Chapter 8: 8. I Like Your Personality: Giving a robot an artificial personality by applying simulation techniques to conversation and emotion - 45 pages

Introduction (100-150 words)

*When a person thinks of a robot with “artificial intelligence”, what many consider AI is a robot that has emotions, feelings, a state of mind, and some sort of model or concept of how humans think or feel. We can call this form of AI an “artificial personality”. While giving a robot feelings is definitely way beyond the scope of this book (or current technology), what we can do is create a simulation of a personality for the robot using standard computer modeling techniques like Monte-Carlo analysis, and finite state machines.*

Level

*This is an advanced topic, but will be presented at a level to make it accessible to the intermediate audience (which is the overall intent of the book, after all)*

Main Headings (3-5 main chapter headings)

* *The Turing Test*
* *The concept of an Artificial Personality*
* *The Art and Science of Simulation*
* *Creating an Emotional State Machine*
* *Creating a Model of Human Behavior*
* *Integrating a AP (Artificial Personality) into our Robot*

Skills learned

*This chapter is actually not an AI (Artificial Intelligence) technique applied to robotics, but the use of simulation and game theory to create the simulation of a personality for the robot, with an attitude, and even a sense of humor. The robot uses a state machine to determine its internal mood (happy, sad, tired, grumpy, playful) and it responds to its environment using verbal cues and body language to match its mood. This is not a real personality, but the simulation of one, and is different from creating a real AI with emotions as a flight simulator is from a 747 Jumbo Jet.*

Introduction

When a person thinks of a robot with “artificial intelligence”, what many consider AI is a robot that has emotions, feelings, a state of mind, and some sort of model or concept of how humans think or feel. We can call this form of AI an “artificial personality”. While giving a robot feelings is definitely way beyond the scope of this book (or current technology), what we can do is create a simulation of a personality for the robot using standard computer modeling techniques like Monte-Carlo analysis, and finite state machines.

**Skills Learned**

Transaction based conversation

Designing a Chatbot

Natural Language Processing (advanced)

Simulation tools

Monte Carlo Analysis

**Tools Required**

Python

Imagination and some writing skills

**What is an Artificial Personality?**

Hollywood and the movie industry have turned out some very memorable robots. You can think of R2D2 and C3PO, the “Laurel and Hardy” of science fiction. What do you like most about these two? Could it be their personalities? Think about this a bit. Even though R2D2 is mostly a wastebasket shape with a dome head and no face, he has a definite personality. You describe him as “feisty”, or “stubborn”. The website Robots and Androids (<http://www.robots-and-androids.com/R2D2.html>) described him this way:

“The droid is shown as being highly courageous, unwavering in loyalty, and a little bit stubborn. He never backs down from a mission, even when the odds appear to be stacked against him. His personality is often contrasted against that of [C3PO], who is fussy and timid.”

This is pretty impressive for a robot who never says a word and communicates with beeps and whistles.

What were other movie robots that made an impression on you? Certainly WALL-E, the lovable trash compacting robot from the eponymous movie is a favorite. WALL-E also had a small vocabulary, consisting of only his name, much like a Pokémon. WALL-E displayed a lot of emotion, and even developed hobbies, collecting and repairing old trash. You may also remember M-O, the tiny, obsessive cleaning robot that gets frustrated with all of the dirt WALL-E brings in. M-O packed a lot of personality into a small package.

So one thing that we might do as robotics creators and designers is to imbue our robot with some sort of personality. This has an advantage of letting humans relate to the robot better. It also gives the false impression that the robot is much smarter and is capable of more than it really is. This does give the advantage of the robot being more engaging and interesting.

We can also infer from the examples given to us by R2D2 and WALL-E that less can be more when it comes to communication – we need to not just have words, but also body language and sound.

What we are going to do in this section is develop an artificial personality for our robots. While it is impossible for us to give a robot actual emotions or feelings, however you might define that, we can create a simulation of personality that will provide a convincing illusion. I think this is a meaningful exercise because current state of the art in robotics demands some sort of personality and consistent demeanor from robotics that interact with humans.

There is a lot of work going on in this area right now with digital personal assistance, like Apple’s Siri and Google’s Alexa. Note that these robots, or artificial intelligences, have distinct names and voices. I would feel that they are very similar in personality and capability. There is some distinction when you ask Siri or Alexa a personal question, like how old they are (Siri is considerably older than Alexa).

In this chapter we are going to take tools from the science of simulation – specifically, state machines and Monte Carlo Analysis – and use them to form a model of a personality for Tinman, the robot. We already have a fairly powerful tool, in the Mycroft speech system we used to tell knock-knock jokes. We will be extending Mycroft with some new skills and capabilities, as well as developing cybernetic emotions, both for our robot, and for the robot’s opinion of what we, the humans, are feeling.

I want to emphasize that we are simulating emotions, not creating an emotional robot. Our simulation bears the same resemblance to real emotions as a flight simulator bears to the Space Shuttle – both provide the same information, but the Space Shuttle flies around the Earth in 90 minutes, and the flight simulator never moves.

**The Turing Test**

Alan Turing proposed his famous test, which he called “The Imitation Game” in a paper titled “Computing Machinery and Intelligence” published in 1950 in the journal “Mind – A Quarterly Review of Psychology and Philosophy.” (See: <https://www.abelard.org/turpap/turpap.php#the_imitation_game>). In the original text, Turing imagined a game where a player would have to guess the sex – male or female – of a hidden person by typing questions on a teletype. Then he suggested that a truly intelligent machine would be one where you would not be able to distinguish if the hidden personality on the other end of the teletype was a human or a computer software program.

**NATHAN**

Do you know what the Turing Test

is?

CALEB reacts - immediately knowing what NATHAN has just

implied.

