William Kluge

Abstract

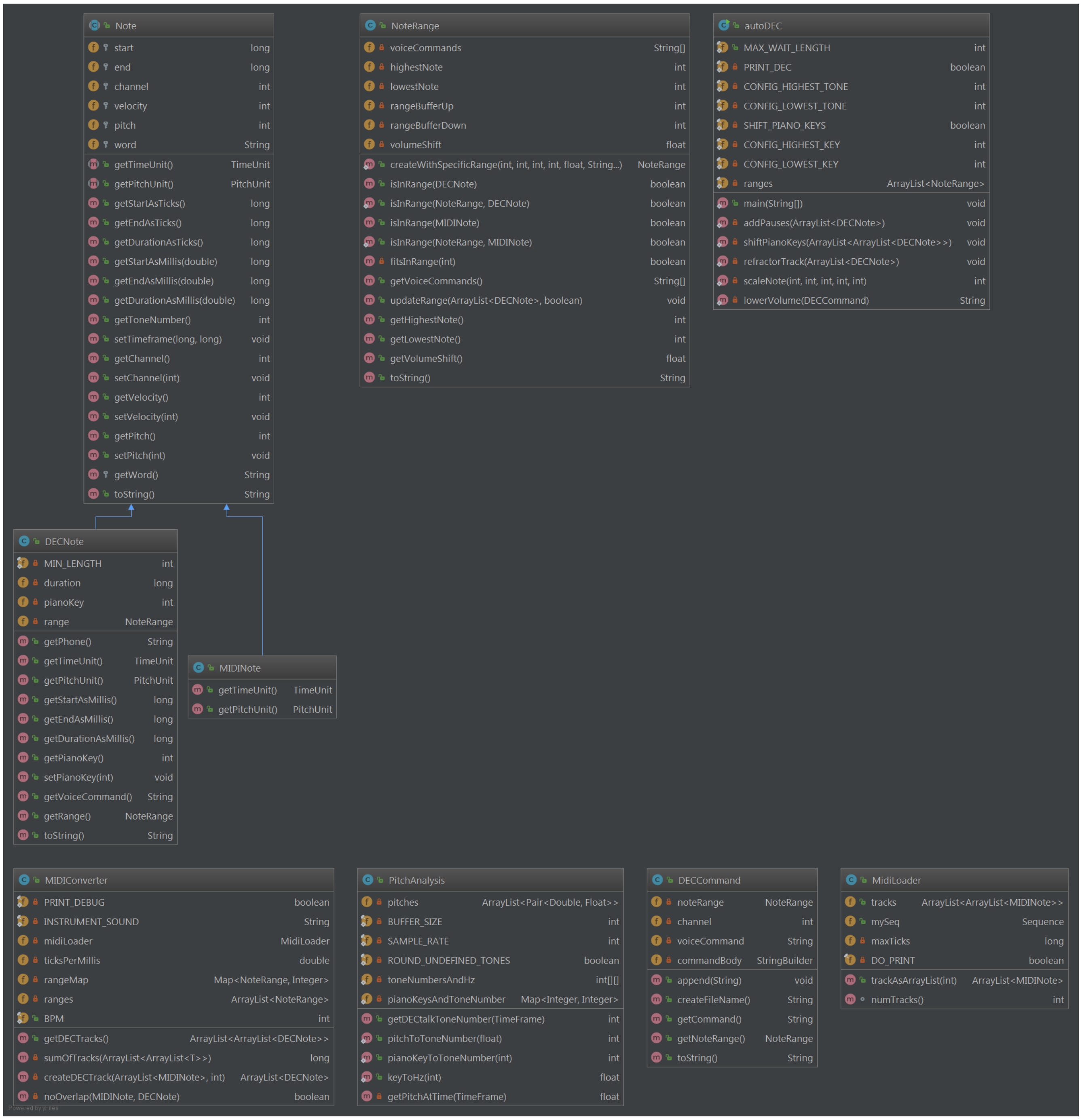
autoDEC is a program that turns a MIDI file into WAVE files of the famous voice synthesizer DECtalk singing the MIDI's song. MIDI files are used to save a instrument part to a file as a series of notes. These notes specify the velocity, duration, and pitch of the note. autoDEC uses this information to split up the song into pieces for DECtalk to sing. DECtalk is not able to play multiple notes at once, so autoDEC needs to split the MIDI file into multiple parts that never play a note at the same time.

