CPSLP Programming assignment

1 Introduction - Speech Synthesiser

Your task for this assignment is to create a speech synthesiser! Your Python program will take text input from a user and convert it to a sound waveform of intelligible speech. This will be a *very basic* waveform concatenation system, whereby the acoustic units are recordings of diphones. You will be provided with several files to help you do this:-

simpleaudio.py
: This is a version of the `simpleaudio.py` module that we have used in the lab sessions. The `Audio` class contained therein will allow you to save, load and play `.wav` files as well as perform some simple audio processing functions. You should **not** modify this file.

synth.py
: This is a skeleton structure for your program, with a few hints to get you going. Your task is to fill in the missing components to make it work. You are free to add any classes, methods or functions that you wish but you must **not** change the existing argparse arguments.

diphones/
: A folder containing `.wav` files for the diphone sounds. A diphone is a voice recording lasting from the middle of one speech sound to the middle of a second speech sound (i.e. the transition between two speech sounds). If you are unfamiliar with diphones, try listening to some of them to understand what this means. (Hint: you cannot rely upon the fact that all possible diphones have been given to you!)

examples/
: A folder containing example `.wav` files to compare how your synthesiser sounds with respect to a reference implementation.

2 Task 1 - Basic Synthesis

The primary task for this assignment is to design a program that takes an input phrase and synthesises it. The main steps in this procedure are as follows:-

- normalise the text (convert to lower case, remove punctuation, etc.) to give you a straightforward sequence of words
- expand the word sequence to a phone (or "speech sound") sequence – you should make use of `nltk.corpus.cmudict` to do this, which is a pronunciation lexicon provided as part of NLTK. It is already imported in the skeleton script for you, but you will need to find out yourself what the `cmudict` object is, what it can do and so how to use it for your purposes. Don't forget, utterances should always start and end with a silence phone! For example, saying the sentence "Hello!" requires a phone sequence of [PAU, HH, AH, L, OW, PAU] (where PAU is the label for the short pause or silence phone).

- expand the phone sequence to the corresponding diphone sequence (e.g. to say “Hello!” we need [PAU-HH HH-AH AH-L...] etc.)
- concatenate the necessary diphone wav files together in the right order to produce the required synthesised audio in an instance of the Audio class. Your aim here is to create a single new waveform (e.g. one array of audio data in an Audio class instance), and not just to quickly play one diphone sound after the other, for example.

A user should be able to execute your program by running the `synth.py` script from the command line with arguments, e.g. the following should play “hello”:-

```
python synth.py -p "hello"
```

If a word is not in the **cmudict** then your program should take appropriate action (e.g. print an informative message for the user and exit).

You can listen to the examples `hello.wav` and `rose.wav` in the `./examples` subdirectory, which were created as follows:-

```
python synth.py -o hello.wav "hello nice to meet you"
python synth.py -o rose.wav "A rose by any other name would smell as sweet"
```

If you execute the same commands with your program and the output sounds the same then it is likely you have a functioning basic synthesiser! You could also compare waveforms sample by sample, to ensure your synthesiser functions entirely like the reference implementation.

3 Task 2 - Extending the Functionality

Implement **at least two** of the following extensions:-

Extension A – Volume Control [easy]

Allow the user to set the volume argument (`--volume, -v`) to a value between 0 and 100 (minimum and maximum loudness respectively) to change the amplitude of the synthesised waveform.

Extension B – Spelling [easy]

When the user gives the spell command-line argument (`--spell, -s`) synthesise the text as spelled out instead of read normally. Do this by converting a string into a sequence of letters, and then to an appropriate phone sequence to pronounce for each letter in its alphabetic form. At its simplest, you can assume this should only work for single words, but you can also go further if you wish. [hint: cmudict contains entries for letter names]

Extension C – Speaking Backwards [fairly easy]

Have fun by making the synthesiser speak backwards in one of three ways:

signal When the user gives the commandline option `--reverse signal`, switch the waveform signal for the whole synthetic utterance back to front.

words When the user gives the commandline option `--reverse words`, reverse the order of the words that will be synthesised (e.g. `"oh hi there" -> "there hi oh"`).

phones When the user gives the commandline option `--reverse phones`, reverse the order of the phones that will be spoken for the whole utterance.

Extension D – Punctuation [more challenging]

If the input phrase contains a comma, insert 200ms of silence. If it contains a period, colon, question mark or exclamation mark, then insert 400ms of silence. Strip and ignore all other punctuation. Note, when inserting the silence, you will also have to change the phone sequence to include a PAU phone. For example, if you needed to put a 200ms silence between phones "... AH HH ...", then your sequence of waveform "chunks" should be "... AH-PAU 200ms-silence PAU-HH ...". Don't worry about the duration of the PAU segments in your solution, but do just insert an extra 200 or 400ms of silence in the right places!

