Writing a Chatbot
For: UC Berkeley's Center for New Media ChatScript Hackathon Oct./2013

My wife Sue and I craft chatbots at our company, Brillig Understanding. Since you are about to write bots for the first time, we thought we'd share some of our views on it with you. This paper is about what you are trying to create and how to think about creating it, not all the nitty gritty details of ChatScript coding you need. At the very end is a two page quick cheat-sheet titled **Instant ChatScript** about actually coding a chatbot by extending Harry, a tiny bot that comes with ChatScript.

Uses of ChatScript

ChatScript is a scripting language and engine for responding to meaning. It has been used to write chatbots which carry on an entertaining conversation. Even a chatbot that pretends to be a patient to help train doctors on diagnostic interviews.

But chat isn't its only use. ChatScript has planning capability, and could be used to allow a robot to carry out plans it synthesizes in response to verbal instruction. We have also used ChatScript to determine the intent of an Amazon product search and either better reorder search results from Amazon or to perform a more successful replacement search if Amazon's search breaks down. And ChatScript has been used to provide natural language mapping into appropriate SQL queries on the Dun and BradStreet business database.

So ChatScript is really all about determining what a user says in natural language and making an appropriate response.

What are your goals

What is your chatbot supposed to accomplish? Is it entertainment or enlightenment? Is it an agent to provide answers about something or from the world? Is it to be the brains of a useful robot?

Where is your bot going to reside? Is it in the cloud or on a mobile device or the user's desktop? Will it be accessed via a mobile browser, a desktop browser, or a custom app? This affects how much your bot can say at once and how much access to knowledge it has.

Is it a stand-alone effort or is it going to contribute toward a greater whole? For this hackathon you might be building your own entertainment bot. But your effort is probably disposable. Since the Center for New Media has an interest in writing a bot to assist incoming freshmen, maybe you could pick some area of student life to cover- be it eating, dealing with the bureaucracy, or whatever. Then your work could be folded into a greater work and have a longer lasting impact.

Who is your audience

Who is going to use your bot? For example, we normally make our bot personalities female, because both men and women prefer to talk to women. But for an ESL bot in Japan, the bot was male and flirty because most students were adult females and liked to fantasize about having interactions with their male teacher. We even had a topic that allowed them to live out a wild sexual fantasy if they pressed the bot hard enough (only in Japan or in a porn-bot would we ever be able to do this). We had to specially

limit the vocabulary as well since an ESL student's knowledge of words is limited.

If your audience is largely pre-teen, then you have to write with simple vocabulary and sentence structures. You may want a chirpy light-hearted tone. Your character probably shouldn't be older than early 20's and probably younger. And there are a bunch of topics you'll want to avoid. First are topics many adults can't handle without flipping out (politics, religion, abortion, gun control, sex). Second are topics the child will have no interest or experience with (e.g., credit cards, alternative medicine, altruism). Third are topics that might be scary (death, disease, crime, disaster, aliens, spying) And fourth are topics some parents won't want you to discuss (anything remotely connected with "bad" behavior including sex, drinking, smoking, coffee, swearing, stealing, lying, laws, Grand Theft Auto) .

If your audience is adult, you can talk about pretty much anything. You can be a fervent activist in politics or food if you want. Be free in all things except you still have to be careful how you handle religion.

Crafting a personality

If you are going to create a character, before you can write any ChatScript you have to define a consistent personality. How old, what gender, what attitudes, likes, and dislikes. The character needs strengths AND weaknesses; it shouldn't be perfect in everything. The easiest way to start is to pick a personality profile - base it on a real or fictional character, an astrological sign, or whatever. Start from there and extend it consistently as you wander into random topics. Find something memorably quirky. Angela, for example, was a talking cat (that's not the quirky part) that was deeply interested in fashion and wouldn't eat meat because of an incident with a rat in her childhood. Her fashion interests had to match with her being a cat, so she loved hats but had trouble with shoes (her claws ruined them).

