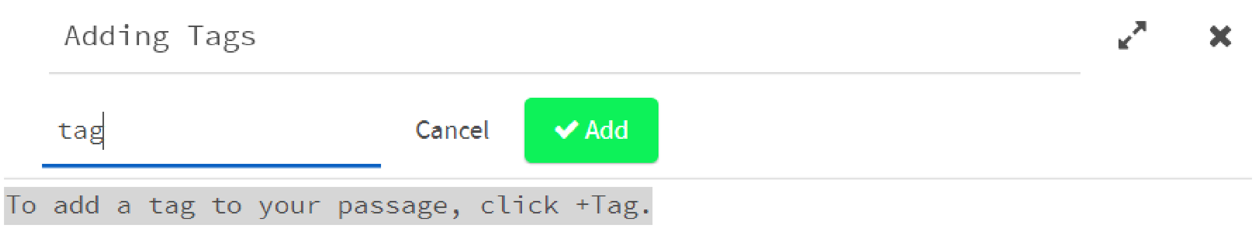
Adding Tags

To start, decide which passages will be "left" and which will be "right."

From one of your existing passages, **create a link** to each of these so it'll be easy to test.

Then:

1. Double click your passage
2. Click **"+Tag"**
3. Type the name of your tag exactly as you plan to use it
4. Be sure to hit **Enter** or click the **Add** button



To add a tag to your passage, click +Tag.

You've now assigned your passage tags. You'll need to do this for every passage that requires certain tags.

➥ Styling Your Tags

To use these tags, we'll have to work them into the stylesheet. The CSS will be different depending on your format.

SugarCube 2.x & 1.x

```
body.tagname .passage {

   /* Your CSS Here */

}

body.othertagname .passage {

   /* Your CSS Here */

}
```

Replace "**tagname**" and "**othertagname**" with the tags you're using, which in this case will probably be "left" and "right."

Harlowe 1.x

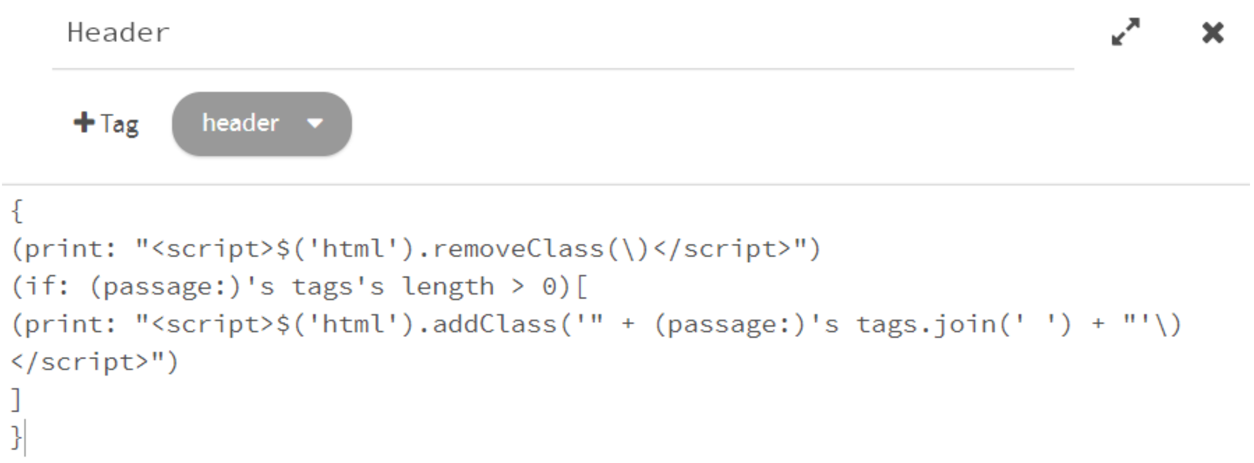
Ignore this and skip to the next subsection if you are using a later version of Harlowe.

To make tag-based styling work in **Harlowe 1.x**, there are **a few simple steps you need to do first** before you implement CSS:

1. Create a new passage called "Header" (the name doesn't matter, but it's easier to keep track of your passages this way)
2. Add "**header**" as a tag to this passage
3. In the passage body, copy and paste the following:

```
          {

          (print: "<script>$('html').removeClass(\)</script>")

          (if: (passage:)'s tags's length > 0)[

          (print: "<script>$('html').addClass('" + (passage:)'s
          tags.join(' ') + "'\)</script>")

          ]

          }
```

Here’s how that will look:



```
{
(print: "<script>$('html').removeClass(\)</script>")
(if: (passage:)'s tags's length > 0)[
(print: "<script>$('html').addClass('" + (passage:)'s tags.join(' ') + "'\)
</script>")
]
}
```

Now you’ll be able to **implement your CSS** like so:

```
          html.tagname tw-passage {

             /* Your CSS Here */

          }
```

```
html.othertagname tw-passage {

   /* Your CSS Here */

}
```

Don’t forget to **replace** “**tagname**” and “**othertagname**” with your own tags.

Harlowe 2.x & 3.x

Harlowe 2.x and later versions have a **tags** attribute added to the **tw-story** element, meaning that the way tag-based CSS styling is handled in Harlowe 1.x does not work for these versions, because the **tags** attribute needs to be accounted for in a different fashion.

Here is the correct way to style tags in the **current version** of Harlowe:

```
tw-story[tags~="tagname"] {

   /* Your CSS Here */

}

tw-story[tags~="othertagname"] {

   /* Your CSS Here */

}
```

As before, **replace** “**tagname**” and “**othertagname**” with your own tags.

➥ Creating a Container Element

Another possibility: you want your passages to mostly look the same, but you want your links (or titles, or certain lines of text, or etc) to appear OUTSIDE of the passage block.

Because everything you input into Twine is technically located inside a passage, using the basic passage styling I've outlined above will still leave you stuck with all your text inside that element.

What you can do instead is **create a new div class** in the stylesheet that you'll apply to your text using in-line HTML. (This is similar to how you created the emphasis span class in the first tutorial.)

