Toolkit - the Making of Duck Life

What is this game?

Duck Life: Save the Pond is a game in the form of an interactive digital narrative (like a choose-your-own-adventure book) built in Twine. It was created for a course on science games at the University of Arizona. You can play the game online here: https://alyssafink.itch.io/duck-life-save-the-pond. You can also download the game (and this handy guide to how it was made) from this GitHub repository.

In this game, you are trying to convince other ducks that some very fashionable scarves are causing environmental problems, and that they all need to stop wearing them. To do this successfully, players are tasked with using a sociological framework called legitimation code theory that breaks down the different qualities that make claims to knowledge legitimate in different circumstances. Players aren't loaded with a lot of terminology, but are rather taught how to use tools from the framework. They then have the chance to talk to three different ducks in whatever order they want. After talking to all three ducks (successfully or not), players move to the end of the game, where the result is determined by how many they were able to convince.

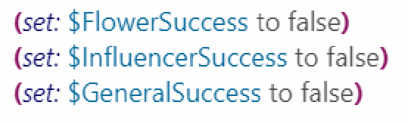
How to get started with Twine

Twine is a free, open-source tool for creating interactive, nonlinear stories. You can use it online by visiting twinery.org and clicking "Use in your browser," or you can download it for Windows, macOS, or Linux by selecting "Download" on the same site. This project uses Harlowe, which is one of several Twine story formats. This doesn't impact anything about the user interface, but rather the types of code that you use and what those things do.

To edit or look at the Twine file for this game, download the .html file from this GitHub repository. Open up in either way, then go to Library > Import. Click on the downloaded file to open it.

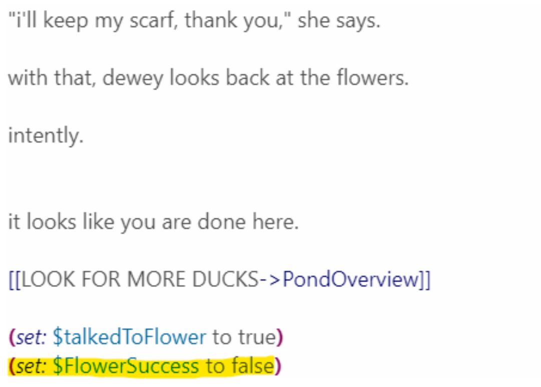
Keeping score

The player's final score comes into play at the end of the game. It is meant to total the number of ducks (0-3) that they were successful in convincing, and trigger a different ending based on that number. There are three components to setting up this type of score keeping. First, I created a boolean variable for each of the three ducks that the player is trying to convince early on in the game. It's important that these are created in a passage that the player will not be revisiting, or the command to set the value to false could erase game progress.



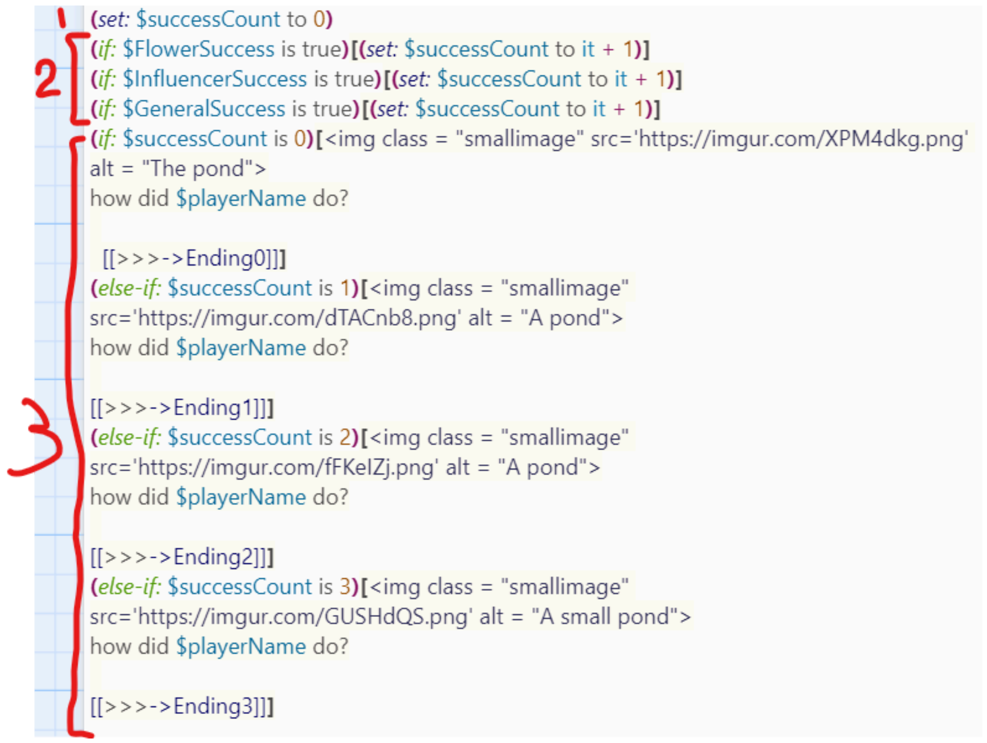
```
(set: $FlowerSuccess to false)
(set: $InfluencerSuccess to false)
(set: $GeneralSuccess to false)
```

