# SciCommbat, spring 2023

SciCommbat is a game that we make as a full class in the science information and its presentation course at University of Arizona. It is a myth busting game built in twine. If you would like to see the live version of SciCommbat you can visit [www.scicommbat.com](http://www.scicommbat.com) and see how it has changed through the years. This document is going to describe the SciCommbat from spring 2023, as that was the most complicated version of the game I think we created – fall 2023 was also pretty complicated but it had more individual aspects and less group work.

## How to install and use twine

twine is a free software that you can find at <http://www.twinery.org/> and you can either download it or use it in your browser. This particular game was built using Harlowe (twine has multiple different languages that you can use within it).

Download the.html file from the same place that you found this readme. Then, in twine, you will see a menu item called library. Click on that, and then click on import. Find the.html file and import it.

A screenshot of a computer

Description automatically generated

## How the game works

The game itself is the player trying to convince a character that climate change is real – a very big ask. His name is Henry, your name is whatever you want to input in the input box. From there you make choices about what to say to Henry in response to his conversational decisions in an attempt to convince him to change his mind. Regardless of what route you choose, once the game ends you are sent to a scorekeeping card called "the end".

A screenshot of a computer screen

Description automatically generated

Inside of the end, there are some calculations that go on in the background that determine the number of points you have (which determines how good your ending was) and it determines whether there were techniques you tried a lot more techniques you didn't try. These techniques are things like knowledge deficit approach where you just try to give more information. Or fear appeals, where you try to scare Henry into making a decision.

At the end of the scorecard, you have the option to click and learn more about different techniques. This includes things like what at guilt appeal is, how economic appeals could be used, etc. These are not game content they are to teach the player a little bit more.

## Variable counters

Throughout the game each choice you make has a variable counter associated with it. These are essentially tallying up the numbers of choices you make and whether they were good or bad decisions. At the start of each passage there is a little counter that adds or subtracts points to these counters. Here is an example of a decision that was good because it uses a technique called motivational interviewing which tends to be more effective than other techniques:



As a result, the trust counter gains one, and so does the counter for motivational interviewing because we want to keep track of how many times you have picked those sorts of options.

In the ending passage, there are different chunks of text that are only displayed if you have made those decisions more than others. These appear in different boxes depending on what you chose. Here is an example of one that only displays if you made several rationality appeals or knowledge deficit model choices – more than or equal to three of them. These are not particularly good forms of science communication but they are very common, so that's why the player gets feedback on them and the fact that they aren't very effective. If they did not make a number of these decisions, they get told that there was a technique they didn't try and to go back and play again to see if they can figure out what technique it is.

A screenshot of a computer

Description automatically generated

## Keeping track of trust points

The variable counters for techniques were pretty easy to keep track of because you only need to know what they are at the end. The trust points were a little bit more complicated because we needed to know what they were in each passage because they could change what you were offered as options, or how Henry reacted to the player. We also had a point bar at the top that showed how many trust points you had.

Initially this was a problem because it was showing you how many trust points you had in the previous passage because it wasn't updating at the correct time. We also wanted to make sure that the number of bars was consistent every single time so we created a separate passage to keep track of trust points. It's called trust bar in the file.

A screen shot of a graph

Description automatically generated

The way we got it to work and show you the number of trust points you had at that moment one time was to use the after: function. Essentially it just created a delay and allowed you to calculate the points before showing the trust bar. It's such a small amount of time that you wouldn't notice as a player.

A computer screen shot of a black background

Description automatically generated

We also wanted the point bar to not show up in the first two passages as you don't actually have any points and you haven't fully chosen the things you need to choose to get points. We use the tag system in twine to say that if it contained the tag **noheader** then it wouldn't show up the trust bar. You can add tags of all kinds using the tag button in each passage:

A screenshot of a computer

Description automatically generated

## Tags and how we use them

The no header tag was used to identify which passages we did not want to have the trust bar in. The trust bar popped up as a header. When you open the game you'll notice that there are number of colored tags. Noheader doesn't have a color because it didn't need to, the colors were there to indicate the class what sections needed to be worked on, which sections were good decisions, and which sections were bad decisions.

This helped us determine when we needed to write potential endings. If you made something like three bad decisions in a row, Henry would get really mad at you. But it was hard to tell where those points were unless we had a visual signal like red/green tags.

A computer screen shot of a computer network

Description automatically generated

A few of them have orange tags because they were not completed in one of the final drafts. They were completed by the time of publication but we use that to indicate to each group who needed to do what.