CSS

Open up the stylesheet and input the following:

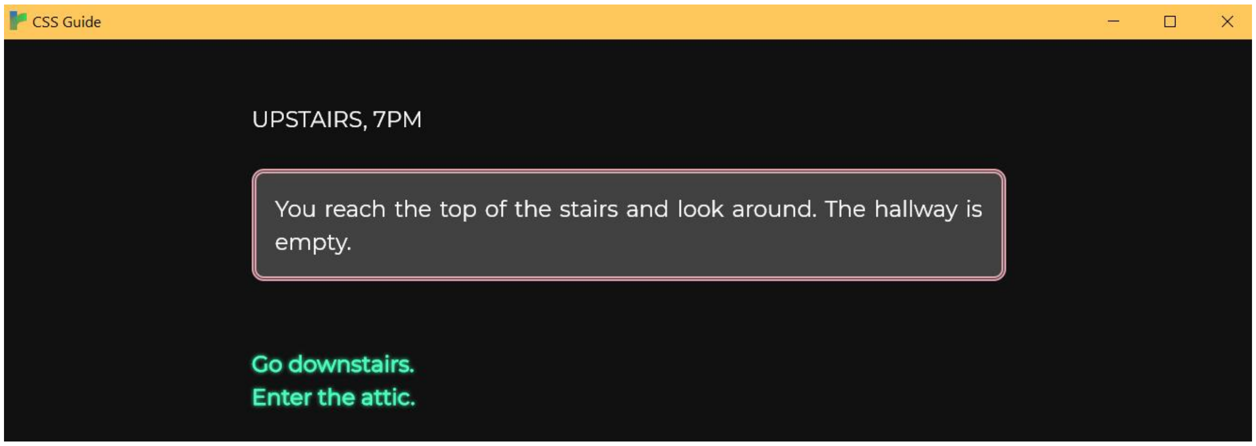
```
.psg {

   /* Your CSS Here */

}
```

Define the font, padding, border, background, and other elements you want your passages to have between the braces.

HTML

Now return to your passage and wrap your "passage" text in the following HTML tags. Try something like this:

```
Your text here.

<div class=”psg”>Your text here.</div>

[[Your link here.]]
```

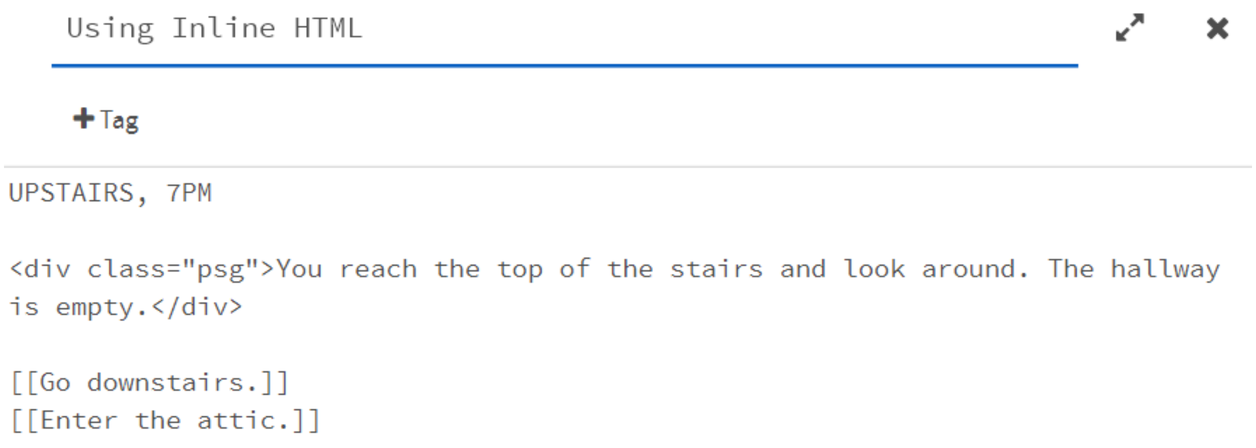


UPSTAIRS, 7PM

You reach the top of the stairs and look around. The hallway is empty.

Go downstairs.
Enter the attic.

(Above: Harlowe 2.x.)

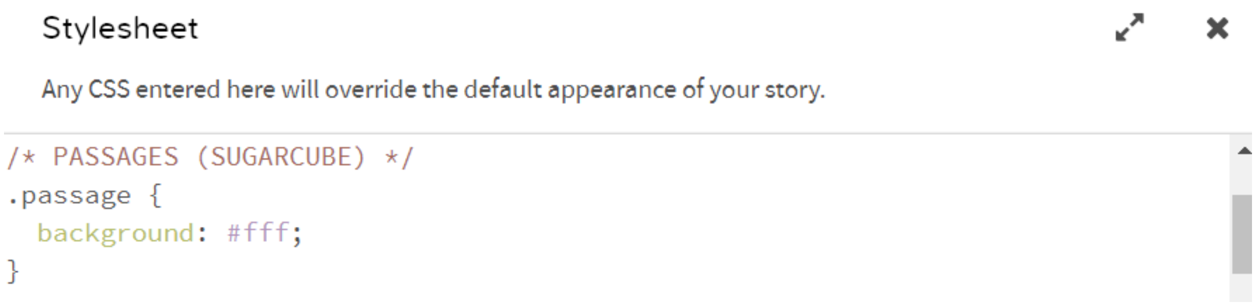


```
UPSTAIRS, 7PM

<div class="psg">You reach the top of the stairs and look around. The hallway
is empty.</div>

[[Go downstairs.]]
[[Enter the attic.]]
```

The pros to this method are that you can get some very granular customization going on from passage to passage, and that you can separate certain elements from your main text if you want to do so in a visually interesting way.

The downside is that you'll have to apply this HTML manually to every passage in your project that requires it. (This is why I usually call the class "*.psg*" versus anything longer.) But although it can be tedious, it's not a horrible undertaking, especially for a game with under 50-100 passages!