Second, I changed the value of each boolean variable once the player reached a passage that indicated that a conversation with one of the ducks was over. If the passage indicated success, I would set the value to true. If it indicated failure, I would set it to false. The value is actually already false, but I included it anyway so that all the concluding passages had the same structure. See example below.



```
"i'll keep my scarf, thank you," she says.

with that, dewey looks back at the flowers.

intently.


it looks like you are done here.

[[LOOK FOR MORE DUCKS->PondOverview]]

(set: $talkedToFlower to true)
(set: $FlowerSuccess to false)
```

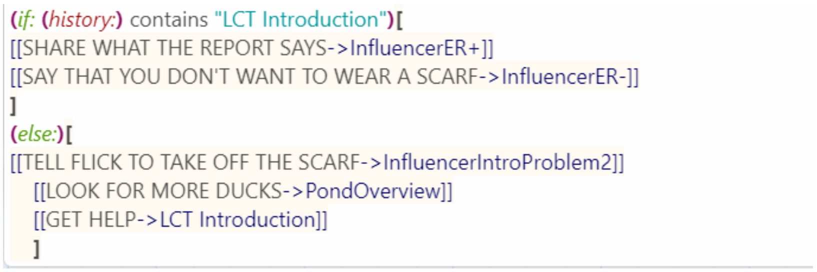
The third part is a little more complicated. After players select the option to “CALL IT A DAY,” I total up all the successes into a new variable, called $successCount. Please refer to the annotated passage below to see how this works. In section 1, I create the variable $successCount, and set its value to 0. Then, in section 2, I go through each of the duck success variables one by one and increase the success count for each one that is true/successful. And section 3 lays out a series of options. If the first option ($successCount is 1) is true, then that dialogue will be used. The dialogue includes a link to the corresponding ending passage. If the first option isn’t true, Twine will go to the next option and check if it is true. And on and on.



```
(set: $successCount to 0)
(if: $FlowerSuccess is true)[(set: $successCount to it + 1)]
(if: $InfluencerSuccess is true)[(set: $successCount to it + 1)]
(if: $GeneralSuccess is true)[(set: $successCount to it + 1)]
(if: $successCount is 0)[<img class = "smallimage" src='https://imgur.com/XPM4dkg.png'
alt = "The pond">
how did $playerName do?

 [[>>>->Ending0]]]
(else-if: $successCount is 1)[<img class = "smallimage"
src='https://imgur.com/dTACnb8.png' alt = "A pond">
how did $playerName do?

[[>>>->Ending1]]]
(else-if: $successCount is 2)[<img class = "smallimage"
src='https://imgur.com/fFKelZj.png' alt = "A pond">
how did $playerName do?

[[>>>->Ending2]]]
(else-if: $successCount is 3)[<img class = "smallimage"
src='https://imgur.com/GUSHdQS.png' alt = "A small pond">
how did $playerName do?

[[>>>->Ending3]]]
```