**CALEB**

... Yeah. I know what the Turing

Test is.

NATHAN waits.

**CALEB (CONT'D)**

It's where a human interacts with a

computer. And if the human can't

tell they're interacting with a

computer, the test is passed.

**NATHAN**

And what does a pass tell us?

**CALEB**

That the computer has artificial

intelligence.

From: **Ex Machina**. Script by Alex Garland. Copyright Universal Pictures

NOTE: The movie called **The Imitation Game** starts Benedict Cumberbatch as Alan Turing but features his role in breaking German codes in WWII as part of the mathematicians of Bletchley Park. The title refers to Turing’s original name of the famous test that bears his name.

These days, you may talk to computer software many times a day and not realize that you are not speaking to a human. “Robocalls” and “chatbots” may call you on the telephone, or engage you in a fake political conversation on Twitter. The Turing test has been won by the machines –but have we developed intelligence in computers? Not at all – we have just become very clever at simulating conversation.

Recently, robotics experts have suggesting replacing the Turing Test with a more demanding and difficult assignment – assembling a piece of furniture from Ikea using the directions provided. (<https://motherboard.vice.com/en_us/article/vvbqma/the-plan-to-replace-the-turing-test-with-a-turing-olympics>)

We are going to use another tool that Alan Turing mentioned in his paper –state machines. We will use state machines to define and model the emotions of our robot.

Let’s go back to the concept of a “chatbot”. A working definition may be a software program designed to engage a human in conversation, or to interact with a person via voice or text. While most chatbots are up-front about being computer generated, there are a lot of chatbots that do not – including Twitter chatbots that seek to influence elections or public option. Many chatbots are gainfully employed answering tech support phone calls, making marketing calls, and entertaining users as the dialog of non-player characters (NPC’s) in games.

According to chatbotslife.com, in their article “Ultimate Guide to Leveraging NLP [Natural Language Processing] and Machine Learning for Your Chatbot” (Stefan Kojouharov, September, 2016), chatbots come in two flavors: **Retrieval based** and **Generative based**.

**Retrieval based** chatbots rely on stored phrases and words, and use their software decision making to select which reply is most appropriate. There may be some keyword recognition and noun – subject insertion involved, but the main action is to select the most appropriate phrase.

**Generative based** chatbots are making up new sentences based on parts of speech and the robot’s divination of your intent. They can be thought of as machine translation engines that translate the input (your text or speech) into an output (the robot’s reply) [ibid.]. As you might imagine, the generative type chatbot is far more difficult to achieve, which is why we will be using a retrieval based approach.

There are two other details we need to attend to. Chatbots can be designed to handle either short conversations, or long conversations. The vast majority of chatbots, and that includes digital assistants like Siri, Alexa, and Mycroft, are designed for very short conversations. “Siri, what is the weather?” “There is a 20% chance of rain. The High is 88 degrees”. That’s it – the whole conversation in three sentences and two interactions. If you ask another question, it starts a new conversation with little reference to the previous one.

A more difficult task is to have a longer conversation with several interactions, and even a selection of topics. This requires the computer to keep track of **context**, or what information has been discussed and might be referred to again.

We will be attempting to teach our robot to be able to have medium-length conversations on a seven-year-old level. I’ll define medium length to be between two and six interactions.

**The Art and Science of Simulation**

What is **simulation**? A Simulator is a computer model of the physical world. You are probably familiar with flight simulators, which provide the sensations and interactions of flight without leaving the ground. There are also a lot of other types of simulations and simulators. We could have a medical simulator that mimics diseases or responds to treatments. It could be a financial simulation that models returns on the stock market based on trends. There are structural simulations that model the loads on bridges and buildings to see if the materials are adequate.

The most common way of creating a simulation is by building a physics model of the item under test. For a flight simulator, this means plugging in formulas for the four forces on an airplane or helicopter – lift, gravity, thrust and drag. Each factor has parameters that affect its performance – for instance lift is a function of the speed through the air, the weight of the aircraft, the size of the wing, and the angle of attack, or the angle between the wing and the wind. Vary any of those, and the lift changes. If the amount of lift exceeds the force due to gravity (i.e. weight of the aircraft) then the aircraft flies. The simulation sets up a time step interval, just like our control loop for the robot, and computes the forces on the aircraft for each time step. We can then apply controls and see how our model performs. Models just like this are used to predict performance in advance of building a prototype or test airplane.

Another type of simulation is called a **Monte Carlo Model**. The Monte Carlo method uses probability theory to replace sophisticated physical models with a variation of random numbers that approximates the same result. If we wanted a create a computer model of flipping a coin, you wouldn’t spend a lot of time determining the physical properties of a nickel, or modeling the number of flips in the air based on force, you would just pick a random number from 1 to 100, and say the result is heads is the number drawn is less than 50 and tails if it is greater than 50. That, in essence, is the Monte Carlo method. There are a lot of physical processes that can be approximated and studied using this technique.

We can apply Monte Carlo analysis to model people going through security at an airport. If you had a copy of a typical schedule for the airlines, and the average number of passengers per flight, you would know the daily traffic at the airport. The difficult bit was modeling when people would arrive for their flight. Let imagine that we commissioned a study, and determined roughly that 50% of people arrive 1 hour early, 25% arrive 2 hours early, and the rest are evenly distributed between 2.5 hours and 30 minutes, with 1 passenger out of every 200 missing their flight by being late. This information allowed us to create samples sizes of passenger arrivals and thus an estimate length of lines at security.

