Crafting Simulated Worlds of Interactive Text:
The Basics of Twine for History, English, and Language Teachers

Version 1.0: 1/9/2017
Jeremiah McCall: gamingthepast.net
@gamingthepast
jmc.hst@gmail.com

Note: This text uses the default design format for Twine 2.0, Harlowe.

Starting Twine:

Once you have installed twine (find different versions at www.twinery.org) run it. Skip any offer of an introduction and you will be brought to the starting screen. It will be empty of stories at first. Once you have written some it will look more like the image to the right. To create a story, click on the green +story button on the right side of the starting screen.

When you create a new story in Twine, the story map will appear with a first passage, marked untitled passage. This will be the start of your text. Edit this untitled passage by double-clicking on it.

Twine Design, Stage 1:
The basics of a Twine text: Passages and links

When you first open a project in Twine, the story map will appear with a first passage, marked untitled passage, as shown in the picture on the left. This will be the start of your text. Edit this untitled passage by double-clicking on it.

Double-clicking brings up the passage editor screen, which looks like this. Replace the “Untitled Passage” text with your own title, simpler is better. Ignore the +Tag button for now.

Then, as the passage editor suggests, double click on the passage text and create your own passage. Since this is the start of your interactive text, you will want to establish the situation in which the player/reader finds herself. Generally (though it is not required) the Twine reader is referred to in the second person (“You are on a long and winding road.”).

When you are done with the passage the player will see, you’ll need to provide some links that allow the player to make a choice in the story. Let’s suppose your passage text says, “You are on a long and winding
road” and you would like the player to have the choice to go to Cincinnati or to return home. Each of the choices must be made into a link that, when selected, will take the player to a new passage.

There are two parts to a basic link in Twine Harlowe. The first part is a word or words that the player will see and be able to click upon in the game. The second part is the name of the passage the player goes to after clicking on the link. These two parts are connected with an -> made up of the minus sign and the greater than sign. The whole is enclosed within [[ ] ] like so: [[Text Player Sees->Destination Passage]]

To make a link to a new passage

Format:

[[Text Player Sees->Destination Passage]]

Examples:

[[Walk to Cincinnati->Cincinnati]]
[[Give up and go home->Home]]

Notice that if you typed the links correctly, two new passages with the names “Cincinnati” and “Home” have been created in the story map. To continue the story, write text in each of the new passages that the player will select their respective links. That’s it for the basics. The new passages you have created can have their own links to more passages and so on. Done this way, a story map could resemble the picture below:

Exit out of the passage editor. Be sure to use the black X to exit the passage editor. If you use the red X at the top right of the window, you will close the program and lose whatever you typed in this passage.
Twine Design, Stage 2: Using variables and if-statements to vary the story for the player

Using simple passages and links alone is an excellent way to design a simple Twine text. For texts that want to provide the player with more input, a more varied experience and a greater number of choices without an exponential increase of unique ending passages, just using links to determine what text the player sees can get cumbersome. They require the writer to create all manner of different endings and to keep mental track of each step the player might have taken to reach a particular end. If one wants to design a Twine with links only that gives the player three different decision points, the designer must create 15 passages and 8 distinct endings, as shown in this figure. It’s those eight unique endings that can start to pose a writing problem. Using some basic computer code, we can make interactive texts that respond more to player decisions and allow a series of choices to result in fewer endings that still have significant differences between them. We can also create a greater amount of variety in individual passages based on choices the player makes.

Keeping track of user choices and results using variables and if-statements

The key to more sophisticated interactivity in Twine is to write some basic Twine instructions that perform three actions in the text:

1. Keep track of an “event” that may or may not have happened previously as the player plays the Twine (a particular player choice; a score increase; etc.)
2. Checks later to see whether that “event” occurred.
3. Display some text or some other text (or no text) depending on whether the “event” occurred.

Step 1: Keeping track of user choices and events using variables

A variable in Twine is a named space in the computer’s memory that can store a piece of information for use later in the Twine. Each variable must have its own unique name, have no spaces, and begin with the dollar sign $ character. Usually, variable names of more than one word capitalize the first letter of each word after the first for readability.
Examples of variable names: $score, $name, $romeoIsDead, $presentedDemands

The three common types of information stored in a variable are:

1. An integer, when for example, keeping track of something like a score
2. A value of true or false (called a Boolean value) when keeping track of whether something in the text did or did not happen
3. A string of characters when, for example, keeping track of a name or some other text.