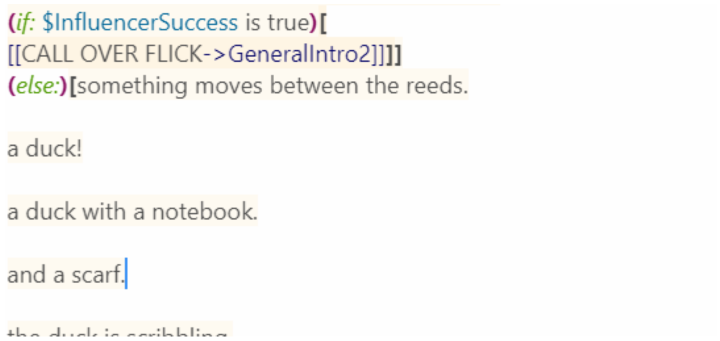
Triggering new options

One of the key parts of the game is that players can do some things that create new choices for them in other places. For example, once players talk to the special snail, they have new dialogue options with each of the ducks that allow them to move forward in conversation. This is also used so that, if the player successfully convinces a certain duck, they have the option to call them over for help in a different conversation. To make this work, I used some if/else statements that control which choices the player has.



```
(if: (history:) contains "LCT Introduction")[
[[SHARE WHAT THE REPORT SAYS->InfluencerER+]]
[[SAY THAT YOU DON'T WANT TO WEAR A SCARF->InfluencerER-]]
]
(else:)[
[[TELL FLICK TO TAKE OFF THE SCARF->InfluencerIntroProblem2]]
   [[LOOK FOR MORE DUCKS->PondOverview]]
   [[GET HELP->LCT Introduction]]
   ]
```

The code above means that, once players have visited the first passage of the special snail conversation, they will have two dialogue options. But if their history does not contain that passage yet, they have three different dialogue options.

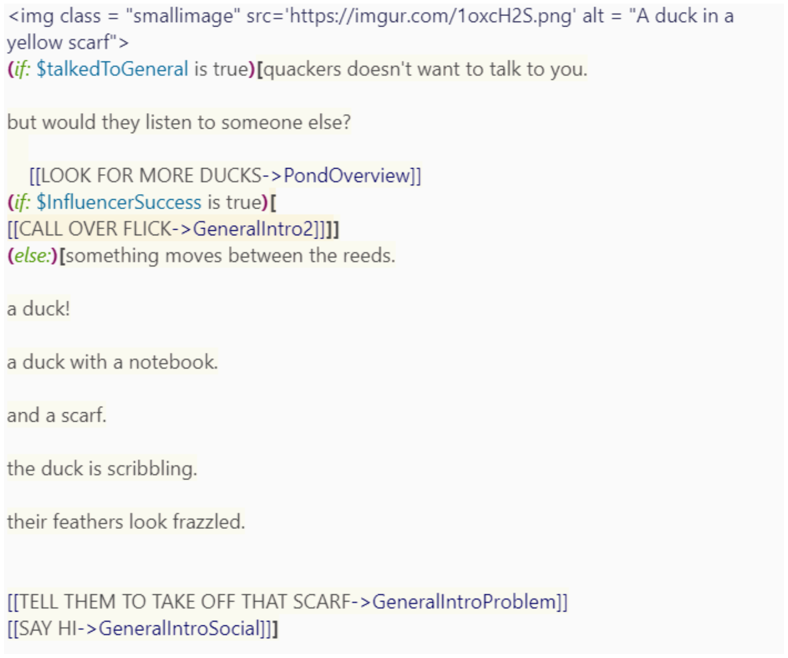


```
(if: $InfluencerSuccess is true)[
[[CALL OVER FLICK->GeneralIntro2]]]
(else:)[something moves between the reeds.

a duck!

a duck with a notebook.

and a scarf.
```

I used the if/else statement differently here so that a new option opens if the player has success with a different duck. That's the (if: $InfluencerSuccess is true) part.

New dialogue after completion

I wanted the game to feel like the players were actually in this pond world, with some degree of freedom of movement. To create this feeling, I had an overview page that players could access each of the ducks from, and that they returned to after each conversation. I wanted the players to be able to revisit the areas of the pond that the ducks were in, even after they had finished the conversation. To do this, I used more if/else statements. See below for an example and explanation of how it works.