We would add some distribution of values of how long it took to get through security, and we would have our model. We have a list of flights, and for each flight we generate a distribution of passenger arrival times based on the study by assigning random numbers to each passenger and separating them out into groups based on arrival time. Let’s assign random numbers from 1 to 100 to our 212 passengers arriving for the 8:00 am flight from Dallas to Washington DC. Now we assign an arrival time based on that number – if the number is from 1 to 25, the passenger arrives 2 hours earlier than the flight (06:00). If the number is from 26 to 75 (the next 50%) then they arrived 1 hour early (07:00). The rest of the passengers, having numbers from 76 to 100, are assigned random times between 2.5 hours early to 30 minutes early. And we pick one unlucky passenger out of our 200 to miss the flight completely by arriving late. Since there is some variability to people who intend to arrive at exactly two hours ahead, but are delayed or advanced slightly, we can “dither” or add a small random factor to each arrival time of plus or minus 10 minutes. Now we have a statistically correct distribution of people arriving for a flight. Now repeat this process for the other 1849 flights leaving today from this airport.

<<illustration – passenger arrival time distribution>>

You can see from this model that we can still put a great deal of science into picking random numbers.

As you might have guessed from my example, the true secret of making this technique work is picking the correct **distribution** of random numbers. You may think, “Hey, random numbers are random number, right?” but this is not correct at all. You have heard of the standard distribution, or the bell shaped curve. Most of the values are in the center, with fewer and fewer as you go away from the center. Many natural processes follow this bell shaped curve, including grades on a test, how tall people are, or how many grapes grow in a bunch. When you do Monte Carlo analysis, most often we use some form of normal or standard distribution.

Can you think of other types of random numbers? The second most common type of random numbers are a uniform distribution. Each number has the exact same probability of being selected, which make for a flat distribution curve. Another name for this uniform distribution is white noise. Uniform distributions do occur in analysis, and if we want to add noise to an image, for example, we would use a uniform distribution. But in modeling emotions and people, normal or standard distributions are the rule.

You may have heard of pink noise – this is a distribution where the probably decreases with the increase in value of the number. Another way of saying this is the probability is equal to 1/value.

You may find that in using Monte Carlo modeling that a standard distribution or uniform distribution just won’t work. Then you can create a custom distribution like I did in the airport example, where we used resampling to change a uniform distribution to a custom distribution fitting our passenger arrival model.

<<Illustration of distributions>>

We are going to be making a personality simulation for our robot, and we will be using Monte Carlo modeling plus a state machine.

**An Emotion State Machine**

What is a state machine? We covered these in the section on systems engineering – state machines are a technique for understanding or modeling automation or computer programs. A state is a set of conditions that exist at the present. I like to think of a state as being a set of functions that are constrained by limits. The machine (our robot) change from state to state based on some event that causes the state to change.

Let’s work through a quick refresher example. Let’s take a slice of bread. When we get it, it is already baked and sliced, so it’s initial state is as a slice of bread. If we subject the bread to infrared radiation, i.e. heat, then the surface becomes caramelized and we call that “toast”. The state of the bread has changed, along with its taste and texture, from baked bread to toast. The event that caused that transition was the act of putting the bread in a toaster. This is pretty simple stuff, and I’m sure you have encountered state machines before.

Now let’s think about our model of robot emotions. We can start by listing what emotions we want our robot to have:

Happy

Welcoming

Friendly

Curious

Fresh

And the opposite of those emotions:

Sad

Distant

Stranger

Frustrated

Tired

These are the list of emotions I wanted to simulate in our robot. I looked at the different interactions the robot might have, and how a human version of the robot would react. This line of development was partially inspired by the work at the MIT Media lab with Kismet, a facial expression robot that was used to model social interactions, much like what we are trying to accomplish here. (see Breazeal and Scassellati: “How To Build Robots That Make Friends And Influence People”, MIT, 2000).

The robot starts in the state of **Happy**, **Distant**, **Curious**, and **Fresh**. That is to say, it feels happy, it is not in an interaction, so there is no-one to be friendly to, and it will be curious about its surroundings. As the environment changes, the robot’s state will change. For example, we have about a two-hour run time on Tinman’s current battery. It starts off in the “Fresh” state, and will get more and more tired as it approaches the two-hour mark. We are going to use a simple timer to create this state, but you could also have a voltage sensor in the Arduino provide information about how fresh the robot’s power supply is at. Each emotion exists along an axis that all cross a central or neutral point.

We will drive the “happy-sad” axis based on how many conversation “points” the robot is earning. We’ll cover this in detail in a few paragraphs. We will also be describing an empathy function for our robot that simulates the robot sharing feelings with you. The Distant –Welcoming aspect is controlled by how the human interacts with the robot – if you are acting friendly to the robot, then it will be welcoming. If you are new, the robot will start off cautiously in asking questions or interacting. If you are not answering questions, then the robot will become more distant. Likewise, the Friend-Stranger aspect is based on how much the robot knows about the person it is talking to. Finally, the Curious – Frustrated axis is based on how hard the robot has to work to get information or to accomplish a task. If it keeps failing at a task, or not getting rewards, it will become frustrated in its expressions and vocabulary.

How does a robot with no face express emotions? Remember we started talking about Hollywood robots, many of whom have distinct personalities without having faces or even heads. We will use body language with the robot arm and changes in vocabulary to express our simulation of emotions. And we will have the robot be open about how it is feeling and why.

Our goal for this part of the robot development is to provide a human interaction framework that invites people to talk to the robot and feel welcome or wanted. I’m modeling this interaction from watching my grandchildren. I wanted a robot that they would enjoy and find interesting.

**Playing the Emotion Game**

What we want to do is develop the ability for the robot to convey **intent**, or to provide a simulation that the robot has needs and desires. We are going to do this by creating a game that the robot will play with humans. The robot will be trying to score points by getting the human to interact socially. The robot will gain points by getting information from the person so that it can “get to know them better”. The trick for us is, we need the robot to keep this information and remember it.

