

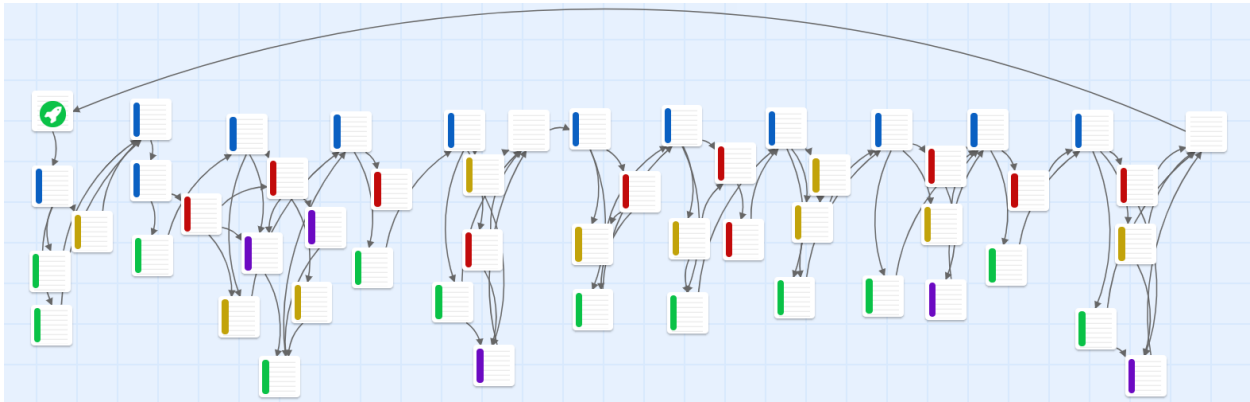
FAST TRACKING YOUR BRANCHING SCENARIOS

Learning Solutions Conference

April 19, 2022

Christy Tucker

christy@christytuckerlearning.com



Download files	4
Questions to Ask	4
Identify the need.....	4
Identify the desired behavior	4
Gather details about the behavior	5
Uncover mistakes and problems.....	5
Identify the consequences.....	6
Project Ideas	7
Work project.....	7
Your expertise	7
Sample project ideas	7
Activity 1: Objective and Desired Behavior	8
4 Cs in Scenarios	9
Characters.....	9
Context	9
Challenge	9
Consequences.....	10
Other models	10
Activity 2: 4Cs	11
Main Character (Protagonist)	11
Other Characters.....	12
Context	12
Challenges.....	13
Consequences.....	13
How to Use Twine	14
Download and install Twine.....	14
Caution for the browser-based version	15
Create a story in Twine.....	15

Add and link passages.....	17
Tags in Twine	19
Story and proofing formats.....	21
Test your prototype.....	22
Activity 3: Intro and First Decision.....	24
Branching Structure.....	25
Time Cave.....	25
Limited Branching or Gauntlet	26
Branch and Bottleneck	27
Activity 4: Branching Structure.....	30
Visual Design in Twine.....	31
Harlowe Toolbar	31
Formatting text.....	31
Sample visual design in Twine	34
Page Settings.....	34
Passage settings.....	36
Link settings	37
Images	38
Sample enchant macros.....	39
Hide the sidebar	40
Publish a Twine Story.....	41
Export to Plain Text	41
Install and Use Entwee	42
Example entwee plain text format.....	43
Activity 5: Finish Your Branching Scenario.....	43

Download files

Download the files for this workshop, including this workbook:

ctuckerlearning.com/fast

Questions to Ask

Normally, you would ask these questions of SMEs and stakeholders during your analysis. For this workshop, you'll identify a training need that you can write about on your own without input from a SME. Use this list to think about the problem you can address with your branching scenario. Keep this list of questions for reference for the future.

Identify the need

Start with a bit of needs assessment.¹ It doesn't need to be extensive, but this helps with the scoping as well as pointing to where you might follow up for stories from the SME.

1. What is the problem?
2. How will you measure success?
3. What are people doing that they shouldn't?
4. When does the problem happen?
5. Where is this a problem?
6. How big is the problem?
7. What else should I have asked?

Identify the desired behavior

This clarifies what you identified during the needs assessment.

- What do you want people to do differently as a result of this training?
- If this training is successful, what will that look like in their day-to-day work?
- What is happening right now that shouldn't be happening?

¹ Adapted from "How to Conduct a Lightning-Fast Needs Assessment Clients Will Love" by Sardek Love
<https://www.td.org/insights/how-to-conduct-a-lightning-fast-needs-assessment-clients-will-love>

- What isn't happening currently that you want to happen?

Gather details about the behavior

Keep drilling down to get more details on that desired behavior. SMEs will often start with general ideas like "provide quality customer service" or "improve communication." That's not enough, especially for a branching scenario. You need to continue to ask follow-up questions until you get more concrete information.

- If you took a photo or video of that behavior, what would it look like?
- What would it sound like in a conversation?
- Can you give me an example of how someone used this technique successfully? What were they able to accomplish by doing it right?
- Tell me about a time when you saw this happen in a real situation.
- Walk me through this process. What would it look like if they did everything perfectly?
- Are there any exceptions or edge cases where you might handle it differently?

Uncover mistakes and problems

Besides talking to the SMEs, it's also helpful if you can talk to actual users (or people who have learned this skill recently). They will often identify other issues than what the SMEs might think are the primary problems.

- What are the common mistakes people make?
- Tell me about a time where someone got stuck in this process.
- Tell me more about that mistake. What do you think is going through people's heads when they do that?
- What does it look like when they make this mistake?
- What problems do users report?
- What makes this hard for newbies?
- How do you know when it hasn't been done correctly?
- How are people doing it wrong currently?

- What is most confusing about this to people doing it for the first time?
- What are the most common issues reported to the help desk?

Identify the consequences

In addition to the mistakes and problems, you need to identify the consequences. You can also flip these questions around to get the consequences and effects of good decisions. For example, "What results do you see that tell you that things went well?"

- What happens if someone makes that mistake?
- Can you give me an example of a time when someone did this wrong? What happened because of this mistake?
- In that situation, what happened next?
- If someone [makes a specific mistake], what happens? What's the effect?
- What does that consequence look like in practice?
- What results do you see that tell you something went wrong?

Project Ideas

Think about a performance problem or organizational need that you can address through a branching scenario.

Work project

Do you have a current project for your work that you can use? That's great! You can use that here.

Your expertise

If you don't have a specific work project in mind, think about your own expertise and background. Is there something you know enough about to be a SME?

Sample project ideas

You can also use one of these sample project ideas.

New managers

- Conflict resolution
- Providing constructive feedback
- Project management

Objection handling

- Cost objections
- Concerns about difficulty of use
- Reservations about quality

Troubleshooting

- Spreadsheet problems
- Machinery errors
- New employee account doesn't have needed access

Activity 1: Objective and Desired Behavior

What is the objective of your branching scenario? Focus on a single objective with one-specific action or behavior.

What does the desired behavior look or sound like?

At a high level, what are the steps if someone does the process or behavior perfectly?
What is the "ideal path" in the branching scenario?

4 Cs in Scenarios

When you create branching scenarios, plan these four elements: characters, context, challenge, and consequences.

Characters

The main character of your scenario (also called the protagonist) who drives the action should generally be someone similar to your learners. Even if the main character isn't named and the scenario is in second person (e.g., What do **you** do next?), the role of that character should be familiar to your learners. Give your main character a goal that aligns to the learning objectives and that your learners share.

The other people your main character interacts with should be typical and mostly realistic, with perhaps a little exaggeration. If you're doing customer service training, think about the different types of customers employees interact with. If you're creating manager training, the other characters might be employees and coworkers.

Context

The context is the background or environment in which the scenario happens. Think of it as the setting in a story. The context can be portrayed through language, images, and other cues.

Your learners want to know that their training is relevant to them and the work they do. Aligning the context to their work environment helps them see themselves in the story and apply what they learn.

Challenge

Your characters face challenges in the scenario. Those are the points where learners have to make a decision or take an action. The challenges are where the learning happens. Think about the frequent obstacles: faulty technology, impatient customers, or a limited budget.

Common mistakes are good challenges to include. If sales associates often forget to provide a recommendation at a specific point in the sales process, include that point in the scenario. Give learners a choice to make a recommendation or not.

You might also include challenges that happen less often but are critical to address correctly. Sales associates won't often have to deal with a customer so angry that they threaten violence, but it's important to know how to handle that volatile situation.

Consequences

Especially in branching scenarios, the feedback is generally part of the scenario rather than something you just tell them. A customer gets angry, a patient refuses to follow your recommendations, the technology continues to malfunction, or you run out of budget two months before your project is finished.

Show learners the consequences of their mistakes rather than just telling them. You might also provide coaching or instructional feedback, especially for novice learners, but don't neglect the consequences of their actions.

Other models

We all stand on the shoulders of giants who have shared their ideas before us. My list of the 4Cs overlaps with concepts in Michael Allen's CCAF (Context, Challenge, Activity, Feedback) model.² I do prefer Consequences to Feedback, as designers too often assume feedback is only verbal (although Allen explains the difference in his great books). Tom Kuhlmann's "Challenge-Choice-Consequence" model³ also has similarities to this model of 4Cs.

² <https://www.alleninteractions.com/services/custom-learning/ccaf/elearning-instructional-design>

³ <https://blogs.articulate.com/rapid-elearning/build-branched-e-learning-scenarios-in-three-simple-steps/>

Activity 2: 4Cs

Plan your scenario in more detail with the characters, context, challenges, and consequences. Because you don't have access to a SME, you may need to use your imagination for some details.

Main Character (Protagonist)

Who is your main character? Make sure their goals align with your objective.

Personal details	Goals and motivation
Job, role, work environment	Problems, challenges, worries, fears

Other Characters

Who are the other characters in the scenario? Use as many rows as you need (or skip this if your scenario only has a single character).

Name	Role	Details

Context

How could you set the context for your scenario? You may not use all of these ideas; just brainstorm for now.

Images	
Language	
Other ideas	

Challenges

What are the challenges your character faces in your scenario? What mistakes could your main character make?

Challenges

Consequences

What are some of the consequences that could happen in the scenario?