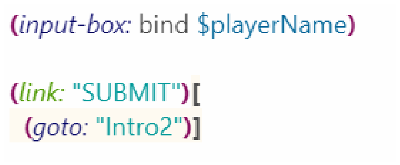


```
<img class = "smallimage" src='https://imgur.com/1oxcH2S.png' alt = "A duck in a
yellow scarf">
(if: $talkedToGeneral is true)[quackers doesn't want to talk to you.

but would they listen to someone else?

   [[LOOK FOR MORE DUCKS->PondOverview]]
(if: $InfluencerSuccess is true)[
[[CALL OVER FLICK->GeneralIntro2]]]]
(else:)[something moves between the reeds.

a duck!

a duck with a notebook.

and a scarf.

the duck is scribbling.

their feathers look frazzled.


[[TELL THEM TO TAKE OFF THAT SCARF->GeneralIntroProblem]]
[[SAY HI->GeneralIntroSocial]]]
```

The image link is on the top, and will show up no matter what. If the player has completed a conversation with this duck, everything within the set of brackets that immediately follows the if statement will show up. Otherwise, everything within the set of brackets immediately following the else statement will show up.

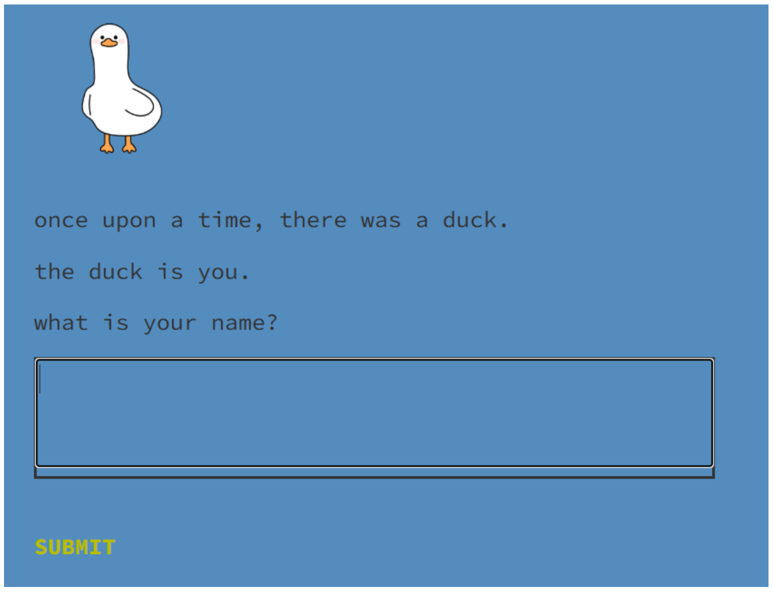
Custom names

I decided to let players set their own character name, which the other characters could use in dialogue throughout the story. To do this, I used the following code on the page where players could choose their name:



```
(input-box: bind $playerName)

(link: "SUBMIT")[
  (goto: "Intro2")]
```

This is what that looks like from the front end:

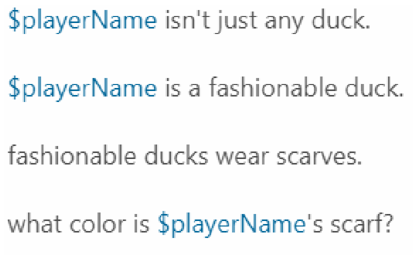


once upon a time, there was a duck.

the duck is you.