Finally, to show off your punctuation handling, add code to enable the user to give the `--fromfile` flag. The user can use this flag to specify a text file containing text to synthesise – and in this case, your programme should open the file with the given filename and synthesise whatever is contained there. Note, however, your programme should process the text *sentence by sentence*. For example, your programme should synthesise the first sentence and then play it if the `--play` flag is given, then synthesise the second, and then play that one, and so on. If the `--outfile` option is given by the user, then each of the synthesised waveforms must be concatenated to a single waveform prior to saving it at the end.

Extension E – Emphasis markup [more challenging]

One way emphasis can be indicated on a word is by increased loudness, duration and some pitch (f0) accent. Implement a simplified version of this in your synthesiser. Allow the user to put curly braces around any 1 word in the input text, and increase the loudness (don't worry about duration or pitch) of that word noticeably compared to the rest of the utterance. So, for example:

```
  "The {cat} sat on the mat."
 -> the word 'cat' should sound louder than the rest of the utterance.
```

Extension F – Smoother Concatenation [more challenging]

Simply pasting together diphone audio waveforms one after the other can lead to audible glitches where the waveform "jumps" at boundaries between diphones. Implement a simple way to alleviate this by "cross-fading" between adjacent diphones using a 10 msec overlap. To achieve a cross-fade, you need to lower the amplitude at the end of one diphone down to 0.0 over 10msec and then overlap and add in the signal from the start of the next diphone which is similarly tapered from 0.0 to normal amplitude over the same 10msec period. Note this will only mitigate one cause of "choppiness" in the synthetic speech and so can only do so much to make it sound better! Audio examples are provided for you to compare cross-faded concatenation with simple concatenation. Compare sizes of those files with ones produced by your code. to make sure you're not losing or gaining any samples. [hint: you will probably find numpy very useful to implement this extension in a succinct and efficient way!]

4 Rules and Assessment

Your submission will comprise a single file of Python code and should abide by the following rules:-

- all submissions must be written individually and be your **own work**.
- the University's penalties for plagiarism are **potentially very harsh**. Googling how to use a particular Python object or syntax feature is to be expected, and finding information in that way is no problem at all. If you find a one-liner to achieve some neat "trick", it's also fine to include that, as long as you attribute it (e.g. provide the URL for the StackOverflow page you found it on in a code comment for example). In contrast, cutting and pasting whole sections of code, or even whole functions/classes is definitely **not** in the spirit of the exercise and will be penalised. Note, Python code plagiarism detection software can spot copied code even if the names and formatting are changed. So, please, do just come up with your own code.
- your submission may only use **numpy**, **nltk**, the provided files, and any packages that are **built-in** to Python. I must be able to run your code on my computer using just the one file and without installing anything else.
- you may **not** change any of the existing argparse arguments provided in `synth.py` – when you view that file you will see argparse has already been set up to take all the correct command-line arguments for you.

The assignments will be graded out of 100 and will be assessed according to the following criteria:-

Task 1

Your system:-

- is able to synthesise several test phrases that contain only words (no punctuation, numbers, etc.).
- is reasonably robust - for example it can handle out of vocabulary words, or malformed input, or a missing "diphones" directory, or even missing diphones, in an elegant and/or informative way rather than just falling over or breaking
- is able to play and/or save the output to a file

IMPORTANT: just because your code might run and work as specified above, it does not mean you will necessarily get full marks! Many aspects of your code will be considered beyond simply whether it works or not, including for example:

- has the task clearly been understood with clear evidence of a sensible attempt to implement solution?
- does it work entirely as specified?
- is it efficient?
- is the code sensible/logical?
- is the code legible, sufficiently documented and "friendly" for others
- is the code robust?

Task 2

As a bare minimum, you must implement **at least two of the extensions** in order to pass. For a higher grade, you can choose to implement more of the extensions, taking care to present strong solutions. For example, 3 extensions implemented to a high standard could achieve the same grade as 4 extensions implemented less well.

Also note that the extensions vary in terms of the effort required to implement them. Extension A is far simpler than Extension E or F, for example, and the marks awarded will naturally reflect the intrinsic difficulty of each of the extension tasks. Again, keep in mind the key marking criteria: functionality, design, legibility, efficiency, and robustness.

Style and Design

Further marks will be awarded based on how well your code is designed and presented overall. Therefore, take care to use appropriate object-orientated design as well as suitable code formatting, naming and plenty of appropriate comments and docstrings. You can also earn extra marks by being 'pythonic' (i.e. keeping your code succinct, well-organised and easily-readable; good use of nice Python language features (e.g. comprehensions); etc.)

You will be required to submit your modified `synth.py` file once you have finished.

Submissions are marked anonymously. You **must** prepend your exam number (the 'B' number on your student card) to the file prior to submitting it, e.g. your submitted file should be in the format:

`B123456_synth.py`

In addition, do **not** include any identifying content (e.g. Matriculation number, name etc.) in the submitted file. Submit only one Python code file in the above format (and **not** a zip file for example).

Finally, if you should have any queries about the task, or questions more generally, then please do not hesitate to ask!