Consequences

How to Use Twine

Download and install Twine

You can download and install Twine from twinery.org. If your workplace doesn't allow downloads, try the online version instead.

Note for Mac users: you may see a warning that the Twine app wasn't opened because it's from an unknown source. Select **Open anyway** to launch it. If you're still having trouble, try changing your OS settings to allow unsigned apps.

The first time you open the application, click **Tell Me More** for help. Read the information and click **OK** for each part until you finish the introduction. The online guide⁴ provides documentation.



Hi!

Twine is an open-source tool for telling interactive, nonlinear stories. There are a few things you should know before you get started.

[Tell Me More](#)

[Skip](#)

New here?



If you've never used Twine before, then welcome! The [Twine 2 Guide](#) and the official wiki in general, are a great place to learn. Keep in mind that some articles on the wiki were written for Twine 1, which is a little bit different than this version. But most of the concepts are the same.

If you have used Twine 1 before, the guide also has details on what has changed in this version. Chief among them is a new default story format, Harlowe. But if you find you prefer the Twine 1 scripting syntax, try using SugarCube instead.

[OK](#)

⁴ <http://twinery.org/wiki/twine2:guide>

Caution for the browser-based version

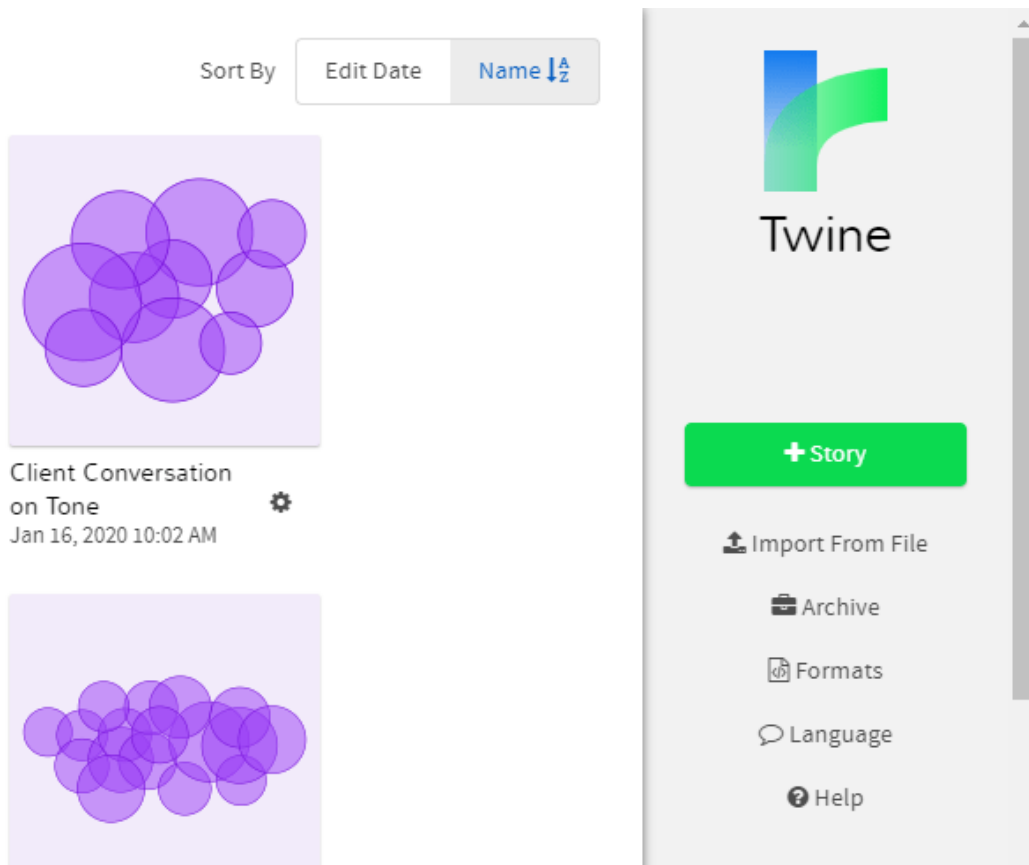
If you can install Twine locally, I strongly recommend it. While the browser-based version of Twine will work for all of the activities in this workshop, your work will **only be saved in your browser**. If you clear the saved data from your browser, your story will be gone, likely with no way of recovery.

Therefore, if you use the online version of Twine, please use the **Archive** button regularly. This will save a local copy of your work.

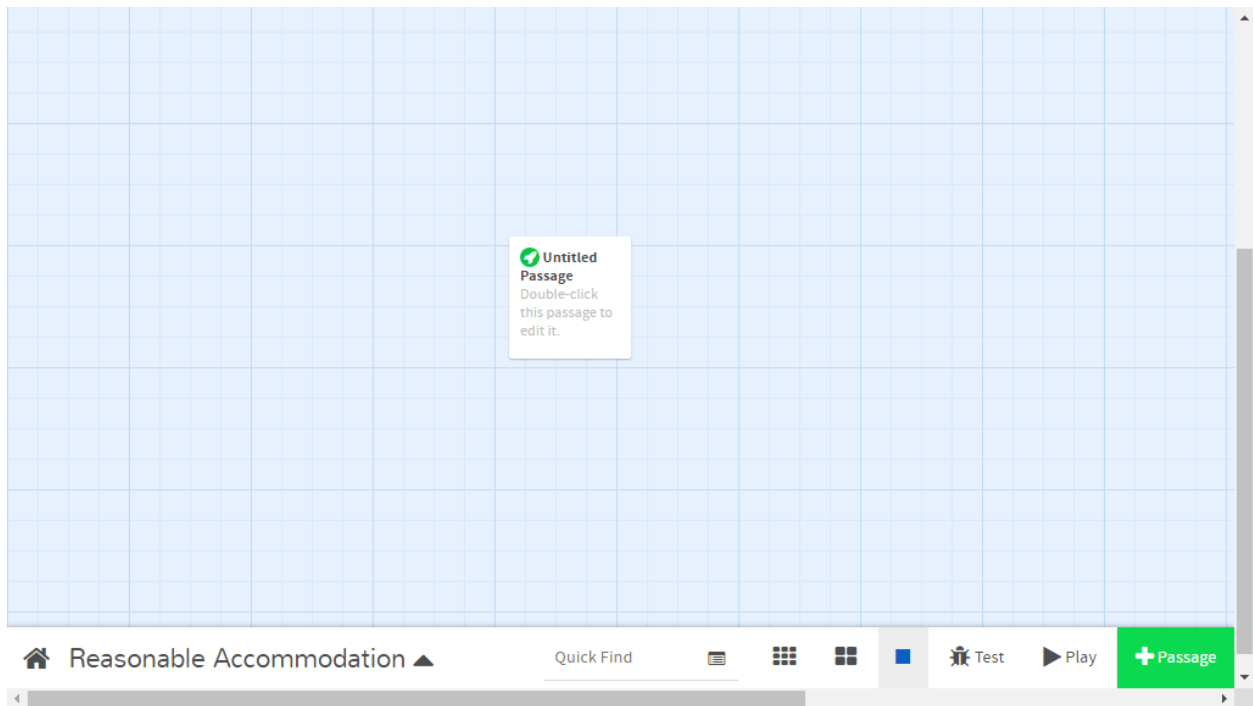


Create a story in Twine

Click the +Story button to create a new file. (The screenshot below shows some of the existing stories, but this will be empty the first time you use Twine.)



Give your story a name and open it up. You'll see a single Untitled Passage in the center to start.



Double click the Untitled Passage to open it.



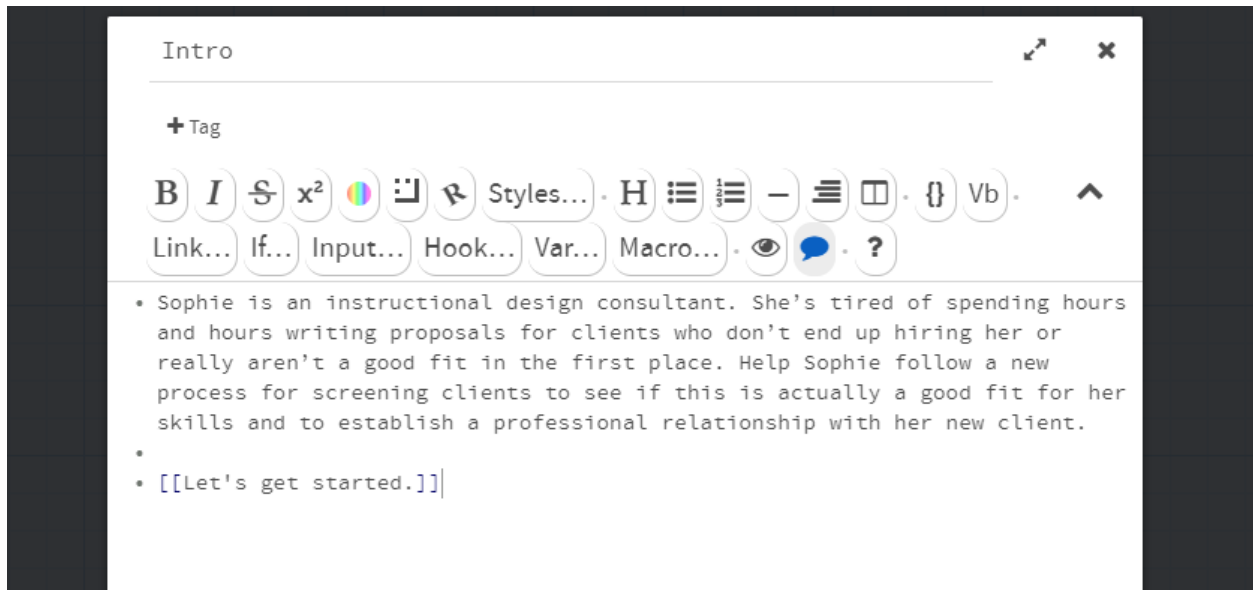
Enter a title for your passage. Enter your first text (the introduction to the scenario).