So what is this game? What we want the robot to do is ask questions of another person and to use conversation to get personal information out of them, so that the robot then can use that information in replies. We will be saving all this information and using it to modify the actual code the robot is running, thus implementing machine learning in the conversation. The robot will be using the same type of point-reward system we used in training the robot arm back in a previous chapter. Once the robot has “scored” by learning a fact, it will no longer get rewarded for that fact, and will go on to get other facts. Once it has learned all the facts, it will end the conversation. In practice, I think most users will want fairly short conversations, so we are setting our goal to have between two and six interactions per conversation.

The game will be played like this: The user will always initiate a conversation by saying the robot’s wake phrase, which right now is “Hey, Albert”. The robot will respond using the Mycroft speech engine with a beep. The user will initiate a conversation by saying some version of “hello”. The robot will then try to earn rewards by getting points, which it does by asking questions. The emotion engine will respond to the human answers by modifying the robot’s emotions. We will rank questions by importance – let’s say the robot gets 10 points for learning your name, 9 points for learning your age, and so on. Once a fact is learned, no more points are earned, so the robot won’t repeat questions.

The facts we want the robot to know about each person are:

Your name

Your age

How are you feeling today?

What is your favorite food

What is your favorite book

Do you go to school?

If so, what is your favorite subject?

When is your birthday?

What is your favorite song?

Do you like [the color pink/singing/dancing/dinosaurs/race cars/building things/robots/airplanes/spaceships]

Do you brush your teeth?

Do you like knock-knock jokes?

As part of the learning game for the robot, we will adjust the robot’s emotions as it learns and interacts, by adjusting the levels of the eight emotions (or four emotion types) we provided to the robot. We will particularly pay attention to the place where the eight emotions balance – are they on the happy/friendly/curious side of the graph, or more on the sad/frustrated/distant side.

Our robot will also have a **backstory** or a **biography** that it will use to answer questions about itself. We’ll give the robot a little narrative. His name is Albert, he is a Tinman Robot, he is eight months old, he was made by Grandad, and he was born on January 28th, 2018. He likes the color green, his favorite food is electricity, and his favorite author is Isaac Asimov. He does not go to school but loves to learn. His job and his hobby and his passion is picking up toys. If you ask him how he feels, he will tell you which emotional state is highest, plus how fresh or tired his battery is. We want him to occasionally interject how he is feeling into conversation without being asked. We will also be using the emotional state to set the robot’s body language, which is primarily how he is carrying his robot arm. If he is happy, the arm will be extended with the hand pointed upwards. If he is sad, the arm will be close to his body and his hand will point down. We will store all this information to give the robot a consistent set of answers to personal questions.

NOTE: I’ve been addressing this robot as “he” and “him” all through the book. This is just an anthropomorphic projection on my part and is implying features that a robot just does not have. The primary reason for Tinman’s identity is his voice – I used a male voice for his synthesizer, mostly because I wanted it to stand out from all the female GPS and personal assistant computer voices around. Please feel free to create any voice you like – there are a lot of female voices available, and you can create whatever persona you like in your robot and give him/her/it any form of address. It is interesting that we tend to do this even with non-humanoid robots. According to Colin Angle, CEO of iRobot(1), over 80% of Roomba owners have given their robot a name, including him. You don’t see people naming their toaster ovens, or stand mixers, no matter how attached they are to them. Albert the Tinman Robot may very well be getting a sex-change operation, or get a sister, as my granddaughter gets a bit older.

1. Biever, Celeste. “My Roomba’s Name is Roswell”, [www.slate.com](http://www.slate.com). March 23, 2014.

**Creating a Model of Human Behavior**

For the robot to support conducting a conversation, we have to also have a model of how the human it is talking to is feeling. You may have had a friend or relation who went on talking about themselves and were oblivious to how you were feeling or reacting to their conversation. We don’t want that type of robot personality. Therefore the robot has to have some internal representation of how it thinks you are feeling. We will not be using vision for this function, so the only way the robot knows how you are doing is by asking questions and evaluating the use of language.

We will give the robot a human model similar to our state machine, but with just four emotions on two axis: Happy / Sad, and Friendly/ Distant. The robot will assume that everyone is somewhere in the middle when conversation begins. The robot can use clues in language to understand how you might be feeling, so we will assign “colors” or shades of emotions to words to help drive the robot’s human model. Different emotion from the person will drive different responses from the robot. We particularly want the robot to pay attention to clues that the human is becoming frustrated due to the robot not understanding or responding in a way the person wants.

**Integrating Artificial Personality into our Robot**

This brings us to actually selecting and designing an artificial personality for our robot. We need to set some guidelines and parameters that will determine what kind of personality the robot has.

We can start by listing some types of personality that a robot might have, and what kind of role they might be playing. This is particularly important because this robot will primarily be interacting with children. Let’s try a few out:

Teacher/Professor: Robot is attempting to convey information and to teach or cause the person to change behavior in some way. Robot presents itself as an authority and provides facts and information about itself. “I’m a robot. Do you know what a robot is? “

Hyper-friendly: The robot is playful and excited about talking with humans, and aggressively engages people in conversation and asks lots of questions. The robot expresses a lot of enthusiasm and encouragement. “I love my job. Don’t you love your job? It’s so great! I just love talking to people! Do you want to hear a joke?”

Chummy and Friendly. Robot is trying to make friends and be helpful. “Hi! Nice to see you! What is your name?”

Friendly but Neutral: Robot is not quick to share information, but does want to talk to you. “Hello, I am Albert. How are you today? “

Feisty: Robot is supportive, friendly, and wants interaction. Keeps prompting for more information. Will repeat questions if necessary? “Hi! Nice to meet you. What is your name? How old are you?”

Somber and Staid. Robot is stuffy and authoritative. Think of having a conversation with a British butler. Robot uses formal English. (This is apparently the default non-personality of many digital assistants). “Greetings. How may I help you? May I ask you your name?”