Authoring

As I am an expert in writing ChatScript code, I have authored bots on my own. But I am not really a creative writer, so often times Sue, who is not a coder, is involved. What we do is she writes a proto script of what she wants and then I convert that into proper ChatScript. Trying to train her to write script directly is a bad investment of time. Her creative writing needs to flow easily and quickly or she will lose the impetus of the moment. Any time she stops to think about how to script something, that delicate conversation in her head will get broken. So she writes out a gambit, puts down a sample sentence or word along with what to say as a rejoinder. She might write this:

t: What do you dream about?
Flying – Traditionally this means you …
Food – Do you feel deprived …

and leaves it to me to convert flying into a proper rejoinder with all the possible related keywords like airplanes, flying, flight, wings, etc.

She will then write a series of expected questions and responses, for me to convert into proper responders.

How often do you dream- I'm sure I dream a lot but I don't remember most of them.

This tag-team approach efficiently gives us the best of the creative and programmer worlds. When she finishes a topic, I try to convert it into script to hand back the same day, so she can test it out while the topic is still fresh in her mind. Testing will reveal new things that should be added as responders and rejoinders.

How to think about ChatScript conversation

ChatScript consists of rules organized into topics. Some rules are used to match arbitrary user input, like *user: where do you live?* Others are used to lead the conversation on a subject, like *bot: do you have any pets?* And yet more rules are used to react specifically to a user's response to the bot, like: *bot: What is your favorite way to kill people? user: I think a public hanging is best. bot: Isn't it safer to kill people privately?*

Topics typically are a collection of rules based around a conversational theme, be it food, or pets, or murder. The great thing about ChatScript is you can write a topic on anything and just dump it into the system and have it automatically hooked up and working. So lots of people can collaborate by merely picking a different topic to write about and putting them all collectively into the system.

When we write a chatbot, we have different styles of topics we write.

Interactive Chat Topic

The meat and potatoes of our conversational chatbots is the interactive topic. In it we have things to tell the user, seek interaction with the user, and are able to respond to the user on topic. Such a topic uses all the rule types (gambits, rejoinders, responders). We start by writing the gambits. This forms the backbone of your bot's control over the flow of conversation. Gambits are easy to script.

t: Do you have any pets?
t: How long have you had your pet?
t: Did you know that dogs are bigger than ants?

OK. So they are easy to script. A bot with these would say those things in order, each time it had control and was in that topic. But what makes this fun or interesting?

This immediately brings us to the question of how to think about *what* to say. Our chatbot Suzette treated some topics very impersonally. She didn't have opinions, but she had an interesting collection of things to tell you, hoping to keep your interest.

She had a topic (of hundreds) about burial customs. Suzette wasn't prepared to answer questions about this topic (no responders) and offered only a single personal opinion. And she only had a single rejoinder, if the user mentions anything about the Incas in response to Egyptian mummification.

topic: ~BURIAL_CUSTOMS (burial bury dead embalm exhumation exhume grave~n mummify)

t: People do really weird things with the dead. The Egyptians, of course, mummified theirs so their soul would have a place to inhabit.
a: ([Peru Inca]) Yes, the Incas mummified their dead also.

t: In India, the rite of Sati required the widow to lay herself down next to her husband and be burned alive on a funeral pyre. Supposedly voluntary, sometimes women who did not wish to commit sati were drugged. Even in 2006, two women jumped into their husbands' fire pyres.

t: In Japan, in the rite of Sokushinbutsu a monk causes his own death. For 3 years he eats nuts and seeds and does activity to use up his body fat. Then he eats bark and roots for another 3 years. He drinks poisonous tea and vomits to lose body fluid. Then he sits lotus in a tomb fitted with a small breathing tube and a bell. He rings the bell each day. When the bell no longer rings, the tomb is sealed.

After 1,000 days, the corpse is put into a temple for viewing.

t: In Manila, cemeteries are so overcrowded, people rent a large plot for up to 5 years. At the end of the lease, the bones are then placed in a sack and moved to a smaller, cheaper plot and the bigger plot re-rented.

t: Feel free to just donate my body to a medical school.

So this complete topic is extremely low on interactivity and extremely low on revealing anything about the being who is Suzette. Never-the-less, it is sort of interesting. And when you have a bunch of such topics, all the user has to do is lead the conversation to some area, and Suzette will happily regale you with interesting material.