Speaking of groups, you might notice that all of them have crocodilian passage names. These were just the group names that I assigned. So each animal is a section of the tree that I handed to a group of students to complete as their section. I then combined all of them as the version control officer.

## Climate Jokes

I don't fully remember why we wanted to include climate jokes but we decided that we did. So I think initially almost all of the passages you have the option to just randomly tell a joke, and this includes both the joke and Henry's potential reaction.

We have a separate passage called climate jokes that keeps track of all of the jokes and the potential reactions. It looks like that did not make it to the final draft is I can't see anywhere that you are allowed to tell Henry a joke. They weren't very funny jokes so that seems fine.

## Getting to pick your name

We decided that one of the pieces of control that we wanted the player to have was being able to input their name. To do that we just used:

**(set: $Name to (prompt: "Hey! What's your name?", "Enter It Here!"))**

And then any time that we were going to use the player's name you use the variable $Name and it would pop up with either enter it here if the player had not changed anything, or whatever the player had entered, e.g.:



## A few other random things

We found that it was hard to keep track of what decisions people had made in order to get to a passage. In order to avoid having our students click back-and-forth as they were trying to remember what they should be writing, we entered HTML tags at the top of each passage. Initially, in the first few phases of the game, these had more instructions in them. Things like, Henry should be mad here.

But because we had included HTML tags and variable counters, we found that there was a really big blank space at the top of each passage when you are playing. To collapse the blank space we started using {} as a wrapper which got rid of it. So the full hidden header would look something like this if opened up in twine:

A screenshot of a computer

Description automatically generated

You have the HTML comment, a little bit of instruction to whoever's writing that passage, the trust and the variable counters, and it's all wrapped up in the curly brackets to make sure that there's not a massive white space at the head of the webpage when it's being played.

## External URLs

Linking to an external website on a twine game can be a bit of a pain in the butt. You can use the (go-to-url) function, but the problem is that when the player clicks the link, it removes the text that was the link.

I have also run into some problems with Google domains. In the last year or two (2023ish) Google domains has really decided that it gets sketched out by twine files. This is exacerbated when you include external links because it thinks that the webpage is trying to infect someone's computer essentially. So right now every time I publish a new twine file or an updated twine file on my website, I have to then go and immediately start appealing the fact that they block me from showing up in search sites or from leading Google Chrome users view that webpage at all. So far, all the appeals go through so it's really just an annoyance but I suspect it has mostly to do with these external links.

## Combining files

one of the hardest parts of working with an entire class to create a single game is version control – making sure we don't overwrite one another's work. The students in this class don't necessarily have a lot of coding background so we needed an easy way to ensure that would work that didn't require teaching GitHub essentially.

The way that I found that this works best was to have me be the version control person and to split the game into chunks for different groups of students. So for example in the final version we have alligator, camin, and crocodile – each of those are a group of 3 to 4 students. They were responsible for fixing their section of the tree and combining their tree as a group , and then I was responsible for combining all the files from each of the groups.

I found a relatively easy way of combining them using **Tweego.** https://github.com/ChapelR/tweego-installer/releases

this combiner automatically will combine them if they are.the files and not.HTML files. But to make sure that they don't overlap, any passage that that group didn't work on should be deleted from their file. So for example, alligators group file should only have alligators passages and not the starting passage, the end, or crocodiles or caimen's.

I have two hard drives on my computer so I had a few problems with figuring out where I was supposed to install Tweego. Once I figured that out it got a lot easier.

Instructions on how to use it are here:

<http://www.motoslave.net/tweego/docs/>

## Double checking variable names

I got really interested in creating some code to automate version control in twine – one of the problems that happens a lot is that students don't stick with the exact same variable names and they don't remember that capitalization is important. So things like $Rationality isn't equivalent to $rationality. So I created some code in R that pulls out a list of variables. I've included the RMD file but I'm guessing it probably isn't all functional – I think I got stuck on some other task as well and so some of this might not work. But it was at least exporting a list of variable names so I could see if there was weird spelling problems.

## References

I pretty exclusively used the twine a Harlow reference vignettes page. Harlow is the language that we wrote this game in. <https://twine2.neocities.org/>

there is a little bit of CSS code in this game as well which you can see if you open up the story stylesheet:

A screenshot of a computer

Description automatically generated

This allowed me to center different graphics. The CSS information on W3 schools was pretty useful: <https://www.w3schools.com/css/css_align.asp>

We didn't end up using much of this for this particular game but I have in the past and this is a great resource for CSS as well in twine: <https://www.adamhammond.com/wp-content/uploads/2017/03/2_css_twineguide2-1_hammond.pdf>