# Game Breakdown

The game was created using [Twine](https://twinery.org/), a free open-source tool for creating interactive non-linear stories.

Twine has multiple possible [story formats](https://twinery.org/reference/en/getting-started/basic-concepts.html#story-formats) (languages used to create a game). Library Quest is written with the **SugarCube 2.36.1 format.**

When you open the story in Twine, you will see an **interconnected web of passages** that makes up the main story of the game. To the right, **two columns of passages** function as extra information that is imported into the main story if players interact with certain links. **StoryCaption** and **StoryInit** are special passages used by SugarCube for certain types of game information.

***Languages used:*** SugarCube markup language, HTML, CSS, Javascript (primarily through SugarCube’s built-in functions)

***Important development resources:***

[Twine Cookbook](https://twinery.org/cookbook/index.html)

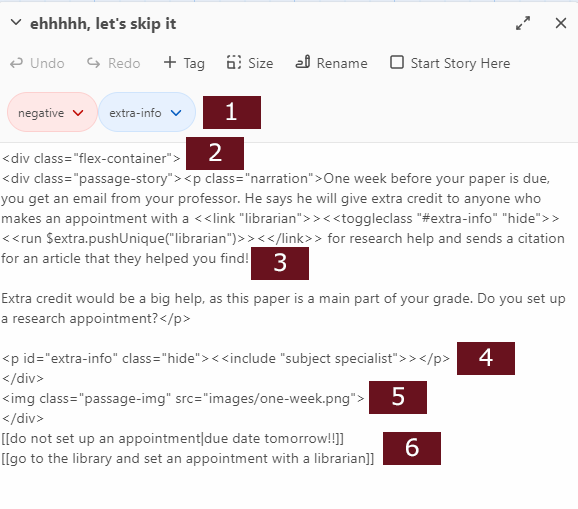
[Twine Reference](https://twinery.org/reference/en/index.html)

[Interactive Fiction Community Forums](https://intfiction.org/c/authoring/twine/46)

[SugarCube documentation](https://www.motoslave.net/sugarcube/2/docs/)

Twine also has a Discord server where you can ask for help [linked to on their website](https://twinery.org/)

# Basic Passage Breakdown



1 Passage tags - primarily used in Library Quest to track whether choices have a positive or negative effect on the outcome and whether a passage contains an extra interactive piece of text that displays information from the separate passages in the columns to the right of the interface

2 The simple HTML included in passages is mostly there to help control the structure of the page and to add ID or class selectors for CSS styling purposes. I relied on flexbox layout for flexibility and to help the game adapt to different sized screens. [A Complete Guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) is a great resource to understand how this works.

3 Special clickable text that reveals additional information. This was achieved with a little bit of Javascript filtered through SugarCube’s markup language. Very basically, the text is formatted as a clickable link. The effects of clicking the link are included between the starting <<link>> and ending <</link>> tags. See the Secret Ending section for more details about how this works.

4 This line sets up a paragraph container that is hidden by default. In this example, the container is pulling in the contents of the passage with the title “subject specialist.” It will remain hidden until the class “hide” is removed.

5 This line sets up the image included in each passage. For this game, images are stored in a folder called “images” at the same file structure level as the Twine story.

6 At the bottom of each passage are clickable links that advance the story, formatted with SugarCube’s markup language. Typically the text of the link matches the title of the passage it connects to. In some cases, the link text and the linked passage name differ, like in the first link in the example provided. (Link text → “do not set up an appointment” // Linked passage name → “due date tomorrow!!”)

# Secret Ending

It’s only possible to see the secret ending if a player reaches the Great Ending or Good Ending. The content for the ending can be found in a passage with the title **“secret ending”** in the columns to the right. The game checks whether the player has interacted with at least three of the special “extra info” elements; if not, the passage **“wrap up”** will be shown instead.

An array is used to count players’ interactions with special elements. The array (**$extra**) is created in the **StoryInit passage** with the following code:

<<silently>>

<<set $extra to [ ]>>

<</silently>>

Each time a player clicks on one of the “extra info” links in a passage, two things are triggered:

<<link "librarian">><<toggleclass "#extra-info" "hide">><<run $extra.pushUnique("librarian")>><</link>>

First, the function**toggleclass()** is run on the HTML element with the ID “extra-info.” Toggleclass() removes the existing class called “hide” so the paragraph of content will no longer be hidden.

Next, a string is added to the **$extra** array. In the example above, the string is “librarian” and it’s being added with the **pushUnique()** function. This works better than simply using the **push()** method, which would add the string to the array multiple times if a user clicks the special text more than once. **PushUnique()** will check if the string already exists in the array before adding it.

When a player reaches either the Great Ending or Good Ending, the following code checks the array and triggers the secret ending if the criteria are met:

<<silently>>

<<timed 3s>>

<<if $extra.length gte 3>>

<<toggleclass "#secret" "hide">>

<<else>>

<<toggleclass "#wrap-up" "hide">>

<</if>>

<</timed>>

<</silently>>

This code uses Javascript and is formatted with SugarCube’s markup language. After waiting 3 seconds, the code checks the length of the $extra array with the **length()** method. If the array is greater than or equal to 3 items (i.e. the player has clicked at least three special elements), the **toggleclass()** method removes the “hide” class from the element with the ID “secret” to reveal the secret ending content. If the condition is not met (the array contains less than 3 items), this code is skipped and the **toggleclass()** method instead removes the “hide” class from the element with the ID “wrap-up” to reveal alternate content.

# Questions?

If you’d like more explanation of anything or have questions, feel free to reach out to Sharona Ginsberg at [sginsbe1@umd.edu](mailto:sginsbe1@umd.edu).