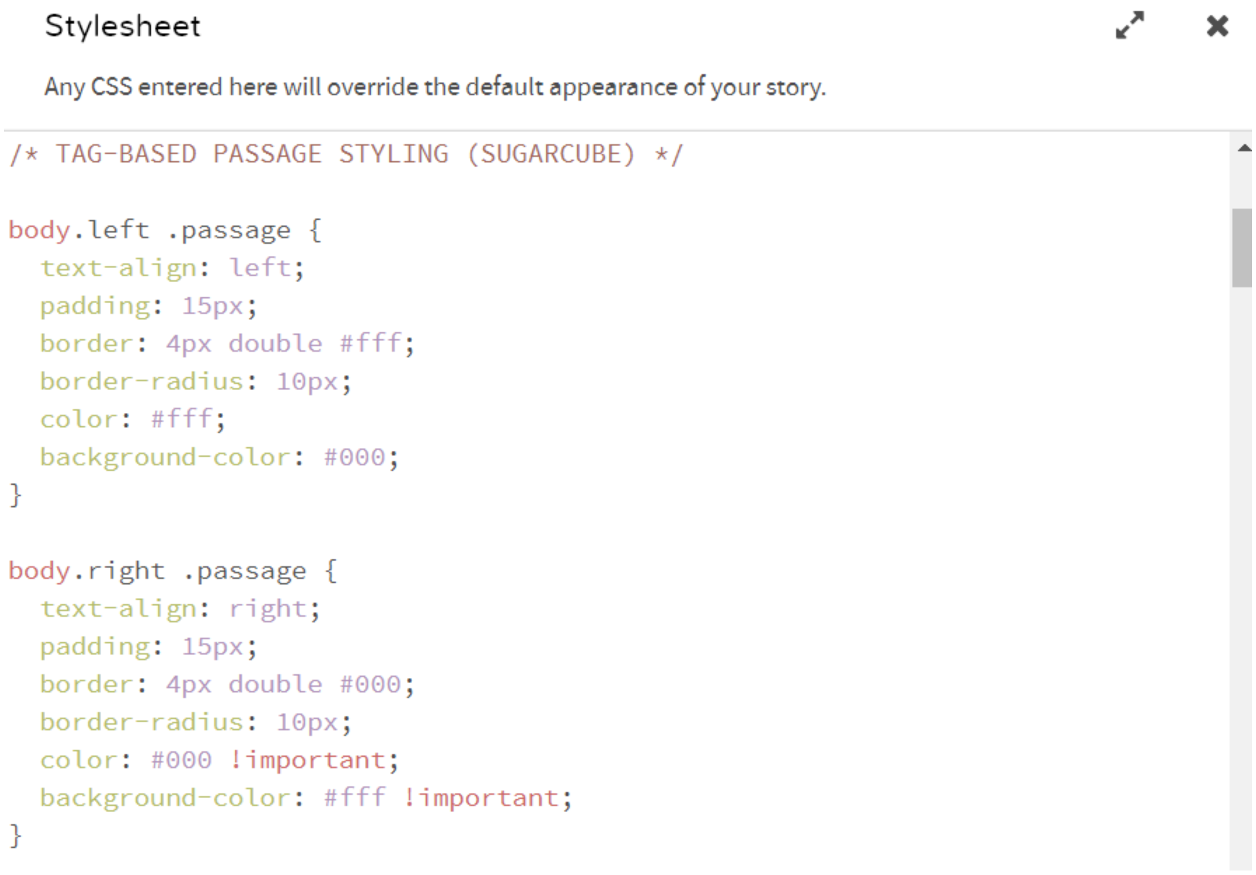
➥ CSS Samples & Screenshots



Stylesheet

Any CSS entered here will override the default appearance of your story.

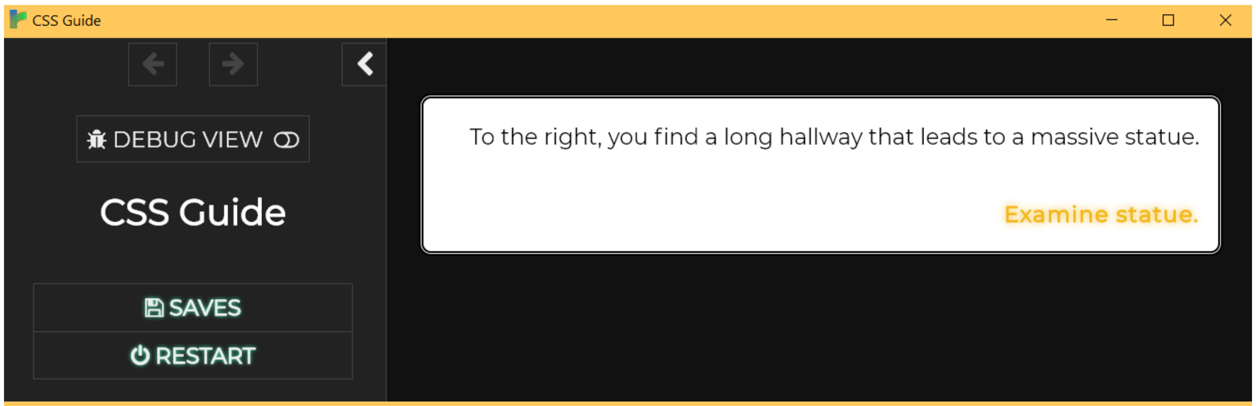
```
/* PASSAGES (SUGARCUBE) */
.passage {
  background: #fff;
}
```



Stylesheet

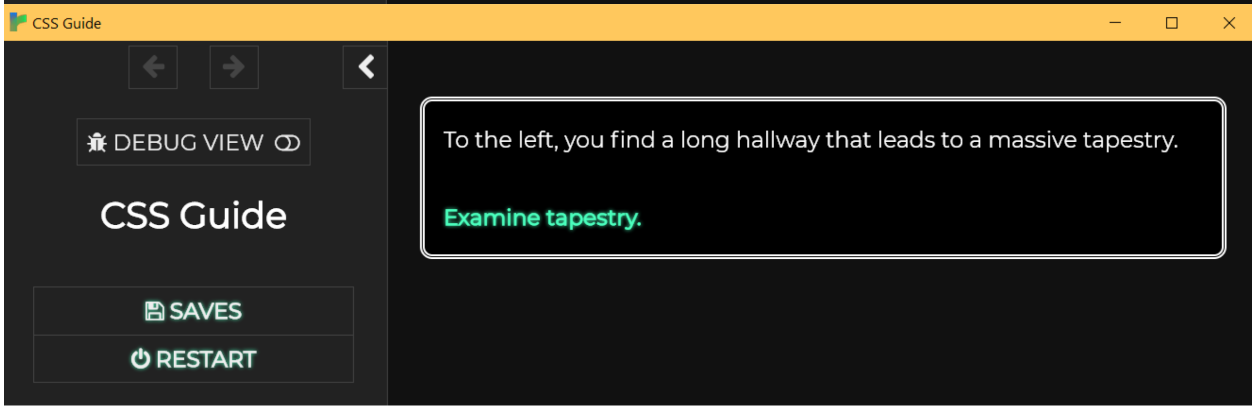
Any CSS entered here will override the default appearance of your story.

```
/* TAG-BASED PASSAGE STYLING (SUGARCUBE) */

body.left .passage {
  text-align: left;
  padding: 15px;
  border: 4px double #fff;
  border-radius: 10px;
  color: #fff;
  background-color: #000;
}

body.right .passage {
  text-align: right;
  padding: 15px;
  border: 4px double #000;
  border-radius: 10px;
  color: #000 !important;
  background-color: #fff !important;
}
```



To the right, you find a long hallway that leads to a massive statue.