Introduction

I was inspired to make autoDEC by videos of songs redone in DECtalk. I found these to be very cool because I like how DECtalk sounds and the creator clearly put a lot of work into making them. autoDEC was made so that significantly less effort is needed to translate a song into DECtalk. This paper will describe out autoDEC is structured, exactly what autoDEC is solving, other work that is similar to autoDEC, how to use autoDEC, and the sources for information used to make this program.

System Description

autoDEC has no user interface. While it was originally going to use one, due to the many changes in how autoDEC actually works it is not needed. To use autoDEC, a user just needs to run the program with a path to a MIDI file as the only argument and autoDEC will do the rest. autoDEC then takes this MIDI files, uses a program to convert it all to one static BPM (if this is not done then the tempo gets very messed up if it ever changes during the course of the song), splits it into multiple tracks, then converts the pieces of those tracks into DECtalk commands. autoDEC then uses a specific distribution of DECtalk known as say.exe to convert the commands into WAVE files. These WAVE files are put under the dectalk/generated directory. The following is a UML diagram of the interaction of classes in the autoDEC program:



Note	
start	long
end	long
channel	int
velocity	int
pitch	int
word	String
getTimeUnit()	TimeUnit
getPitchUnit()	PitchUnit
getStartAsTicks()	long
getEndAsTicks()	long
getDurationAsTicks()	long
getStartAsMillis(double)	long
getEndAsMillis(double)	long
getDurationAsMillis(double)	long
getToneNumber()	int
setTimeframe(long, long)	void
getChannel()	int
setChannel(int)	void
getVelocity()	int
setVelocity(int)	void
getPitch()	int
setPitch(int)	void
getWord()	String
toString()	String

NoteRange	
voiceCommands	String[]
highestNote	int
lowestNote	int
rangeBufferUp	int
rangeBufferDown	int
volumeShift	float
createWithSpecificRange(int, int, int, int, float, String...)	NoteRange
isInRange(DECNote)	boolean
isInRange(NoteRange, DECNote)	boolean
isInRange(MIDINote)	boolean
isInRange(NoteRange, MIDINote)	boolean
fitsInRange(int)	boolean
getVoiceCommands()	String[]
updateRange(ArrayList<DECNote>, boolean)	void
getHighestNote()	int
getLowestNote()	int
getVolumeShift()	float
toString()	String

autoDEC	
MAX_WAIT_LENGTH	int
PRINT_DEC	boolean
CONFIG_HIGHEST_TONE	int
CONFIG_LOWEST_TONE	int
SHIFT_PIANO_KEYS	boolean
CONFIG_HIGHEST_KEY	int
CONFIG_LOWEST_KEY	int
ranges	ArrayList<NoteRange>
main(String[])	void
addPauses(ArrayList<DECNote>)	void
shiftPianoKeys(ArrayList<ArrayList<DECNote>>)	void
refractorTrack(ArrayList<DECNote>)	void
scaleNote(int, int, int, int, int)	int
lowerVolume(DECCommand)	String

DECNote	
MIN_LENGTH	int
duration	long
pianoKey	int
range	NoteRange
getPhone()	String
getTimeUnit()	TimeUnit
getPitchUnit()	PitchUnit
getStartAsMillis()	long
getEndAsMillis()	long
getDurationAsMillis()	long
getPianoKey()	int
setPianoKey(int)	void
getVoiceCommand()	String
getRange()	NoteRange
toString()	String

MIDINote	
getTimeUnit()	TimeUnit
getPitchUnit()	PitchUnit

MIDIConverter	
PRINT_DEBUG	boolean
INSTRUMENT_SOUND	String
midiLoader	MidiLoader
ticksPerMillis	double
rangeMap	Map<NoteRange, Integer>
ranges	ArrayList<NoteRange>
BPM	int
getDECTracks()	ArrayList<ArrayList<DECNote>>
sumOfTracks(ArrayList<ArrayList<T>>)	long
createDECTrack(ArrayList<MIDINote>, int)	ArrayList<DECNote>
noOverlap(MIDINote, DECNote)	boolean