Add and link passages

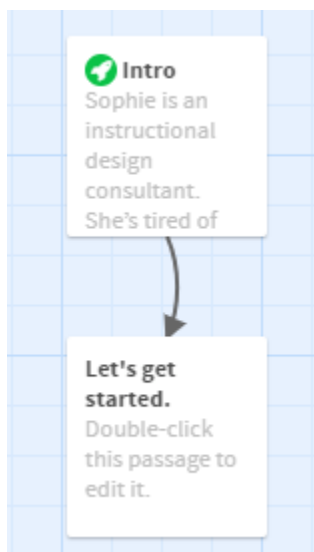
The power of Twine for branching scenarios is in the simplicity of creating additional passages with links between them. Unlike many other tools, Twine is built for nonlinear structures.

To create a link to a new passage, just type double brackets around the text of the link.

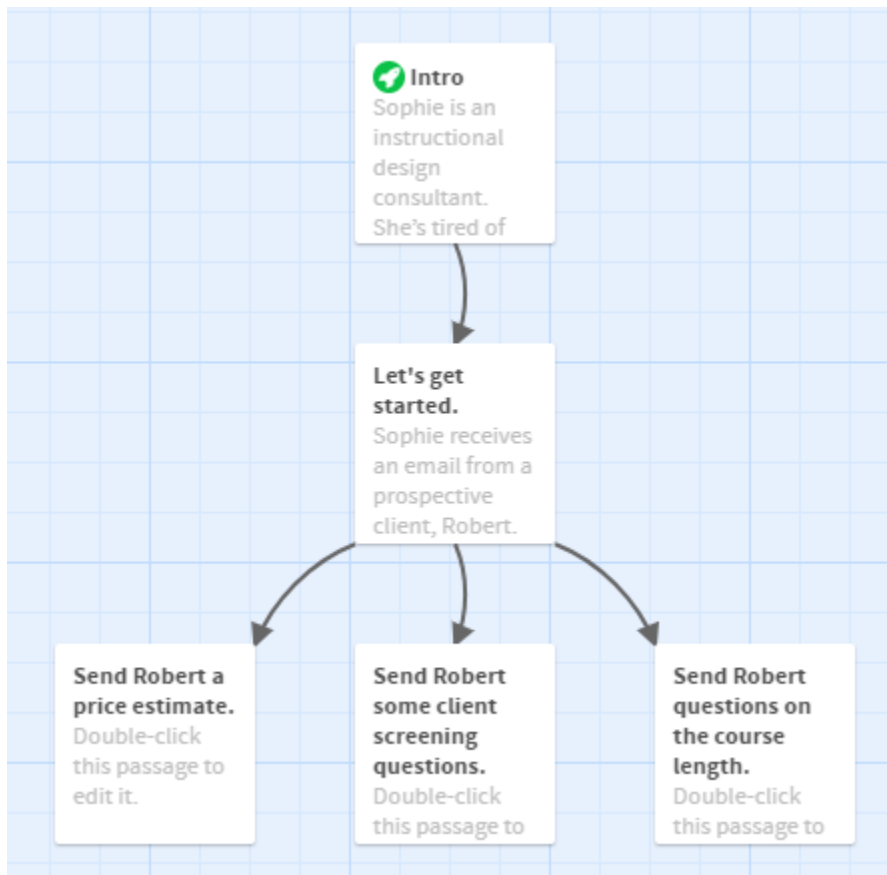
In the screenshot below, `[[Let's get started.]]` is the new link.



When you close the first passage, you'll see a new passage with a link from the first passage.



Continue this process to add a question with multiple options. Adding multiple links creates multiple new passages.



Different link text and passage name

The text shown in the link can be different from the passage name. You can use an arrow -> or a vertical pipe | to separate the displayed text from the passage name.

```
[[Try again->Let's get started]]
```

```
[[Try again|Let's get started]]
```

Both of the above links will display "Try again," but link to the passage "Let's get started."

Multiple links to a single passage

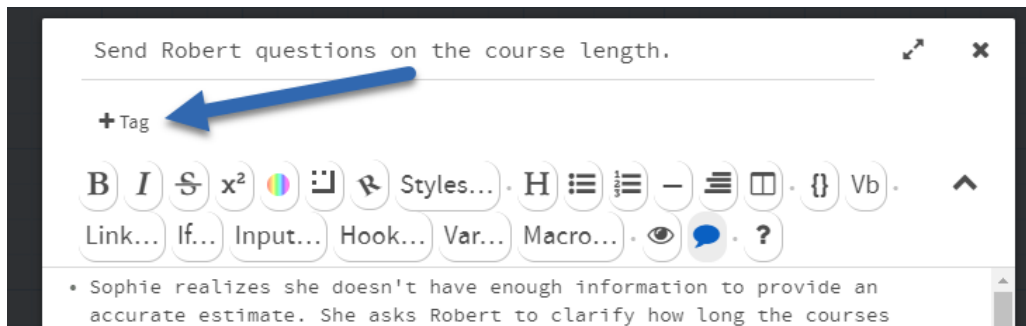
To link to a single passage more than once (like returning to the beginning), make sure your link target text exactly matches the passage name. Any difference (including punctuation or capitalization) will create a new passage.

Tags in Twine

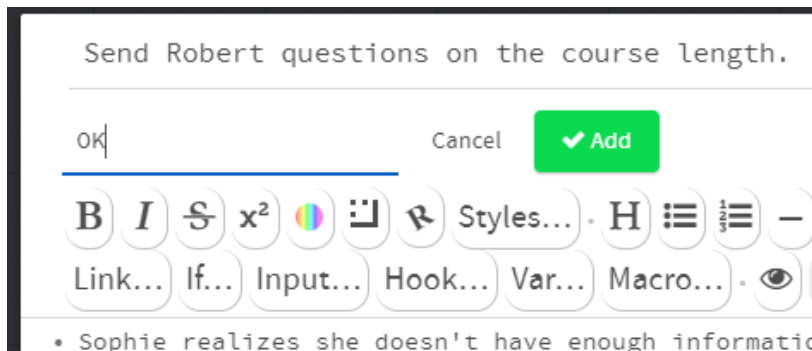
You can add tags to passages. I often tag passages as "Good," "Bad," and "OK," making it easy to see how choices are classified. You can also create tags for "Question" or "Info," depending on the structure and what's useful to you.

Tags may also be used for adding functionality like formatting or controlling variables, but you usually won't need those for a prototype.

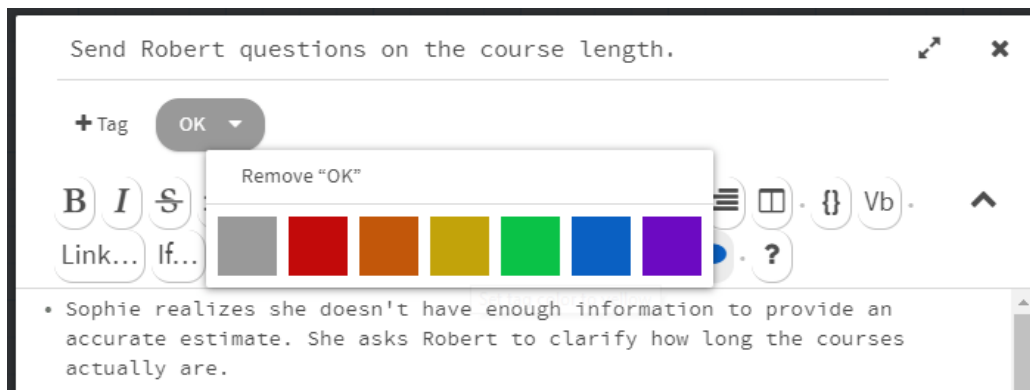
Select the **Tag** button.



Type the text you want for the tag. Select **Add**.



You can assign a color to each tag.



Story and proofing formats

Twine works a little differently from other elearning tools you might be familiar with. Tools like Storyline, Rise, and Captivate are mostly WYSIWYG (what you see is what you get). You work in a visual environment, and you change the layout and appearance directly along with the content.

Twine separates the content from the presentation, like using HTML to build a website. You work directly with the content, and then you apply a format on top of it. It's similar to creating a PowerPoint presentation on blank white slides with black text, and then applying a theme to automatically reformat the entire presentation.

Advantages

This gives us a couple of advantages for branching scenarios. First, when you're writing, you can focus just on the writing and the relationships between passages. You're not splitting your attention between writing and visual design.

Second, you can change the format for the entire story at once. That means you can create a basic design quickly, without having to design each and every passage.

Third, you can create different versions of your story by using different formats. For example, you can have an interactive version of the story for testing, but also a plain text export for reviewing and editing.

Story formats

Story formats control how the story is displayed. They also control functionality, so if you're using variables, macros, or more advanced features, it's best to pick a format at the beginning of development so you don't break anything by changing.

Harlowe

The default format for Twine is Harlowe. The Harlowe story format for Twine provides a number of built-in features to make it easier to enhance your stories, even if you have no experience with HTML, CSS, or Javascript. The toolbar makes these features easier to access, providing simple ways to format text, set conditional states, or manage variables. However, Harlowe has so many options that it has its own learning curve.

Consult the full documentation for Harlowe⁵ for details on what's possible. Note that the official documentation assumes some prior experience and background, rather than

⁵ <https://twine2.neocities.org/>

being a step-by-step guide (which is why I'm giving you some more detailed instructions here to get started).

All of the instructions in this workshop are based on Harlowe unless otherwise noted.

Sugarcube and Snowman

If you are comfortable working in CSS and Javascript, you may find Harlowe too limiting. Sugarcube has more flexibility if you're comfortable with programming. Snowman is another story format that relies heavily on Javascript knowledge.

While the directions in this workshop will all use Harlowe, you're welcome to use Sugarcube or Snowman if you have the skills to take advantage of their power.

Other formats

Twine also has a number of specialized story formats with different purposes and features. For example, botscripTEN and Trialogue are designed for creating chat simulations. Other Twine formats are meant for game design, with inventory and health tracking. While you probably don't need any of these to create your first branching scenario, you can check the catalog of Twine formats⁶ for inspiration.

Proofing formats

Proofing formats are different from story formats. While story formats are designed for end users, proofing formats are designed for developers and reviewers.

One way I commonly use a proofing format is for sending a text-only version of a story for review. You can put this into a Word document so reviewers can use Track Changes.

Test your prototype

To just walk through your entire scenario, select **Play** in the lower right. Play mode is usually fine for basic branching scenarios.