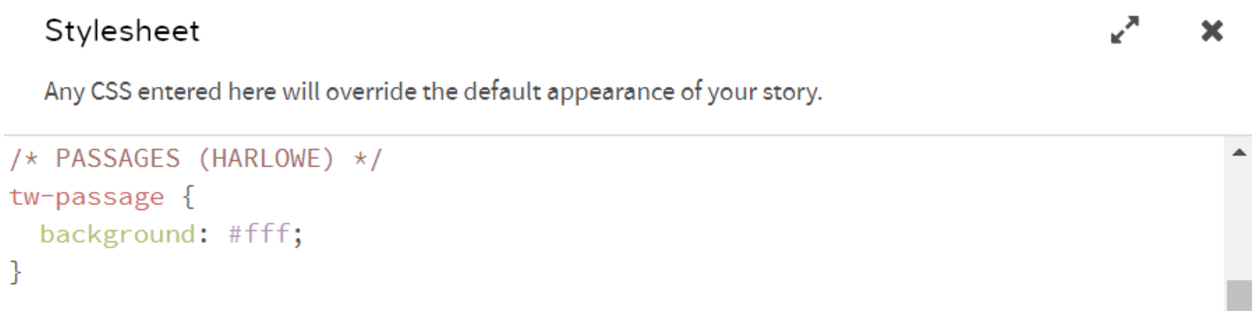
Examine statue.



To the left, you find a long hallway that leads to a massive tapestry.

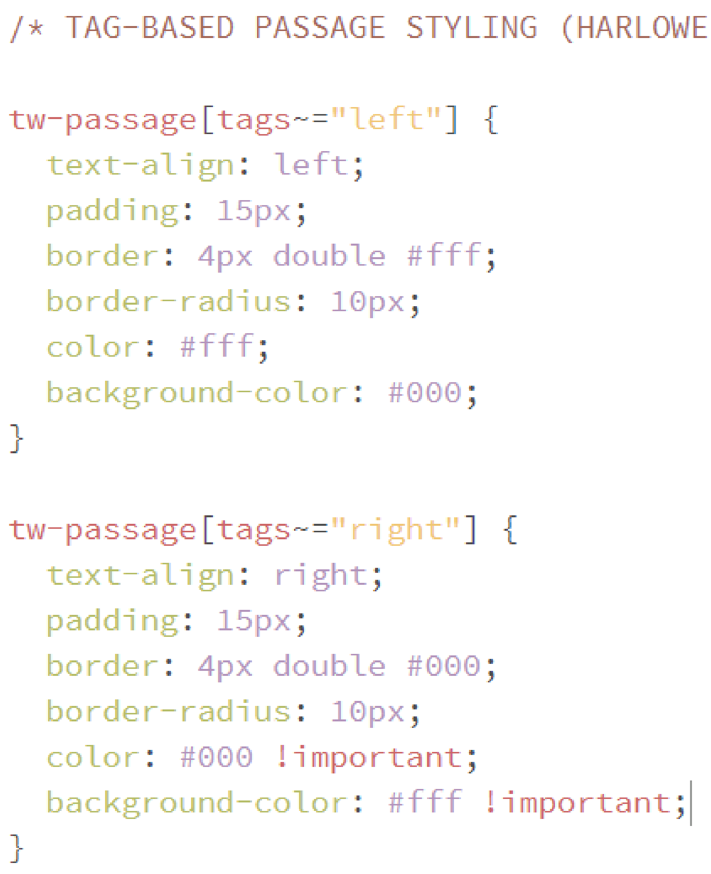
Examine tapestry.

(Above: SugarCube 2.x.)

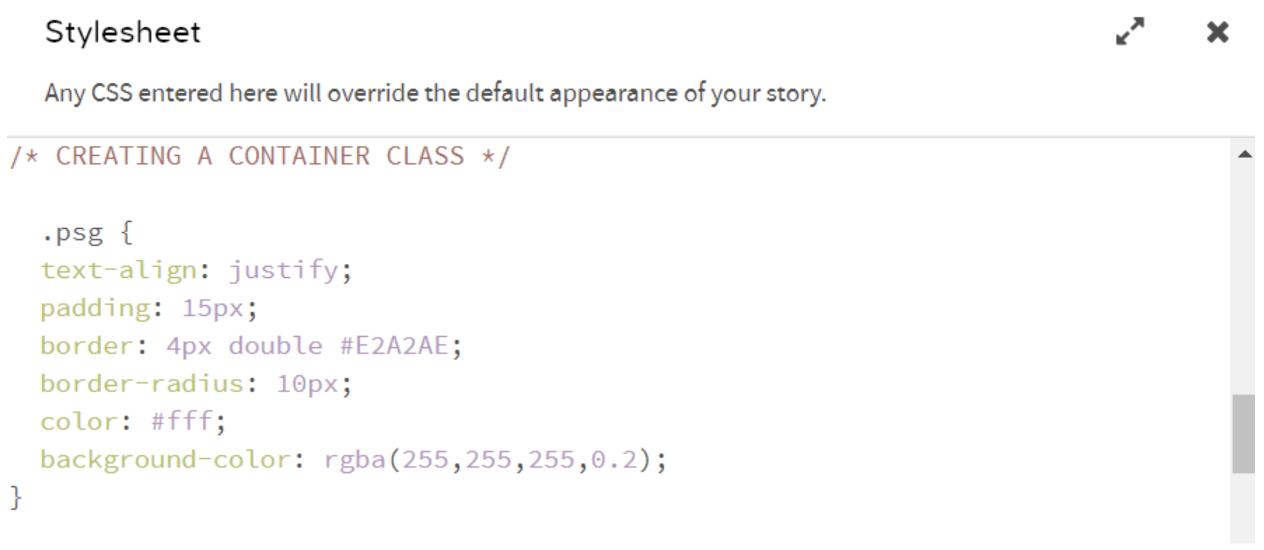

Stylesheet

Any CSS entered here will override the default appearance of your story.

```
/* PASSAGES (HARLOWE) */
tw-passage {
  background: #fff;
}
```



