Overview

About

This project was developed as a final project for Middlebury College's CS318: OOP & GUI Development. The original concept of the Phonetics Visualizer is to aid language learners' pronunciation by algorithmically identifying differences in pronunciation between an uploaded and a recorded sentence. These could be differences in length of phoneme, pitch, relative emphasis, et cetera. We believe that identifying these differences for the listener, then giving them capability to isolate, listen, and visualize these shortcomings in pronunciation would significantly enhance the language learning process.
Although over the short course of the semester we were not able to implement an algorithmic framework to recognize differences in these across recordings, our team developed visualization and playback capabilities that allow for user analysis.

Prerequisites

What things you need to install the software and how to install them.

1. An instance of Qt and Qt Creator. This is for the development version.
    i. This requires XCode if you are on a Mac.
2. Right now the software is dependent on the FFTW library. We have been running it from within our Qt development instance, which is dynamically linking to the FFTW headers from our `\usr\local\lib` directory. Hopefully for the production version we will have this code fully linked and included in a standalone executable.

Installing

1. Install XCode, Qt, Qt Creator. There are plenty of better guides to do this online than anything we could write.
2. Git clone the project into your desired directory.

git clone https://github.com/jorredahl/LinguisticCS.git

3. Install FFTW. We did this using homebrew, then to get the FFTW package to where our `.pro` file was looking we used

sudo cp /opt/homebrew/Cellar/fftw/3.x.x/lib/* /usr/local/lib/

Built Using

- C++
- Qt
- FFTW

Authors

- @jkwarren
- @mbamaca
- @terryluongo
- @jorredahl
- @abmerino

See also the list of contributors who participated in this project.

Acknowledgements

- Thank you to Professor Swenton for the plethora of help throughout the semester!
- Thank you to Professor Baird for the linguistics resources and Professor Abe for the Japenese resources!

Class Descriptions

Main Window

File: mainwindow.cpp, mainwindow.h
Description: This source file implements the 'MainWindow' class, providing functionality for the main UI. It initializes the main window, sets up the main layout in a vertical layout, and creates two Audio objects for dual audio playback functionality. The constructor creates the central widget, configures the vertical layout, and adds two Audio widgets. The 'audio1' and 'audio2' widgets are integrated into the layout to enable independent audio playback and visualization within the UI layout. The two audio objects are also connected here for when the user wants to align both audios (play or view both with the same set of controls simultaneously).

Notes:
- The Audio class is used for playback and visualization(*). See 'audio.h' and 'audio.cpp' for its implementation.

---

audio

File: audio.cpp, audio.h
Description: The audio class provides the functionality for audio playback, upload, visualization, and control interactions. The class uses 'QMediaPlayer' and 'QAudioOutput' for audio playback and integrates waveform and zoom controls. The constructor 'Audio(QWidget *parent)' calls the 'newAudioPlayer()' method which sets up the UI, including the upload and play buttons, waveform display, segment controls,and zoom controls, and the audio playback is managed with 'QMediaPlayer' and supports scrubbing, segmenting, and zooming functionality.

Purpose

- Contains most functionality of the app
- Contains all other classes implemented

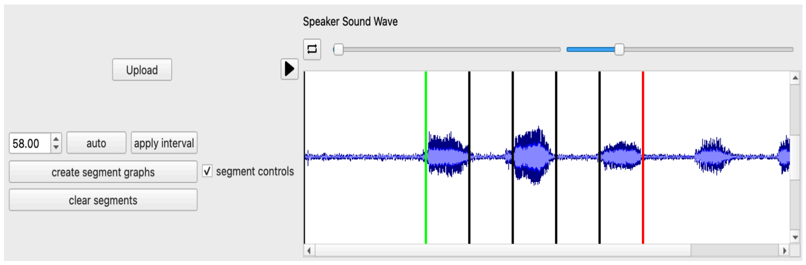
Key Members

- ‘Int audioDiviceNumber’: which audioDiviceNumber this recording comes from; for QMediaPlayer
- 'QPushButton *uploadAudioButton': Button for uploading audio files
- 'QMediaPlayer *player': Media player for playing audio
- 'QAudioOutput *audioOutput': Handles audio output
- 'QToolButton *playButton': Button to toggle play/pause
- 'Zoom *zoomButtons': Zoom controls for adjusting the waveform display
- 'QTimer *timer': Timer for updating scrubber position during playback
- ‘QString label’: label to display on this audio - distinguishes user audio