PitchAnalysis	
pitches	ArrayList<Pair<Double, Float>>
BUFFER_SIZE	int
SAMPLE_RATE	int
ROUND_UNDEFINED_TONES	boolean
toneNumbersAndHz	int[][]
pianoKeysAndToneNumber	Map<Integer, Integer>
getDECtalkToneNumber(TimeFrame)	int
pitchToToneNumber(float)	int
pianoKeyToToneNumber(int)	int
keyToHz(int)	float
getPitchAtTime(TimeFrame)	float

DECCommand	
noteRange	NoteRange
channel	int
voiceCommand	String
commandBody	StringBuilder
append(String)	void
createFileName()	String
getCommand()	String
getNoteRange()	NoteRange
toString()	String

MidiLoader	
tracks	ArrayList<ArrayList<MIDINote>>
mySeq	Sequence
maxTicks	long
DO_PRINT	boolean
trackAsArrayList(int)	ArrayList<MIDINote>
numTracks()	int

Requirements:

autoDEC does not do all the work, but saves a considerable amount of time for people who are trying to translate a song into DECtalk syntax. A user could just take the output WAVE files and be happy with the result, but those who wish to tweak the output to increase the quality (for example adding lyrics to the song) can use the DECtalk commands that autoDEC outputs. Even if the output is far from correct, which is very unlikely, a user still gets a strong base of note lengths and pitches to go off of. autoDEC is only as good as the input a user gives it. If the user gives a MIDI file that was made very poorly, then the output will sound just as foreign as the input. autoDEC needs a properly created MIDI file that uses actual markers for tempo changes, note endings, and part separation to work properly. There are some, in fact quite a few, MIDI files that keep all parts in one track. This is an issue because autoDEC wants parts to already be separated so that any notes that are supposed to stay together stay together and any that are meant to be different stay different. autoDEC will still work perfectly fine, but the quality of output will decrease.

Literature Survey:

There has been surprisingly little discussion of using MIDI files as a base for DECtalk singing. There is one[1] repository on GitHub that seems to entertain the idea, but it is not fully implemented and seems more like someone fiddled with the concept in an afternoon. There is also a tutorial[2] of how to convert a MIDI file into DECtalk very easily, but it does not seem to account for multiple notes being played at the same time and is intended for very simple songs.

User Manual:

autoDEC is extremely easy to use. All a user needs to do is open autoDEC with the one and only command line argument being a path to a local MIDI file. autoDEC will then output a series of WAVE files to the dectalk/generated directory. Each one of these files represents one track in the DECtalk song. These files are separated so that the user can select which tracks will actually be

1 https://github.com/moustacheminer/midi-to-dectalk
2 https://www.freelists.org/post/dectalk/dec-talk-tracks-from-midi-files,2

included in the final version. Usually higher pitched notes (anything above the 65th key on pianos) do not sound very good in DECtalk, so if a user chooses they can leave these out. autoDEC works best on songs with primarily low notes. There is an example of this at data/Skyrim Example.wav, which is the Skyrim Theme put through autoDEC. To make a song sound good with autoDEC the user will likely need to adjust the gain of various tracks. If this is not done the low notes will completely overpower the high notes.

Conclusion:

Overall autoDEC accomplishes a different goal than what I originally set out for it to do, but the new way to do things works so much better that it is worth it. The original goal was for the input to be an mp3 and use voice to text and pitch recognition to reconstruct the song in DECtalk, but this method did not work well without intense user interaction that took an extremely long time. The entire point of autoDEC was to save time, so it did not make sense to use a method that made the user do so much work. autoDEC works with almost any MIDI file given to it (just try to stay below 1,000,00 notes for the sake of your computer), which I think makes it a very cool program.

Resources:

DECtalk Commands
How To DECtallk
Many many many SO posts referenced in the code wherever they were used.
DECtalk Phonemic Symbols
Sphinx