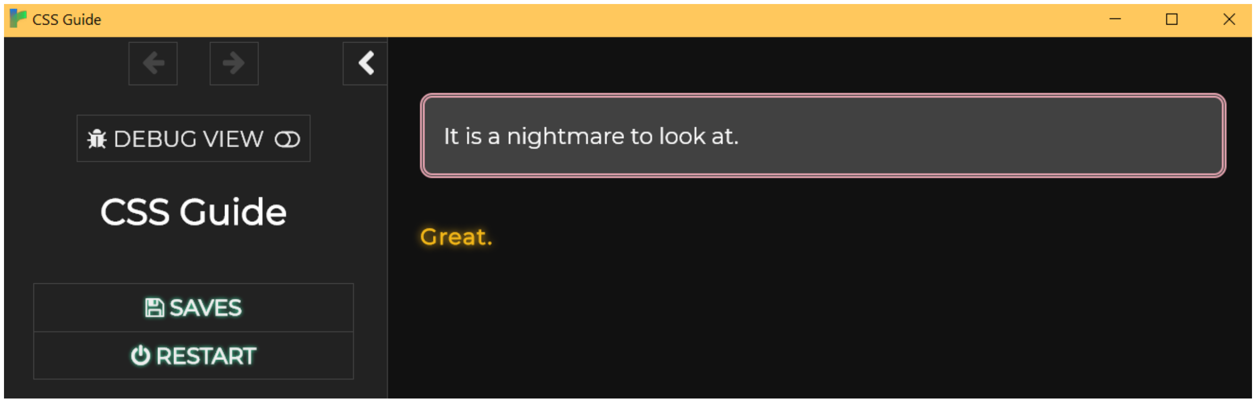
```
/* TAG-BASED PASSAGE STYLING (HARLOWE) */

tw-passage[tags~="left"] {
  text-align: left;
  padding: 15px;
  border: 4px double #fff;
  border-radius: 10px;
  color: #fff;
  background-color: #000;
}

tw-passage[tags~="right"] {
  text-align: right;
  padding: 15px;
  border: 4px double #000;
  border-radius: 10px;
  color: #000 !important;
  background-color: #fff !important;
}
```



Stylesheet

Any CSS entered here will override the default appearance of your story.

```
/* CREATING A CONTAINER CLASS */

  .psg {
  text-align: justify;
  padding: 15px;
  border: 4px double #E2A2AE;
  border-radius: 10px;
  color: #fff;
  background-color: rgba(255,255,255,0.2);
}
```



It is a nightmare to look at.

Great.

(Above: SugarCube 2.x.)

➥ Exercises

You can choose to do only one depending on how you're designing your project, or you can experiment with multiple if you want to:

EXERCISE 1:

- Design a **default appearance** for all of the **passages** in your game. Try sketching it or creating a mockup in an image editor. Think about properties like padding, borders, and colors. Once you have a design in mind, try **implementing it** in your Story Stylesheet and **testing it**.

EXERCISE 2:

- Design and implement two **different passage designs** using **tag-based styling** and CSS. Each design will require a separate tag and a separate section in your stylesheet.

EXERCISE 3:

- Design and implement a **custom passage container** using **CSS *and* inline HTML**. You will need to define it in your stylesheet and implement it by wrapping the desired text in the correct HTML tags.

BACKGROUNDS

In this section:

(You'll be seeing the CSS implementation throughout these sections.)

➥ Background Colors

Just to refresh, here is how to change the default background for your entire project:

SugarCube

```
body {

    background-color: #******;

}
```

Harlowe

```
tw-story {

    background-color: #******;

}
```

➥ More Tag-Based Styling & CSS Implementation

You now know how to customize passage backgrounds as well as how to change the main body background of a project. The latter, though, applies to your entire project, and you may want **different backgrounds** for different sections of your game.

To do this, we'll be using **tag-based styling** again.

Let's keep using the "left" and "right" tags from last time along with some new tags. You can use **several tags together** so long as their CSS doesn't conflict.

Just to experiment, we'll create tags for "red" and "blue."

(You can use whatever different tag name and different color hex code or rgb value you want, if you feel comfortable.)

- Apply "red" to your "left"-tagged passage using the tagging method from the previous tutorial.
- Apply "blue" to your "right"-tagged passage.
- Open up the Story Stylesheet.

SugarCube

```
body.red {

   background-color: red;

}

body.blue {

   background-color: blue;

}
```

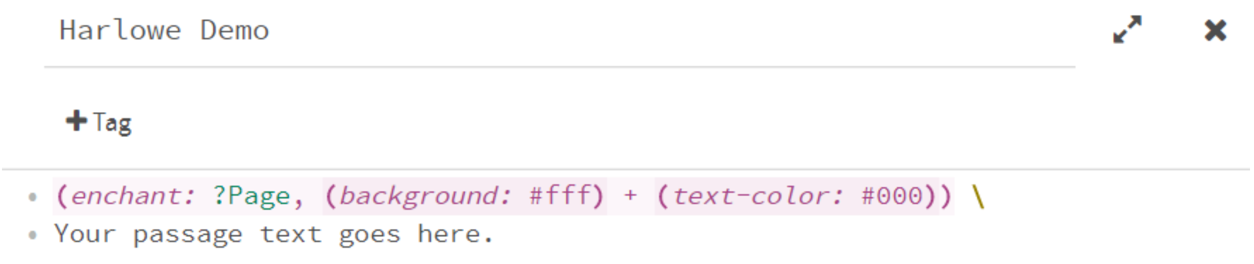
Harlowe

```
tw-story[tags~="red"] {

   background-color: red;

}

tw-story[tags~="blue"] {

   background-color: blue;

}
```

Macros & Named Hooks

If using Harlowe, you can also use the **?Page** named hook.

To change this, attach an (enchant:) macro to the ?Page hook, then attach further macros within the enchant macros like so:



```
(enchant: ?Page, (background: #fff) + (text-color: #000)) \
Your passage text goes here.
```

(The "\" at the end of the code above is **not** part of the macro; it closes the extra line of whitespace you would otherwise see preceding your text without its inclusion. It's super handy when you start using several macros at once, or long ones that make your passage difficult to read in the editor.)

Remember that changes made via hooks and macros will **only apply to the current passage.** To change a property for the entire game, you must override it in the Story Stylesheet.

Now **test** your game!

You should see both tags on each passage functioning simultaneously. The result of this very basic code looks horrible, sure, but it's working, and you can build on it to match your own tastes and ideas.

This also goes to show how different Story Formats behave differently; you may switch from Harlowe 1.x to 3.x or from SugarCube to Harlowe and find that your game looks totally broken, but it really only needs a very small tweak in the stylesheet. You'll learn to feel these adjustments out as you go.

! **If you're using Harlowe 1.x**, you **must** make a **header passage** that includes a certain script; you can find a short walkthrough for this **in the last tutorial (Passage Styling).**

➥ Images

Maybe you don’t want a flat color as your background, and instead you want to use **images.** This is also doable, but there are a few ways to achieve it.