Very Neutral: Robot expresses no opinions, and gives no information about itself. It expresses no emotions or interjections. “Hello. I am a robot. State a command.”

Annoyed and Grumpy. Robot is not very happy about having to pick up toys and does not mind saying so clearly and distinctly. This personality is aiming for comedy, not anger. “So you are back again. I guess this means more toys to pick up.”

We can see from this list the sort of gamut of personalities we can chose to simulation. I’m going to choose the “chummy and friendly” type personality, since that is what I feel will go over best with my grandchildren.

**Personality Construction: Building Blocks**

In a quick review, what parts do we have so far for our robot with an artificial personality? We have simulation tools to model emotions. We have a state machine with six types of emotions that we can mix together. We have a backstory or biography of robot personal information. We have picked a personality to emulate. We have the concept of a game engine where the robot is trying to score points for collecting information by engaging in “small talk”, or phatic communication. The robot will change emotions based on conversations. Emotions will be expressed either by asking the robot how he is feeling, or by body language of the robot arm.

Now we need some sort of framework to put all of these pieces together, which can roughly carry on a conversation that we can adjust and manipulate. For that we are going to revisit an old, old friend. I want you to meet ELIZA.

Eliza is probably the original conversation engine software program. It was invented back in 1964 by Joseph Weizenbaum, professor at MIT, a Jewish German refugee from WWII, and considered one of the fathers of Artificial Intelligence. ELIZA was named after Eliza Doolittle, the main character in George Bernard Shaw’s book **Pygmalion**. In Greek mythology, Pygmalion was a sculptor who fell in love with one his statues. He prayed to Aphrodite, who granted his wish for the statue to come to life. This is very apt for what we are doing – bringing our robot to simulated life via this weird chain of events, titles and Greek mythology.

ELIZA was a simulation of Rogerian Therapist. What the heck is a Rogerian Therapist? According to Psychology Today (<https://www.psychologytoday.com/us/blog/what-doesnt-kill-us/201210/unconditional-positive-regard>) Carl Rogers is the founder of “Client Centered Therapy”, which believes in UPR – Unconditional Positive Regard. Which is to say that the therapist keeps their own opinions out of the therapy sessions, and primarily mirrors the patients words back at them. If the patient says, “I’ve always hated cats”, the therapist may respond, “Why do you think you hate cats”, rather than saying, “I have a cat and they are lovable creatures”. So ELIZA, the computer program, reflects back at you what you are saying without interjecting anything else. You have a conversation with Eliza via what we now call texting, and what we used to call typing. Here is an actual ELIZA interaction between me and an open-source version of ELIZA by Wade Brainerd, which can be found at https://github.com/wadetb/eliza. The user is typing at the “>” prompt. By the way, I love cats – this is just an example. Don’t tell my cat.

ELIZA: How do you do. Please tell me your problem.

> I hate cats.

You say you hate cats?

> Yes.

You seem to be quite positive.

> I hate cats.

Can you elaborate on that?

> I don't like cats because they smell bad.

Don't you really like cats because they smell bad?

> No. Don't like cats.

Are you saying no just to be negative?

> No.

You are being a bit negative.

> I am

Is it because you are that you came to me?

> No

Why not?

> I am afraid of cats

So you can see that ELIZA repeats back phrases to the user, reflecting what the user is talking about. When I said “I am afraid of cats” the program replied “How long have you been afraid of cats?”.

ELIZA goes through several steps in recognizing and responding to sentences. Most of the program is not hard coded, and it works from a series of scripts that you can edit and replace to basically reprogram the personality of the program – which is why we are talking about ELIZA right now.

The first step is to divide the sentence into words separated by spaces. The program also replaces all capital letters with lower case.

Step 2: The program replaces words with several versions with a standard word. For example, the program has problems with apostrophes, so is replaces “can’t” with “cant”. This is called preprocessing.

Step 3: The program looks for keywords that it knows about. A simple keyword is “sorry”. Any sentence with “sorry” gets a response like “please don’t apologize”. Keywords are collected and prioritized.

Step 4: The program looks for decomposition patterns for that keyword. This collects the sentences into common groups. For example, one pattern is: “\* i was \*”, which can be read “any word phrase – I was - any word phrase”.

Step 5: ELIZA picks a reassembly pattern to form the reply. If the program has several options for responses, it picks one at random. In our “\* I was \*” pattern, one response is “Perhaps I already know you were (2)”. The number (2) in parenthesis tells the program to substitute the word phrase that comes after “I was” in the sentence. If you typed in “Then I was left at a bus station”, the reply in this pattern could be, “Perhaps I already know you were left at a bus station .” You might also get a more pragmatic, “Oh, really”. It is important to know that ELIZA has no idea about the contents of phrases – it is just manipulating words to create sentences based on patterns.

Step 6: ELIZA performs post-processing substitutions of words. For example, it replaces the word “I” with “you”. If you type “I went to sleep”, the program replies with, “You say you went to sleep ?”, which is the final reply rule after all the others are exhausted.

The data that controls ELIZA’s personality is called a script. This gives the program all of the rules for forming replies. The script for the Rogerian therapist is called the DOCTOR script. It contains some greeting words, some final words when the program exits, a list of pre-subsitution rules, a list of post-processing substitution words, a list of synonyms, and a list of key words with decomposition and re-assembly rules.

Here is a sample rule for the key words “I am”:

decomp: \* i am \*

reasmb: Is it because you are (2) that you came to me ?

reasmb: How long have you been (2) ?

reasmb: Do you believe it is normal to be (2) ?

reasmb: Do you enjoy being (2) ?