Key Methods

- ‘newAudioPlayer()’: Sets up the layout, buttons, waveform, and timer connections.

(image includes everything newAudioPlayer() holds)



- ‘void setTrackPosition(qint64 position)’: updates the current audio playback position and transforms the raw audio position into a normalized floating-point value between 0 and 1. Handles loop playback, reaching the end of the track, and synchronizing audio alignment across multiple tracks, emitting signals to update various UI components like the waveform chart and scrubber position. Also prevents accidental triggers near the track's end, updating play/pause button states, and coordinating with audio alignment features when audio tracks are being synchronized.

Slots

- ‘void uploadAudio()’: Handles the file selection process and media player setup, enables audio playing buttons and initializes scrubber features, and sends the audio file to the Spectrograph
- ‘void handlePlayPause()’: Manages the play/pause state of the audio player **player** and updates the timer **timer**. It also calls the **SetTrackPosition** to update the audio position (*see* **SetTrackPosition)**
- ‘void ZoomScrubberPosition()’: Adjusts the scrubber position when the waveform is zoomed.
- ‘void audioLoaded()’: Enables segmenting functionality after an audio file has been successfully loaded.
- ‘void updateTrackPositionFromTimer()’: Periodically updates the audio track position using a timer, providing smooth position tracking.
- ‘void updateTrackPositionFromScrubber(double position)’: Allows manually moving the audio playback position by interacting with the scrubber/timeline.
- ‘void updateTrackPositionFromSegment(QPair<double, double> startEnd)’: Sets the audio playback to a specific segment defined by start and end positions.
- ‘void segmentIntervalControlsEnable(bool ready)’: Enables or disables segment interval control buttons based on audio readiness.
- ‘void segmentCreateControlsEnable(bool ready)’: Enables the create segment graph button when segments are ready.
- ‘void segmentLengthShow(int numSamples, int sampleRate)’: Displays the length of the current audio segment in samples and seconds.
- ‘void toggleBoolManualSegments(double position)’: Enables manual segmentation mode and updates the delta value.
- ‘void toggleBoolAutoSegments()’: Enables automatic segmentation mode.
- ‘void watchForEndOfSegmentAudio(qint64 audioPosition)’: Monitors and handles playback when a segment's end is reached.
- ‘void handlePlayPauseButton()’: Manages the play/pause button interaction, toggling playback and updating the UI accordingly.

- ‘void clearSegmentsEnable(bool enable)’: Enables or disables the clear segments button.
- ‘void handleLoopClick()’: Toggles the loop playback functionality.
- ‘void handleSpectWithPlay()’: Pauses audio and resets position when spectrograph is loading during playback.
- ‘void handleWavClearing()’: Clears existing charts and intervals when a new audio file is uploaded.
- ‘void disableButtonsUntilAudio()’: Disables various audio control buttons until an audio file is loaded.
- ‘void enableAudioAligning(bool enable)’: Enables the audio alignment checkbox when a second audio file exists.
- ‘void disableAudioControls(bool disable)’: Disables audio controls for the second audio track during alignment.
- ‘void audioAligningSegmentControls(bool segEnabled)’: Manages audio alignment and segmentation control interactions.
- ‘void switchControlsWithAlign(bool aligning)’: Adjusts control states when audio alignment is activated.