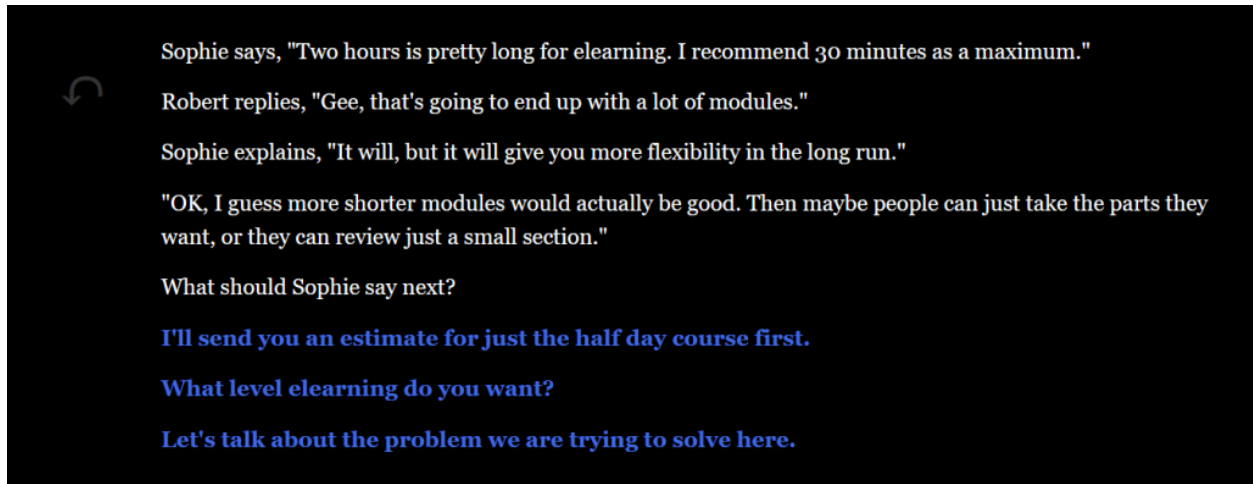
Select **Test** if you want to use debug mode to view where links go (very useful if the link text is different from the passage name).



⁶ <http://mcdemarco.net/tools/hyperfic/twine/catalog/>

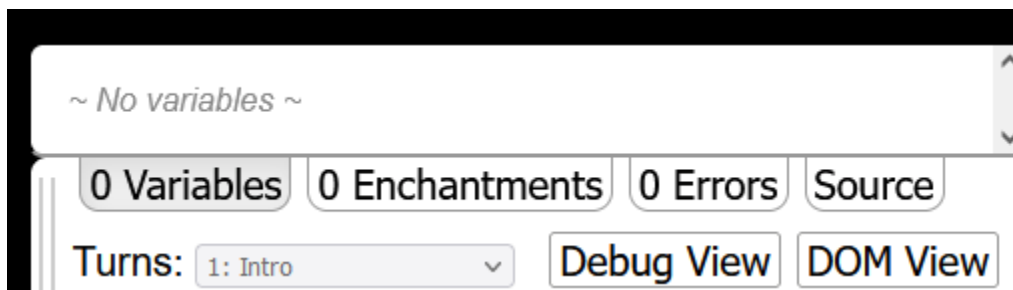
Play mode

In Play mode, you will see a passage with text links, like this. By default, Twine uses a dark background and light text. We'll change that later.



Test mode

Test mode will initially look the same as Play, but with additional information on variables and errors in the lower right. In test mode, you can turn on Debug view via a button in the lower right.



Activity 3: Intro and First Decision

In this activity, you will create the introduction and first decision point of your scenario in Twine.

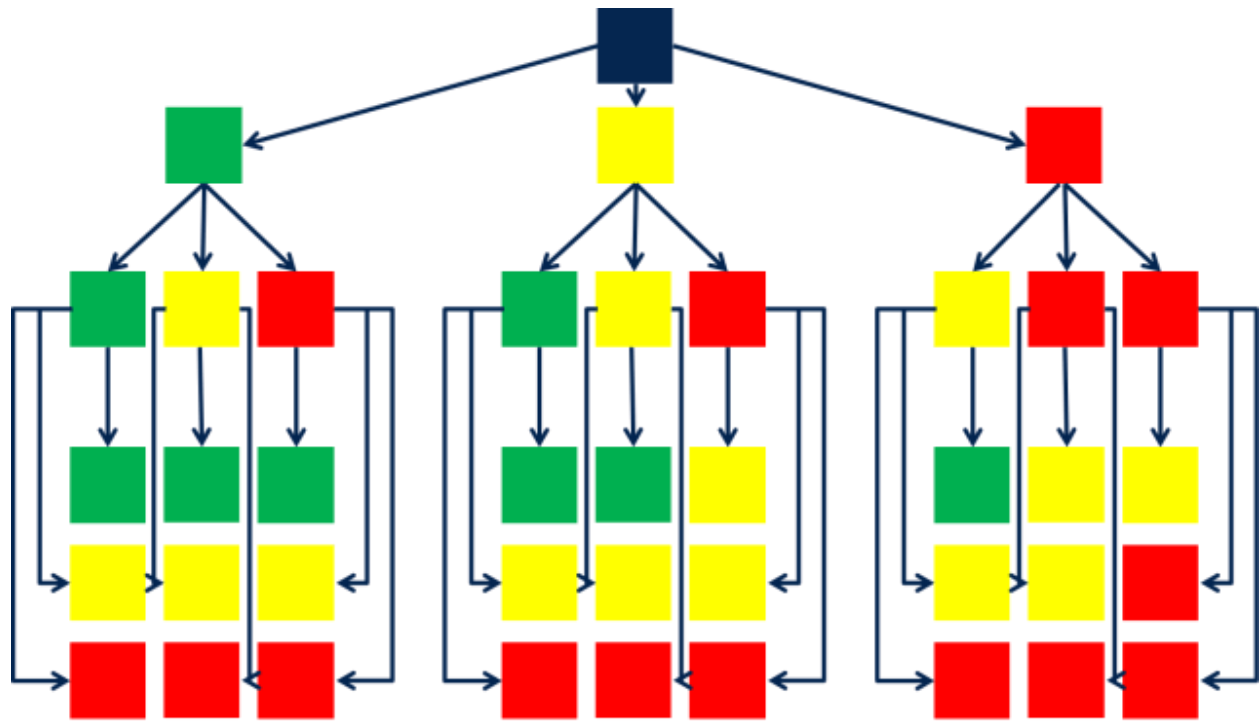
1. Install Twine if you haven't already (or use the online version).
2. Create the introduction in the starting passage of your scenario. You can use your scenario concept from lesson 2 as a starting point, but you may need to revise it so it works for the audience of learners rather than stakeholders.
3. Add a single link or an easy question at the end of your introduction like `[[Let's get started]]` or `[[Continue]]`.
4. Write one question or decision point, including the options (usually 3 choices, but it could be more or fewer) with links to new passages.
5. Write the responses or consequences to each decision from these first three passages.
6. Adjust the formatting to make your scenario easier to read. (We will do more with formatting in lesson 6, so right now you just need to focus on readability and color contrast.)
7. Play the scenario to verify that everything works so far.
8. If you have extra time, start writing the "ideal path," or the path learners would see if they made all of the best possible choices.

Branching Structure

Time Cave

I refer here to terminology from Sam Kobo Ashwell⁷ in his post on Standard Patterns in Choice-Based Games. Although he's writing about structures for games and interactive fiction, his concepts also apply to branching scenarios for training.

The Time Cave is the traditional structure for branching. Each choice leads to more choices, with no rejoining or reusing choices. This leads to numerous endings.



In the example above, players make 3 total decisions, but it's already 40 screens (1+3+9+27). The number of screens in a time cave increases exponentially. To add a fourth decision in the path, you need to add 81 more screens.

By necessity, a time cave is generally limited to a maximum of 3 decisions in each path. It's broad, giving you lots of alternative endings, but it's not very deep. It doesn't do well for showing a longer process or change in a character.

⁷ <https://heterogenoustasks.wordpress.com/2015/01/26/standard-patterns-in-choice-based-games/>

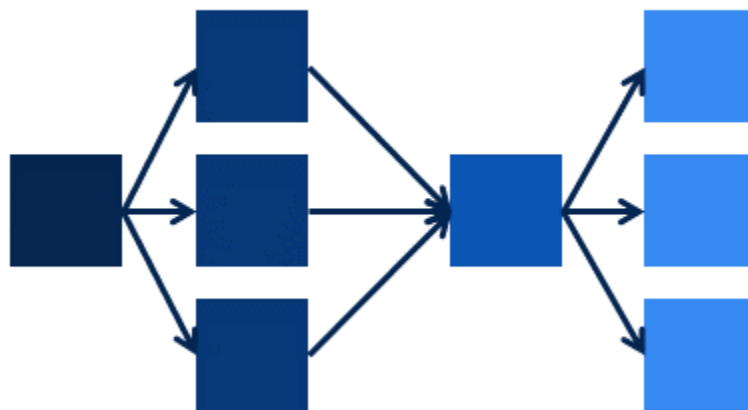
When to use a time cave

Although this is the structure many people think of when they envision a branching scenario, it's not very useful for online training because of its complexity. Generally, I recommend using this structure for a short process with a maximum of 3 steps. For a longer process, look for ways to collapse certain options together to reduce the number of passages or slides needed.

Limited Branching or Gauntlet

Another structure is limited branching, where you always return to the correct path. This structure has different names in different sources. Sam Kabo Ashwell calls it a gauntlet. Cathy Moore calls these "control freak scenarios," since the learners have no real freedom in their choices and everything is tightly controlled by the ID.

Instead of a true branching scenario with multiple endings, this is essentially a single correct path and a single ending. When you make an incorrect choice, you get some customized feedback and perhaps see limited consequences of your decision. There are no long-term consequences for decisions unless something is controlled with variables and states. You are forced back to the correct path, regardless of your mistakes.



Examples of limited branching structure

I use this structure in some of my demos, such as this Twine scenario on responding to instructional writing feedback.