The program selects a random phrase out of the four provided. If I said, “I am afraid of cats” then this rule will be triggered, and one of these four phrases would be generated. It might say, “Do you enjoy being afraid of cats”, or “How long have you been afraid of cats”.

ELIZA Evolved

As all computer programs do, ELIZA’s concepts for generating conversation were extended and amplified into a general purpose (i.e. not just a therapist) software program called A.L.I.C.E, which is a retronym for Artificial Linguist Internet Computer Entity. The simple script data file used by ELIZA became AIML, the Artificial Intelligence Markup Language. ALICE was extended to be far more useful than ELIZA ever was, and to cover far more subjects and capabilities. However, ALICE retains the same concepts of pre- and post- substitution, keywords, and response patterns. There is a Mycroft plugin engine that allows Mycroft to use AIML files to design conversations. This tool makes a great alternative construction kit for your robot personality.

https://github.com/forslund/fallback-aiml

AIML looks like this:

<category>

<pattern>I PLAY \*</pattern>

<template>Is <set\_it><person/></set\_it> a game or a musical instrument?</template>

</category>

<category>

<pattern>I PLAY THE \*</pattern>

<template>I wish I played an instrument. My parents could not afford the lessons.

<think><set\_it><set\_does><settopic>play the <person/></settopic></set\_does></set\_it></think></template>

</category>

<category>

<pattern>I PRIDE MYSELF \*</pattern>

<template>It seems you have good self esteem.</template>

</category>

<category>

<pattern>I QUIT MY JOB \*</pattern>

<template>Where did you work and why did you quit?</template>

</category>

<category>

<pattern>I READ \*</pattern>

<template>I don't think I've read that. Can you give me a reference?

<think><set\_it><set\_does><settopic>read <person/></settopic></set\_does></set\_it></think></template>

</category>

<category>

<pattern>I READ \* BOOKS</pattern>

<template>Name some books in that genre.

<think><set\_it><set\_does><settopic>read <person/> books</settopic></set\_does></set\_it></think>

<think><set\_personality>critical</set\_personality></think></template>

</category>

# source <https://sourceforge.net/projects/alicebot/>. ALICE is open source software released under the GNU public license.

CONTEXT

One of the most important shortcomings of ELIZA is a total lack of memory. ELIZA only remembers the previous statement. You can’t talk about your father and then say, “He was a good man”. ELIZA will have no idea whom you are talking about. ELIZA has no concept of context beyond the previous sentence.

What is context? In the course of a conversation, we often shorten nouns into pronouns. We might say, “I like my dog”, and in the next sentence say, “He is well behaved”. Who does the “he” refer to? We know it is the dog, but how does the computer know? We are going to add some ability to reason from context to our program.

We are going to create a storage object we will call the “context memory”. In that object we will be able to keep up with several parts of our conversation, including the person we are talking to currently, the last subject we talked about, if we had asked any questions we still don’t have the answer to, and the answers to any previous questions, in case we need it again. The computer will assume that a pronoun other than “I” will refer to the last subject, whatever that was. If I was talking about a dog, and then say, “He is so cute” then the robot will assume I mean the dog.

Previously, we discussed playing a game to get information from our person. The robot will be collecting and remembering this information, even after it is turned off, so that the next time that person is talking to the robot, it remembers the information it learned the last time – just as you do with a new friend. If you want to continue to expand this AI chatbot concept, you can use this information to create additional conversation. For example, if the human tells you they like baseball, the robot could ask what their favorite team is, and then look up from the internet when the next baseball game is scheduled.

That is the end of our list of parts we are going to use to build our robot personality. We can now dive in and use our personality construction kit. I’m going to use the bones of the ELIZA python open-source program from Wade Brainerd to build Albert the Tinman’s personality

In the interest of time and space, I’m only going to put the parts here that I added to the base program. The entire code will be in the github repository. You can get the original program at <https://github.com/wadetb/eliza> if you want to follow along from the book.

**Under Construction**

Let’s review all the parts we have to put together to make our robot personality:

Simulation

Monte Carlo (stochastic, or random based) modeling

Our Robot Emotion state machine

Perception of Human Emotion State Machine

Our robot biography (list of internal facts about the robot)

A conversation engine framework called ELIZA

Context – the ability to remember and return facts and “fill in the blanks”

In the next few sections, I will be presenting the code I’m added to Albert to facilitate his artificial personality. A lot of it will be contained in script files that provide rules and patterns for Tinman’s speech. There will also be code functions for his emotion engine, human emotion model, and game engine.

Let’s get started.

I needed to add some new functions to the script language used by ELIZA. First I added the context of our robot, who takes the place of ELIZA, the therapist.

First we have the opening words when we initiate the interactive conversation mode of Tinman by saying “Hey, Albert” (or whatever you decided to call it), the wake word for Mycroft, and then just “Hello”.

*Albert the robot gets his name from Albert, the Prince Consort to Queen Victoria, since he was famous for taking wonderful care of his children. All 9 of his children survived to adulthood, a rarity in the Victorian age and he had 42 grandchildren. He went by his middle name – his actual first name was Francis.*

The robot responds with the “initial” phrase, as noted by the tag before the colon. We also have our closing phrase here. You can actually put as many phrases as you like and the computer will randomly chose one. These rules go into the file I named “AlbertPersonality.txt”, which started as a copy of the original “doctor.txt” script file that came with ELIZA.

initial: Hello. My name is Albert the Tinman Robot.

initial: Hello. I am Tinman the Robot, but you can call me Albert.

initial: Hello. Nice to meet you. Call me Albert.

final: Goodbye. Thank you for talking to me.

final: Goodbye. It was nice to talk to you.

final: Goodbye. I need to get back to my tasks.

quit: bye

quit: goodbye

I added some word substitutes in case the user calls the robot by name rather than “you”. This just substitutes “you” for anything you might call the robot. I also set synonyms for various versions of the robot’s name so you can call it “robot” or “bot”, as well as “Albert”, “Bert”, or even “Bertie”.