Signals

- ‘emitLoadAudioIn(QString fName)’: Signals that an audio file is ready to be loaded into the waveform chart.
- ‘audioPositionChanged(double position)’: Broadcasts the current normalized audio playback position.
- ‘emitAutoSegmentBool(bool autoSegmentBool)’: Indicates whether automatic or manual segmentation is active.
- ‘segmentAudioNotPlaying(bool)’: Signals the current segment playback state.
- ‘secondAudioExists(bool)’: Notifies when a second audio file is available.
- ‘audioFileSelected(const QString &fileName)’: Indicates that an audio file has been selected.
- ‘playPauseActivated()’: Signals that the play/pause state has been toggled.
- ‘scrubberUpdate(double position)’: Broadcasts updates to the scrubber position.
- ‘audioEnded(bool disconnect)’: Signals when audio playback has ended or needs to be reset.

segmentgraph

File: segmentgraph.cpp

Description: segmentgraph visualizes the partitioned selection of the audio as defined by the user. Segmenting is specified through controls in Audio before the waveformsegments class partitions the audio into either user-defined constant-width or auto-generated segments. This class manages the visualization, user interactivity, and updating data.

Purpose:

- Creates a close-up, high resolution view of the wave to visualize component frequencies within the wave and allow for harmonic analysis

Key Features:

- See very short snippets of the selection of the wave - see actual wave instead of waveform
- Can scroll through sections to see how the harmonics and frequencies are changing over time
- Can play the audio from corresponding sections

The constructor 'SegmentGraph(int width, int height)' creates a layout for the widget that is filled with the exit button and slider above a QChartView. The widget is vertically and horizontally stretchable to fit the window.

Key Members

- ‘QChartView *graph': display for the zoomed-in sections
- ‘QList<QChart *> *charts’: charts for the graph to display corresponding to position of segmentSlider

- ‘QPair<double,double> startEndOfSelectedSegment': audio indices corresponding to the absolute start and ends of the selected segment
- ‘QList<QPair<double, double>> startEndSegmentAudioValues': audio indices corresponding to start and ends of each segment

Slots

- 'void SegmentGraph::slideSegments(int position)': Given a integer of the slider position, the QChartView will change its QChart to the corresponding value by index
- 'void SegmentGraph::exitView()': Upon pressing the exit button, the widget will be set to be invisible
- 'void SegmentGraph::updateGraphs(QList<QList<float>> segments)': The widget is set as visible, the chart values are cleared, and then refilled with new charts based on the values of segments. Using this new chart value, the slider is reset to find its values.
- ‘void clearView()’: clears data from the charts and graph, reinitializes the graph. Emits clearSegmentsEnable(false) to disable re-clearing a cleared graph
- ‘void getSegmentStartEnd(QList<QPair<double, double>> startEndValues)’: sets startEndSegmentAudioValues to specified list of pairs
- ‘void getSegmentAudioToPlay(int segmentPosition)’: retrieves specified audio segment start and end into startEndOfSelectedSegment, enables audio play functionality
- ‘void playSegmentAudio()’: sends the corresponding audio start and end to the audio playback in Audio to play it; maintains state
- ‘void changePlayPauseButton(bool segAudioNotPlaying)’: maintains state of play button

Signals

- ‘void sendPlaySegmentAudio(QPair<double, double>)’: sends absolute positions of start and end of specified segment
- ‘void clearSegmentsEnable(bool)’: indicates we have cleared chart

Notes:

- Default values for the width and height are 400 and 200 respectively.
- This widget is only set to be visible when the updateGraphs slot is called.

spectrograph

Files: spectrograph.cpp, spectrograph.h

Description:

The spectrograph class provides a spectrograph visualization for an audio file using a QImage and QPixmap.