To store a value in a variable, use the (set:) macro in the passage that you want to set the variable

**Format:**

(set: $nameOfVariable to value)

**Examples:**

(set: $score to 0)
(set: $name to “Josephine”)
(set: $mercutioDead to false)

Using the (set:) macro in these ways will store the listed piece of information (The integer 0, the string “Josephine”, and the Boolean value true) in the named variable.

The value stored in a variable can be shown to the user just by typing the name of the variable in the passage text. Suppose $score were set to 5. A passage with the following text in it:

Your score is @score

Would display the following text to the player

Your score is 5

The value in a variable can be changed whenever the designer wishes, simply by using the (set:) macro again. If a designer types in a passage

(set: $name to “Josephine”)
(set: $name to “Rudolf”)

When the Twine runs, the variable $name will end the passage with the name “Rudolf”

A number stored in a variable can also increase or decrease, useful when trying to raise or lower a score, or some other value by a certain amount. So if $score is set to 0 in the first passage of the Twine, a later passage can increase the score using the += operator like so:

(set: $score += 1)

will add one to the current value in $score. Now $score will have a 1 in it. Any integer amount can be used, so the following line of code

(set: $score += 12)
would add 12 to the current value in $score. Finally, to decrease the amount in a variable, use the -= operator like so:

(set: $score -= 5)

**Steps 2 and 3: Checking the value stored in a variable and changing the text the player sees accordingly.**

**Storing and using a number value**

Once a variable has been set to a value, the Twine designer can check it (once or many times) later and, using an if-statement, can display different text depending on what is in the variable. The basic form for this is as follows:

(if: $nameOfVariable is someValue) [Text to display if the if-statement is true]

The terms in the if-statement, a variable name, <, >, or is and a value is called an *expression*. The expression contained in an if-statement is evaluated by Twine to be true or false. If the if-statement is true, Twine displays the text attached to the if-statement in square brackets [ ]

So, for example, if you are keeping track of the player’s score by adding to a variable called $score, and the value of $score increases when the player makes the right choices in the text. In the final passage of the text, you could customize the text if the player had not scored any points like so:

(if: $score is 0) [You have failed in your epic quest]

An if-statement can be followed by an (else:) statement. If the expression in the if-statement is not true, then text attached to the (else:) by square braces [ ] will be displayed

(if: $score is 0) [You have failed in your epic quest]
(else:) [You have succeeded]

The expression contained in an if-statement can also use > or < instead of is. So, for example, say you are keeping track of the player’s score by adding to a variable called $score, and the value of $score increases when the player makes the right choices in the text. In the final passage of the text, you could customize the text depending on the score the player had reached.

(if: $score > 5) [You have successfully made it through the horrors of your epic journey.]
(else:) [You will never achieve your quest.]

**Storing and using a boolean value**

Storing boolean values in variables is an extremely useful method of keeping track of things that happen in a Twine. Essentially create a variable that represents some important event or status in the text and set it to true or false depending on whether the event occurred or status is true.

**NOTE:** When using Boolean values, the values are written as true and false without quotation marks. They are not written “true” and “false”
A designer can use almost any name for a variable containing a Boolean value, but it is best to use a name that clearly reminds the designer what the variable is supposed to keep track of. So if you want one kind of ending passage if a character named Sparky is alive at the end of the Twine you might name a variable $sparkyIsHappy and set it to true in the first passage of the Twine text.

    (set: $sparkyIsHappy to true)

Later, if as the result of a choice the player makes a decision that causes the Sparky to be unhappy, the passage would set the value in $sparkyIsHappy like so.

    (set: $sparkyIsHappy to false)

Still later, Twine checks the value in $sparkyIsHappy and displays different text depending on whether the value stored in $sparkyIsHappy is true or false. The if-statement might look like this:

    (if: $sparkyIsHappy is true)[You managed to remodel all of Sparky’s house without making Sparky unhappy. Well done!]
    (else:)[Sparky is highly dissatisfied with the job you did remodeling the house. Sparky will not recommend you to anyone.]