Image Hosting

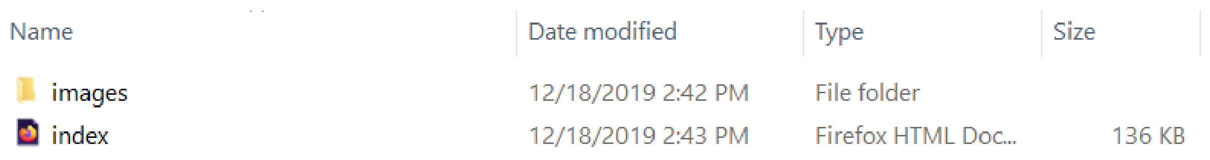
You could certainly host your images somewhere like Imgur and link to them directly, but **I don’t recommend this**.

I did this in my early games, and it’s functional, but only if your game is being played online; **without an internet connection, none of your images will display**. With a slow or interrupted connection, they also could take an excessively long time to load or may not load at all.

Instead, I recommend creating an **archive** for each of your projects if you plan to include any images (or sounds, other media files, etc). You can do this by making a **folder** for each project with subfolders for your assets.

➥ Organizing Your Files

1. Create a New Folder with your project's name
2. Move your index.html file (or whatever you've named it) into this folder; **it must remain in the top level**, not within another subfolder
3. Create a New Folder in this same folder called "images"
4. Download images you want to use for backgrounds into the "images" folder (to start, check out Unsplash or Pexels for free high-res images suitable for BGs)



Name	Date modified	Type	Size
images	12/18/2019 2:42 PM	File folder	
index	12/18/2019 2:43 PM	Firefox HTML Doc...	136 KB

Here's my archive for the interactive CSS Guide that I use to test & compile these tutorials!

You now have a little archive for all of your project's associated media. I recommend **large, high resolution images** for backgrounds; these can be photos, textures, tiled images, etc.

! When you upload your game to a website (like itch.io) later on, you **must have an index.html file in the top level of this folder**. *It cannot be named anything else, and it cannot be located in a subfolder.*

➥ Applying a Background Image

SugarCube

```
body.yourtagnamehere {

   background-image: [img[images/yourfilenamehere.png]];

   background-size: cover;

   background-repeat: no-repeat;

   background-attachment: fixed;

   background-size: 100% 100% !important;

}
```

Harlowe

```
tw-story[tags~="cover"] {

   background-image: url(images/cover.jpg);

   background-size: cover;

   background-repeat: no-repeat;

   background-attachment: fixed;

   background-size: 100% 100% !important;

}
```

Above are some useful image properties I use to ensure that my background image will neatly **cover the screen**.

You’ll want to use a high-res image for this rather than stretching a very small image by using “background-size: cover;” — keep that in mind as you’re looking for images.

You’ll also notice that **Harlowe and SugarCube locate images differently** — you need to use “**url(yourfolder/yourfile.png)**” for Harlowe and you can use “**[img[yourfolder/yourfile.png]]**” for SugarCube.

! In order to check that your backgrounds are displaying correctly, you need to run your index.html file in your browser! You will not be able to see the images if you test your project natively in Twine!

➥ Applying a Tiled Background Image

SugarCube

```
body.pattern {

   background-image: [img[images/pattern.png]];

   background-repeat: repeat;

   background-attachment: fixed;

}
```

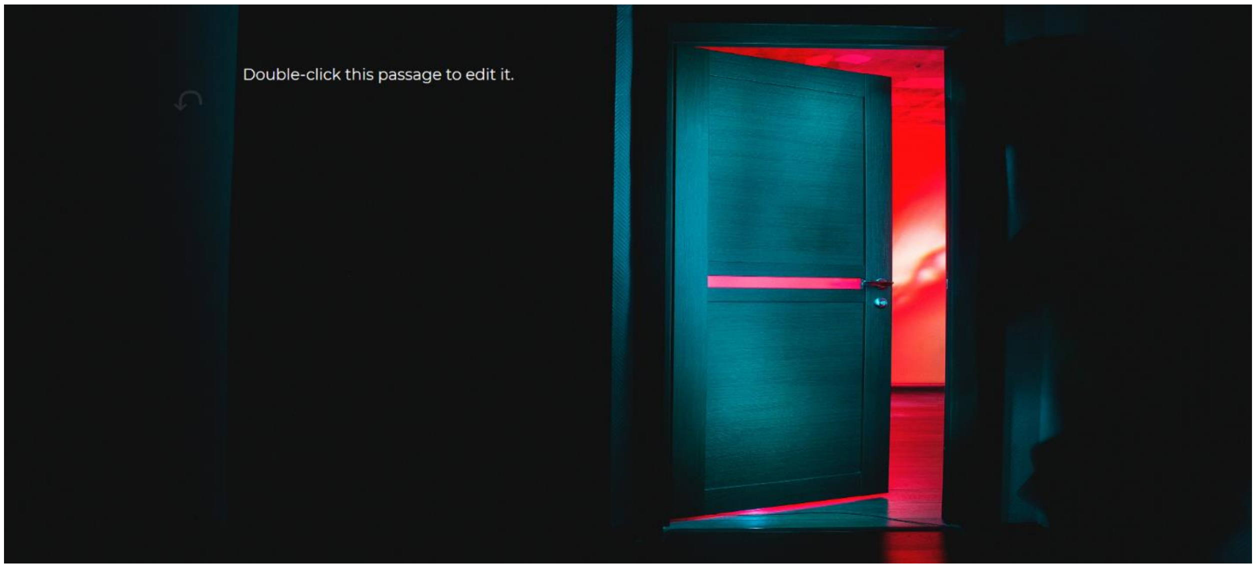
Harlowe

```
tw-story[tags~="pattern"] {

   background-image: url(images/pattern.png);

   background-repeat: repeat;

   background-attachment: fixed;

}
```