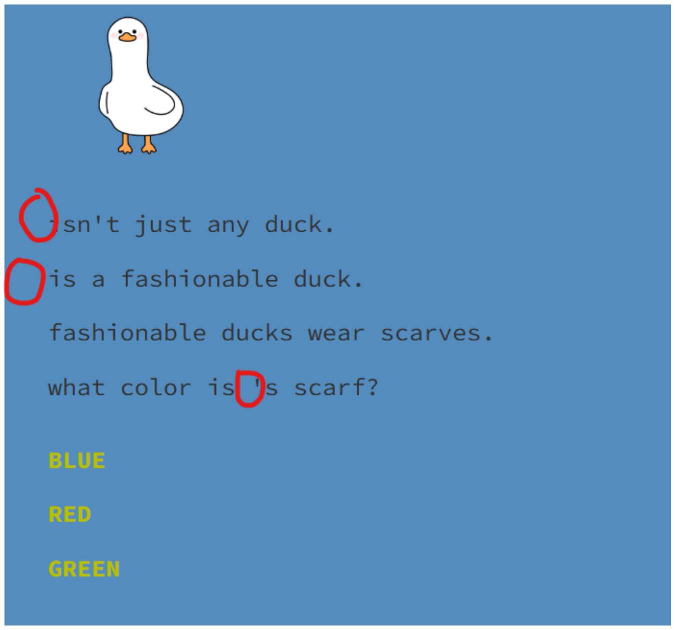
what is your name?

Whatever players enter in this box is then registered as $playerName. So if I use $playerName as text on another passage it will show up as whatever players have entered (see code below).



```
$playerName isn't just any duck.

$playerName is a fashionable duck.

fashionable ducks wear scarves.

what color is $playerName's scarf?
```

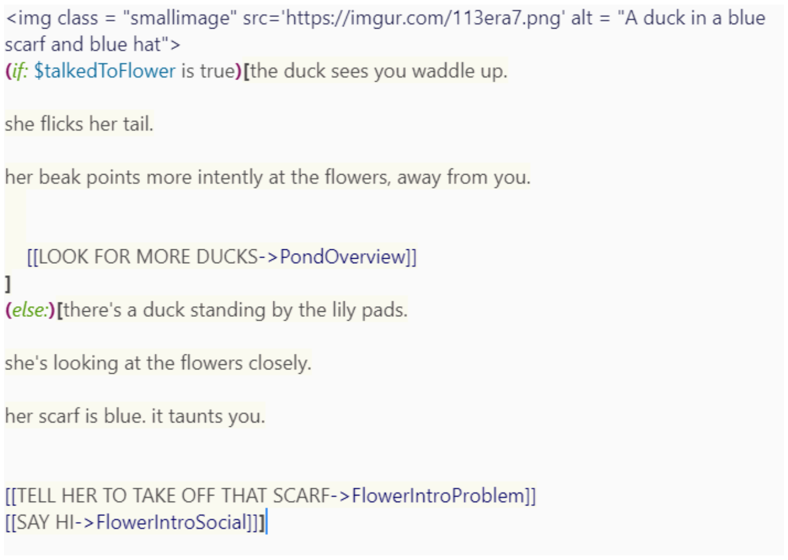
I couldn't figure out a way to require players to enter a name, though. If they click SUBMIT without entering a name, it shows up as blank anytime $playerName is used (see example below - red circles indicate where a name should be.



```
isn't just any duck.
is a fashionable duck.
fashionable ducks wear scarves.
what color is 's scarf?

BLUE
RED
GREEN
```

Keeping track of variable dialogue

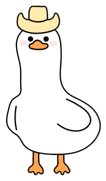
Each passage started to get pretty cluttered in Twine, between variable dialogue options, keeping score, and images. You can see in the image below how that might make it difficult to keep track of the story.



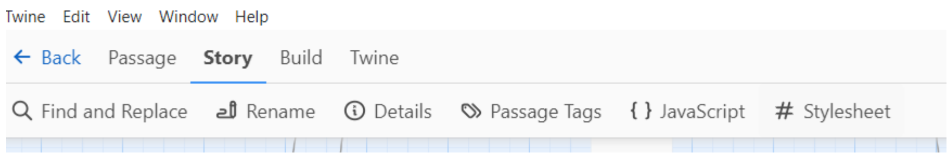
```
<img class = "smallimage" src='https://imgur.com/113era7.png' alt = "A duck in a blue
scarf and blue hat">
(if: $talkedToFlower is true)[the duck sees you waddle up.

she flicks her tail.

her beak points more intently at the flowers, away from you.


    [[LOOK FOR MORE DUCKS->PondOverview]]
]
(else:)[there's a duck standing by the lily pads.

she's looking at the flowers closely.

her scarf is blue. it taunts you.


[[TELL HER TO TAKE OFF THAT SCARF->FlowerIntroProblem]]
[[SAY HI->FlowerIntroSocial]]]
```

If I were to start this game over again, I would hold off on adding all those fancy bits and pieces until after the story was more complete to make it easier to keep track of.