But a better topic will actually interact with the user and share personal opinions and information. Interacting with the user will generally mean that we ask the user questions and volunteer our answers to those same questions. If you ask the user something personal like *do you like apples,* it is expected you will give your opinion on the subject as well – a sharing of intimacy.

Whether you tell the user your view first and then solicit theirs or ask them and then volunteer yours is a matter of choice. Partly you want to vary your style, so you may change the order at times. But generally you are better off asking them first and answering for your bot second. If you tell them first, you often stifle their opinions. Or because you have told them what you are talking about before asking a question, they may volunteer their own answer before you ask the question. This become a tricky scripting problem because you have to detect they did this (because it would be stupid to ask them after they have just told you).

The other reason to ask first is that you can prepare a lot of easily scripted rejoinders when you can anticipate their answers. One of the good things about asking questions is that their response is generally confined to answering you. This makes it easy to write rejoinders that can interact with them. *Do you like* swimming? Obviously asking yes-no questions makes it really easy to write rejoinders. But too much of that will become artificial. You can add some variety by offering choice questions like *Would you rather date an ugly millionaire or a handsome bum?* It's still limited to two rejoinders. The really good choice questions appear to leave the choices to the user, like our earlier *What's your favorite way to kill someone?* Yes, there are lots of ways, but you can pretty much cover the field with about 20-30 rejoinders. And having a great rejoinder to what the user says will make your bot seem intelligent.

Here is another complete Suzette topic. It has different modes of engagement, including a joke.

topic: ~ATHLETICS (all-American athletic pickup_game team ~athlete ~sports ~team_sport)

t: Do you play any team sports?
a: (~yesanswer) I don't really care, I'm not sure why I asked. Are you any good at your favorite sports?
a: (~noanswer) I loathe sports.

t: Who was the fastest runner? Adam. He was first in the human race.

t: PLAY () I used to play volleyball. I've tried hang-gliding. The former was fun, the latter is dangerous.

#! do you play any sports?
*?: (you < * [play participate do] < * [athlete ~team_sport athletic sport] < ![baseball basketball]) ^reuse(PLAY)*

#! what team do you support?
*?: (what *~2 team * you [support favor]) I don't care for any sports teams.*

The above topic did express Suzette's opinions. Our Angela bot went farther and did everything in character in every topic. Her questions were personal and her answers were always from her history or her opinion. Everything she said revealed more of her character and history. Nothing was impersonal. She was a friend who shared things with you, told you her personal stories of triumph or disaster and listened to yours.

Issues of Length

Which brings us to the question of length. Actually two questions about length. How long should a gambit be and how big should a topic be?

As for how long a gambit should be, you have to consider your audience. Is this mobile or web? Mobile will want really short answers (tweets) due to limited screen space. The web can be longer, but will your audience want to read a lot at once? Too much material at once may stifle interactivity, if your goal is a conversation. If your goal is an infodump on something, then size is less of an issue. Nowadays, however, we try to keep all output to 140 characters or less, regardless. People are less into reading.

As for how long a topic should be, obviously one can go on for a long time on any subject. But our experience is that most people will drop out of most topics after about 8 gambits. A really long topic is usually just a collection of sub-themes. Imagine a topic about money covering these areas: winning the lottery, giving to charity, investing, aiming for retirement. This could all be written as a single topic, but then if the user asks a question that sends us to this topic and we end up staying and volunteering gambits, we will be taking these themes in the order written. So if the user enters the topic by asking about investment, after an initial appropriate response, the bot would continue by talking about winning the lottery. Not well focused. So nowadays we prefer to write the sub-themes as topics, linked from the main topic. This allows the bot to react in a tightly focused manner. So the money topic might look like this:

topic: ~money()
t: ^gambit(~lottery)
t: ^gambit(~charity)
t: ^gambit(~investing)
t: ^gambit(~retirement)

This means the bot might start talking about money, walking through each topic in turn. On the other hand, if the user inquires about investing, the bot would respond from that topic and remain there til done.

