The TWINE® Grimoire

G.C. "GRIM" BACCARIS

Intermediate methods for customizing **Twine** 2 projects with **CSS** and **HTML**.

Volume Two

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:

- About The *Grimoire*
- About Twine
- About the Author
- RECAP: Volume One
- Twine Resources
- CSS Basics
- CSS Resources

➥ About the *Grimoire*

Hello (or hello again) and thanks for cracking open ***The Twine® Grimoire***! This arcane tome contains another series of tutorials created to help Twine® users learn how to create custom stylesheets for their projects.

The Twine® Grimoire is an unofficial guide to using **CSS** (Cascading Style Sheets) and **HTML** (Hypertext Markup Language) to customize projects made in Twine 2. The *Grimoire* demonstrates how CSS and HTML can be used in a variety of Twine 2 Story Formats as a method of fine-tuning a game's **appearance** and **behavior**, allowing users to develop new skills and create visually and mechanically unique text-based games.

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** along with each tutorial, which you can use for practice or inspiration.

The information in these tutorials is not always compatible with previous versions of Twine such as Twine 1.4.2, and is not intended to be used with previous versions for this reason. The tutorials in *Volume 2* are suitable for intermediate users, and will be geared toward features that many users may be interested in working into their projects.

However, *Volume 2* will assume that you are already confident with concepts and skills introduced in *Volume 1*, including:

- CSS & HTML
- Story Formats
- Editing the Story Stylesheet
- Customizing Text
- Customizing Links
- Incorporating Images

(More information about what is covered in *Volume 1* will be discussed later in the Introduction.)

If you aren't yet confident with the topics listed above, you can download *The Twine Grimoire: Volume 1* any time for free.

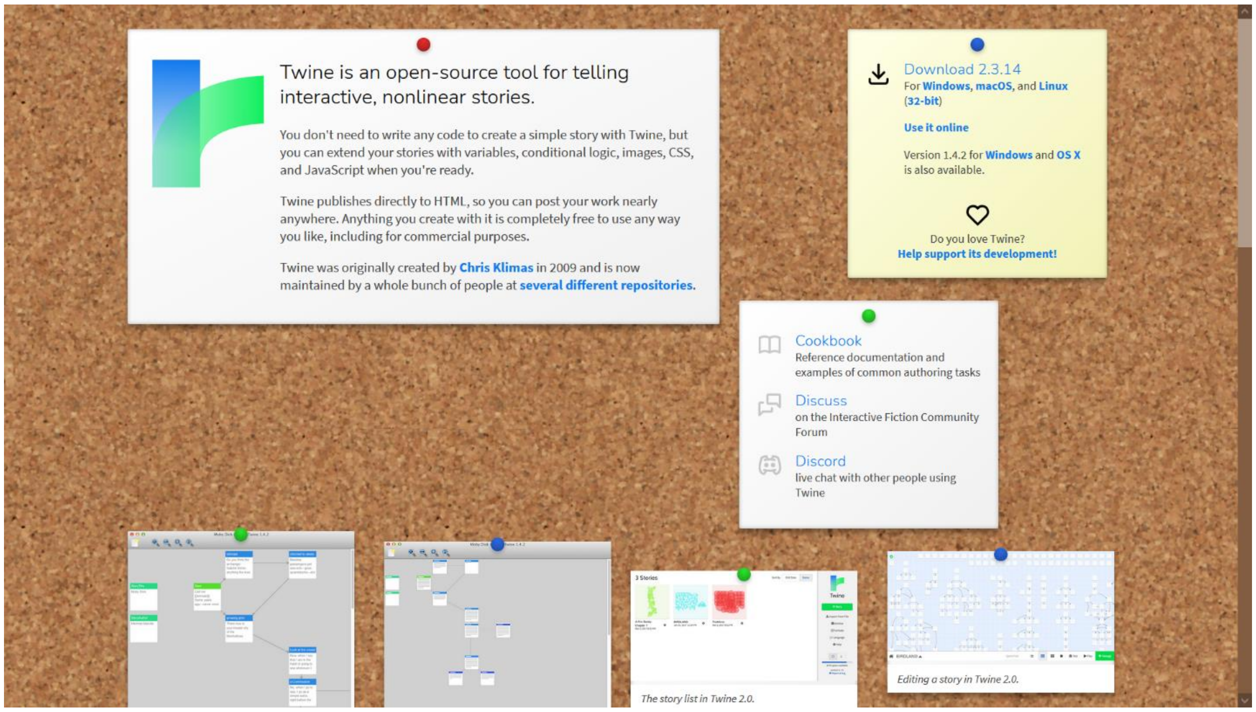
The syntax and methods for the features you'll be creating from the *Grimoire* can **differ between versions and formats of Twine**, so this document contains demonstrations and exercises geared toward multiple formats. We'll be using **Twine 2 exclusively** and working with the most recent **SugarCube** and **Harlowe** formats primarily.

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

➥ About Twine

Twine® is an extremely powerful, versatile, and accessible tool at your disposal as a game developer. It 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. I am not personally affiliated with its development process.

Twine is a registered trademark of the Interactive Fiction Technology Foundation (IFTF), which "helps ensure the ongoing maintenance, improvement, and preservation of the tools and services crucial to the creation and distribution of interactive fiction, as well as the development of new projects to foster the continued growth of this art form."



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 HTML, so you can post your work nearly anywhere. Anything you create with it is completely free to use any way you like, including for commercial purposes.

Twine was originally created by **Chris Klimas** in 2009 and is now maintained by a whole bunch of people at **several different repositories**.

Download 2.3.14
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!

Cookbook
Reference documentation and examples of common authoring tasks

Discuss
on the Interactive Fiction Community Forum

Discord
live chat with other people using Twine

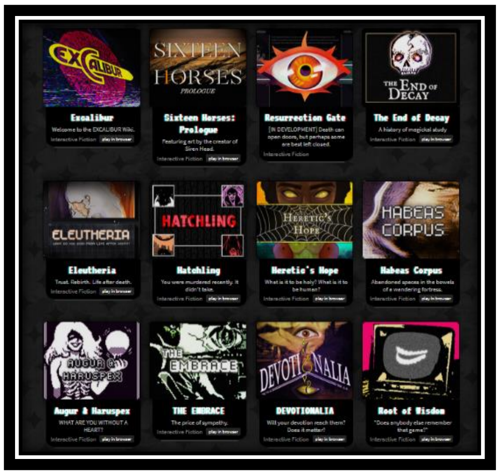
The story list in Twine 2.0.

Editing a story in Twine 2.0.

➥ About the Author

I'm G.C. "Grim" Baccaris, and I'm a writer, game developer, and UI designer.

My work has been featured in *Sub-Q Magazine*, *Indiepocalpyse*, PAX Online, WordPlay, and elsewhere, and I've worked for clients representing National Geographic and Boston University, among others.

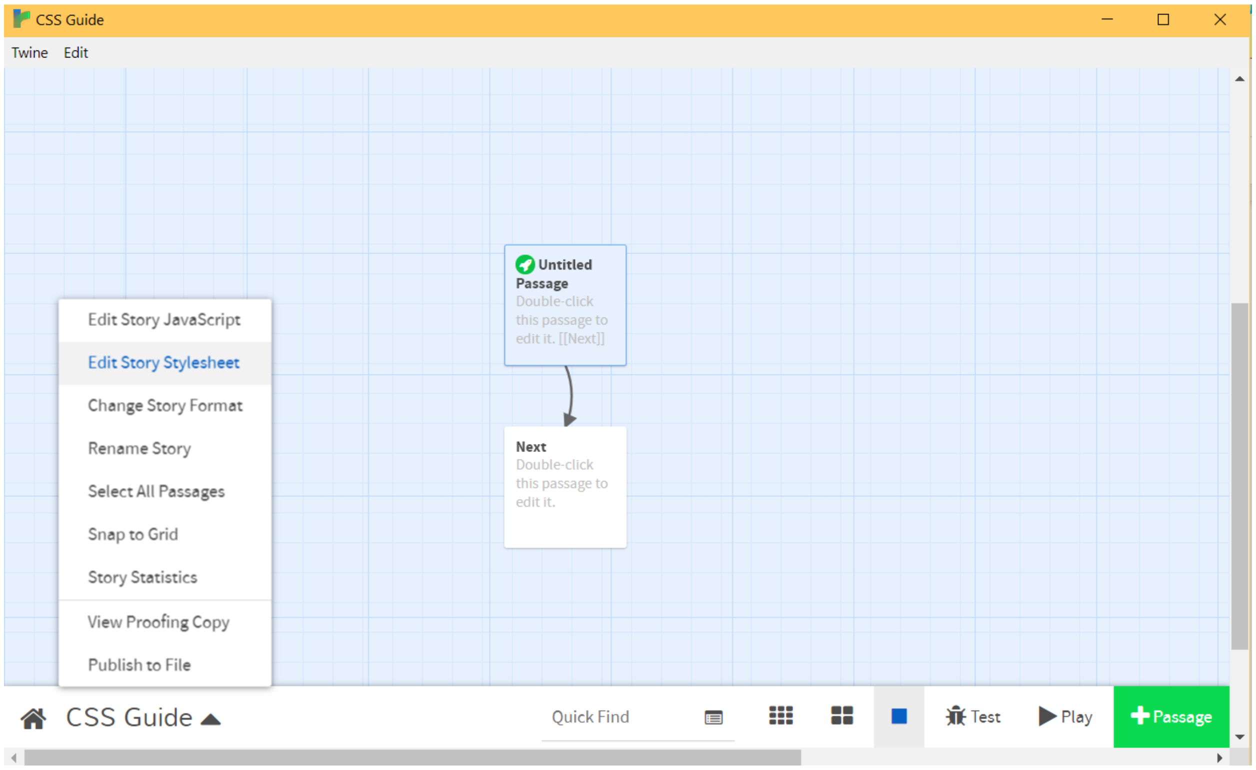


I have a degree in English Literature and Classical & Near Eastern Civilizations, and I'm influenced by an interest in the uncanny and grotesque, as well as ritual, masks, and the macabre. You can check out my work on itch.io by clicking the image on the left.

I am **not** a developer of Twine itself, and am not involved with its development process, but I have contributed to the Twine Cookbook, and I am a developer who primarily works with Twine. I'm passionate about its utility to developers of all backgrounds, skill sets, and skill levels!