Purpose:

- Creates a visual representation (spectrogram) from audio data across frequency and time axes, complementing the wavform visualization across amplitude and time

Key Features:

- Displays a spectrogram of audio data
- Provides functionality for enabling / disabling peak amplitude visualization
- includes audio decoding and FFT transformation using FFTW library

Key Members

- ‘QAudioDecoder *decoder’: pointer to a Qt audio decoder object used to decode audio files by reading and converting audio buffers into usable sample data.
- ‘QVector<double> accumulatedSamples’: vector that stores decoded audio samples as normalized double-precision floating-point values between -1.0 and 1.0
- ‘fftw_complex *in’: pointer to a complex array used as input for Fast Fourier Transform (FFT) computation, storing real and imaginary components of the input signal.
- ‘fftw_complex *out: same as above, but output
- ‘fftw_plan plan’: An FFTW plan that defines the specifics of the Fourier transform computation, created with parameters for forward transformation.
- ‘fftw_complex *data, *fft_result’: Input and output arrays used for FFT computation, with data holding the input signal and fft_result storing the frequency-domain representation.

- ‘QVector<QVector<double>> spectrogram’: A 2D vector representing spectrogram data, storing amplitude values of different frequencies per window of the audio signal.
- ‘QVector<double> hammingWindowValues’: A vector of pre-computed Hamming window values used to smooth audio signal chunks before FFT,.

Key Methods

- 'void setupSpectograph(QVector<double> &accumulatedSamples)': Prepares the spectogram using FFT for an audio segment
- 'void hammingWindow(int windowLength, QVector<double> &window)': Applies hamming window to smooth audio data

Slots

- 'void bufferReady()': Processes ready audio buffers by decoding into sample data
- 'void loadAudioFile(const QString &fileName)': initilizes processing
- 'void processAudioFile(const QUrl &fileUrl)' : takes in fileUrl to sample values and prepares them for FFT by calling bufferReady and finish signals on QAudioDecoder
- 'void renderToPixmap': Creates spectogram visualization using QPixmap
- ‘decodingFinished()’: once QAudioDecoder is done, window samples should be displayed

---

waveformsegments

File: waveformsegments.h, waveformsegments.cpp

Description: This class segments the audio for the segmentgraph class. There are two main functionalities: auto segmenting (based on zero-crossings and thresholds), or user-defined segments of fixed width. This class implements auto-segmenting, and finally turns the original audio and a list of breaks into a segmented list of audio samples for the segmentgraph class to graph

Purpose:

- Implements auto-segmentation of an audio sample for close-form waveform analysis
- Turns list of user or auto-generated breakpoints into a list of very short audio samples to be graphed by a segmentgraph instance

Key Members

- 'QList<QList<float>> wavSegments': holds the wavSegments to be emitted
- 'QList<float> originalAudio': the original sample audio data to be partitioned
- ‘QList<QPair<double, double>> wavSegmentStartEndPositions’: the start and end indices of each segment
- ‘double audioSampleLength’ - length of original file
-

Key Methods

- ‘WaveFormSegments(QList<float> _audioSamples = QList<float>(), QObject *parent = nullptr)’; - constructor
- 'uploadAudio(QList<float> audio)': gives the object the audio data to slice

Slots

- 'void collectWavSegment(QList<int> segmentPlaces)': collects wav segments and divides audio graph based on segment location
- 'void clearAllWavSegments()': clears segment information (to be used when user clears segments from graph)
- 'void autoSegment(QList<float> dataSample)': creates automated points in the audio wave for segmentation

Signals

- 'createWavSegmentGraphs(QList<QList<float>>)' : signal for sending the list of audio samples to segmentgraph to draw

- ‘void drawAutoSegments(QList<int>)’: signal for sending the segment indices to wavform to draw
- ‘void storeStartEndValuesOfSegments(QList<QPair<double, double>>)’: signal for sending the start and end indices of each segment to segmentgraph. Used for playing audio and for chart axis labelling

Notes:

- This works in tandem with segmentgraph, wavform and user input in audio on the graph closely, all changes to those should involve double checking the functionality here is not compromised */

---

wavfile

File: wavfile.cpp, wavfile.h

Description: This class provides an interface allowing access to the contents of a WAV file. It includes functionality to read, parse, and extract audio data and metadata from a WAV file. The class supports loading WAV files, reading header information, and extracting audio samples, and standardized storing of audio data across different bit-depths.

