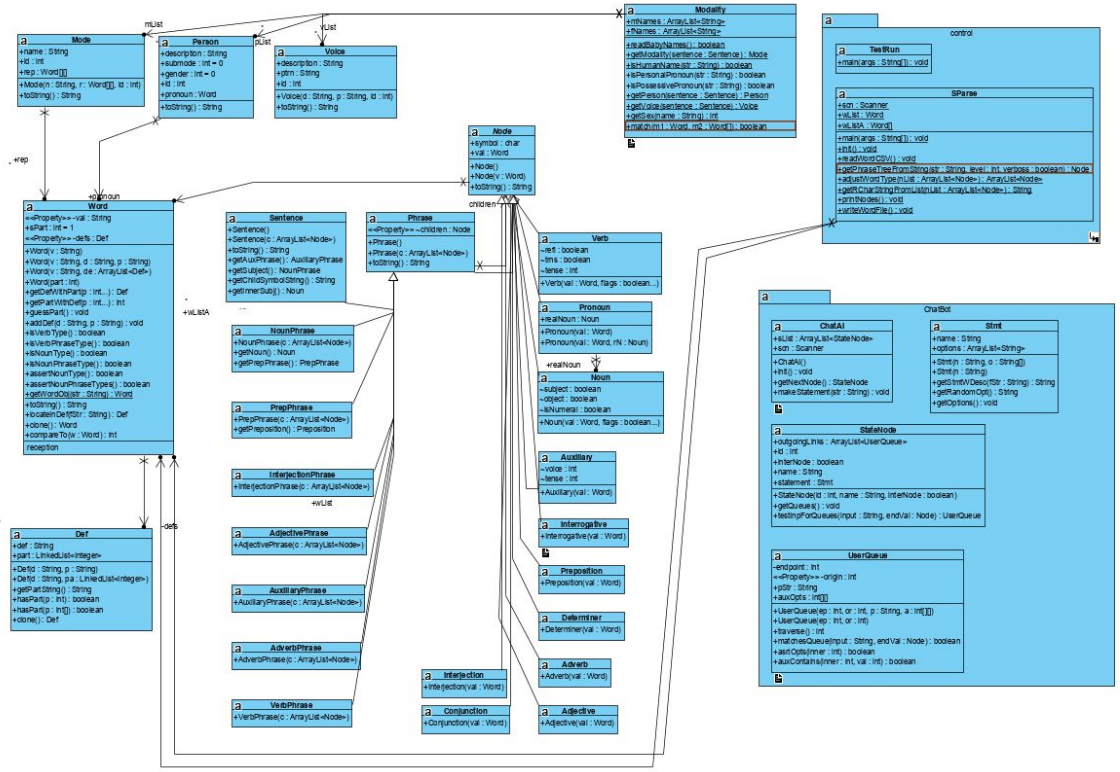
Project Final - README

Chatbot

Team: Group 3 - Cody Tyerman, Adam Fox, Aleksander Konstantopulous, Matt Duroche

Class Organization:

Below is our class hierarchy diagram in order to further explain the organization of each class:



*See the attached file called: ***classdiagram1.png*** for a hierarchical view of each class and the ***classdiagram2.png*** for an enlarged picture of the above diagram.

Class Roles:

TestRun: the entry point for the application, and the GUI control center (extends javafx application). Employs a singleton to allow control of JavaFX GUI elements from anywhere.

ChatAI: the class responsible for mimicking conversation

StateNode: The Node class for the graph of Chatbot behaviours

UserQueue: Class modelling links in the above-mentioned graph

Stmt: Statements that the bot can make

SParse: top level of speech parsing system which encapsulates and analyses the statements to determine metadata.

Word: wrapper class for dictionary word, the definitions associated with it, and the parts of speech associated with each definition

Def: struct to store a definition and part of speech qualifier
Node: Base class from which all other Sparse tree nodes extend
Phrase: extends node, represents a list of nodes which together form a type of phrase
About 15 subclasses of **Node** and **Phrase** representing the various parts of speech and phrases found in an English sentence. Little specific functionality is implemented in many cases, but room to do so exists.
Sentence: extends phrase, top-level class in the SParse phrase tree. It contains a noun phrase and a verb phrase typically. Can contain only a verb phrase, or only an interjection depending on the voice/mode
Modality: class which determines the modality, person and voice of the sentence/subject.
Mode: Enumerated instance of various types of mode (tense and optionality)
Person: same as above, but for 1st, 2nd, 3rd person, singular, plural, etc
Voice: imperative, declarative, interrogative, salutative (we made that last one up)

Methods:

Sentence SParse.getPhraseTreeFromString(String):
This method takes in a String of words (presumably forming a syntactically correct statement), splits it into its constituent words, encapsulates a matching word from the dictionary in a type of node matching it's likely role in a sentence, assigning each different type of word a different letter of the alphabet, allowing a regex match to be performed to determine which patterns of words forming phrases (themselves replacing words in other phrases higher up in the hierarchy) can be encapsulated, and performing this operation (like a macro processor but in reverse) until the entire sentence is reduced to a single phrase (the Sentence type) containing all of the others. Further operations are performed on this end product in the Modality class to determine metadata

boolean UserQueue.matchesQueue(String, Node):
This method is used to test the regex corresponding to each link on a node against the input given, in either the form of a String or a phrase tree as described in the method above.

Mode Modality.getMode(Sentence):
Checks a double array of nodes representing the possibilities for each of a number of predefined modes against the encapsulated phrase tree to determine which is employed in the sentence.

int UserQueue.traverse():
returns the identifier of the Node at the opposite end of the link.

Compilation Instructions:

In order to compile our program, you will need to have JavaFX 11 along with the Java Development Kit 12.0.2. These can be found on their respective websites with the JDK being found at:
https://www.oracle.com/java/technologies/javase/jdk12-archive-downloads.html
and JavaFX 11 is found at:
https://gluonhq.com/products/javafx/
after these files have been downloaded and placed in the correct directories they should be added to the project build path in your Java IDE of choice. After they have been added the program should compile and run without issue.

List of Features Programmed:

For each feature:

- (explain how we used the feature to improve the agent's responses)
- (show snippet of the feature working)