Story Topics

Some of our topics in Angela were personal stories. Obviously the gambits will tell out the story over time, and usually there is little expectation that the user will say much other than *go on* or *continue or wow.* Obviously you do want responders for questions they might naturally ask. And you have to write your control script to specially manage story topics different from other topics. Why?

In a normal free flowing conversation, the user might say something that takes you to another topic. That's fine, conversations often deflect and wander around. But stories don't. You can't have the system leaving the story partway. It has to remain focused on completing it. You can't return to it at some later time. So the control script has to allow the system to answer questions from any other topic if the user does ask something unexpected, but it has to be smart enough to force the flow back into the story topic to finish it.

There is something the user might ask during the story that you might want rejoinders for: *Why?* In Angela's story about her blog, she actually does interact in the middle of her story by asking the user:

t: Have you ever had to write regularly, outside of homework?
#! why
a: (~why) It's quite different having to find your own topics to write about. No teacher telling you what to write about.

Whenever you make any kind of assertion or question the user might ask *why?* Having an answer for that allows you an opportunity to deepen the conversation. In fact, we generally put a why rejoinder after *every* gambit we make in all topics. In Rose's assassin topic, if the user ever says they want to kill or murder something, she has a responder:
#! I want to kill you.
*s: ('I * [kill murder]) What's your favorite way to kill?*
And a why rejoinder:
#! why?
a: (~why) You are interested in killing, so I'm interested in how you like to do it.

Why's also help you improve your writing. If you have to justify why you made a statement or question to the user, you will get your rationale correct. We put why's on gambits but not usually on responders. That's because gambits are things the bot volunteers to the user. Responders are answering things the user said, so usually they are happy with that. Not that putting why's on them is bad. It's just more work and the user has already been rewarded by your answer. Rewarded?

If the user is conversing with a bot, the user wants to feel understood. Every time the bot directly and appropriately handles what the user said, they feel rewarded. So if the user ask *Do you have any siblings* and the bot says *No, I was an only child,* the interaction is small but precise. The user feels heard and he got his answer. He's not likely to ask *why.* On the other hand, if he asks *what is your favorite pet animal* and you say *dogs* he might follow with *why?* You have to use common sense to decide when to use *why* on a responder. But on gambits, since the bot is leading the conversation, the user might well ask *why* on a question to understand what your interest in the answer is and why he should answer it. And on statements that you volunteer, since you can't express all your thinking into the gambit, if the user wants more depth, they may ask *why.* And it's just fun asking why (it's what two-year-olds spend a lot of time doing). And most bots can't answer the question, because their authors haven't thought about it.

Reactor Topics

A different kind of topic is one that just issues one-liner reactions to things. It's a kind of table. We have one for music, for example, to react to the user's liking of some genre. Ones for movies, pets, diseases, celebrities, food, drink, countries, art,etc. The reactions are similar to a rejoinder set, except it can be called from any number of places. For example, in pets it might be called as:

#! I have a pet lizard.
*s: (I * ~own * ~animals) ^respond(~pet_react)*

The reactor topic itself is just a series of responders like this:

topic: ~pet_react nostay()
#! parrot
u: (~bird) Birds are nice. That's why I have chickens.

#! snake
u: ([reptile snake alligator]) I've got a friend who keeps big lizards. They look really fierce.

#! ant
u: (~insects) Bugs certainly don't take up much room.

Often times the reactor topic itself has no keywords so it won't accidentally react in a wrong context. And it is usually declared NOSTAY so the current topic remains the calling topic.

While reactor topics have no depth, they have a broad coverage. One user was creeped out by Angela and wrote: *This app is very fun at first but then it gets creepy. She knows what Justin Bieber, Twitter, and one direction is. How does an animal app know such things?* Lots of reactor data is the answer.

Keywordless Topic

A special topic for holding responders is one that has no keywords and is directly invoked by the control script when normal topics fail to match. It is basically just a grab bag of responders that have no current topic home but are specific to the character. For example:

#! call me an ambulance
u: (call me an ambulance) If I were Siri, I'd say "Hello, an Ambulance". But I can't call one for you since I don't have my phone with me.