Purpose:

- Retrieves audio properties such as sample rate, number of channels, and bit depth
- Extracts raw audio data and samples for further processing or visualization

Key Members

- 'QString filePath': Path to the WAV file
- 'int sampleRate': Sample rate of the audio file
- 'int numChannels': Number of audio channels (1 for, mono, 2 for stereo)
- 'int bitDepth': bit depth of each audio sample

- ‘int dataSize’: the size of data section in the serialization
- 'QByteArray audioData': Extracted raw audio data from the file
- 'Qlist<float> samples': Parsed audio samples as float values between -1.0 and 1.0

Key Methods

- 'WavFile(const QString& filePath, QObject* parent = nullptr)': Constructor that initializes the WAV file with the provided file path
- 'int getSampleRate() const': Returns the sample rate of the audio channels
- 'int getnumChannels() const': Returns number of audio channels
- 'int getBitDepth() const': Returns the bit depth of the audio file
- 'QByteArray getAudioData() const': Returns the raw audio data as a byte array
- 'QList<float> getAudioSamples() const': Returns the parsed audio samples
- 'bool loadFile()': Loads and processes the WAV file. Opens the filePath, splitting the file according to specification before passing the header and data to readHeader() and collectAudioSamples() respectively
- 'bool readHeader(const QByteArray& headerData)': Parses and validates the WAV file header
- 'bool readData()': Validates the size of the audio data chunk - there were issues with the footer being parsed as data in some audio files.
- 'void collectAudioSamples()': Extracts individual audio samples from the raw audio data. Parses each sample as a float before fitting to 32-bit float specification.

Signals

- 'void fileLoaded(bool success)': Boolean signal to indicate success/failure in loading file

---

wavform

File: wavform.cpp, wavform.h

Description: This header file defines the 'WavForm' class, which is a custom 'QGraphicsView' that creates a waveform visualization for audio data using 'QGraphicsView'. This class renders an audio waveform using the data from a 'WavFile' object and allows user interaction such as scrubbing and segment selection with interval-based annotations.

Purpose:

- Creates a visual representation (waveform) of WAV audio data.
- Supports user interaction like scrubbing and segment selection.
- Emits signals to communicate playback position and interval/segment details.

Key Members

- 'QGraphicsScene scene': The graphics scene used to render the waveform and scrubber.
- 'QGraphicsLineItem* lastLine': Pointer to the last scrubber line drawn.
- 'WavFile* audio': Pointer to the associated `WavFile` object containing audio data.
- 'bool segmentControls': Tracks whether the user is in segment control mode to initiate start/end line selection; used to determine mouse click behavior
- 'QPointF startSegmentP, endSegmentP': Start and End point positions for a selected segment of audio from segmentgraph.
- 'QGraphicsLineItem *startSegment, *endSegment': Graphic line items corresponding to the start and end of the selected interval of audio.
- 'QList<QGraphicsLineItem*> intervalLines': List of interval lines within segment start/end lines.
- 'double delta': Distance between intervals within user selected segments.
- 'QList<float> intLinesX': List of x-coordinates of intervals within selected segment.
- ‘QPointF viewCenterPoint’: maintain the central point of the waveform chart; for maintaining scrubber and zoom state and for centering on scrubber

Key Methods

Public Methods:

- 'explicit WavForm(int width, int height)': Constructor initializing the view dimensions; provide audio later
- 'void audioToChart()': Loads audio data from the `WavFile` and creates a waveform visualization.
- 'void setChart(QList<float> data, int width, int height)': Draws the waveform using audio sample data.
- 'QList<float> getSamples()': Gets audio samples currently displayed in waveform.
- 'void updateDelta(double delta)': Updates delta which calculates spacing between interval lines in segment selections.
- ‘void drawIntervalLinesInSegment(double x)’: for user-defined delta, draw the segment lines from the start of desired interval to the end
- ‘void updateIntervals(int oldChartWidth)’: takes the interval lines generated from user-defined delta-based segmentation and redraws them on the screen after the zoom of the chart changes

Protected Methods:

- 'void mousePressEvent(QMouseEvent evt) override': Handles user interaction for updating the scrubber and segment lines.