Anna Sabramowicz used this same structure for her interactive storytelling model. Her Broken Co-Worker example uses a central narrative; you're faced with the same series of decisions no matter what choices you make.

When to use limited branching

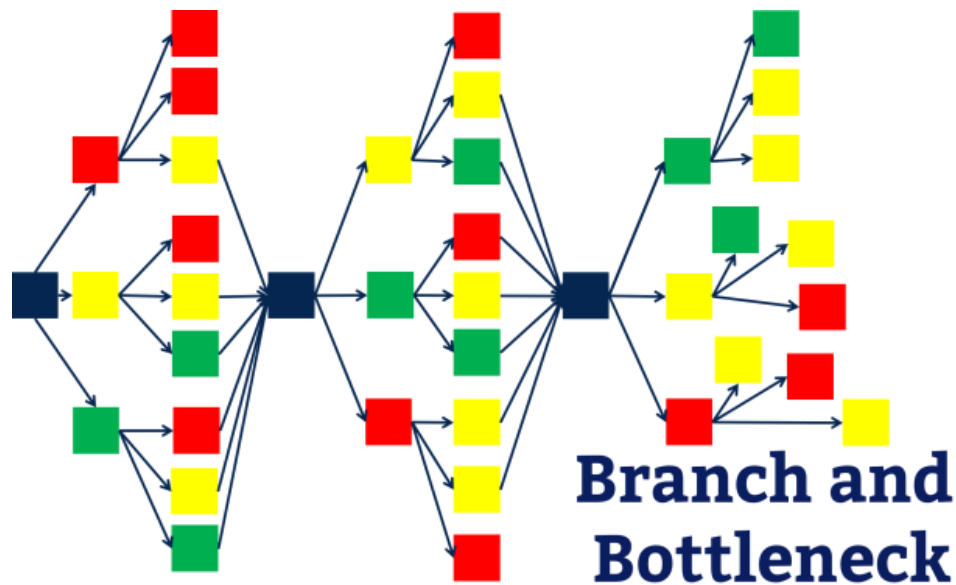
In limited branching, you can get the wrong answer every single time, and the scenario still propels you forward. That can be helpful for teaching skills to less experienced learners, where you're mostly using the scenario for instruction rather than decision-making practice.

This structure also works OK if your scenario is a series of independent decisions rather than multiple decisions in a single large scenario. If you're teaching a process with multiple steps, where each step is contingent on the previous step, this method doesn't create as realistic of an assessment.

Sometimes, you might use this structure because you're short on time and resources, and this is faster to build. You might choose this for fast tracking your branching scenario.

Branch and Bottleneck

One of the challenges of branching scenarios is that you can generate so many options and paths that it quickly becomes unwieldy. A branch and bottleneck scenario structure is one strategy for keeping the complexity manageable.



In a branch and bottleneck structure, you branch into different options for a while, but then all paths return to a single bottleneck. The bottleneck is an event or choice that happens the same in every path. In the example above, there are 39 screens. Most paths have 6 decisions (plus two bottlenecks), although a few shorter paths end in failure.

This is another example of terminology I learned from Sam Kabo Ashwell in his post on Standard Patterns in Choice-Based Games.⁸

The branch-and-bottleneck structure is most often used to reflect the growth of the player-character: it allows the player to construct a somewhat-distinctive story and/or personality, while still allowing for a manageable plot.

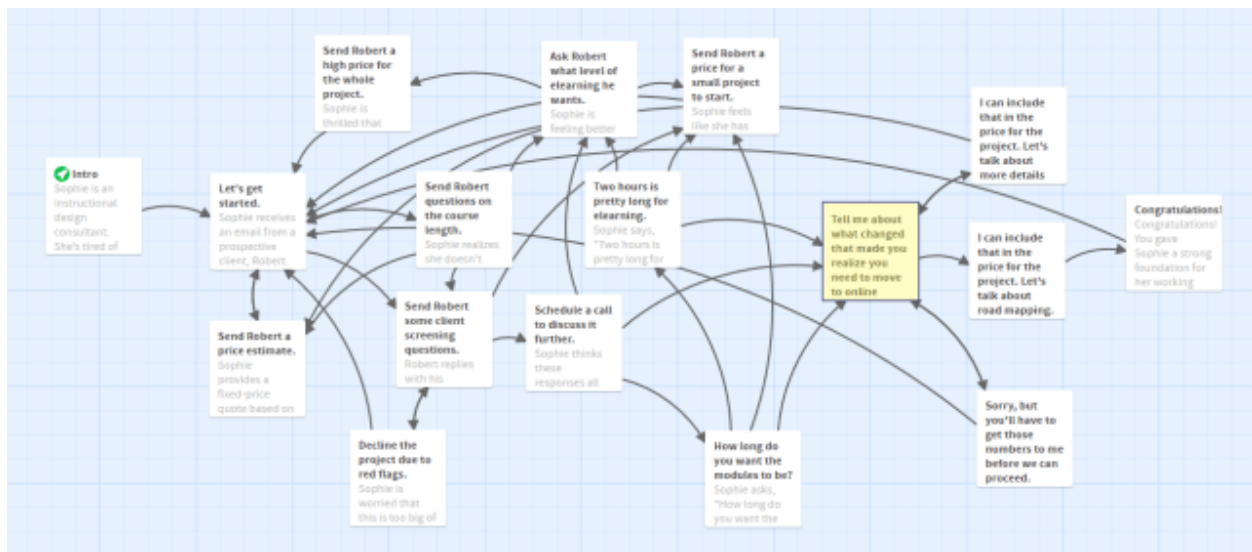
Sam Kabo Ashwell

Ashwell notes that this structure often relies on states to maintain continuity in the narrative. You might track an overall score or adjust another part of the consequences or intrinsic feedback based on earlier choices.

Examples of branch and bottleneck

Client screening

In practice, the branch and bottleneck doesn't usually look as "clean" as the graphic above. I usually reuse some choices. That gives players a chance to correct their mistakes without continuing down the wrong path indefinitely. It doesn't immediately force them back to the right path or to restart; it lets them learn from the feedback and adjust.



In my client screening branching scenario, I have one major bottleneck near the end, highlighted in yellow ("Tell me about what changed..."). Several other paths are failures

⁸ <https://heterogenoustasks.wordpress.com/2015/01/26/standard-patterns-in-choice-based-games/>

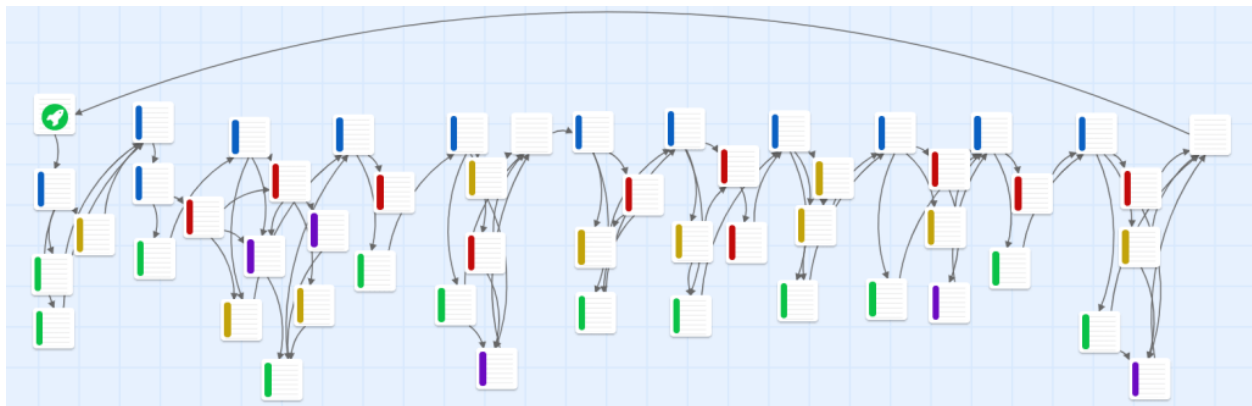
that cause a restart. Mostly, the paths cross and converge multiple times as players have the opportunity to correct mistakes and get back to the ideal path.

The ideal path is 4 decisions deep. The complete scenario ended up as 20 slides in Storyline, 1/6 of the number of slides it would have taken to use a time cave structure.

Teaching something new

In this example, the branching scenario was a simulation to teach new information. It's essentially a simulation with lots of decisions and immediate feedback. There are no failure points; learners get feedback and then continue to the next decision.

In this scenario, the questions and bottlenecks are tagged blue. Choices and feedback are tagged green, yellow, and red (Good, OK, Bad). Additional info is tagged purple.



Some of this scenario is effectively a gauntlet. Especially at the end, the scenario has several decisions that follow the pattern of bottleneck decision point – feedback – bottleneck decision point.

However, it has some complexity that makes it a little more interesting than a pure "control freak" scenario. The beginning has some more complex structure, with multiple crossing paths and options before returning to a bottleneck. Plus, even at the end, learners often have the end to select a different option if they make a mistake. The opportunity to fix mistakes and try another option can be very helpful for learning.

When to use branch and bottleneck

Branch and bottleneck is a good structure if you have two options that could be done in either order (where you then backtrack to the other branch after completing one). It's also good if certain key events will drive the plot forward regardless of prior decisions, or if there's a particular point you have to reach in order to continue.

Branch and bottleneck can be very effective in scenarios designed to teach new skills and information, rather than to practice or assess existing skills. It gives you a little more complexity than a limited branching or gauntlet scenario, but it's still manageable.

Activity 4: Branching Structure

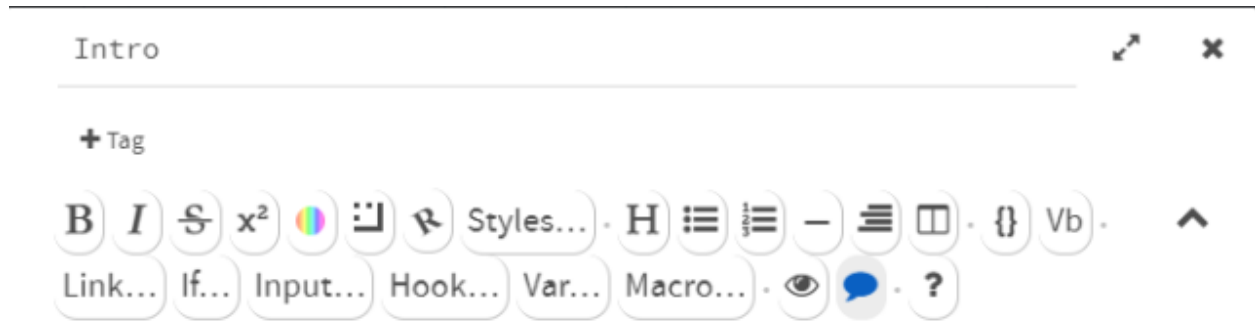
Next, you will plot out the overall structure of your branching scenario in Twine. For this fast track workshop, aim for around 12-15 passages in Twine, including any intros and feedback passages.