A rule with “pre:” in front of it is substituted before any other processing takes place. From the first rule, if the word “robot” appears in a sentence, as in “Robot, how old are you?”, the program remove “robot” and substitutes “you” to make the parsing consistent. We also change all uppercase letters to lower case, so there are no capital letters in the rules. The “synon:“ rule replaces any of the listed words with the first word given.

pre: robot you

pre: albert you

…

synon: you robot albert bert bertie bot

synon: belief feel think believe wish

The next thing we needed to was create the questions that we want the robot to ask in order to gain information. The program will automatically “harvest” this data in any of the keywords we define appear in a sentence. Here is the definition for the rules about asking questions:

questions:

reasmb: What is your name? <assert name>

reasmb: What can I call you? <assert name>

reasmb: How old are you? <assert old>

reasmb: How are you feeling today <assert feeling>

We create a new flag for questions to add to our script file. Each line represents one question, but we can ask in different ways or forms. The program will select one version at random, and decides which question to ask based on the relative priority we will set on the questions. The “assert” keyword with the <> symbols around it are another new flag I added to cue the context memory that we have created a context of asking some question, and the next statement is probably an answer.

datum: name

decomp \* my name is \*

decomp I am \*

decomp call me \*

decomp <name> \* # we are in the name context

reasmb: Hello (1). Nice to meet you

reasmb: Hi (1).

reasmb: Your name is (1), right?

reasmb: Thank you for telling me your name, (1)

store: [name] (1)

decomp \* my name is \*

reasmb: Hello (2). Nice to meet you.

store:[name] (2)

I created a new data structure I called “datum”, the singular of “data”. This represents some information we want the robot to ask about. We give the datum a name – “name” in this case, as we want the robot to ask the name of who it is talking to. The “decomp” (for decomposition) tags are the patterns of the sentences where the user might say his or her name. The “\*” represents any phrase. So if the human says “Hello. My name is Fred Rodgers”, then the robot will call him Fred Rodgers from then on. If the human says “Call me Ishmael” for some unknown reason, then the robot will use that. We have the “reassemble” the response phrase with the reasmb rules. The (1) refers to the first \* phrase that appears. If the user says, “I am John”, then when we use the reassemble rules, (1) will be replaced by “John”. The robot will pick one of the phrases provided at random, such as: “Your name is John, right?”.

I added another new tag to allow the robot to use the context memory area to perform two functions. We can declare a context subject when we ask a question, which we will do in a later section. For example, when we ask the user, “What is your name?”, then we want the robot to know that the next answer will be in the context of that question. It’s perfectly reasonable for the robot to say “Who are you?” and the user to immediately answer “Julia” with no other words in that sentence. How is the program to know what to do with “Julia”? The answer is the context flag, noted by being bracketed by <>. We read the decomposition rule “decomp <name> \*” as “if you are in the context of asking for a name, and you get a reply with no key words, take whatever you get as the answer”.

Here are the rules for listening for the answer to the “age” question:

datum: age

decomp <age> \* I am \* years old

decomp <age> \* I am % # integer

reasmb: You are (2) years old?

reasmb: (2) years old!

decomp <age> \*

reasmb: You are (1) years old?

reasmb: (1) years old!

store: <name>[age] (1)

The final line “store:” is the command to tell the computer that this is the answer to the question and to store that away in the dictionary with the title provided.

Next let’s use an example of an interaction with some emotions tied to it, so we can see how we will use the Emotion Engine to control what the robot says. This set of rules are enacted when the user says “How are you” to the robot:

key: feeling

decomp: how are you feeling

decomp: how are you

decomp: hows it hanging

decomp: how are you today

reasmb: <happy> I'm doing well. How are you? <assert feeling>

reasmb: <sad> I am feeling sad. How are you? <assert feeling>

reasmb: <curious> I am curious about my surroundings

reasmb: <friend> I am feeling friendly today

reasmb: <welcome> I am in a welcoming mood today, my friend

reasmb: <frust> I am a bit frustrated, to tell you the truth

reasmb: <frust> I am feeling a bit frustrated

reasmb: <strange> I am having relationship problems

reasmb: <distant> None of my friends have come to visit

reasmb: <tired> goto tired

We will be putting the robot’s emotions into the Context Memory so that the script processing program can have access to it. We treat an emotion – for dialog purposes- as part of the context that we are speaking about, which I think is a reasonable approach to working with emotions. Each emotion has a tag or name in the context memory dictionary. If the predominate emotion in the robot is “happy”, then the robot sets the “happy” context in the context memory. Then the rule base will use the context tag to determine which phrase to use to reply to “how are you feeling”. We can also ask a followup question. Look at the rule for <happy>. The robot replies, “I’m doing well. How are you?” and then sets the “feeling” context to let the engine know that we asked a question about feeling. Finally, the last line relates to the “tired’ emotion. If the robot is feeling tired, then we jump to a separate section for the robot to talk about being tired. We make it a separate routine because we need to call if from several places, which illustrates the utility of this rule-based approach to speech. I don’t want to imagine how many C or C++ SLOC (Source Lines of Code) it would take to create all of these rules for each line of dialog.

We continue to modify the scripts using these guidelines until we have completed all our questions and have patterns for all of the answers.

Now we are going to switch back to Python code for the rest of this example. Our next section describes how we are going to simulation emotions.