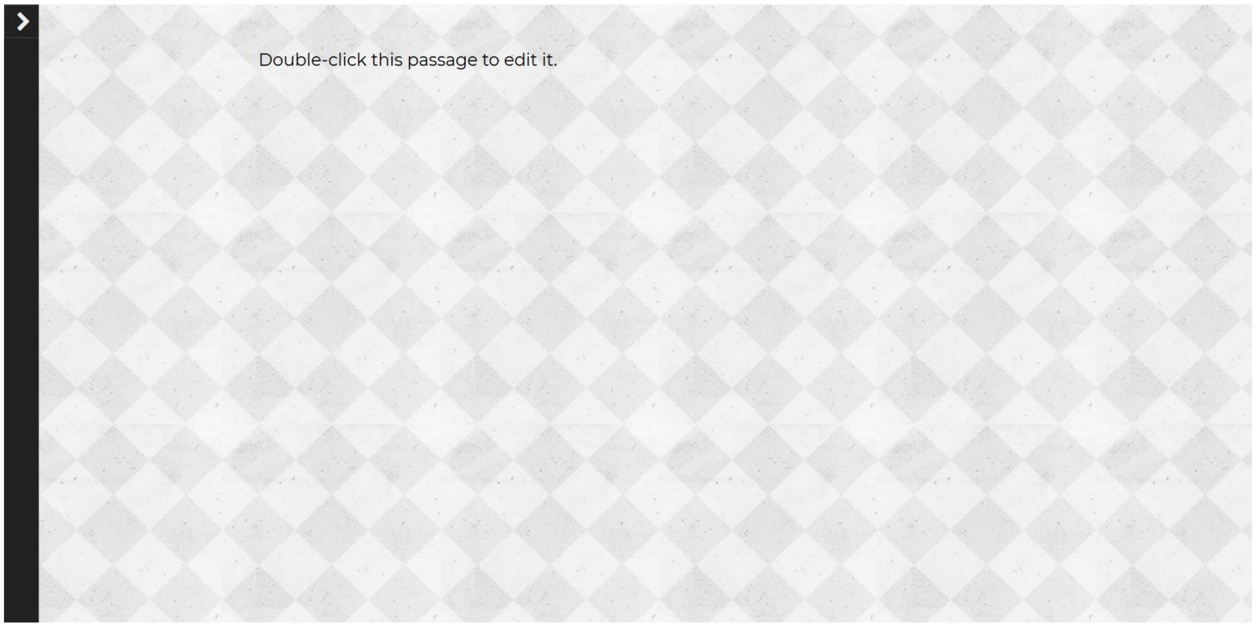
Note: I've been using "cover" and "pattern" as sample classes here, but you can call your classes and filenames whatever you want as long as you keep track of them. I do recommend using **short filenames** for your images, however; you may have to rename the images you download.

➥ Screenshots

Here’s an example of an image that **covers the entire page background** in **Harlowe** using the CSS above:

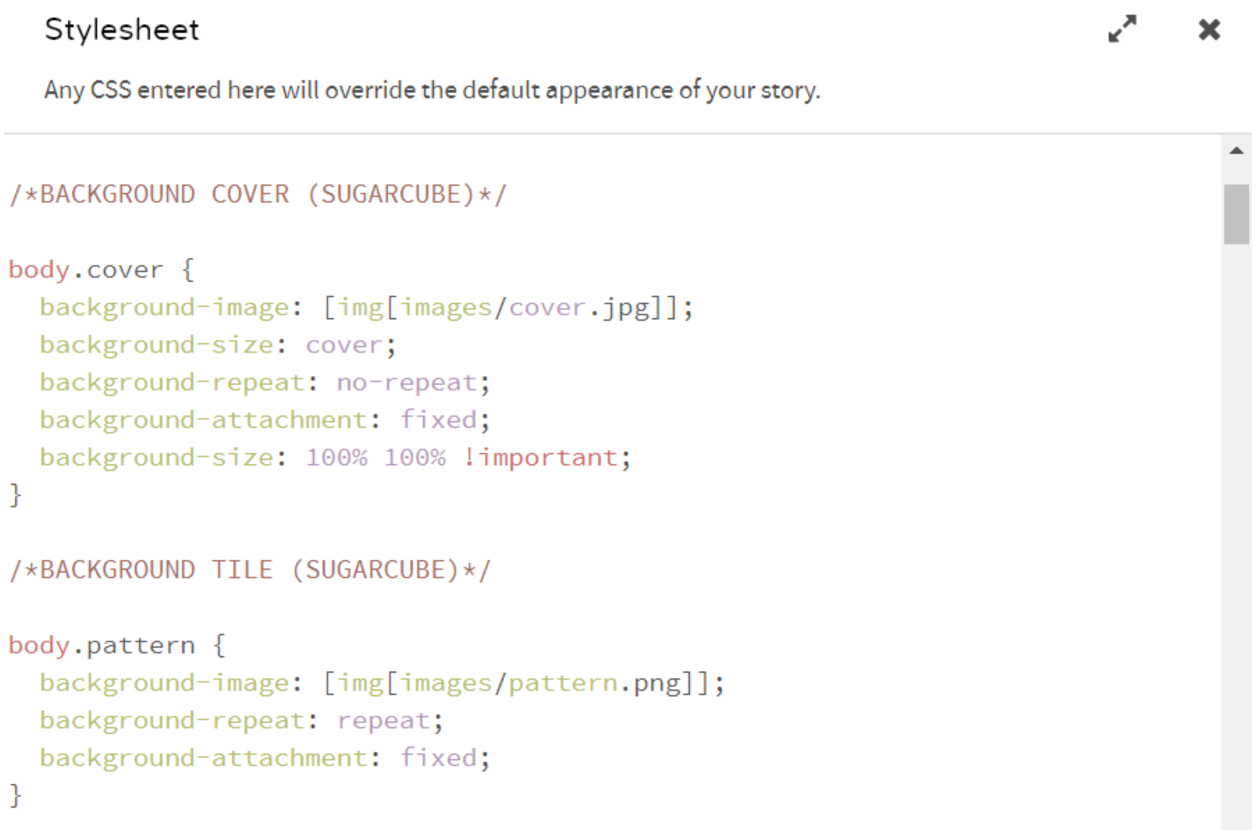

Double-click this passage to edit it.

And here’s an example of **a tiled background** in **SugarCube 2.x**:



(Note that this shows the sidebar while collapsed rather than in its default expanded state.)

Here's a quick look at the stylesheet as a reference for how some of the code above will look in Twine:



Any CSS entered here will override the default appearance of your story.

```
/*BACKGROUND COVER (SUGARCUBE)*/

body.cover {
  background-image: [img[images/cover.jpg]];
  background-size: cover;
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: 100% 100% !important;
}

/*BACKGROUND TILE (SUGARCUBE)*/

body.pattern {
  background-image: [img[images/pattern.png]];
  background-repeat: repeat;
  background-attachment: fixed;
}
```

➥ Exercises

EXERCISE 1:

- Apply a **background color** to **one** of your passages using **tag-based styling**.