1. Starting from the intro and decision point you created in activity 3, add linked passages for each step in the "ideal path" for your branching scenario. This should be 3-5 passages. You may want to start with just key points to develop the structure, rather than drafting all of the text right away.
2. Once the structure of the ideal path is built, start back at the beginning and add one more choice. Plan a single alternate path from beginning to end.
3. Each time you reach an ending point or a bottleneck where the scenario rejoins another path, return back to the beginning. Find a dead end and flesh it out. Keep repeating this until you have the full scenario structure built. As you build it, think about the structure. How does your structure support learning? Are you keeping the complexity manageable?
4. At all endings in the scenario, provide a link to restart the scenario or go back.
5. Play through the scenario, trying several different paths. Correct any errors or dead ends in the structure.
6. With any remaining time, go back and start fleshing out the text of your branching scenario in any placeholders.

Visual Design in Twine

Harlowe Toolbar

The Harlowe toolbar gives you access to a number of features without having to know how to code them.



In past versions of Harlowe (and current versions with other story formats), a passage appears with a title, tag, and space for the passage text.



Formatting text

Formatting text with bold, italics, and strikethrough styles works like other text editing programs. You don't have to remember the markup language, but it will appear that way in the passage.

If you want to change the overall look and feel of the whole story, it's a little more complicated than just formatting a short passage of text. For example, the default styling for Harlowe is a black background with white text. I don't personally like this, plus I think the blue links are hard to read.

What topic do you want to use as the focus for a branching scenario prototype?

New manager training

Objection handling

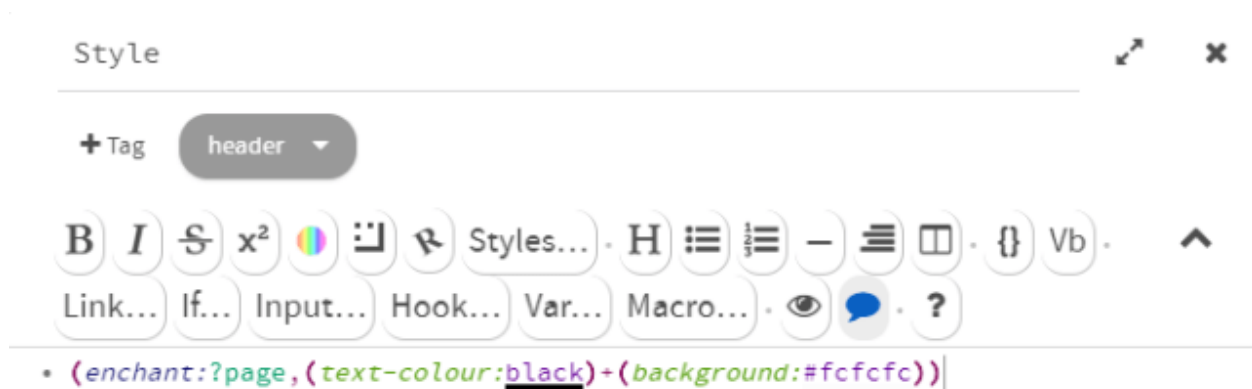
Troubleshooting

I'd rather have a white background with black text so the links are more readable.

1. Create a new passage. This passage doesn't need to connect to any other passages.
2. Add the title "Style."
3. Select the **Text and background color** button on the toolbar.

The screenshot shows the 'Style' configuration panel in the Harlowe editor. At the top, there is a title 'Style' and a '+ Tag' button. Below this is a preview box containing the text 'Example text preview' in a serif font, centered on a black background. The 'Text colour' section has 'Flat colour' selected, with a color palette showing various colors and a 'Harlowe built-ins' dropdown. The 'Background' section also has 'Flat colour' selected with a similar color palette and dropdown. Below these are options for 'Affect' (The attached hook, The remainder of the passage or hook, The entire passage, The entire page) and a 'Resulting code' field containing the code `(enchant:?page, (text-colour:black)+(backg`. There are 'Cancel' and 'Add' buttons at the bottom right. A note at the bottom left says 'Double-click this passage to edit it.'

4. Change the text color and background.
5. Set the **Affect** to "The entire page." This is what makes the change format the whole page instead of just a passage.
6. Select **Add**.
7. Remove the placeholder passage text if needed.
8. Add the tag "header" to the passage. This tag adds the style to every passage in the story automatically. (Note: tags are case sensitive. "Header" is not the same as "header.")



Changing the color and fonts makes it more readable.

What topic do you want to use as the focus for a branching scenario prototype?

New manager training

Objection handling

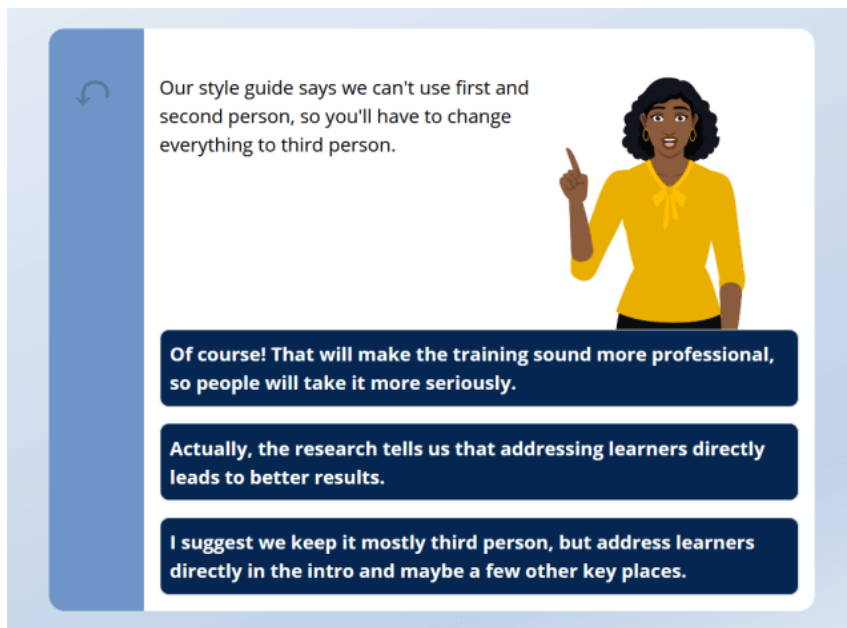
Troubleshooting

Could you accomplish the same thing without using the toolbar? Yes! You could enter the same "enchant" code shown below to make these adjustments. However, if you aren't used to using Twine, this format may be hard to remember. That's what Harlowe tries to accomplish; making it easier to generate these codes without consulting the documentation each time.

```
(enchant:?page,(text-colour:black)+(background:#fcfcfc))
```

Sample visual design in Twine

While I often use Twine for simple text-based prototypes of branching scenarios, it's also possible to create a more polished look and feel. With some work on the visual design, you can use Twine to create branching scenarios for end users without migrating the content to another tool for final development. This requires changing the formatting with "enchant" macros on three separate elements: the page, passage, and links.



Page Settings

The page settings affect the entire background and story.

1. Select the **colors** button.
2. Change the text color to the **flat color** black.
3. Change the background to a gradient. I used some light blues for a subtle gradient. You may need to play around with the gradient options until you get exactly what you want.
4. Set it to affect the entire page.

That generates the following code:

```
(enchant:?page,  
  (text-color:black)+  
  (background:(gradient: 157,0,#EAF0F7,0.7102,#C1D2E7,1,#F8F7FF))  
)
```

I also changed the font to Open Sans by adding +(font:"Open Sans"). This command works with any font available to the browser, so any standard online fonts should work.

This is the final code for the page:

```
(enchant:?page,  
  (text-color:black)+  
  (background:(gradient: 157, 0,#EAF0F7,0.7102,#C1D2E7,1,#F8F7FF))+  
  (font:"Open Sans")  
)
```

Our style guide says we can't use first and second person, so you'll have to change everything to third person.

Of course! That will make the training sound more professional, so people will take it more seriously.

Actually, the research tells us that addressing learners directly leads to better results.

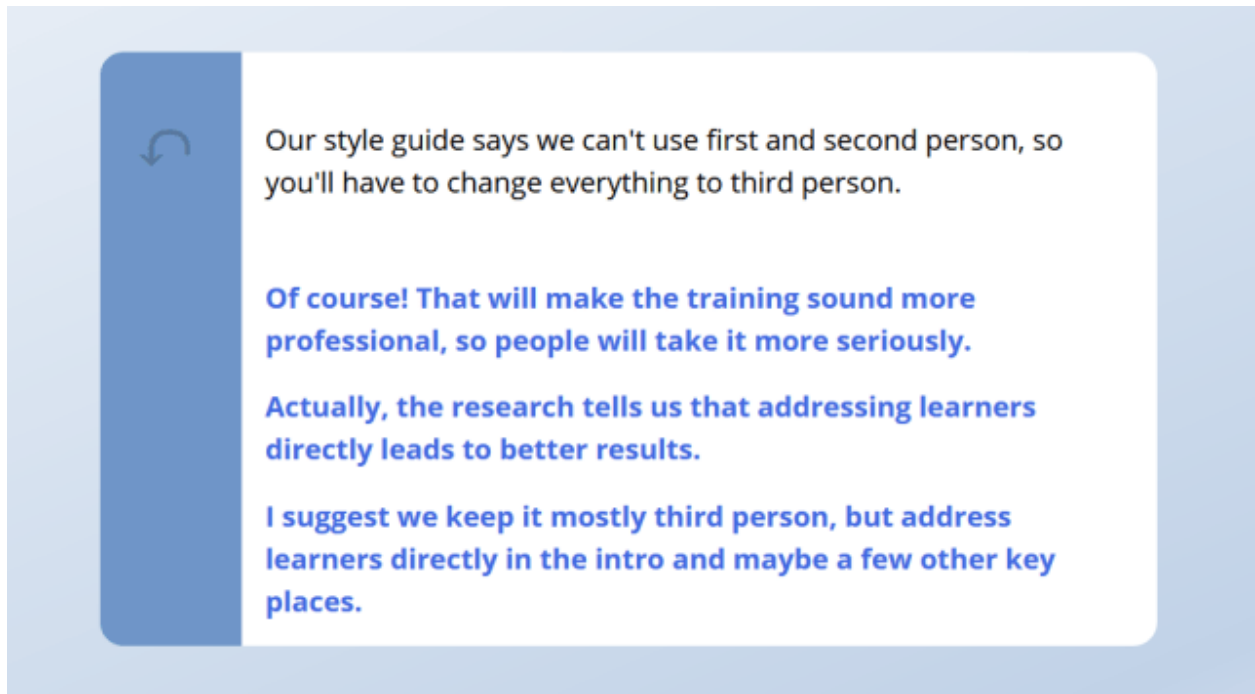
I suggest we keep it mostly third person, but address learners directly in the intro and maybe a few other key places.