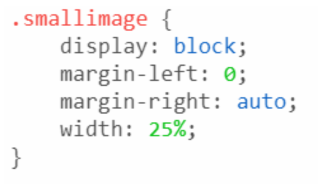
Adding artwork



I created my artwork (like the little duck pictured above) in Rhinoceros, which is a 3D design tool. I exported each drawing separately, as a PNG file of the same size. I used PNGs because they are able to have a transparent background, so there wouldn't be a white box behind each graphic when it was laid over the blue background. Unfortunately, my Rhino PNGs still had the white background, and I'm not sure why. I ended up opening each file in Photoshop, deleting the background, then saving over the original file and that worked fine. I used files that were all the same size (dimensions) so that I would only have to set one image type in Twine and they would all show up the same way.


Twine Edit View Window Help
← Back Passage Story Build Twine
Find and Replace Rename Details Passage Tags { } JavaScript # Stylesheet

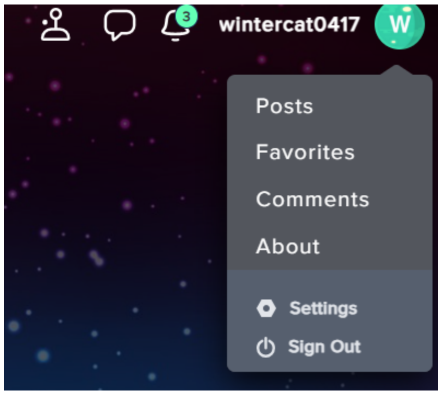
The dimensions of the images are controlled from the stylesheet (see the image above to find where the stylesheet is). I used the following CSS code, which I borrowed from another Twine game that I had downloaded.



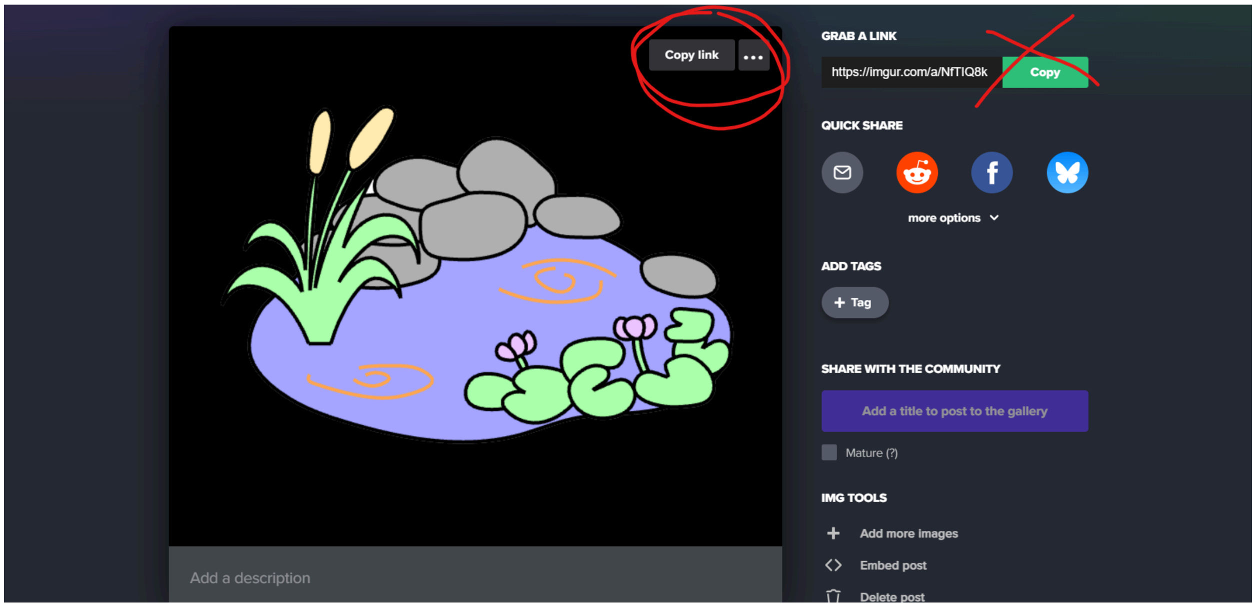
```
.smallimage {
    display: block;
    margin-left: 0;
    margin-right: auto;
    width: 25%;
}
```