➥ RECAP: Volume 1

The Twine Grimoire: Volume 1 is intended for those new to Twine®: users who may have had some previous interaction with it as a player or developer, but 1) are still learning their way around the application itself, and 2) who have not yet familiarized themselves with the basics of CSS or HTML. Users do ***not*** need previous experience with web development, programming, or Twine®.



A screenshot of a Twine user navigating to the "Edit Story Stylesheet" menu option.

The Twine Grimoire: Volume 1 is an 84 page, 100% **free** PDF which contains **6 detailed tutorials** on:

- Installation & Getting Started
- Text Styling
- Link Styling
- Customizing Passages
- Customizing Backgrounds
- Embedding Images

Each section discusses both **SugarCube** and **Harlowe** Story Formats, with emphasis on the most recent versions.

Other features discussed include: **tag-based styling,** using the **CSS Stylesheet**, usage of **macros** and **named hooks** in Harlowe, how to **organize your files**, and other topics. There are **screenshots** and **examples of code** provided throughout each tutorial, along with **exercises** that you can do on your own. Additionally, *Volume 1's* introduction provides an overview of the basics of CSS.

Resources, including links to other helpful Twine®-related websites and guides, Story Format documentation, and places to get help with Twine or talk to other users, have also been included in the introduction to *Volume 1*. For convenience, I'll replicate the Resources sections in this volume as well.

➥ 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 Homepage
- 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 many 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

*For the purposes of *The Twine Grimoire*, focus is placed on the **most recent versions** of **SugarCube** and **Harlowe**, either of which is recommended to use as you follow along with the tutorials. However, if you're very confident with your knowledge of CSS, HTML, and JavaScript already, 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.

➥ 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 users.

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 professionals do regularly, and you're encouraged to do it as often as you need to. 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 six more tutorials!

HOVER STATES

In this section:

➥ RECAP: ":hover" Pseudoclass

If you're unfamiliar with pseudo classes or hover states, you can refer to *The Twine Grimoire: Volume 1* for more information. Here's a quick recap of what was established in *Volume 1*:

Per w3schools, "a **pseudo class** is used to define a **special state** of an element." Pseudoclasses can be used to **style an element when a user mouses over it,** style visited and unvisited links differently, and style an element when it is in focus (such as when clicking into a text box).

In this case, the "special state" we're interested in is the **hover state**, visible when the player brings their mouse cursor over an element in your game.

➥ Ways to Use ":hover"

Parts of your game that may benefit from a hover state include **links**, **images**, and **<div> elements**, but you'll most often find the :hover pseudo-class useful for **links**, since not all Twine projects use everything else listed.

To apply a hover state to something, :hover must be appended directly to the element's selector in the stylesheet, like so:

LINKS

```
/* SugarCube */
.a:hover {
   color: #ff0000;
   text-decoration: underline;
   letter-spacing: 1px;
}
```

```
 /* Harlowe */
tw-link:hover {
   color: #ff0000;
   text-decoration: underline;
   letter-spacing: 1px;
}
```

The code above will change text color to red, add an underline, and add 1px of letter spacing to all links when the mouse cursor hovers over them.

<DIV> ELEMENTS

For this example, I'm creating a simple <div> element called "links" that can be used as a container.

```
/* The un-hovered element */
.links {
   display: inline-block;
   padding: 5px;
   width: 50%;
   background-color: #333;
   border: 4px double #a8a8a8;
}

/* Works for both SugarCube & Harlowe */
.links:hover {
   background-color: #111;
   border: 4px double #fff;
}
```

The code above changes the link container's background and border each to a lighter color when hovering over the element itself. Using this method in a passage will look like this:

```
<div class="links">Your text here.</div>
```

LINKS WITHIN <DIV> ELEMENTS

If you want the links inside of a <div> to look different than the links in the previous example:

```
/* SugarCube */
.links a:hover {
   color: gold;
}
```

```
/* Harlowe */
.links tw-link:hover {
   color: gold;
}
```

Implementing this in a passage will look like this:

```
<div class="links">[[Your link here.]]</div>
```

IMAGES

Maybe you're using an image instead of a text-based hyperlink, and want the image to respond when the player hovers over it. In this case, you might be putting your image inside a <div> class — because if you apply :hover to just "img," it will affect every image in your project.

I won't be styling this hypothetical <div> (which I'll call "nextpage"), but here's a demonstration on how to style the image within such a class:

```
/* Works for both SugarCube & Harlowe */
.nextpage img:hover {
   box-shadow: 3px 3px 10px 10px #fff;
}
```

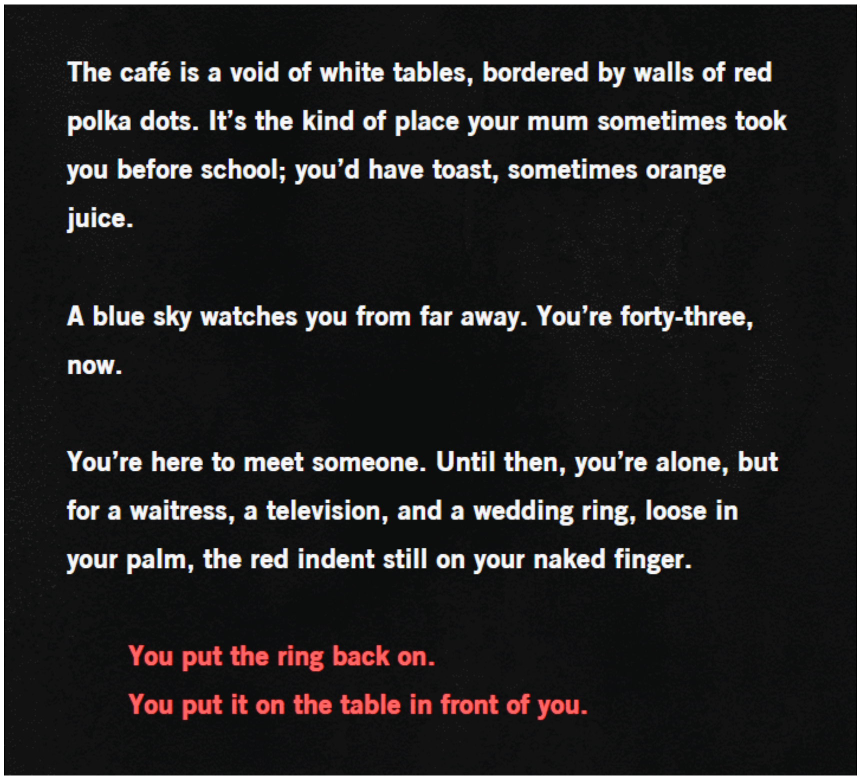
This creates a diffuse white shadow when hovering over the image. Implement this by putting the <img src=""> containing your image within the tags:

```
<div class="nextpage"><img
src="images/yourimage.png"></div>
```

➥ Hovering & Interactivity

Links, especially in a Twine project, should immediately **communicate interactivity** in their active state in order for the player to feel inclined to click on them. The hover state also helps to do the same. When a link changes in response to the player's attention, namely contact with their mouse cursor, it **reinforces** its interactivity, demonstrating that clicking on it will change something.

In some cases, the hover state can convey a preview of what's about to happen: for example, in Greg Buchanan's *Paper Brexit* (which I programmed), certain links shudder and shake all the time; some links begin to shake violently only when the player is hovering, about to click on them.


The café is a void of white tables, bordered by walls of red polka dots. It's the kind of place your mum sometimes took you before school; you'd have toast, sometimes orange juice.

A blue sky watches you from far away. You're forty-three, now.

You're here to meet someone. Until then, you're alone, but for a waitress, a television, and a wedding ring, loose in your palm, the red indent still on your naked finger.

You put the ring back on.
You put it on the table in front of you.

Some links in Paper Brexit by Greg Buchanan ***shake*** *with different degrees of intensity when hovered over.*

It's ominous and unnerving; it creates a sense of intensity and disruption. (This is an option that can be turned off in accessibility settings, but we thought it would be cool to offer!)

Different ways of styling hover states can convey completely different things. In some games, the color of the font might brighten to something more whimsical and fun for some links, or a white text-shadow might envelop another link in holy gravitas.

For certain games, you may not want to style your hovered links, or may want to do so only minimally. It's kind of a matter of personal taste, but I would caution against having **no** hover state at all; even a very slight change can be useful in helping the player more easily identify what to click and when.

Consider the websites you use daily; if a lot of the elements you went to click on didn't respond to the cursor, would you think something was broken or disabled? Many players will still click to check anyway, but eliminating that potential moment of pause or confusion is an extra bit of polish.

It's worth noting that a lot of the time, hover effects will be more understated on **mobile devices**, since a tapping finger doesn't usually linger on a link or other element for long enough to trigger an animation or even a quick change. (However, if the player does happen to linger a little, they'll still see the change. Similarly, some hover-based features like tooltips will appear on tap and disappear on the next tap.) This is why it's also important to have clearly-designated **default states** for links, as described in *Volume 1*.

➥ Designing a Hover State

It doesn't have to be a complex animation or a significant color shift; just a minute change in a link's letter spacing or text-shadow could do the trick if you want to keep things subtle, but color changes are often helpful too. A gray link or button usually appears to convey that something has been disabled or "greyed out," so this kind of element gaining color would be significant. (On the other hand, depending on the game, an opposite, de-saturating effect could be cool, too.)

Again, poke around on websites that you frequent and take note of what elements change when you hover, and how. A lot of these changes are subtle; for example, some links in my Patreon sidebar just change from gray to black in an almost-immediate transition, but they're completely effective at communicating what they need to.

Also consider your **cursor** behavior. Using the **cursor property**, you can control the appearance of the player's cursor in several ways with only CSS. As an example, when something isn't meant to be clicked on, I like to use *cursor: no-drop;* on hover. There are a lot of other cursor values to try out as well.