Passage settings

Next is the passage settings. The passage settings control the container for the story text itself. You can copy and paste the settings below into the Style passage.

```
(enchant:?passage,  
  (background:(white)) +  
  (corner-radius:20) +  
  (border:"none", "none", "none", "solid")+  
  (border-size:120) +  
  (border-color:#6F95C8)  
)
```

background	the background of the passage
corner-radius	the setting for how rounded the corners are
border	the border around the passage. The only border shown is on the left side, and it's a solid color. That's the background for the sidebar with the previous button.
border-size	the width of the sidebar
border-color	the color of the sidebar



Our style guide says we can't use first and second person, so you'll have to change everything to third person.

Of course! That will make the training sound more professional, so people will take it more seriously.

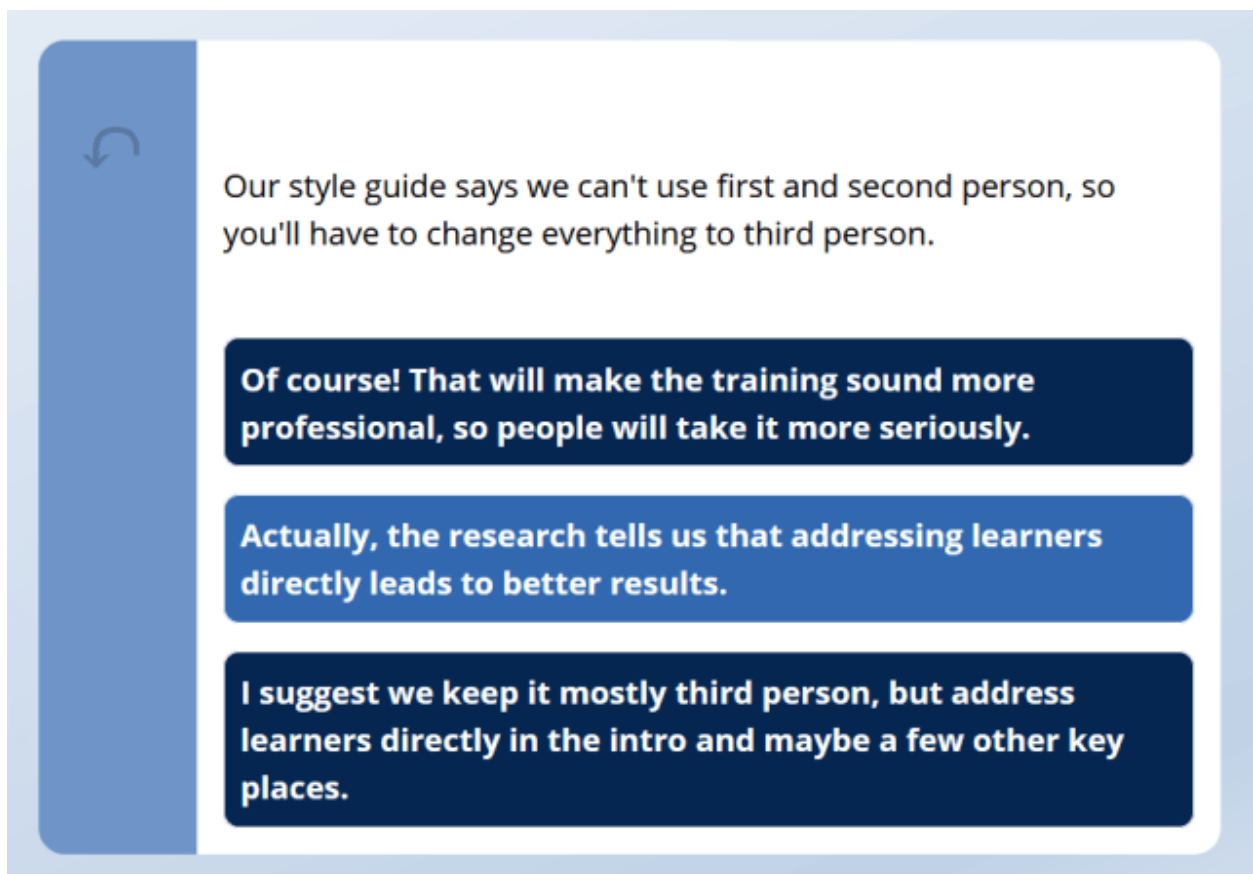
Actually, the research tells us that addressing learners directly leads to better results.

I suggest we keep it mostly third person, but address learners directly in the intro and maybe a few other key places.

Link settings

To make the links look more like buttons, I changed the background and text colors. I also added rounded corners and a hover style with a different color. These enchant settings should be attached to the link.

```
(enchant:?link,  
  (background:#052650) +  
  (border:'solid') +  
  (color:white) +  
  (corner-radius: 12) +  
  (border-color:#F8F7FF) +  
  (border-size:1) +  
  (hover-style: (bg:#3168b0) + (border-color:white))  
)
```



Images

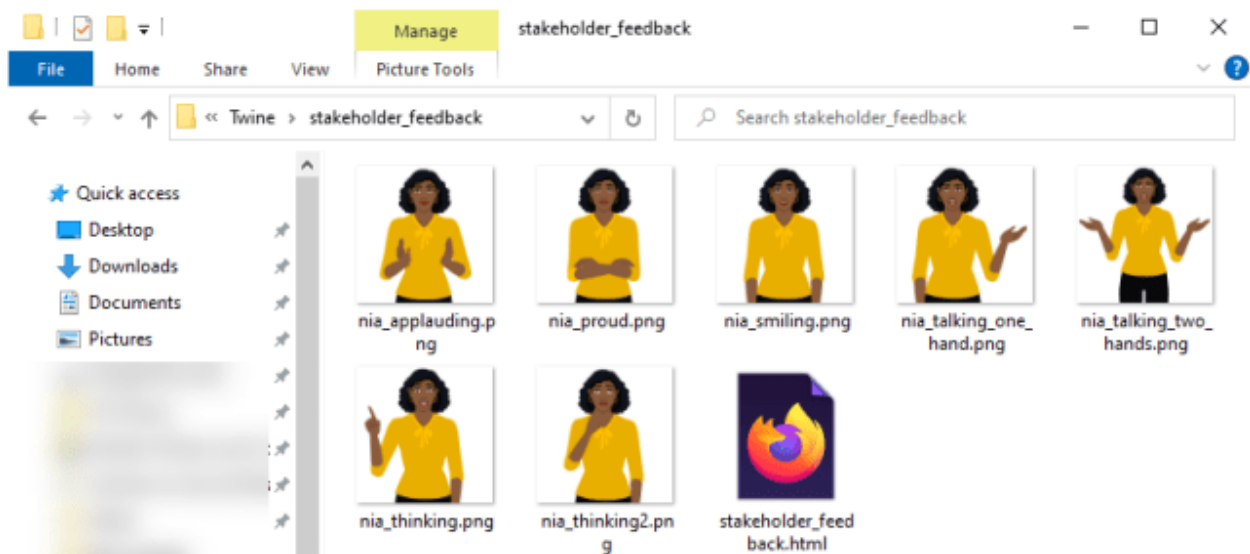
For fast tracking a branching scenario, it may be easiest to skip images, or to only use a single image for the background. Unless an image is important to set the context or part of the scenario, consider skipping it.

To add images in Twine, use the **img** HTML tag. You will need to add images in each individual passage, not the style passage, if you want to show changing character poses.

```

```

You can either link to images online using a full URL or link to images with a relative link. In this case, I put all of the images in the same folder as the published scenario. The images in this scenario are from the eLearning Art⁹ "designer realistic" illustrated character set.



Note that when you **Play** a scenario in Twine, it will only show the alt text for the image. You must publish the scenario for image links to work correctly.