I'm honestly not sure what display: block does. To have the image left-aligned, I set margin-left to 0 and margin-right to auto. I think that, if it weren't set to auto, it might stretch the image or something. I messed around with the width percentage until I found one that looked right.

The trickiest part of adding artwork was making it so that everyone could see the artwork. Twine files can't actually hold artwork - it needs to be linked in. I ended up doing this using imgur. I made a free account. Once I was logged in, I went to my "posts" section (see image below).



I then dragged and dropped the PNG files (one-by-one) onto the screen to upload them. After that, you need to get a link to the image that you can use in the Twine file. To get this link, click on a post/image. Then hover your mouse over the image, and click the “copy link” box that pops up. There is also a green “copy” button off to the side - I found that this produced a different link that didn’t actually work in the Twine file, so I recommend doing it the way I have circled in the image below.



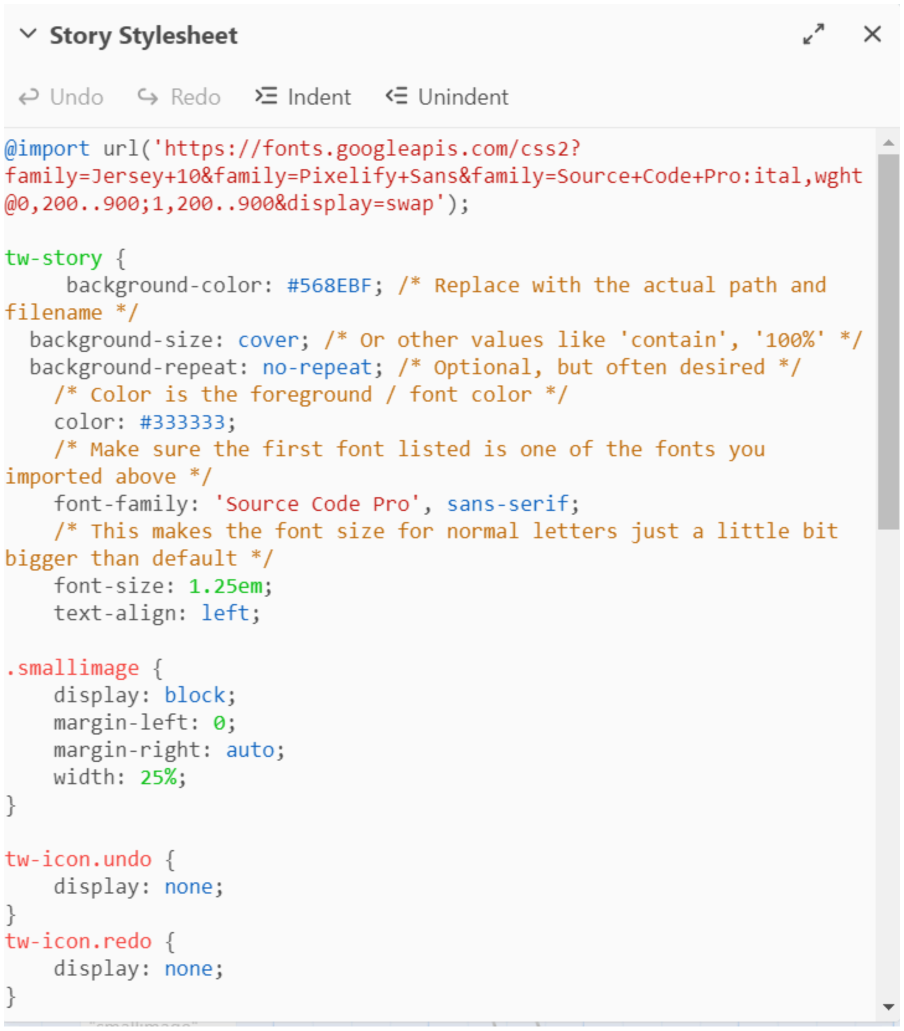
To insert the image in Twine, I used this snippet of code in each passage that I wanted it to show up in:

```
<img class = "smallimage" src='https://imgur.com/idLdDbH.png' alt = "A Duck">
```