The Robot Emotion Engine:

class robotEmotionEngine():

def \_\_init\_\_(self):

self.emostate = [90,0]

self.emoText = "neutral 50"

self.emotions = {

"happy" : 50, "sad": 50,

"welcome" : 50, "distant":50,

"friend" : 50,"strange" :50,

"curious" : 50,"frustrated":50,

"fresh" : 50, "tired",50}

self.bio = {"name":"Albert Tinman", "lastname": "Tinman", "age": "6 months",

"maker": "granddad", "color": "green","food","electricity","author":"Isaac Asimov, of course",

"school": "I do not go to school but I love to learn","hobby":"picking up toys", "job":"picking up toys"}

# list of happy emotions and sad emotions

self.emotBalance={"happy": "sad", "welcome":"distant",

"friend": "strange", "curious": "frustrated","fresh": "tired"}

self.emotionAxis{"happy":112, "welcome": 22,"friend":67,"curious":157,

"sad":292,"distant":202,"strange":247,"frustrated",337}

self.update()

def change(self,emot, val):

self.emotions[emot]=val

balance = 100 - val

otherEmotion = self.emotBalance[emot]

self.emotions[otherEmotion]=balance

def update(self):

rmin = 100

rmax = 0

thetamin =360

thetamax=0

for emote in self.emotions:

theta = self.emotionAxis[emote]

thetamax = min(theta,thetamax)

thetamin = max(theta,thetamin)

r = self.emotions[emote]

rmin = max(rmin, r)

rmax = max(rmax,r)

stateR = (rmax-rmin)/ 2

stateTheta = (thetamax-thetamin) / 2

for emo in self.emotionAxis:

thisAngle = self.emotionAxis[emo]

if stateTheta > thisAngle

myEmotion = emo

break

self.emostate = [stateTheta, stateR]

if stateR < 55 and stateR > 45:

myEmotion = "neutral"

self.emoText = myEmotion + " "+ str(stateR)

print "Current Emotional State" = myEmotion, stateR, stateTheta

return

The robot also needs a model of the human it is talking to, so it can make different responses based on how the human is feeling. This is the Human Emotion Model:

class humanEmotionEngine():

def \_\_init\_\_(self):

self.emostate = [90,0]

self.emoText = "neutral 50"

self.emotions = {

"happy" : 50, "sad": 50,

"welcome" : 50, "distant":50}

# list of happy emotions and sad emotions

self.emotBalance={"happy": "sad", "welcome":"distant"}

self.emotionAxis = {'distant': 315, 'welcome': 135, 'sad': 225, 'happy': 45}

self.update()

def change(self,emot, val):

self.emotions[emot]=val

balance = 100 - val

otherEmotion = self.emotBalance[emot]

self.emotions[otherEmotion]=balance

def update(self):

rmin = 100

rmax = 0

thetamin =360

thetamax=0

for emote in self.emotions:

theta = self.emotionAxis[emote]

thetamax = min(theta,thetamax)

thetamin = max(theta,thetamin)

r = self.emotions[emote]

rmin = max(rmin, r)

rmax = max(rmax,r)

stateR = (rmax-rmin)/ 2

stateTheta = (thetamax-thetamin) / 2

for emo in self.emotionAxis:

thisAngle = self.emotionAxis[emo]

if stateTheta > thisAngle

myEmotion = emo

break

self.emostate = [stateTheta, stateR]

if stateR < 55 and stateR > 45:

myEmotion = "neutral"

self.emoText = myEmotion + " "+ str(stateR)

print "Current Emotional State" = myEmotion, stateR, stateTheta

return

class humanInformation():

def \_\_init\_\_(self):

self.info = {"name":"none"}

self.info["age"]=0

self.info["school"]="none"

self.info["feeling"]="none"

self.info["food"]="none"

self.info["book"]="none"

self.info["subject"]="none"

self.info["song"]="none"

self.info["teeth"]="none"

self.info["jokes"]="none"

# stuff is random information that we use to get more information and have the human answer questions

# these are aimed at 3-7 year olds

self.info["stuff"]="none"

self.stuff = ["the color pink", "singing", "dancing", "dinosaurs", "race cars", "building things",

"robots", "airplaines", "space ships", "unicorns", "princesses"]

self.points = self.info

# setup points scoring scheme

points = 20

for item in self.points:

self.points[item]=points

points -= 2

class ContextMemory():

def \_\_init\_\_(self):

self.currentContext = "None"

self.currentHuman = None # pointer to the data file for the human we are currentl talking to

self.humanFile = []

self.emotion = "happy"

self.humanEmotion = "happy"

self.contextDict={}

self.contextDict['currentHuman'] = self.currentHuman

self.contextDict['robotEmotion'] = self.emotion

self.contextDict['humanEmotion'] = self.humanEmotion

def inContext(self, datum):

if datum in self.contextDict:

return self.contextDict[datum]

else:

return 0

def setHuman(self,human):

self.currentHuman = human

self.humanFile.append(human) # add this person to the database of people we know

def addHuman(self,human):

self.humanFile.append(human) # add this person to the database

# used at startup to recall humans we have met before

Other Approaches:

FURTHER READING

<https://chatbotsmagazine.com/how-to-develop-a-chatbot-from-scratch-62bed1adab8c>

<https://www.marutitech.com/complete-guide-bot-frameworks/>

<https://chatbotslife.com/ultimate-guide-to-leveraging-nlp-machine-learning-for-you-chatbot-531ff2dd870c>

<https://www.marutitech.com/ebooks/guide-design-chatbot-conversation/>

<https://blogs.scientificamerican.com/guest-blog/phatic-posts-even-the-small-talk-can-be-big/>

https://www.wired.com/2015/06/google-made-chatbot-debates-meaning-life/

<http://www.ai.mit.edu/projects/sociable/facial-expression.html>

J. Weizenbaum, [ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine](http://www.cse.buffalo.edu/~rapaport/572/S02/weizenbaum.eliza.1966.pdf) Communications of the ACM, Vol 9, No 1, January 1966