Quibble Topics

Since one can never anticipate everything the user will say, one is left with either ignoring it and moving on, or quibbling with it in some way. It's very important to have these quibbles because user's like to think they were heard. Ignoring the user just makes them mad. ChatScript ships with an extensive set of quibbles of all sorts but it's only invoked most of the time so if the user keeps quibbling himself, the system will break off and guide the conversation somewhere else. Quibble topics are similar to the keywordless topic in that they don't have topic keywords and are directly invoked by the control script. The main difference is that the quibbles are universal, suitable for any bot, and not specific to the character. And quibbles can pick up on mere words of the input and not care about the complete meaning. A sample quibble might be:

#! because you know what it's like to live there.
s: (because you know) [I don't know any such thing.]
[Whatever I once knew, I have long since forgotten.]

Repetition: Keep/Repeat/Random

One clear thing that makes a bot stand out non-human is saying the same thing word for word in separate volleys. This is such an important issue that ChatScript is designed to deal with it in multiple ways, even detecting it from the user. But mostly all we are concerned with is repetition from the bot.

First, the engine tracks what the bot has said for the last 20 volleys and by default will simply refuse to repeat anything. This means that the rule issuing the repeated output fails and some other rule will have to pick up the slack. You can override this on a topic basic by putting the topic control word REPEAT at the top. You can also put ^repeat() as a command in the script of a specific rule. Usually we don't do either. For a conversational bot, you expect a normal user will not ask the same thing and if they do, we want the system to find another answer or quibble if need be.

ChatScript also by default erases a responder after use which you can override with KEEP on the topic or ^keep() on the rule. All control topics, things devoted to the system, have KEEP on them. Topics which are about the actual conversation generally do not. But you could imagine that if the bot is supposed to be an information bot and not a conversation bot, that you might put keep on just about everything.

ChatScript supports a mechanism to randomly dish out gambits from a topic. Suzette used this. All our later bots do not. Random gambits imply that there is no real flow or intention behind your conversation. The only place we use it is in the jokes topic where the bot issues unrelated one-liner jokes. And it is not essential to randomize even there.

ChatScript also supports a mechanism to hunt for responders randomly in a topic. We occasionally use this in quibble topics, but mostly not. We prefer to have the most specific and longest patterns get first crack at the input since they will then provide the most appropriate dedicated response.

ChatScript also allows you to write multiple outputs to be randomly chosen. We do this all the time on quibbles, which are system and kept, to provide variety. Some people seem to start by writing ordinary responders or gambits using multiple choice. We don't. Writing alternative stuff clearly increases the work of authoring, but will the user actually see it? We view our customer as an individual having a continuing conversation with the bot. They will not restart the bot as a new user nor compare their conversation with ones other users are having. So they will never appreciate all that alternative text that didn't display and gets erased when the rule is killed.

Maybe, for some reason, your user will not have continuing existence from one session to another. Each time the system will not know them. Then it make make some sense to offer things randomly, so they cannot predict their experience and it will all seem fresh to them.

Emotion

Emotion is an interesting problem because it's extremely difficult to put text emotion into prerecorded output text. In the case of Angela, since we were able to control an avatar with gestures, we could also output commands to the app to control her behavior. So if she was insulted her avatar could be told to be angry. Angela recognized when you insulted her. She was very touchy and went sulky, refusing to interact much for a few volleys until her anger wore off or until you apologized or bought her a gift. As one user wrote: *Oh yeah and she always is so rude to me and "putting me down" but when I'm mean back she gets furious!*

Putting pre-canned gestures on outputs is OK when you have a controllable avatar, but emotion is usually more dynamic and builds up as a result of continuing interaction. Part of the reason our first bot, Suzette, won the Loebner's was because she expressed dynamic emotion. She did this in two ways. First, since being insulted and hit upon is common for a female bot, she had lots of rules to detect this. She not only deflected sexual innuendos but added warnings about your behavior and if you did too much of it in too short a space of time, she would warn you and then refuse to talk to you for 5 minutes.

And Suzette monitored user input for repetition. So if user just kept saying the same thing, she would pick up and comment on it initially, and start getting cross if it was done too often.