As far as **transitions** go, the following note is a recap from *Volume 1's* Link Styling tutorial:

An abrupt hovering effect can be neat in some situations, but a state change will look and feel smoother and more attractive with a transitional effect. There are many forms transitions can take, but my most simple go-to is the following:

```
-webkit-transition: all 500ms ease;
-moz-transition: all 500ms ease;
-ms-transition: all 500ms ease;
-o-transition: all 500ms ease;
transition: all 500ms 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 (1s/1000ms) or two (2s/2000ms) at the most; shorter than that is great, as in the example above. 500ms, half a second, is a brief moment that still gives a nice fade-in.

Save longer, 1-2s transitions for things that really warrant it. Generally, you won't want a visually elaborate or time-consuming animation on an element with which the player will be regularly interacting.

There's so much you can do with hover states; the possibilities are just as limitless as they are for regular link styling. When you're designing this kind of thing, there are a few questions worth asking:

- **What parts of this element should change?** (Font, color, spacing, shadows, etc?)
- **Should this be a significant change or a subtle change?**
- **Should this affect any other elements?** (Like links within a <div>?)
- **What will this change be communicating?** (Ominous, playful, glitchy?)

Go hog wild and mess around. Hover states are some of my favorite things to implement.

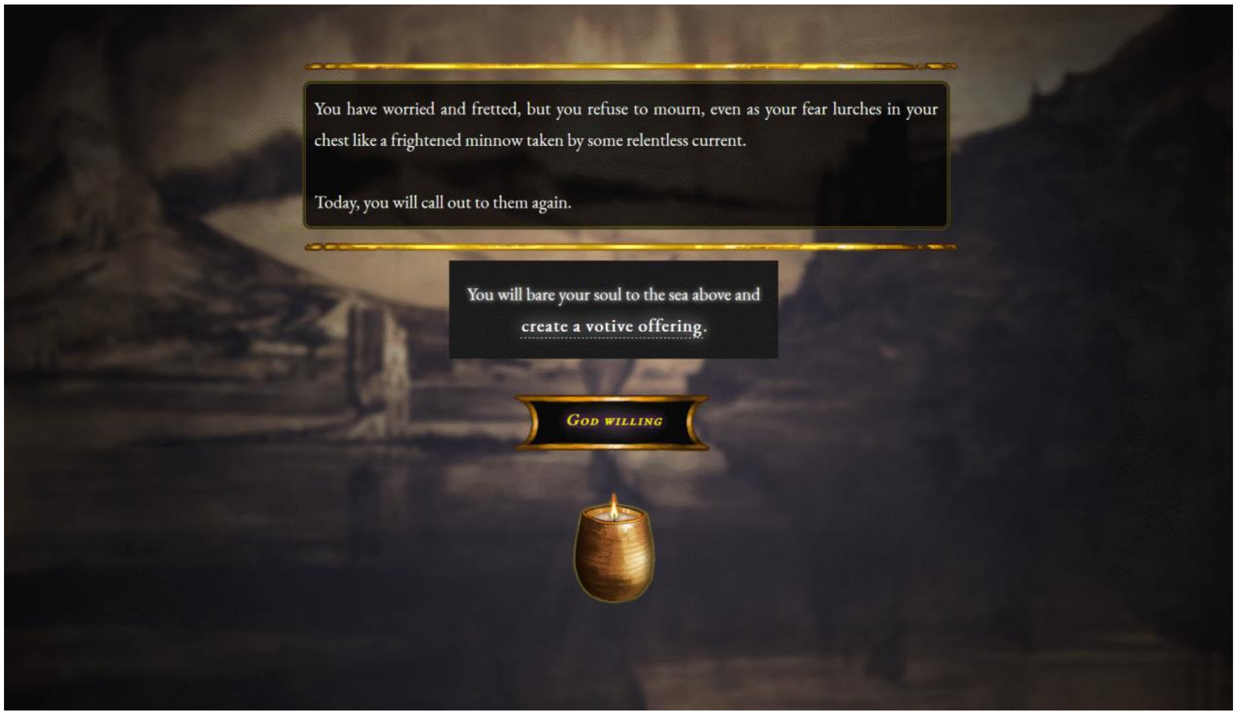
➥ Examples & Screenshots

Here are two examples from my own games, and one from the page hosting one of my games on itch.

I'm not going to fully demonstrate all of the CSS in this section, but I will share thoughts on why I designed things the way I did (and there will be a look at *DEVOTIONALIA*'s CSS in the next tutorial if you're curious).

DEVOTIONALIA

DEVOTIONALIA has several different types of links, all of which utilize hover states. Its buttons and some of its <div> elements also use hover states. I'll focus on one specific type of link here.


You have worried and fretted, but you refuse to mourn, even as your fear lurches in your chest like a frightened minnow taken by some relentless current.

Today, you will call out to them again.

You will bare your soul to the sea above and create a votive offering.

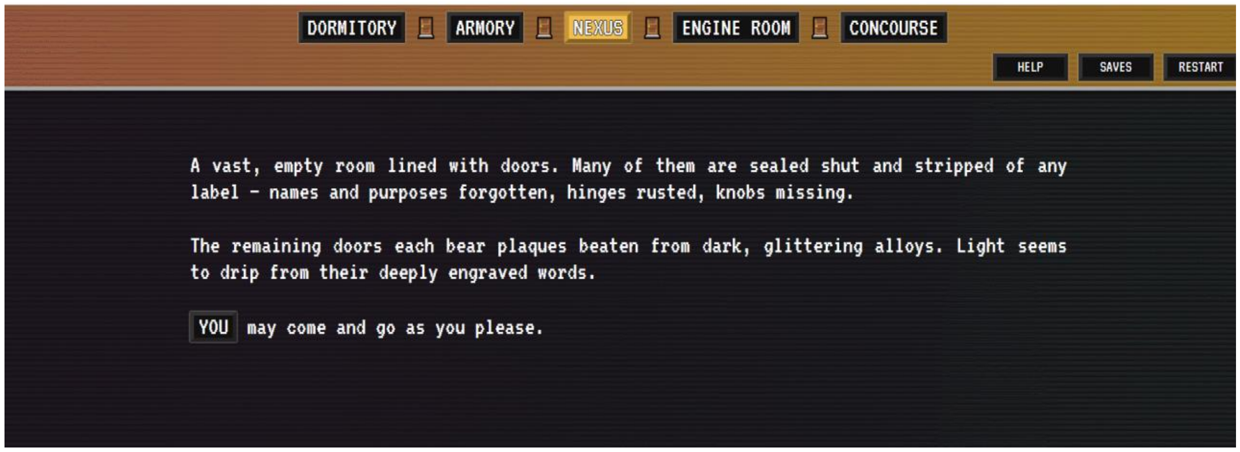
The "*GOD WILLING*" link is the main "decision making" link in *DEVOTIONALIA*, which is always contained within a separate <div> element; this element also changes on hover, gaining a shadow when the mouse is inside the <div> contents.

The "*create a votive offering*" link is a **cycling link** which uses a dissolve transition between different versions of the link's text. (There will be a full tutorial on cycling links later in this volume.) The animation that dissolves each option in the <<cyclinglink>> macro is a different animation applied to a different selector, but there is a simple transition between the regular hover and active states as well.

Because the "*GOD WILLING*" link is the most "important" kind of link in the game, it's the most elaborate; I think this conveys more of a sense of consequence, of action and reaction. The player is meant to get the feeling that this choice "matters" just from the way the link looks and responds to their touch.

HABEAS CORPUS

This game uses hover states for its links as well as its buttons. Since I talked about links for *DEVOTIONALIA*, I'll focus on buttons here.



A vast, empty room lined with doors. Many of them are sealed shut and stripped of any label - names and purposes forgotten, hinges rusted, knobs missing.

The remaining doors each bear plaques beaten from dark, glittering alloys. Light seems to drip from their deeply engraved words.

YOU may come and go as you please.

I didn't want to go too wild with elaborate buttons for this game, so I gave all of them essentially the same appearance and behavior. There are CSS differences between the small versions of the buttons (dialog API-triggering) and the larger buttons (straightforward navigation and choices), but generally their behaviors are the same on hover: a simple background color shift to show that they can be clicked on. The buttons themselves are custom <div> classes.

I also wanted the button color to change to show which room the player is in, but that's separate from the hover state. When the player is already in a room, that room's button doesn't change on hover; this is intentional, letting the player know that there's no need to try clicking on it, since they're already there, which is also indicated by the button's bright color.

Keep this in mind for the next tutorial, which will cover how to create custom buttons of your own.

THE EMBRACE

This is a Bitsy game in which the game itself contains no clickable links or hover-able elements, but I added a bit of custom CSS to the game's actual page on itch which affects the header image on hover.

I wanted to go buckwild on a bit of gore for this one, but I also didn't want to really disturb anybody who wouldn't be expecting to see an eviscerated alien guy's intestines front-and-center on some random game while browsing Bitsy collections.

THE EMBRACE

Instead, I made the header image's default state blurred, de-saturated, and color-shifted using the *filter* property. Color and detail fade in if the mouse cursor hovers over the image. (I still put a mosaic over the gore in an image editor to be polite.)

This filter effect is accomplished with the following properties:

```
        filter: blur(3px) hue-rotate(90deg) sepia(60%);
        transition: 500ms linear;
```

The hover state is created using these properties:

```
        filter: blur(0px);
        transition: 500ms linear;
```

! To enable the CSS editor on your itch.io account, you must email Support! For more information about this process, refer to this link.

➥ Wrap-up

Hover states are some of my favorite components to work on for my games. I really like tweaking my links until I feel like they exude the game's vibes just the way I want them to, and hover properties are part of that. Sometimes I'll come back to a game's stylesheet a few weeks or months later and overhaul hover behaviors entirely after playtesting. Lately, I'm experimenting with more subtle changes, but more elaborate ones are interesting as well.

It's fun to design this aspect of a game partly because the CSS is very simple, with nothing extra to learn or install; there's no JS or complex ruleset to keep in mind. It's worth doing because these properties can give a greater sense of interactivity and polish to a game in very subtle ways.

➥ Exercises

EXERCISE 1:

- Brainstorm how you might **communicate interactive text** using a hover state. When the player's cursor is over a link, how should the text **respond**?

EXERCISE 2:

- **Apply** the state you designed in Exercise 1 to a link in your game using CSS. Did everything go according to plan? When you implemented your design, did you find yourself adjusting anything after playtesting? **Why** or **why not**?

EXERCISE 3:

- The next time you're **playing a game** or even just **visiting a website** – this can be any game on any platform, or any website – observe the behaviors that clickable elements exhibit when your cursor is hovering over them. What do you find **effective** or **ineffective** about the way they're presented?

CUSTOM BUTTONS

In this section:

In this tutorial, we aren’t going to run into any CSS that’s exclusive to SugarCube or Harlowe. Unless noted otherwise, these methods will work in **all** formats that the *Grimoire* has discussed.

➥ Design Tips

First and foremost, buttons aren’t a necessity; you don’t have to have buttons in your game. The project you have in mind might not be suited to having them at all, and that's fine. But if you're interested in experimenting with them for other projects, or just want to know how to make them, this will come in handy.

Start by thinking about how you want your game to look and feel.

- Do you want a **chunky, sharp, retro** aesthetic?
- A **smooth, bubbly, friendly** look?
- A **dark and ornate** vibe?
- **Something else entirely?**

You'll want to then consider things like font size, colors, animations, and how you want links to behave, because your buttons will inherently contain links. If you're stuck, it might help to review the Link Styling tutorial from *Volume 1*, but I'll discuss some specific properties to consider after demonstrating some CSS, and there will be a full tutorial on animations later in this volume.

➥ CSS Implementation

There are four selectors you'll find useful:

```
.button {

   /* Your CSS Here */

}
```

! The period before your selector name is important. Delete the **entire** " */* Your CSS Here */* " portion! SugarCube 1.0.35 uses "button" already (without a period) as part of its UI, but ".button" creates a custom class.

This is the basic structure of your button. It can be named anything you want. It'll contain properties like your background color, border, etc.