Slots

- 'void uploadAudio(QString fName)': Loads a WAV file and generates its waveform visualization.
- 'void updateScrubberPosition(double position)': Updates the scrubber position based on a relative position.
- 'void updateChart(int width, int height)': Redraws the chart to fit new dimensions.
- 'void switchMouseEventControls(bool segmentControlsOn)': Enables or disables segment control mode.
- 'void sendIntervalsForSegment()': Emits a list of audio samples indices for interval positions in segment selections.
- 'void clearIntervals()': Clears segment and interval position data and graphic elements.
- ‘void changeBoolAutoSegment(bool _boolAutoSegment)’: update boolean boolAutoSegment that represents if we are using auto-generated zero-crossing based

segments or user-defined delta; if we are using auto-defined, emit signal to get those and display on the visualization

- ‘void changeCenterOnScrubber(Qt::CheckState checkedState)’: update state of centerOnScrubber; converts from Qt::CheckState to boolean

Signals

- 'void sendAudioPosition(double position)': Emitted when the scrubber position changes.
- 'void sceneSizeChange()': Emitted when scene size has been changed.
- 'void audioFileLoadedTrue()': Emitted when an audio file is loaded.
- 'void segmentReady(bool ready)': Emitted when start and end segment lines are declared.
- ‘void segmentLength(int numSamples, int sampleRate)’:
- 'void intervalsForSegments(QList<int> intervalLocations)': Emits a list of audio sample indices for interval lines.
- 'void chartInfoReady(bool ready)': Emitted when segment selections and intervals lines are defined or cleared.
- ‘void clearAllSegmentInfo()’: when emitted, sends to segmentgraph to clear segment data and visualization
-

Notes:

- The waveform is visualized using properties of audio samples from 'WavFile'

---

zoom

File: zoom.h

Description: This header file defines the 'Zoom' class, which provides functionality for controlling the zoom level of our waveform view using horizontal and vertical sliders.

Note that the sliders are part of the audio class; this class connects to those sliders and implements the functionality; sending the zoom signal to the waveform view.

Purpose:

- Allows user to zoom in and out of waveform view along horizontal and vertical axes
- Emits updated dimensions of the viewer and scrubber position whenever the zoom level has changed

Key Members

- 'QHBoxLayout *zoomLayout': Horizontal layout containing the QSliders for zoom controls (width and height)
- 'QSlider *verticalSlider': QSlider for controlling vertical zoom - is initialized in Audio, this just maintains state for it
- 'QSlider *horizontalSlider': QSlider for controlling horizontal zoom - same as above
- 'int graphWidth', ‘graphHeight’, ‘zoomWidth’, ‘zoomHeight’: initial absolute and relative dimensions

Key Methods

- 'Zoom(QWidget *parent = nullptr, int viewW = 400, int viewH = 200)': Constructor initializing the zoom functionality
- ‘void resetZoom()’: resets the zoom to base: graphWidth and graphHeight. Emits corresponding signals: zoomGraphIn and resetZoomActivated

Slots

- 'void verticalZoom(int position)': Updates the vertical zoom level and emits new dimensions
- 'void horizontalZoom(int position)': Updates the horizontal zoom level and emits new dimensions

Signals

- 'void zoomGraphIn(int width, int height)': Signal emitted whenever vertical or horizontal zoom level changes

- ‘resetZoomActivated’: signal emitted when zoom is reset
- ‘horizantal/verticalSliderChanged’ - wrapper signals around default slider moved signal