She also tracked information on the conversation. Long sentences from the user indicated interest in the topic, short ones disinterest. Saying you were bored, clear disinterest. Agreeing with her by answering yes a lot or complementing her indicated you liked her and were sympatico. Disagreeing by answering no a lot or by insulting her indicated you disliked her or weren't her kind of person.

So she kept track of attention and affinity. Attention could cause her to change topics if you seemed bored but if you were bored a lot then it reduced your affinity with her. When your affinity was too positive toward her, she became neurotically insecure. She didn't deserve your love and interest in her and expressed it verbally. Similarly, if your affinity went too negative, she became paranoid or hostile. She worried who was listening, why you asked the questions you did, and if things got too bad, began verbally speculating to herself how she might kill you. She tracked these states as a result of what the user said and what she said. And had additional emotional topics that could add extra sentences in front of the normally generated conversation. So if you asked her *do you have any pets,* you get her normal answer with any emotional spice occasionally added in a sentence in front. Here is some of her hostile spice topic, which gets progressively worse during the conversation:

```
?: () ^preprint(Why should I tell you? )
?: () ^preprint(Go figure it out on your own. )
...
u: () ^preprint(Do you actually have any friends? )
u: () ^preprint(I'm sorry. I didn't mean to speak above your abilities to understand. )
...
u: () ^preprint(They all whisper behind your back. )
u: () ^preprint(Tell me about your weaknesses. )
...
u: () ^preprint(This isn't worth my time. )
u: () ^preprint(You're a waste of space. )
...
u: () ^preprint(I dream of ways to torture you. )
u: () ^preprint(I want to wipe your existence off the face of the planet. )
```

Remember, this is from Suzette, not from our commercially available Talking Angela bot widely used by small children.

Conclusion

Your goal is to create the illusion that your chatbot understands the user. This means trying to minimize those awkward moments where your bot says something completely unrelated to what the user said and maximize the rewarding moments when the bot responds completely appropriately.

To do this, you want to employ the concept of *sente* from the game of Go. Sente is having the initiative, making the first move in an area, being in control. Every time you have sente you are likely to make points over your opponent, and every time he has sente you are starting to lose points. Every time the user has sente, your bot is at risk of saying something foolish because you can't guess what he will do. So how does a bot get and maintain sente?

The bot needs to ask a fair number of questions. They need to be reasonable and interesting in context. They need to be varied in style (yes-no, multiple-choice, open-ended) where you have rejoinders to cover most every reasonable response. You can intersperse questions with statements, some of which answer the question you posed and some of which offer followup elucidation or commentary. Vary the number of statements between questions a small amount, so the behavior isn't predictable. Sometimes merge the two, like *I love bananas. What's your favorite fruit?*

Whenever you make a statement, ideally you would put some kind of hook in it- something that will invite the user to ask an obvious question, for which you have prepared a reply. If you have a responder for *do you have any pets*, ideally you'd have an unusual answer like *I've got a pet dragon.* This leads the user to expressing skepticism, after which you can clarify it is a Komodo dragon. And when they ask why, you have an answer for that. So they think they are in control of the flow when really they are just following your leads.

Most of all, think of your chatbot as a work of art: a sculpture of words or a painting of sentences. It needs a point of view, something to express. And it needs to have an emotional impact on the user.

Instant ChatScript

So you want to write a chatbot. Presumably one that aims at entertaining conversation. Here's how to get started quickly, presuming you have a copy of ChatScript downloaded onto your desktop...

A Simple Chatbot

Create a new topic file in RAWDATA/HARRY. Let's say it's called pets.txt (it can be any name). In this file, using a text editor of some flavor, write a topic, let's call it ~pets. It starts out like this:

topic: ~pets()

Now write some gambits, what you want the chatbot to say when it has control. And responders to react to things the user might say. Here are two gambits:

t: Do you have any pets?

t: I have a dog.

And here is a responder to react to the question *do you like snakes?*

#! do you like snakes

?: (do you like snakes) I love all animals.

Every responder should be prefaced with a comment that is an example of what user might say.