EXERCISE 2:

- If you plan to use images in your project, create an **archive folder**; do not rely on online image-hosting. Create a subfolder for images within this folder.

EXERCISE 3:

- Apply a **background image** to **one** of your passages using **tag-based styling**. This can be a tiled image or an image that covers the entire screen; use and adjust the correct CSS accordingly.

EMBEDDING IMAGES

In this section:

You might also be interested in **embedding images within your passages**. Again, I strongly recommend keeping all your media accessible in a single folder for this **instead of** linking to pictures hosted online.

For embedded images, which you may want to use for things like pictures of items, character portraits, etc, I recommend using images with transparent backgrounds (**.png**s with transparency enabled), especially if your project's background colors will be changing in different passages.

Approaches to embedding images in Twine differ based on the Story Format you're using.

➥ How to Embed

Harlowe

Harlowe uses straightforward HTML:

```
<img src="yourfolder/yourfile.png">
```

SugarCube

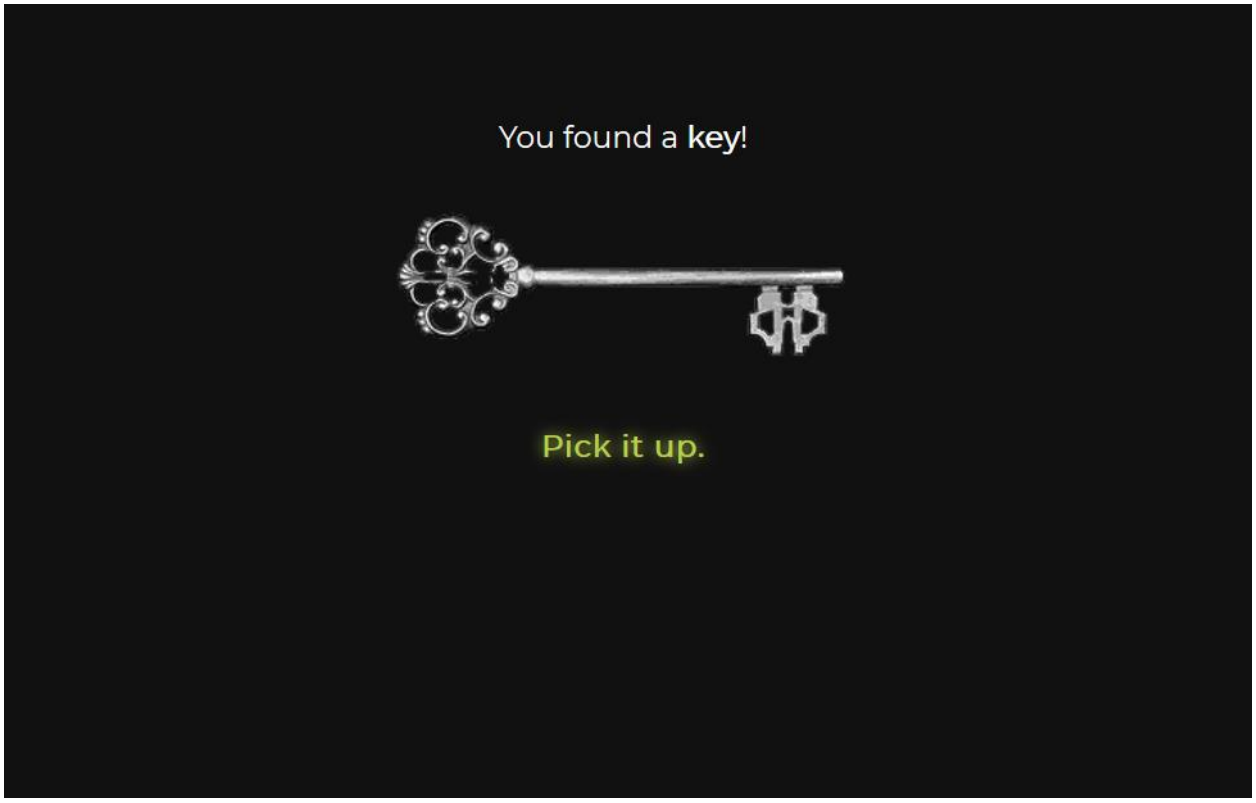
SugarCube can use the above **or** the following:

```
[img[yourfolder/yourfile.png]]
```

(It looks strange at first, and it can be easy to miss a bracket, but I tend to prefer using this method in SugarCube because I almost always have other HTML tags to keep track of.)

➥ Sample Screenshot

Here’s an example of an image **embedded** in a passage in **Harlowe**:


You found a **key**!

Pick it up.

➥ Exercises

EXERCISE 1:

- Find **images** you would like to use alongside your text. Look for images that are **transparent** and the right **size** for your project. Ensure that any images you plan to use are **saved locally** to your project folder.

EXERCISE 2:

- **Embed an image** in one of your passages. If you want, try experimenting with HTML by <center>centering</center> the image.

CONCLUSION

In this section:

- **Future “Grimoire” Volumes**
- **Wrapping Up**

➥ Future Volumes

This concludes the first installment of *The Twine® Grimoire*! The plan is to release at least one more “volume” of the Grimoire, which will cover a similar number of tutorials touching on features like:

- Creating custom buttons
- Fully customizing built-in UI
- “Special” passages and tags
- Experimenting with the new Story Format, Chapbook
- … and more.

Currently, a release date has not been set.

My supporters on Patreon receive early access to the first drafts of these tutorials as they’re written month-by-month, but my tutorials will **always** be released for free after I’ve had the chance to thoroughly revise and refine them. I want these resources to be available in their most polished form to as many users as possible.

➥ Wrapping Up

Twine® is an incredibly versatile tool for creating games of all kinds. It can support a wide variety of mechanics, and has an immensely diverse userbase of all demographics, skill sets, and skill levels. There are so many stories you can tell using Twine, and so many ways to tell them.

Graphics certainly aren't everything in a text-based game, and you may be more interested in the minimalism of Twine's default settings. Even so, it can be both helpful and engaging for new users to learn about which elements they can control and how. In learning to make these changes, you also learn the fundamental pieces of your Twine project and how they fit together.

The visual presentation of your project can have an effect on its atmosphere and mood as well. This guide is intended to help you develop the **basic skillset** needed for beginning to build a complete, custom stylesheet, but there's a lot of leeway left for you to experiment. The samples I've provided are only samples, and you're more than welcome to alter them to suit your needs.

I hope these tutorials have been useful to you, and I hope you enjoy using Twine!

– Grim
G.C. Baccaris