The brackets and “img” indicate that it is an image. The “class” links this image to the characteristics assigned in the stylesheet. “src” is the link to the image on imgur. I found that the links worked when I used single quotations (‘ ‘) instead of double quotations (“ “). I also had to add .png to the end of the url to get it to work. “alt” is the text that a screen reader would read aloud.

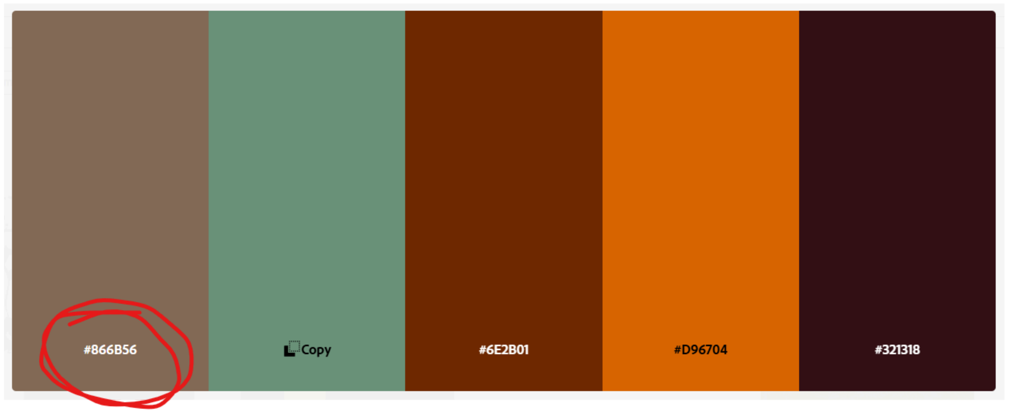
Graphic design in Twine

The graphic design elements of this game (fonts, colors, etc.) are controlled from the stylesheet and written in CSS. To customize mine, I copied this template that I found online into the stylesheet. You can see some of it below. This template is really nice because it includes comments (The things enclosed by /*   */) that explain what each piece does and how you can customize it.



```
@import url('https://fonts.googleapis.com/css2?
family=Jersey+10&family=Pixelify+Sans&family=Source+Code+Pro:ital,wght
@0,200..900;1,200..900&display=swap');

tw-story {
     background-color: #568EBF; /* Replace with the actual path and
filename */
  background-size: cover; /* Or other values like 'contain', '100%' */
  background-repeat: no-repeat; /* Optional, but often desired */
    /* Color is the foreground / font color */
    color: #333333;
    /* Make sure the first font listed is one of the fonts you
imported above */
    font-family: 'Source Code Pro', sans-serif;
    /* This makes the font size for normal letters just a little bit
bigger than default */
    font-size: 1.25em;
    text-align: left;

.smallimage {
    display: block;
    margin-left: 0;
    margin-right: auto;
    width: 25%;
}

tw-icon.undo {
    display: none;
}
tw-icon.redo {
    display: none;
}
```

One of the things that I changed was the colors. To make a good color palette that fit the theme and worked well together, I used Adobe Color. In Adobe Color, you can search for keywords (like pond) and it will pull up some color palettes that you can choose from. The color palettes contain hex codes (pictured below circled in red - a string of six numbers/letters) that you can type into the stylesheet to use. For example, in the image above, I could replace background-color: #568EBF with #866B56 to get a different color.



Publishing online

This game is published on itch.io so people do not have to download it in order to play it. I followed these instructions to do it - they are very comprehensive, and it was easy/quick to do this.