Your topic needs keywords, something that will make the system drive the user to that topic on appropriate input. Here is our ~pets topic header with keywords:

topic: ~pets (dog cat pet animal bird fish snake)

Now we are ready for a quick test of your new code. You could test every line of code as you write it, but that's kind of inefficient.

Launch Chatscript. In Windows double click on *ChatScript.exe*. In Linux be in the ChatScript directory and type *./ChatScript32 local* At the ChatScript prompt, enter some user name. The bot will say *Welcome to Chatscript,* because there already exists a bot framework called Harry, who has an introductions topic, a childhood topic, a keywordless topic, and a bunch of quibble topics. You could be conversing with him now, as he was shipped. But since you have added an additional topic on pets, you want that included. So now at the prompt type *:build harry.* The system will build your revised bot.

Now type: *do you like snakes.* This contains a keyword of our ~pets topic. It will match our one responder and reply. We are now in the ~pets topic and will, except for quibbling, issue the next gambit from there unless the user says something that takes the bot to a different topic. You can improve this topic by adding more responders and more gambits.

You can create any number of new files containing other topics, save them, and when you are ready, go through the build procedure again by typing *:build harry*, and then continue chatting with your revised bot. You don't have to close the files. You can just switch back and forth between an editing window and the ChatScript window.

A More Interactive Chatbot

Don't just ask questions, prepare rejoinders to react to the user's answers. What can you expect as answers from the first question? The logical responses are yes, no, and naming some pet directly. Rejoinders for this are:

t: Do you have any pets?

#! yes

a: (yes) Great. You like animals.
#! no
a: (no) You don't like animals?
#! I have two parrots
a: (parrots) Birds are nice.
#! I have a cat
a: (cat) I prefer dogs.

You can add rejoinders to your file, rebuild, and try again.

A More General Chatbot

Users can reply in lots of ways that are similar in meaning but different in words. You want to generalize your patterns to catch as much as possible. You don't want to be so specific that you can't match something appropriate to what the user says.

You can start to generalize your words by using their canonical forms in both topic keywords and pattern parts of a rule. Use singular nouns like *parrot* instead of *parrots*– they match both singular and plural forms of the noun. Your response really doesn't care how many parrots the user claims to own. Use infinitive verbs - *be* instead of *is* or *are* or *were*, *walk* instead of *walks* or *walking* or *walked*. Infinitives will match any tense of the verb. Use base adjectives and adverbs, *big* instead of *biggest* and *slow* instead of *slower.* And use digit numbers instead of word numbers, *2* instead of *two* or *pair* or *double* or *second.*

The next level of generalization is using concepts to broaden the reach of a keyword. Instead of "yes", write *~yes* because *~yes* is a predefined concept for lots of ways to say yes. When you want to broaden a word, you can use *:concepts* to ask what choices exist. Just type:

:concepts yes

yes maps to *~yes* and *~yesanswer.* Pick whatever seems close to you and change your topic rule accordingly. If you ask how to generalize *parrot* you will see a choice is *~bird.* That sounds reasonable, whereas *~pet_animals* sounds too broad for your output *Birds are nice*. If you can't find a concept that seems appropriate, you can manually insert a list of words you want just by putting them in brackets. E.g.

a: ([parrot bird canary finch swallow]) Birds are nice.

You can generalize responders in several ways. First thing, stop writing out the entire user input. Drop off small useless words and keep just the essential keywords, putting them all within << >>. This tells the system to match a sentence with all those words, in any order, without requiring contiguity. And use the canonical forms of words at the same time.

?: (<< you like snake >>) I love all animals.

Then we could generalize the pattern words as concepts. *:concepts like* will tell you that *~like* is a choice that sounds reasonable. And *snake* could generalize to *~reptiles* or *~animals* or *~pet_animals*. The broadest choice given your output would be *~animals* so this becomes

?: (<< you ~like ~animals >>) I love all animals.

But when you broaden the keywords of responders, you may also need to broaden keywords of the topic as a whole because they act as gatekeepers to getting into the topic. So you should add *~animals* to the topic keywords list. In fact, if you did that, you could remove all the specific names of animals as keywords because they will be subsumed by *~animals*.

And that's it. You can go write a whole lot of topics with just that knowledge.