CSC482 Lab 5: Chat Bot

Charlie Liou
cbliou@calpoly.edu

Daniel Tisdale
dctisdal@calpoly.edu

Caleb Watts
ccwatts@calpoly.edu

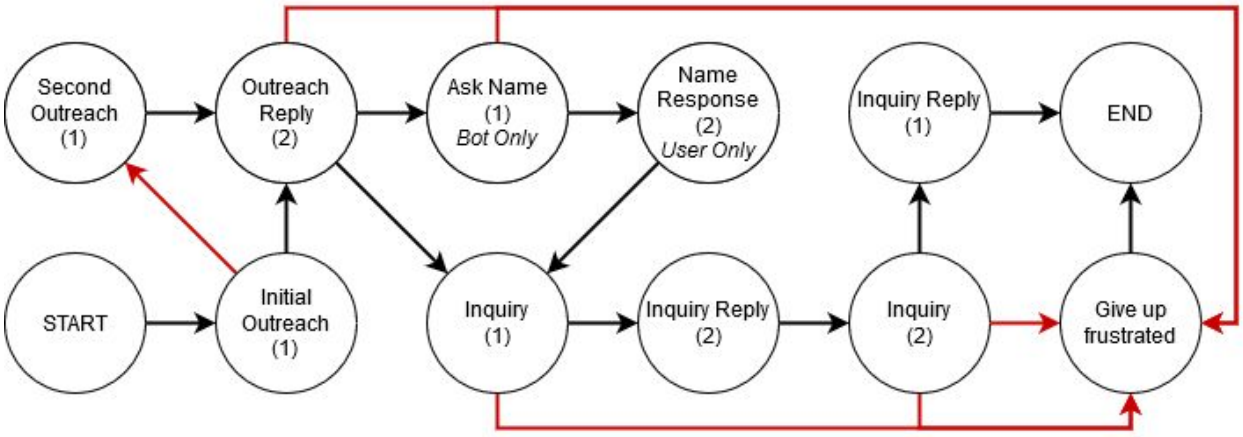
System Description

For our chat bot, the program runs a loop of receiving messages, then handling them in one of five possible ways, depending on the content of the message. All messages to the bot must be prefixed with “spicy-bot:”. The general behavior of each command is listed below:

1. **name all**
   The bot will list all of the users in the current channel.
2. **forget**
   The bot will forget all of the information about its current conversation state and any stored information about messages or users.
3. **die**
   The bot will disconnect from the channel and cease operation. This ends the program altogether.
4. **set timer**
   This command will change the cooldown that the bot has between sending messages.
5. **Any other form of message.**
   These will be handled according to the current conversation stage within the finite state machine. Certain stages will look for certain keywords, while others will look for other information about the messages that the user sent and make a decision based on that.

The finite states of a conversation have been modified slightly in the process of adding additional features for the creativity step. In particular, two stages were added for the feature to use proper names, being Ask Name and Name Response (Celeb’s creative features). Ask Name can only be done by the bot, and Name Response can only ever be done by the user. The bot processes conversations in terms of these stages, with a simplified version of the conversation states being maintained in the bot’s memory.

The modified set of finite states, along with transitions, are listed below. Transitions that occur via timeout are highlighted in red. Additionally, we added a phrase to mark the end of a cycle to better distinguish if the bot was still waiting to timeout or had started a new chat cycle.



Phase Handling

For Phase 1, the non-conversation parts of the phase are detailed above. For the basic responses to user input, the bot looks for a set of greetings to be contained within a message directed to it. That is, it will only respond to messages like “spicy-bot: hello”, or “spicy-bot: what’s up”

For Phase 2, responses are given to the person who it is in a conversation with. Any other users trying to interact with the bot outside of the built-in commands will get a message about interruptions, and the conversation will not be affected. When the bot expects the user to have sent an inquiry, the message is checked to see if the last sentence in the message has a question mark at the end. For the Inquiry Reply stage, the reply is analyzed for the sentiment, which determines how the bot should respond.

Implementation Details

IRC/Networking:

The IRC library is implemented directly via sockets. We implemented this with the idea of a general client: packets are received via the socket and then put on a queue, where a daemon thread is adding packets to a queue. This can be seen as a type of **interrupt-based** handling, which emulates the behavior of clients as a client should not pause to wait for a packet.

NLP:

The primary NLP routines used in the program are tokenization and sentiment analysis. Tokenization was used extensively, both on a sentence and word level. In a similar vein, similarity between user input and potential responses from the bot was measured via the overlap of words used. While it is a naive approach, many of the responses contain one or two key words like “hello” that can inform the bot to reply in kind.

Sentiment analysis is done on the user’s response to an inquiry, in order to determine whether it is good, bad, or neutral. Depending on the sentiment, the bot can then respond with condolences, joy, or neither if the response is deemed to be neutral. The sentiment analysis is done using the Vader Sentiment Intensity Analyzer included with NLTK (nltk.sentiment.vader.SentimentIntensityAnalyzer).

Other data structures:

The information kept in memory by the bot are as follows: The nickname of the bot, the user that the bot is conversing with, the channel it is in, the server it is in, the state according to the finite state machine view, histories of the messages that have been sent and received by the bot, a timeout threshold, a cooldown, a dictionary of username to proper name bindings, and networking connection-specific information. It also contains a sentiment analyzer.

Creativity Part - Extra Features

Three additional features were implemented: proper name remembering and use, lyrics fetching, message recall, and weather detection. Proper name use was implemented by Caleb, lyrics fetching by Charlie, and message recall and weather detection by Daniel. More detail is included below in the reports by each student on the feature that they implemented.

Message recall (Daniel):
The message recall system allows the user to ask for a phrase or partial phrase that the bot or user has said prior during the Inquiry(1) phase (format: time <I/you> said <phrase/partial phrase>). The bot will then attempt to find the earliest timestamp containing the phrase, display information about it during the Inquiry Reply(2) phase, then proceed to continue on with the conversation to the end.

Weather detection (Daniel):
Due to users' IPs not being visible on the Freenode server, the system runs geolocation-db API to get info about the bot host's coordinates and region. Following this the coordinates is then fed to the openweathermap API to get the current day's weather forecast in the area to better augment certain phases of the bots replies, namely: Inquiry Reply(2) and Inquiry Reply(1).

Lyric fetching (Charlie):
The Lyric fetching system allows users to ask for lyrics for a song and artist. This is an addition to the Inquiry (1) phase, as you can ask a question for lyrics. It looks for this specific format "lyrics to <song_name> by <song_artist>". After you enter a query, the bot queries the Genius API for the link to the lyrics and returns the first 15 lines. If the bot cannot find the genius link associated with the song, it assumes the song is on Genius and informs the user it was unable to find a song lyric. This redirects you back to the Inquiry (1) state where you can ask a question (does not have to be for a lyric).