```
.button:hover {

   /* Your CSS Here */

}
```

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

The :hover pseudoclass is back to help communicate interactivity. You'll want to use this only for properties which change; for instance, if you want the font size to increase when you hover over the button, or the background color to shift. This means that you do **not** need to copy and paste all of the properties under ".button" to ".button:hover," only the ones being altered.

The properties associated with your button's default link state go here — font size, color, etc:

```
.button a {

   /* Your CSS Here */

}
```

This is how the link will appear when hovered over:

```
.button a:hover {

   /* Your CSS Here */

}
```

You can go ahead and drop all of those into your CSS Stylesheet if you plan on creating a button.

! A Note About SugarCube: If you're using SugarCube, you might have noticed that the UI sidebar has its own buttons, and that there are also buttons within its UI dialog popups. **These are not customized in the same way.** A **separate tutorial** on **customizing SugarCube's built-in UI,** including those buttons, the UI bar, and how to hide that sidebar if you prefer, is coming up.

➥ Useful Properties

Essential properties you'll want to think about when creating buttons:

- **width:** the width of the element;
- **height:** the height of the element;
- **display:** a value of **inline-block** is necessary for you to **set a width and height** for the element (it also does not add a linebreak after the element, allows you to place elements **next to each other**, and respects top and bottom margins if you need to set any)

Some other properties you can experiment with:

- **color:** your text color, an rgb value or hex code;
- **padding:** the amount of space around your text;
- **background:** or **background-color:** an image, color, or gradient;
- **border:** the border around the button;
- **border-radius:** the closer to 100px, the more rounded the button (it will need to have the same width and height values if you want a 100px border-radius button to be perfectly circular);
- **font-size:** like I mentioned above, it can be neat to have a slight increase in font size on hover to communicate that the button text is interactive;
- **text-align:** buttons tend to look nice with center-aligned text;

➥ Using A Background Image

This is very similar to how we assigned passage backgrounds, but for buttons, you'll want a bespoke image that fits the exact size you want. Check out the Background Images tutorial in *Volume 1* for an in-depth look at background properties and how to organize your image files.

You'll want to use the *background-image* property to set this up, but you'll also want to assign your button the same width and height as your image.

Assuming *yourimage.png* is 180px by 60px, you'd put this in your stylesheet under the *.button* selector:

```
.button {

   background-image: [img[images/yourimage.png]];
   width: 180px;
   height: 60px;

}
```

You can also use this method for your background image values:

```
background-image: url("images/yourimage.png");
```

➥ HTML Implementation

To make your button visible, click into the Twine passage where you want it to be located and wrap your desired link in a div class tag like this:

```
<div class=”button”>[[Button Link]]</div>
```

Play your game within Twine (if not using a background image) **or** save and test your index.html file (if using an image) and check it out! From there, you can experiment with how it looks and feels, then return to the stylesheet to continue tweaking it.

➥ Examples & Screenshots

I’ll use a button from my game ***DEVOTIONALIA*** as an example. Here's the associated CSS:

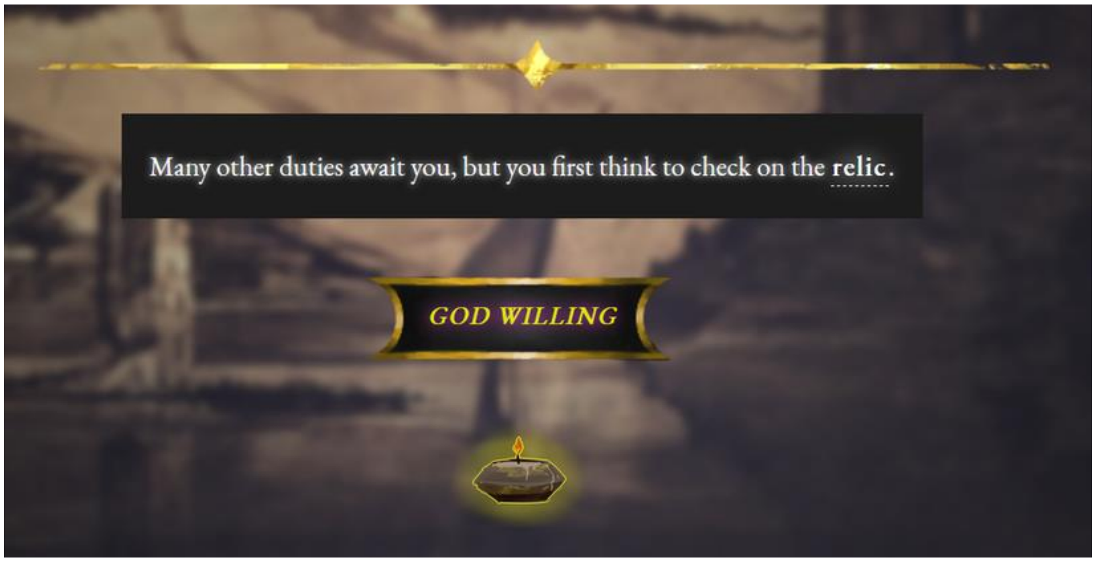
```
.godwilling {
   display:inline-block;
   background-image: [img[images/devoui.png]];
   width: 244px;
   height: 71px;
   font-family: 'EB Garamond', serif;
   line-height: 70px;
   font-size: 18px;
   text-transform: uppercase;
   -webkit-transition: all 2s ease;
   -moz-transition: all 2s ease;
   -ms-transition: all 2s ease;
   -o-transition: all 2s ease;
   transition: all 2s ease;
}

.godwilling:hover {
   background-image: [img[images/devouih.png]];
   font-size: 20px;
   -webkit-transition: all 2s ease;
```

```
               -moz-transition: all 2s ease;
               -ms-transition: all 2s ease;
               -o-transition: all 2s ease;
               transition: all 2s ease;
            }

            .godwilling a:hover {
               text-shadow: 0px 0px 15px #fff !important;
            }

            .godwilling a {
               text-shadow: 0px 0px 10px #ff37ec !important;
            }
```

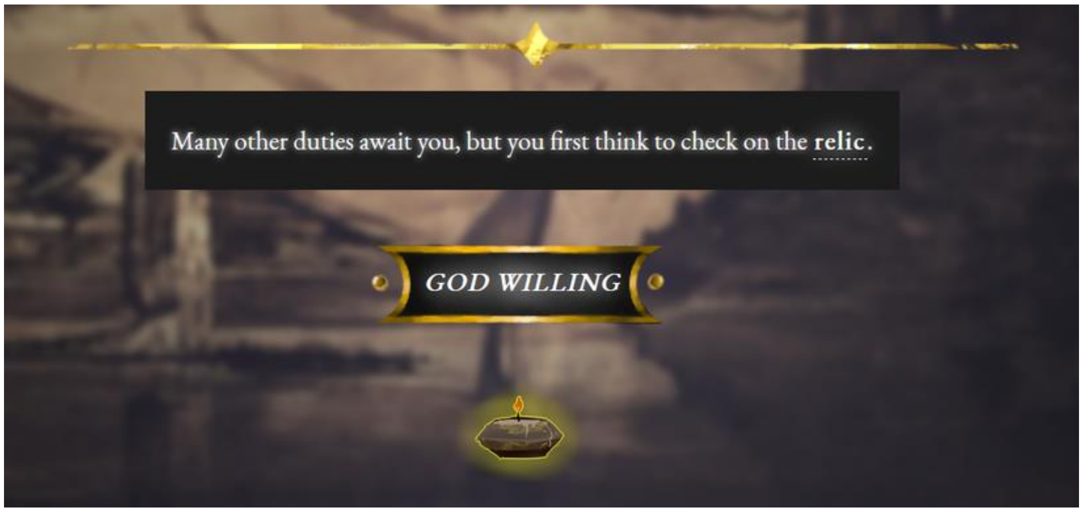
And here is what's going on under the hood:

- **Font size** slightly increases when hovering over the button itself
- **Font color** changes when hovering over the link text
- **Text shadow** changes color when hovering over the link text
- **Background image** changes when hovering over the button itself
- **Line-height** better matches the height of the button in order for the text to stay in place while remaining 18px
- All text is **uppercase**
- All hover effects occur in a 2 second **transition**
- **display: inline-block** property allows me to **set a fixed width and height** to match my art asset

Here’s what all of that looks like in-game:


Many other duties await you, but you first think to check on the relic.

And here’s what it looks like when hovered over:


Many other duties await you, but you first think to check on the relic.

➥ Exercises

EXERCISE 1:

- **Design a button**! Try sketching it on paper or creating a mockup in an image editor. Think about **properties** like padding, borders, and colors, as well as what impression you want your UI to leave on a player.

EXERCISE 2:

- **Design** and **implement** a button with a **custom background**. This can be any image or pattern, but it's recommended that you use an image which will not obscure the link text on top of it.

EXERCISE 3:

- **Design** and **implement** a button with a **hover state**. Think about how you want the button to change in response to the player.

HARLOWE UI

In this section:

➥ Undo/Redo Arrows

If you're a Harlowe user, you've no doubt noticed the arrows that appear alongside your text when testing your project.

These **Undo/Redo arrows**, located by default toward the upper left of the passage, are a handy equivalent to the "history" buttons in SugarCube's UI Bar, which allow you to go back and forth between already-visited passages.

The relevant selector and default values are:

```
tw-icon {
   display: block;
   margin: 0.5em 0;
   opacity: 0.1;
   font-size: 2.75em;
}
```

Both the undo and redo arrows are affected by this selector.

I recommend sticking with the default arrow shapes, as they are pretty unobtrusive shape, but to change their appearance, you’ll want to trying adjusting properties like the *opacity value* and *font size,* and you can also add other properties like *color* or *text-shadow.*

➥ Arrow Alterations

Here’s a sample change:

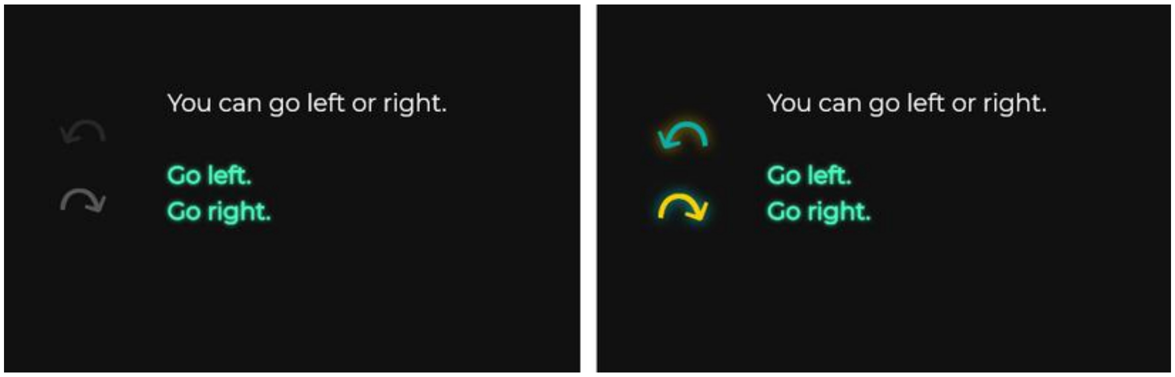
```
tw-icon {
   display: block;
   margin: 0.5em 0;
   opacity: 0.8;
   font-size: 3em;
   color: #03d7d8;
   text-shadow: 0px 0px 15px #ffd000;
}
```

And here’s an added hover class:

```
tw-icon:hover {
   opacity: 1;
   color: #ffd000;
   text-shadow: 0px 0px 15px #03d7d8;
}
```

You can copy and paste the above into your stylesheet (if you’re working with Harlowe) to test and alter it yourself.

Here's a screenshot of default (left) vs. customized arrows (right):



Play around with the CSS above to find something that fits your project!

➥ Removing the Arrows

If you want to **get rid of these buttons** entirely, use the following CSS:

```
tw-icon {
   display: none;
}
```

Sometimes you won't want the player to be able to use this feature; it's up to you to decide whether it's relevant to your game and how to customize it.

➥ Other UI Aspects

Because Harlowe's UI is quite minimalist, these arrows are the only built-in feature you may immediately wish to customize, but this also creates an opportunity for you to experiment with your own sense of design.

Think about the different parts of the page that you see when using your browser's "Inspect" function on a Twine project: the story background, the passages, and the links are the best places to start building a unique look and feel for your project.

➥ Exercises

EXERCISE 1:

- Do the **undo/redo arrows** undermine a mechanic of your game? **Get rid of them** using CSS.

EXERCISE 2:

- **Customize** the appearance of your **undo/redo arrows** using CSS; keep in mind properties like color, size, and shadow.

EXERCISE 3:

- **Design** and **implement** a **hover state** for your undo/redo arrows. How should they respond to the player's cursor hovering over them?

SUGARCUBE UI

In this section:

If your main barrier to using SugarCube thus far has been that the sidebar doesn't look right or has no relevance to your project, I am here to liberate you of this burden by showing you how to either A) obliterate it completely, or B) make it more handsome. This tutorial will also discuss built-in API functions and dialog popups.

➥ SugarCube's API Functions (Save & Reload)

! This section is only relevant if you plan to use SugarCube's built-in Save or Reload API functions in your game. If you **do not** need or want your player to be able to save or reload, **but you still plan to use the UI Bar*, add the following to your CSS:**

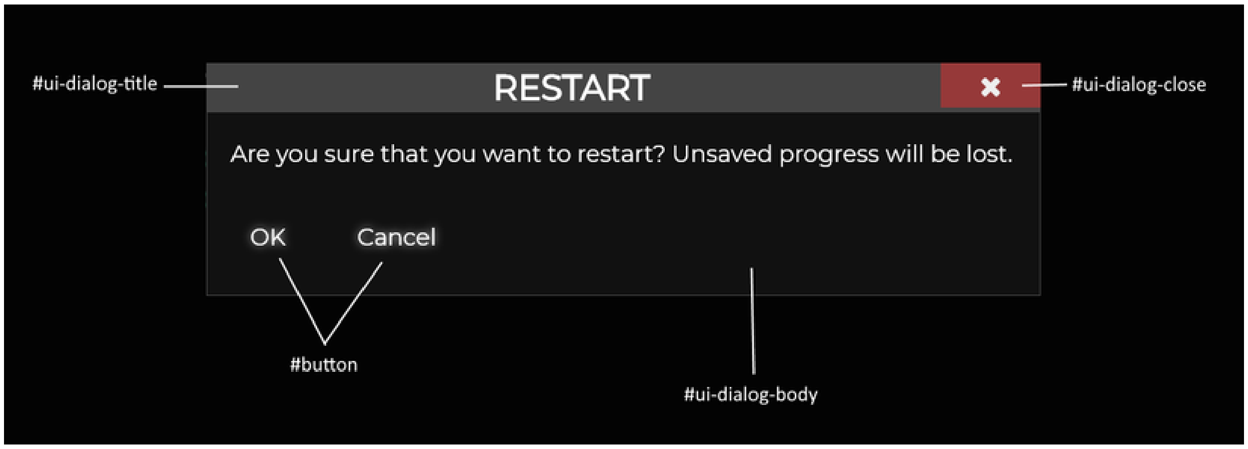
```
#menu ul {
   display: none;
}
```

**If you aren't planning to use the UI bar at all, you don't need to do this; we'll talk about removing that entirely in the next section.*

If you've been using SugarCube, you'll have already noticed the **Save** and **Reload** buttons in the sidebar when you first created your project. In this section of the tutorial, we're going to look at the **popups** that appear when you click either of those options.

➥ Dialogs: Discovering Selectors

There are four main features here; each has a discrete, specific **selector** which isn't immediately obvious without using your browser's **Inspect** tool to discover its name.



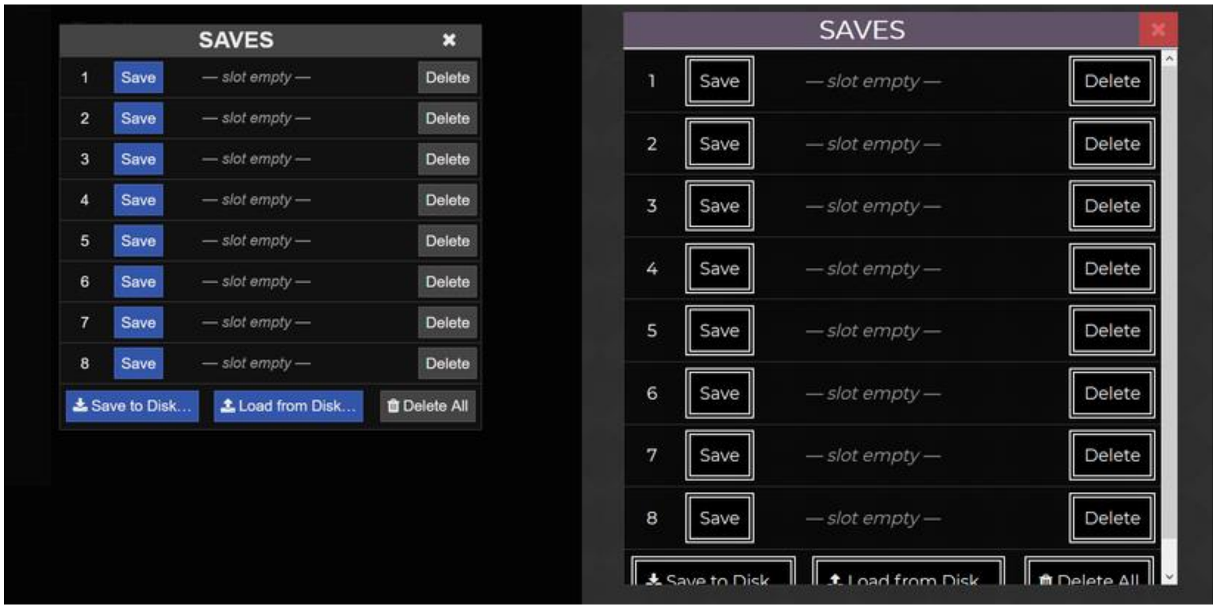
Are you sure that you want to restart? Unsaved progress will be lost.

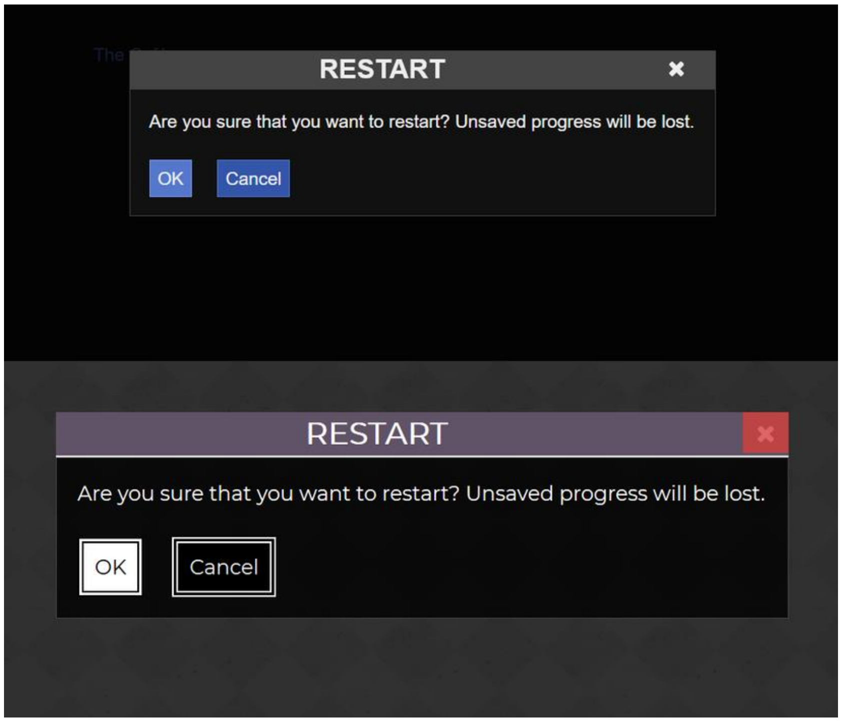
- **#ui-dialog-body** — the dialog box itself
- **#ui-dialog-title** — the box header containing its title
- **#ui-dialog-close** — the X in the upper right corner
- **#button** — regular dialog buttons (OK, Cancel, Save) — this is NOT .button, a separate class which you may have created previously to style your own custom buttons

Here's a sample customization:

```
#ui-dialog-body {
   background-color: rgba(0,0,0,0.8);
}

#ui-dialog-title {
   background-color: #615367;

   border-bottom: 2px solid #fff;
}

#ui-dialog-close {
   width: 40px;
   background: #bb4444;
   color: #dd6666;
}

#ui-dialog-close:hover {
   color: #fff;
   border: none;
}

#ui-dialog-body button {
   padding: 10px;
   background: #000;
   border: 6px double #fff;
}

#ui-dialog-body button:hover {
   background: #fff;
   border: 6px double #000;
   color: #000;
}
```

And here are a couple of **comparisons** between the default Save and Reload dialogs and the results of the code above:


SAVES
1 Save — slot empty — Delete
2 Save — slot empty — Delete
3 Save — slot empty — Delete
4 Save — slot empty — Delete
5 Save — slot empty — Delete
6 Save — slot empty — Delete
7 Save — slot empty — Delete
8 Save — slot empty — Delete
Save to Disk…
Load from Disk…
Delete All
SAVES
1 Save — slot empty — Delete
2 Save — slot empty — Delete
3 Save — slot empty — Delete
4 Save — slot empty — Delete
5 Save — slot empty — Delete
6 Save — slot empty — Delete
7 Save — slot empty — Delete
8 Save — slot empty — Delete
Save to Disk
Load from Disk
Delete All



Are you sure that you want to restart? Unsaved progress will be lost.

Are you sure that you want to restart? Unsaved progress will be lost.

There are a lot of ways you can alter backgrounds, hover properties, and buttons here, but it's up to you and the way you want your project to look. Refer back to previous tutorials for some ideas about how to customize buttons and backgrounds.

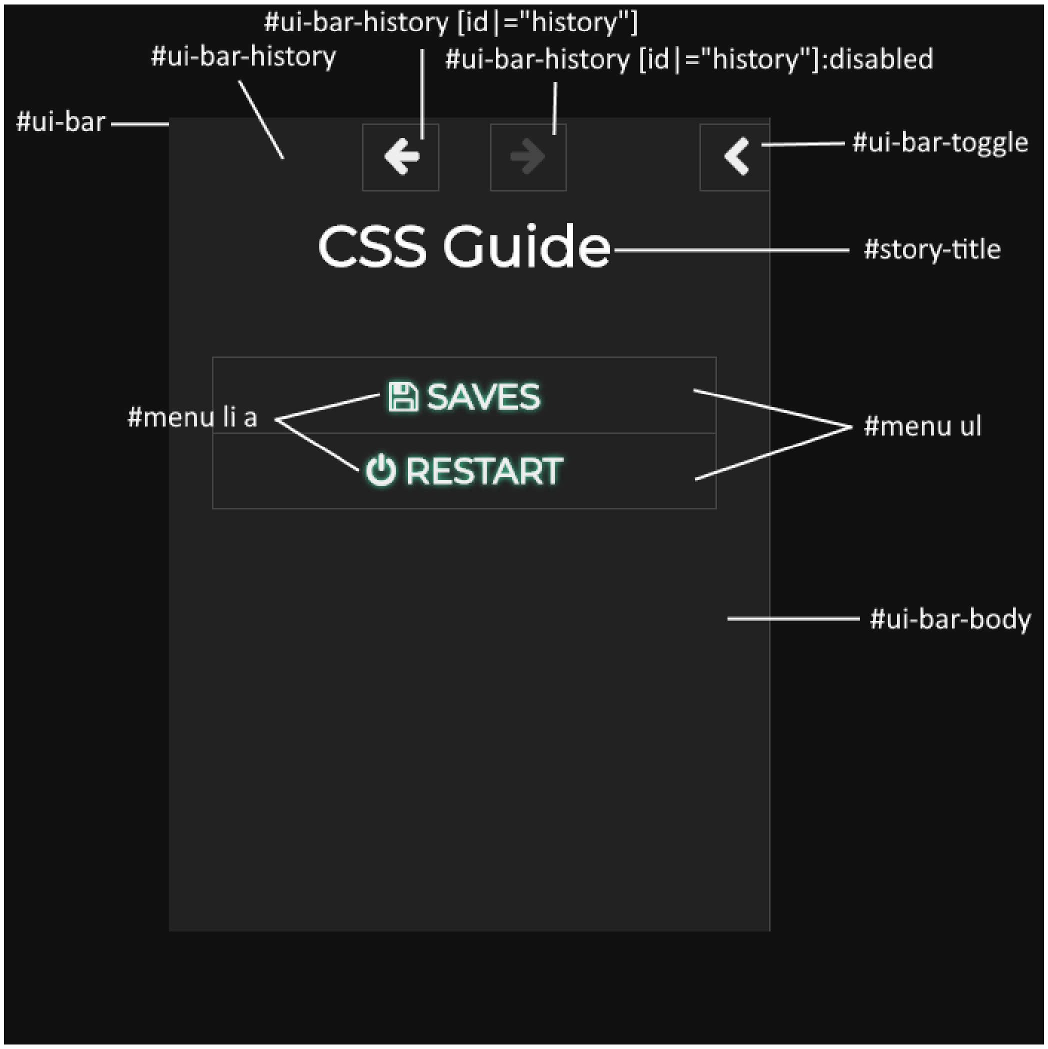
➥ SugarCube’s UI Bar

Right off the bat, if all you want to do is **remove this entirely**, add **the following** to your **Story JavaScript**:

```
        UIBar.destroy();
```

➥ UI: Discovering Selectors

To get into **customizing** the UI bar, though, the next page contains a diagram of **some** selectors you’ll need; this **isn’t** every component of the UI bar, but I've found that these are the most useful ones to mess around with as you’re getting comfortable customizing it.



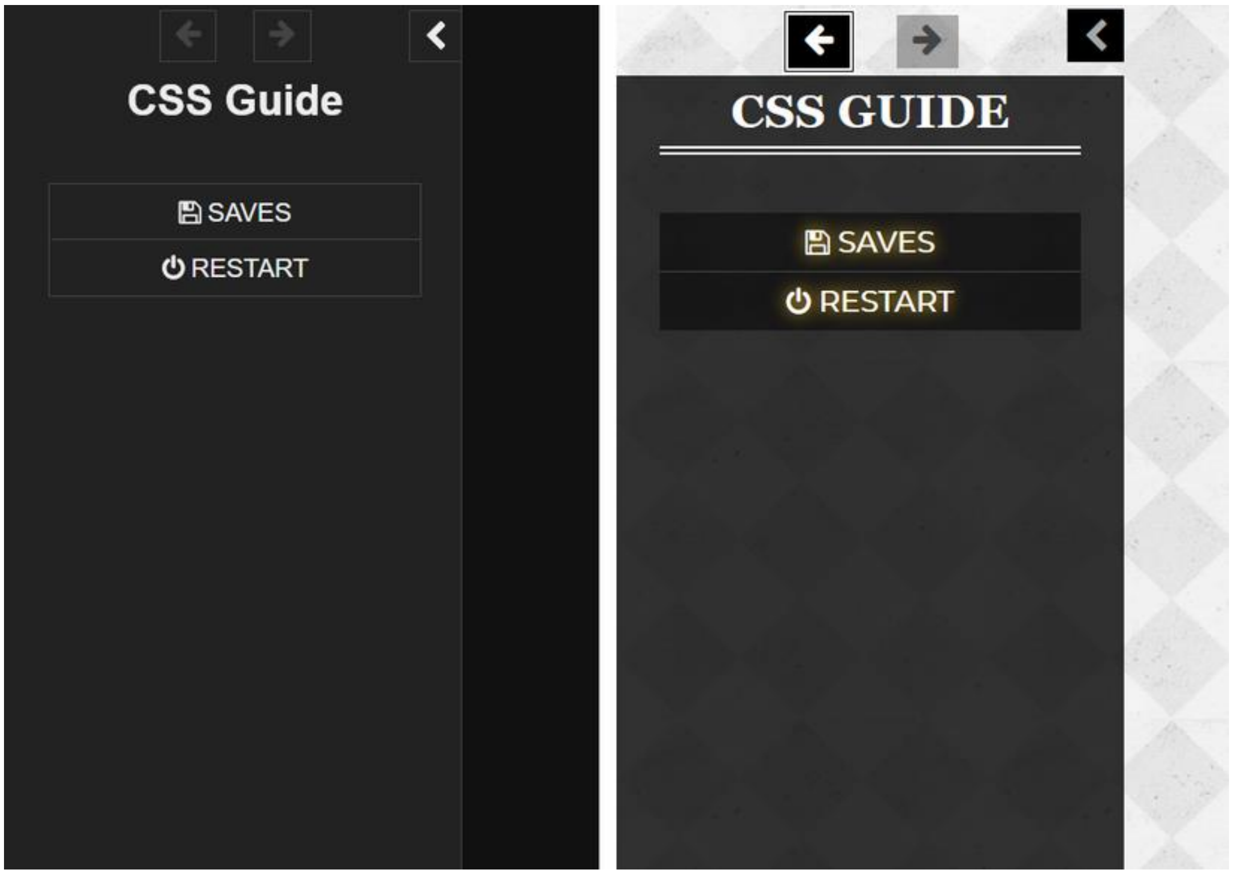
- **#ui-bar** — the UI bar itself; its background, borders, etc
- **#ui-bar-body** — the portion of the UI bar beneath the history/toggle area
- **#ui-bar-toggle** — the button on the upper right that collapses/expands the UI bar
- **#story-title** — the story title at the top of the UI bar
- **#ui-bar-history** — the area around the undo/redo arrows (this is the width of the UI bar and stops just above the Story Title)
- **#ui-bar-history [id|="history"]** — active history button
- **#ui-bar-history [id|="history"]:disabled** — disabled history button
- **#menu ul** — menu buttons (Save, Reload)
- **#menu li a** — menu button link text

Here’s another sample customization:

```
#ui-bar {
   border-right: none;
   background: none;
}

#ui-bar-body {
   background-color: rgba(0,0,0,0.8);
}

#ui-bar-toggle {
   border: none;
   background-color: #000;
   color: #a8a8a8;
}

#ui-bar-toggle:hover {
   color: #fff;
}

#story-title {
   text-transform: uppercase;
   font-family: Georgia;
   border-bottom: 6px double #fff;
}

#ui-bar-history {
   border: none;
   color: #fff;
   background-color: transparent;
}

#ui-bar-history [id|="history"] {
   background-color: #000;
   color: #fff;
   border: 4px double #fff;
}
```

```
#ui-bar-history [id|="history"]:disabled {
   color:#444;
   background-color:rgba(0,0,0,0.3);
   border: none;
}

#menu ul {
   border: none;
   background-color: rgba(0,0,0,0.5);
   color: #fff;
}

#menu li a {
   color: #fff;
   text-shadow: 0px 0px 10px #ffc926;
   border: none !important;
}
```

And here’s a **comparison** of that customization (right) to the default (left):



This code works best on a passage with a light-colored or light-patterned background, to play with the opacity and transparency properties. But you should feel free to change and experiment with the sample above to create the effects you're looking for!

➥ Exercises

EXERCISE 1:

- Does your game **benefit** from the UI bar, or is it **in the way**? Are its features irrelevant to your game's mechanics, serving only to clutter the screen? Using CSS and JS, **remove it** entirely from your game.

EXERCISE 2:

- Are you **keeping the UI bar** around? **Customize it!** Refer to the diagrams in the tutorial above in order to customize **each feature** of the UI bar and become more familiar with its different parts.

EXERCISE 3:

- **Customize** your **dialog popups**, including the title bar, dialog box, and buttons. Refer to the diagrams in the tutorial above.

CYCLING LINKS

In this section:

If you've played any of my games, you have almost certainly encountered a cycling link or ten. If you're interested in including them in your own games, here's a quick primer on how to start using them, and a more in-depth look at how to customize their appearance and behavior.

➥ Cycling Links in SugarCube

The <<cyclinglink>> macro is **not** built in to SugarCube, so you'll need to install it by putting the appropriate code into your **Story Javascript** and **Story Stylesheet**.

Alternatively, you can experiment with <<cycle>>, which is a macro built into SugarCube v2; I'll put up a link to that macro's documentation as well, but I've only used <<cyclinglink>> myself.

This is where I really wish I knew of a way to nicely corral code on Patreon with some kind of textbox, because I'm reluctant to post big walls of Javascript here. Instead, here are a few pages on which you'll find the correct instructions and code for whichever method you're using:

- SugarCube 1.x compatible <<cyclinglink>> macro
- SugarCube 2.x <<cyclinglink>> macro (incompatible with 1.x)
- Using <<cyclinglink>> + Advanced CSS (Same link as 1.x above)
- Using <<cycle>> in SugarCube 2.x (documentation)

➥ Cycling Links in Harlowe

In Harlowe 3.0.1, the **(cycling-link:) macro** was introduced. This is hype as fuck. You don't need to install anything in your Story Javascript and can just jump right into using the macro.

- Harlowe 3.x (cycling-link:) macro documentation
- And here is a page from the Twine Cookbook

This is also built in to Chapbook! Here's a resource for how to use the {cycling link} modifier:

- The Twine Cookbook: Cycling Links in Chapbook

➥ Customizing & Implementing in SugarCube

If you're using the **Glorious Trainwrecks** macro, under the **Advanced CSS** section on that page you'll find some really helpful information:

"Implementation details:

> * For CSS users: the <a> tag has the class names "cyclingLink" and "internalLink".
> * Each option is a <span> inside the <a> tag. Clicking it causes the next one down to have the class "cyclingLinkEnabled", and the others to have the class "cyclingLinkDisabled". **New:** on passage load, each option also has the class "cyclingLinkInit", which is removed as each is clicked.
> * When you click, the "data-cycle" attribute of the <a> tag increases by 1 (wrapping around to 0 at the end). You could select this with CSS, I guess, using [data-cycle=1] or somesuch."

If you do not use the classes above, your <<cyclinglink>> links will take on the properties of whatever you have **already set** for your default internal links. For instance, if you've set all your links to be red, these will also be red; to change that, you can use the classes detailed above.

The main classes you'll want to focus on are **cyclingLinkInit** and **cyclingLinkEnabled.** The **cyclingLinkInit** class controls how your cycling links will first appear before being clicked. The **cyclingLinkEnabled** class controls how your cycling links will look after clicking the initial option in the array. (You may not want these to look different; in that case, just give them both all the same properties and values.)

This means your CSS will look like this:

```
.cyclingLinkInit {
   color: #fff !important;
   text-shadow: 0px 0px 20px yellow !important;
}
.cyclingLinkEnabled {
   color: yellow;
   text-shadow: 0px 0px 2px yellow;
}
.cyclingLinkEnabled:hover {
   text-shadow: 0px 0px 10px yellow;
}
```

This should be reflected when you test or play your project.

➥ Customizing & Implementing in Harlowe

In Harlowe, the (cycling-link:) macro can be styled the same way as your usual links, using the tw-link selector. Your cycling links will take on the appearance of your other links, including any pseudoclasses like :hover.

If you want your cycling links to look different from your regular links, you can assign them each a custom span class like:

```
.cycle tw-link {
   color: #fff;
}

.cycle tw-link:hover {
```

```
                color: #ff0000;
                text-shadow: 0px 0px 5px #ff0000;
            }
```

To implement this, you'll use HTML in your passage to assign the *cycle* class only to specific text:

```
            <span class="cycle">(cycling-link: "Coffee",
            "Tea", "Beer", "Water")</span>
```

Bonus: If you want to create **multiple types of cycling links** (like I do in *DEVOTIONALIA*), you can also use this custom class method for that as well. This is also **compatible with SugarCube and Chapbook**; just set up your classes with CSS, then implement them as tags around specific links as needed.

That's about it for basic cycling link customization. I LOVE this mechanic and I'm excited to see it being implemented into recent versions of different Story Formats. It’s also available in the new Chapbook format!

You can get much more creative with your CSS than I do in these examples, but I wanted to keep it simple so that you can experiment on your own. Have fun and let me know if you have any questions!

➥ Exercises

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

EXERCISE 1:

- Set up a demo **cycling link**. If you're using SugarCube, you will need to download and install the correct custom macro using the Story Javascript and Story Stylesheet. If you're using Harlowe, you will need to examine the relevant macro documentation.

EXERCISE 2:

- Design the **states** of your cycling link. Will cycling links in your game look the same as static links, or will they be differentiated somehow? Will there be a hover state?

EXERCISE 3:

- **Implement** the designs you've created. This will involve either basic link customization, or the creation of a new class, which will also require a small amount of HTML.

SIMPLE CSS ANIMATIONS

In this section:

! GIF images will not display in Microsoft Word or Adobe Reader. As a result, the images featured in this tutorial represent only certain frames of some animations.

➥ About Animations

CSS animations allow you to make an element **change styles** by transitioning from one set of properties and values to another. A definition via Mozilla:

> "Animations consist of **two components**, a **style describing the CSS animation** and **a set of keyframes** that indicate the **start** and **end** states of the animation's style, as well as possible intermediate waypoints."

In contrast to script-based animations, CSS animations are easier to use, perform well in-browser, and shouldn't inhibit a browser's efficiency. They're also easier to create, and can be endlessly variable, allowing you to experiment a lot with them.

I found CSS animations intimidating at first, but they're actually very simple and fun to make!

➥ ❗Harlowe Complications

! If you intend to create CSS-based animations for your project, it is strongly recommended that you use the SugarCube story format. Further information about why this is can be found below.

There is a significant drawback to using CSS animations in **Harlowe**, as far as I can tell. If you run a custom animation in a Harlowe project, you'll notice that when it plays, it seems to be rendered twice in an abrupt manner. It will attempt to carry out your defined animation, then stall, start again, and completes as it was supposed to. This is a jarring visual effect.

According to **this thread**:

> "the way that Harlowe provides some (most?, all?) of its built-in animations seem unfriendly to both other CSS-based animations and/or any modifications of its own animation *[...]*"

This is a bummer. For more context on this conclusion and why this might occur, check the reply by user **TheMadExile** in the thread linked above.

I ran into this exact issue while doing some custom CSS for a client, and we ended up deciding on an alternative, non-animated design feature because of it. The only solution-shaped option I've found here is to disable all other animations, but this impacts the rest of the game negatively by preventing your other passages from fading in at all, so I don't recommend it.

It's unclear whether this is a bug or simply a design limitation unique to Harlowe, so if you want to experiment with animations, **I strongly recommend using SugarCube**.

➥ Defining Animations with Keyframes

First, decide on a unique name for your animation, and set up two or more frames: one for the initial point of the animation, and one for the end point. I'll be calling this animation "example" for now:

```
@keyframes example {
   from {font-size: 18px;}
   to {font-size: 24px;}
}
```

Paste the above into your **Story Stylesheet**.

! The amount of braces ({ }) here is important; don't forget that each set of properties needs to be within a set of closed braces, but so does the entire @keyframes rule itself.

Keyframe rules work like regular CSS. You can use a variety of properties depending on the effect you want to achieve: font color, font size, background colors or images, borders and text decorations, etc...

You can also set up keyframes using **percentage values** rather than "from" and "to;" this is something which will be covered **later** in the tutorial.

➥ Configuring the Animation

To apply the above @keyframe rule to a specific element, you'd use it as a value within the **animation property,** which will be added to the desired selector.

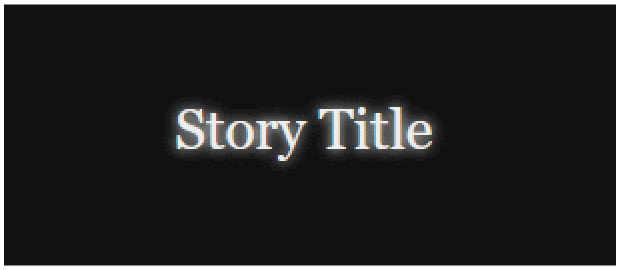
I'll put it into a quick custom element that you can also paste into the Story Stylesheet:

```
.title {
   font-family: Georgia;
   text-shadow: 0px 0px 5px #fff;
   animation: example 2s forwards;
}
```

To test this, add this HTML to a test passage:

```
<div class="title">Story Title</div>
```

Test your game; it should result in an animation which makes the text **slightly larger** over a period of **two seconds**.



Super simple, and probably not something you'd use without some embellishments, but a good way to get the ball rolling.

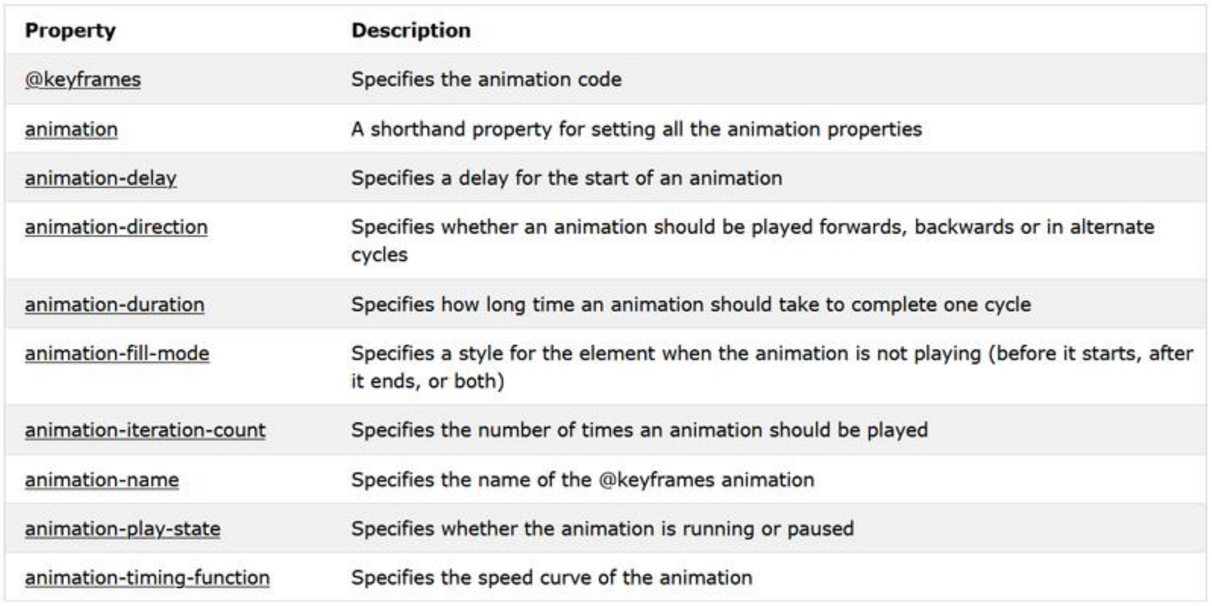
Now that you've created something small that works, you can expand on the idea with more properties.

➥ Animation Sub-properties

Above, we used the **animation shorthand property**, but there are numerous **sub-properties** associated with CSS animations that you can use in a more granular fashion.

We used three sub-properties above: **animation-name** ("example"), **animation-duration** (defined in *s* (seconds) or *ms* (milliseconds)), and **animation-fill-mode** (which can be *none*, *forwards*, *backwards*, or *both*).

Here's a table of properties to be aware of, courtesy of **w3schools**:



Property	Description
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties
animation-delay	Specifies a delay for the start of an animation
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to complete one cycle
animation-fill-mode	Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused
animation-timing-function	Specifies the speed curve of the animation

I'll help break down a few of these as well. Some, like **animation-timing-function,** still aren't extremely obvious; this governs the **pacing** of your animation, and can be set to different predefined options like *ease-in* and *linear* (you can create custom functions using cubic-bezier curves, but that's beyond the scope of what you need to know right now).

It's also good to know that **animation-iteration-count** can be set to *infinite* if you want your animation to **loop** endlessly, rather than play a single time or a set amount of times.

Keep in mind that you don't need to keep all of these properties separate; you can include these values within the **animation** property, but it's good to know what values like "linear" or "backwards" actually correspond to.

The recommended way to order sub-properties within the **animation** shorthand is as follows:

```
animation: [animation-name] [animation-duration]
[animation-timing-function]
[animation-delay] [animation-iteration-count] [animation-
direction]
[animation-fill-mode];
```

You can delete sub-properties as needed; plenty of animations will only really require *name*, *duration*, and *iteration-count*.

To add multiple animations to an element, separate them with a comma:

```
animation: fadeout 2s, rotate 2s;
```

Now let's experiment with more frames and properties.

➥ Example: Simple, Colorful Fade

First, set up **keyframes** to define each stage of the animation itself:

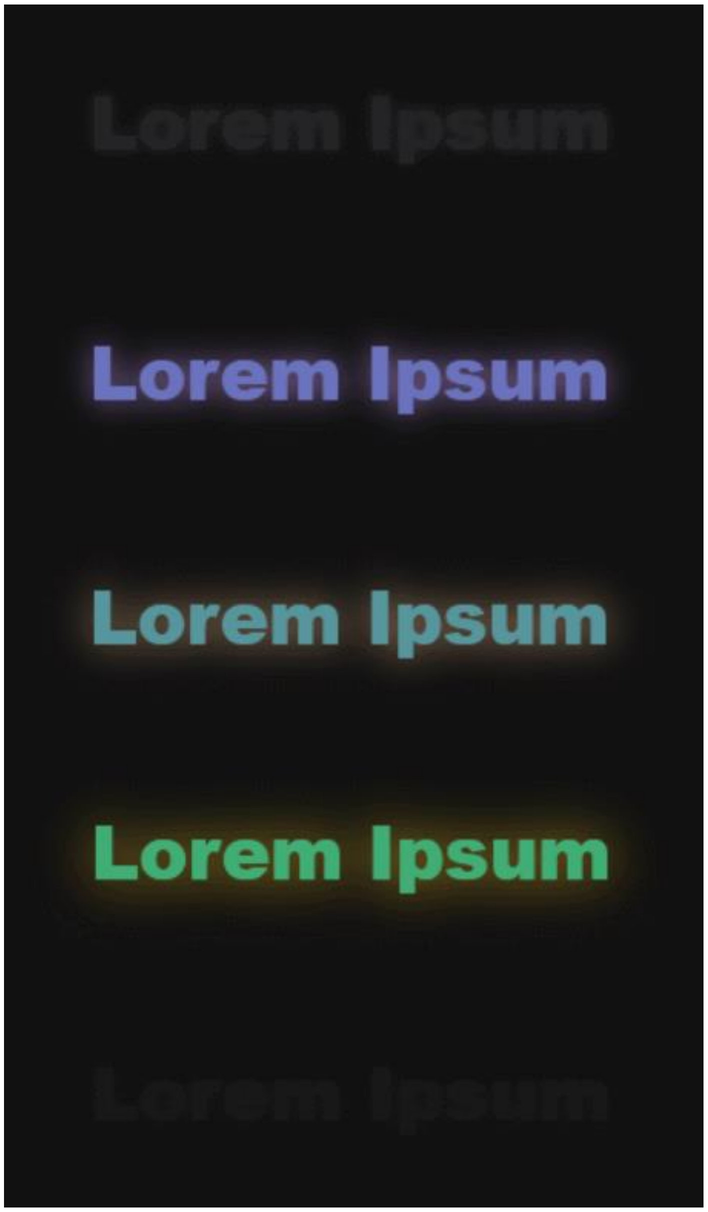
```
@keyframes textfade {
   0% {opacity: 0;}
   25% {opacity: 1; color: #775ad1; text-shadow: 0px 0px
10px #775ad1;}
   50% {color: #57ffa8; text-shadow: 0px 0px 20px #ffcd1f;}
   100% {opacity: 0;}
}
```

If you want to create an animation with **intermediate waypoints** between the start and end states, you'll use **percentage values** to set up **each stage** of the animation.

Once the keyframes are set up, configure the time, duration, and other playback features of the animation by adding it to the selector you want to animate:

```
.fade-element {
   font-size: 28px;
   font-weight: 1000;
   text-shadow: 0px 0px 5px #fff;
   animation: textfade 3s linear infinite;
}
```

The result of the code above will be an animation which **transitions from transparent to opaque** and **shifts smoothly between different text and shadow colors** over a **three second period** for an infinite number of repetitions.



➥ Ways to Experiment

The examples I've supplied today are very simple, but should be helpful to use as foundations for building more animations of your own. There are pretty much endless possibilities for what you can do here: you can make a rotating image, shaking or blinking text (while being mindful of accessibility concerns; violent movement and color shifts can be dangerous for some individuals), a link that cycles through a set of colors, and tons of other stuff with relatively easy CSS.

As you're experimenting, consider things like colors, sizes, fonts, and timing (I recommend avoiding overly-long animations, especially fades). Also think about whether you want your animations to run forwards (end on the last frame), backwards (end on the first frame), or infinitely (loop); this will heavily inform your design.

You can add animations to not only text, but buttons, backgrounds, links, :hovers, images, and pretty much any other element. There is SO MUCH cool stuff you can do with just CSS.

➥ Exercises

EXERCISE 1:

- Create an animation with **two frames**, at 0% and 100%. Decide how you want it to start and end, whether it should progress “forwards” or “backwards,” and other properties.

EXERCISE 2:

- Create an animation with **more than three frames**. This will allow you to experiment with more properties and transitions between them if you’d like.

CONCLUSION

In this section:

- **Plans for the Grimoire**
- **Closing Remarks**

➥ Plans for the Grimoire

This concludes the second installment of *The Twine® Grimoire*! Thank you so much for downloading.

It's possible that a third volume of the Grimoire will be created in the future, but the details are still being ironed out at the moment. In the meantime, thank you again for checking out the second volume! I hope it will be helpful for you as you continue to explore Twine®.

The Twine® Grimoire project is made possible by the support of my **patrons** on Patreon, who enable me to take the time to write these tutorials, revise them, and compile them into free PDFs. All patrons pledging $2 or more gain **early access** to the **first draft** of each tutorial as I write it, and also have the opportunity to make **tutorial requests** every month.

In the interest of making this resource as accessible as possible to many users, the *Grimoire* will **NEVER** have a minimum donation required to download. However, if you would like to support my work, you can do so by either donating on itch, or pledging on Patreon.

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.

➥ Closing Remarks

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 new skills in the pursuit of building 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