ReadMe

LILI: Lehigh Instrument for Learning Interaction
Kira Gobes, Christie Altadonna
CSE 379: Fall 2017

For future Lehigh programmers: please see our report with Professor Chuah or on our Github for more details about project and reasoning behind these implementations.

Downloads and Dependencies:

- To get LILI running, please first view the documentation created by Xin here. Some modification may be made depending on if you get the source code from our Github or if it gets moved somewhere else, but the rest should be the same. We downloaded VirtualBox and created an Ubuntu instance in that.
- For our code, you will need PYQT4 for the GUI
- For speaker recognition, see the external Github page here to get the additional downloads (Scikit learn, talkbox, numpy, scipy).

LILI Code:

In source (src) folder:
(Details included for the folders that Christie and Kira worked on)
Lili_bringup
Lili_description
Lili_audio (src)

- Speech_recog.py: Called whenever audio is expected. This file that uses Google speech API to turn what user says into text. The call to sound_recorder.recordCurrentVoice() on line 70 gets the current voice and turns it into a wav file called currentSpeaker.wav. We added in calls to testSpeakerRecog.predictSpeaker() (around line 82) to use the wav file that the user just spoke to predict the current speaker; note: we moved this to also be in the text_to_speech file so now it is not needed here.
- Sound_recorder.py: code to take sound from the mic and turn it into wav file needed for enroll and predict. There's a few versions of same method in here to do that but to save it in different places based on if it's an enroll or a predict call.
- testSpeakerRecog.py: the class that does most of the work for speaker recognition. This does calls to implement the Github library for speaker recognition and pass it the specific parameters for predict or enroll. The external github library didn't have an efficient way to call methods other than command line, so we do a nifty subprocess.call("command line call here") to get that to work. The main method is not needed, we used it for testing pieces.
    - enrollSpeaker(playerName) makes a call to sound-recorder to record the person speaking. There's a few lines that use PyAudio tools to convert it to the correct kind of wav file. Then it does the process call to enroll the wav file as that player's

name. We created a directory of enrolledSpeakers that this is coded to save into. More details on that below.
    - Essentially, a player is created with a folder for their name, and then predict searches those folders to find the correct one, using the model.out file.
        - Suggestions for debugging: delete model.out and restart and see any differences. Recreate with a simple “touch model.out”. See predict explanation below.
  - predictSpeaker(audioFile) takes in a wav file and returns the name of the predicted player. This is finicky: outside of the LILI codebase, prediction works perfectly. In the actual version, it sometimes works and sometimes it has issues. We’ve attempted to solve this by every time a player is enrolled, re-enroll all the folders of speakers. This should overwrite everything in model.out and has been more correct than just adding another player every time.
  - Note: there’s a lot of junk that gets spewed onto the console during enroll that is not an error, it’s just the result of a lot of things being called. The internet says this is normal.
- Text_to_speech.py: Every time text is processed to say, call predict (around line 66) to figure out who is speaking. We moved this method call here in order to make it easy to have the name of the user spoken by LILI, which we couldn’t easily do from the other methods. This is where we suggest tinkering if you are going to start having username influence gameplay.

Lili_graphics

- Images: directory that holds all the gif/images that are used in story nodes

Lili_storytelling (src)

- Story_telling.py : GUI file, main functional file
- Cinema_story.py: file that contains the hardcoded story nodes and edges for the cinema story
- Park_story.py: file that contains the hardcoded story nodes and edges for the park story
- Test_story: file that contains the hardcoded edges for the zoo story
- Dict_map.py: file that holds the hardcoded locations and scenes for each scene available in the story (used in the GUI to associate certain scenes with certain locations, and certain scenes with the story nodes that need to be played for the scenes). To add locations, scenes, and story nodes to the story, you must create a python file like the above three, and then create a scene object that contains those storynodes and assign them to a location they belong to - this assignment goes in dict_map.py
- Global_nodes.py: file that holds the global story nodes to be added to the created story that encompasses multiple scenes (Ex: the exit node from the whole scene which is connected to the exit nodes of the scenes selected for the given story). Where some of the random nodes can go.
- testSpeakerRecog.py: also in lili_audio; holds the enroll function which enrolls the current speaker and saves their .wav file in the “current directory” which is the .ros file which will be in the user’s home directory in the “.ros” file

.ros

(File within home folder that is considered the current working directory)

- createdStories: file that the created stories are saved to and the stories to play are loaded from (using pickle)
- enrolledSpeakers: directory that holds the enrolled speakers directories and wav file
    - currentSpeaker.wav: file that the current speaker's voice is saved into and compared with the enrolled speakers
- Speaker-recognition.py: github file used to do the speaker recognition (predict and enroll)