⁹ <https://elearningart.com/>
christytuckerlearning.com
©2022 Christy Tucker

Sample enchant macros

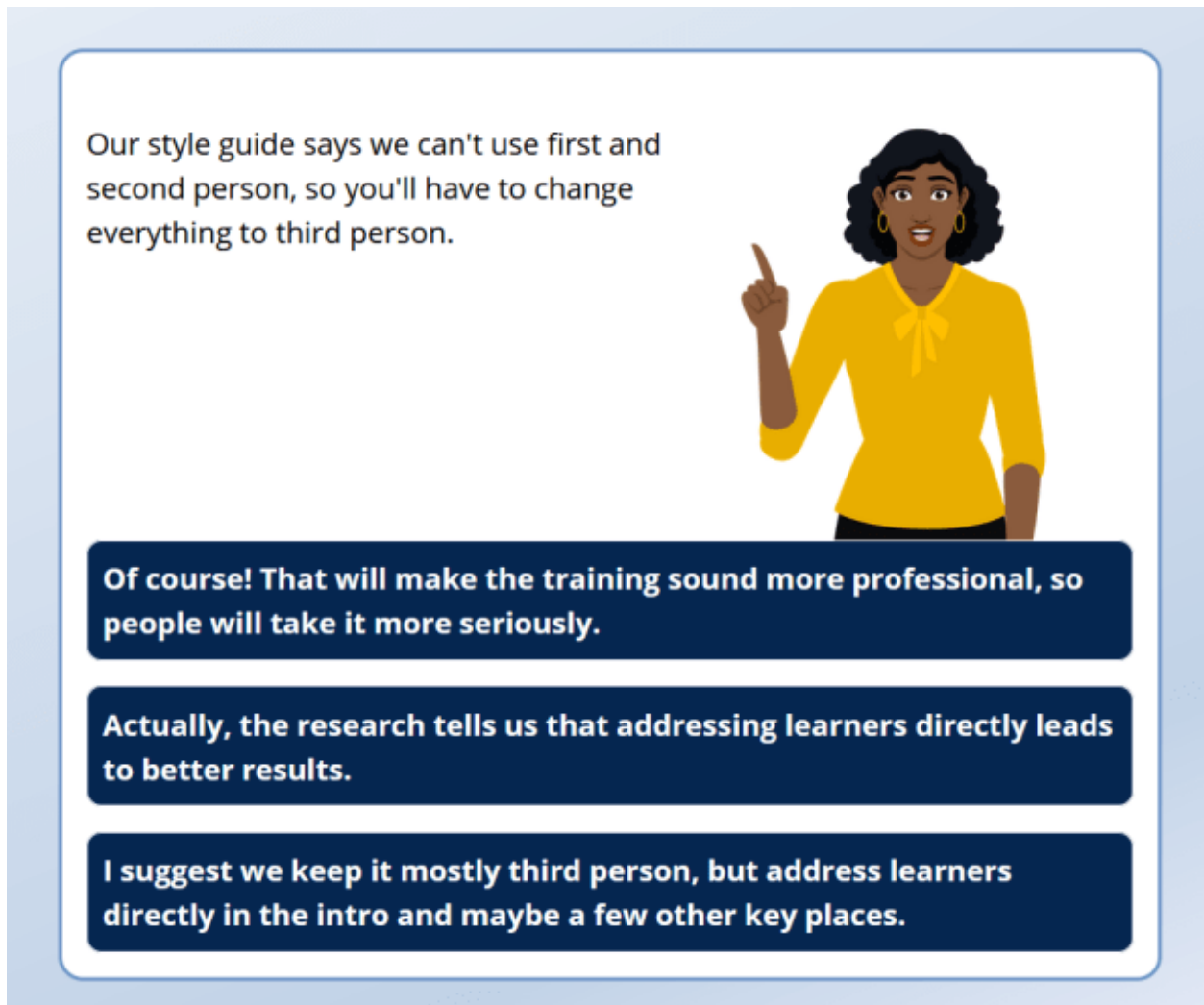
You can use my enchant macros as a starting point for your own visual design in Twine scenarios. If you are using the online version of this workbook, you can copy and paste the entire code below, then edit and experiment with your own variations.

```
(enchant:?page,  
  (text-color:black)+  
  (background:(gradient: 157, 0,#EAF0F7,0.7102,#C1D2E7,1,#F8F7FF))+  
  (font:"Open Sans")  
) (enchant:?passage,  
  (background:(white)) +  
  (corner-radius:20) +  
  (border:"none", "none", "none", "solid")+  
  (border-size:120) +  
  (border-color:#6F95C8)  
) (enchant:?link,  
  (background:#052650) +  
  (border:'solid') +  
  (color:white) +  
  (corner-radius: 12) +  
  (border-color:#F8F7FF) +  
  (border-size:1) +  
  (hover-style: (bg:#3168b0) + (border-color:white))  
)
```

Hide the sidebar

One variation is to hide the sidebar. That will remove the undo and redo buttons. In this case, you need a solid border all around the passage, but it should be narrower. Use the hide macro to hide the sidebar.

```
(enchant:?passage,  
  (background:(white)) +  
  (corner-radius:20) +  
  (border:"solid")+  
  (border-size:3) +  
  (border-color:#6F95C8)  
)  
(hide:?sidebar)
```



Our style guide says we can't use first and second person, so you'll have to change everything to third person.

Of course! That will make the training sound more professional, so people will take it more seriously.

Actually, the research tells us that addressing learners directly leads to better results.

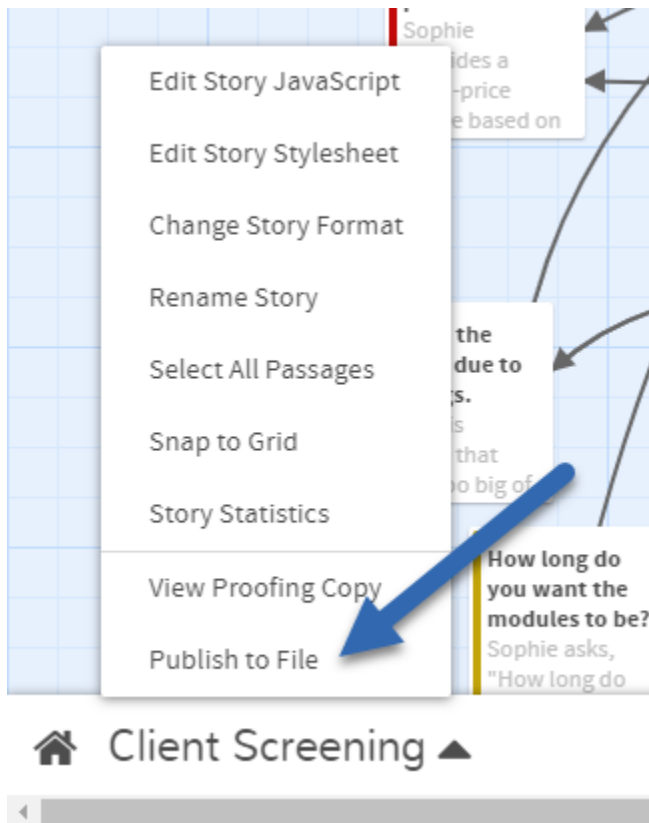
I suggest we keep it mostly third person, but address learners directly in the intro and maybe a few other key places.

Publish a Twine Story

A published Twine file is an HTML file that can be shared on a server for SMEs and stakeholders to review, or to an LMS that accepts HTML.

Select the name of your story.

Select **Publish to file**.



Export to Plain Text

For review purposes, it can be useful to have a plain text export of the Twine story in addition to the published version.

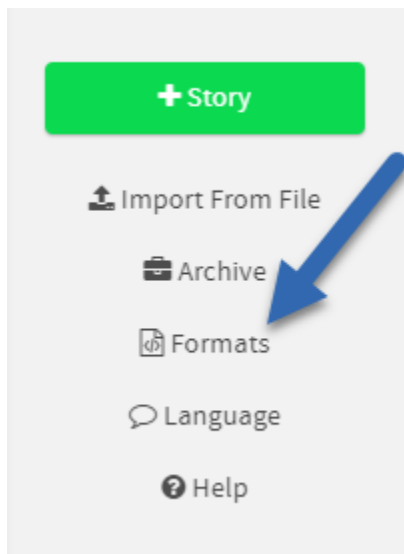
The default proofing format for Twine is Paperthin. I find Paperthin to be challenging if I need to provide a text version of the entire scenario for proofreading or review.

Paperthin's export can't easily be copied to a Word document; it only copies the passage text, not the passage names.

Instead, I prefer the Entwee proofing format.

Install and Use Entwee

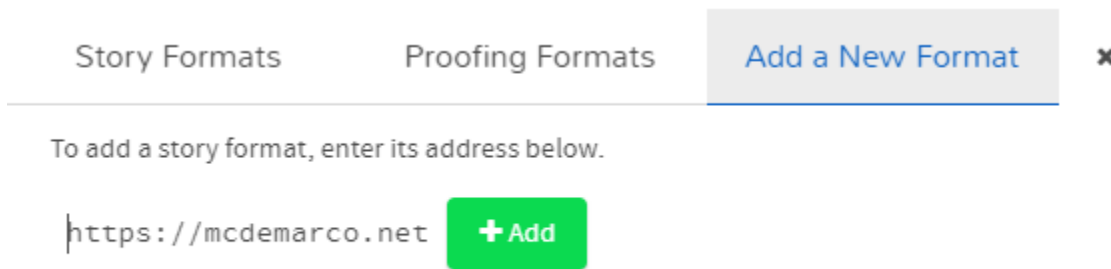
On the Twine home page, select **Formats**.



Select **Add a New Format**.

Enter this address:

`https://mcdemarco.net/tools/entwee/format.js`



On the **Proofing Formats** tab, select Entwee as the default.

Open your story. Select the story name, then select **View Proofing Copy**.

The plain text version will download as a .txt file. I copy and paste the text into Word to send to stakeholders for review. You may want to do a little cleanup before sending it for review. Each passage starts with double colons.

Example entwee plain text format

```
:: Send Robert questions on the course length.[OK]
```

```
Sophie realizes she doesn't have enough information to provide an accurate estimate. She asks Robert to clarify how long the courses actually are.
```

```
Robert replies with the details.
```

```
//Course 1 is a half day (3.5 hours).
```

```
Course 2 is one day (about 6 hours).
```

```
Course 3 is two days (about 12-13 hours).
```

```
Course 4 is four days (about 26 hours).//
```

```
What should Sophie do next?
```

```
[[Send Robert a price for the whole project.->Send Robert a price estimate.]]
```

```
[[Ask Robert what level of elearning he wants.]]
```

```
[[Ask Robert some high level questions about goals and budget.->Send Robert some client screening questions.]]
```

Activity 5: Finish Your Branching Scenario

Now it's time to finish your branching scenario.

1. Finish writing all text for your branching scenario, including dialog, options, and feedback.
2. Adjust the formatting in Twine to get the basic look and feel you want. Don't spend too much time on this; fast tracking means getting it "good enough" without necessarily aiming for perfection.
3. Play your scenario to test it. Try multiple pathways, watching for continuity and flow. Revise your scenario.
4. Share your scenario with someone else and provide feedback.

Congratulations! You now have a functional branching scenario activity!