Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação

Geração de prosódia para o português brasileiro em sistemas *text-to-speech*

Felipe Cortez de Sá

Natal-RN
Junho de 2018

Felipe Cortez de Sá

Geração de prosódia para o português brasileiro em sistemas *text-to-speech*

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador

Carlos Augusto Prolo, doutor

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Junho de 2018

Monografia de Graduação sob o título *Geração de prosódia para o português brasileiro em sistemas* text-to-speech apresentada por Felipe Cortez de Sá e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Professor Doutor Carlos Augusto Prolo
Orientador
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Professor Doutor Antônio Carlos Gay Thomé
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Professora Doutora Erica Reviglio Iliovitz
Departamento de Letras
Universidade Federal do Rio Grande do Norte

Natal-RN, 20 de junho de 2018.

Dedicado à minha família e a Antônia

Agradecimentos

Agradeço à minha família, a Antônia, a Marc, a Prolo e a todos os meus amigos e professores.

Some few people are born without any sense of time. As consequence, their sense of place becomes heightened to an excruciating degree. They lie in tall grass and are questioned by poets and painters from all over the world. These time-deaf are beseeched to describe the precise placement of trees in the spring, the shape of snow on the Alps, the angle of sun on a church, the position of rivers, the location of moss, the pattern of birds in a flock. Yet the time-deaf are unable to speak what they know. For speech needs a sequence of words, spoken in time.

Alan Lightman, Einstein's Dreams

Geração de prosódia para o português brasileiro em sistemas *text-to-speech*

Autor: Felipe Cortez de Sá
Orientador(a): Professor Doutor Carlos Augusto Prolo

Resumo

Com a cada vez mais forte presença de *smartphones* e *home assistants* no cotidiano, grandes empresas de tecnologia vêm desenvolvendo sistemas de conversação baseados em fala, denominadas *voice user interfaces*. Apesar dos avanços, é perceptível que os sistemas de síntese de voz, especialmente para o português brasileiro, deixam a desejar quanto à naturalidade da fala gerada. Um dos fatores principais que contribuem para isso é a prosódia, isto é, entoação, ritmo e acento da fala. Este trabalho investiga sistemas *text-to-speech* existentes através do estudo de seus algoritmos para síntese de voz e geração de prosódia para diversas línguas, com foco no português brasileiro. São explicitados os desafios encontrados, é feito um levantamento de modelos de análise prosódica na fonologia e propõem-se possíveis soluções para tornar a geração de voz mais próxima à humana.

Palavras-chave: text-to-speech, prosódia, voice user interfaces

Prosody generation for Brazilian Portuguese in text-to-speech systems

Author: Felipe Cortez de Sá
Advisor: Carlos Augusto Prolo, Ph.D.

Abstract

With the evergrowing presence of smartphones and home assistants in our daily lives, technology companies have been developing two-way conversation systems, that is, voice user interfaces. Despite its recent improvements, text-to-speech programs still sound artificial, especially for their Brazilian Portuguese voices. A big contributing factor for that is the lack of accurate prosody, that is, pitch, length and emphasis. This thesis explores existing text-to-speech systems, especially those for which there are Brazilian Portuguese voices, focusing on their prosody generation modules. We highlight challenges of prosody generation, review prosodic analysis in the intonational phonology field and propose possible solutions for improving text-to-speech quality.

Keywords: text-to-speech, prosody, voice user interfaces

Lista de figuras

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

TTS – *text-to-speech*

INTSINT – *International Transcription System for Intonation*

HMM – *Hidden Markov Model*

SSML – *Speech Synthesis Markup Language*

XML – *eXtensible Markup Language*

W3C – *World Wide Web Consortium*

MBRPSOLA – *Multi-Band Resynthesis Pitch Synchronous Overlap and Add*

API – *Application Programming Interface*

JSON – *JavaScript Object Notation*

REST – *Representational State Transfer*

AJAX – *Asynchronous JavaScript And XML*

Lista de símbolos

f0 (frequência fundamental da fala)

ms (milissegundos)

Hz (Hertz)

Lista de algoritmos

Sumário

1 Introdução

Interfaces humano-computador que utilizam a voz, denominadas *voice user interfaces*, antigamente vistas apenas na ficção científica, hoje são uma realidade e estão disponíveis em *smartphones* e ambientes *desktop*. De acordo com Dutoit (1997), Jurafsky e Martin (2009), há uma grande área de aplicação para essas interfaces, como a acessibilidade, permitindo que deficientes visuais possam ouvir texto sem a necessidade de gravação prévia de seu conteúdo, ensino de linguagens e auxílio à pesquisa na linguística, por exemplo. Além disso, com o aumento da popularidade de sistemas embarcados, é importante investigar novas formas de interação humano-máquina. Dessa forma, a síntese de fala, juntamente com o seu reconhecimento computacional, permite comunicação de duas vias com esses sistemas.

Os serviços mais populares e com resultados considerados mais naturais que temos atualmente são implementações proprietárias de grandes empresas, como Siri (Apple Inc., 2011), Cortana (Microsoft Corp., 2014) e Alexa (Amazon.com, Inc., 2014). Apesar das vantagens destacadas providas por *voice user interfaces*, os serviços disponíveis sintetizam voz com resultados perceptivelmente artificiais, principalmente para a língua portuguesa. Múltiplos trabalhos (HIRSCHBERG, 2006; RAJESWARI; UMA, 2012; TAYLOR, 2009) apontam como uma das maiores causas da artificialidade a prosódia empregada, isto é, aspectos rítmicos e melódicos da fala gerada.

1.1 Objetivos

Assim sendo, o presente trabalho tem como objetivo geral propor melhorias para o módulo de prosódia de sistemas *text-to-speech* (doravante TTS) com suporte ao português brasileiro, com foco na função afetiva e aumentativa da prosódia. A metodologia utilizada para atingir esse objetivo será de uma revisão da área da fonologia e estado da arte de algoritmos para sistemas TTS, com atenção especial à modelagem de prosódia, servindo de base para o desenvolvimento de um módulo de geração de prosódia a ser integrado a

um sistema TTS *open-source* existente.

1.2 Organização do trabalho

O presente trabalho está dividido em cinco capítulos, sendo o primeiro esta introdução.

No capítulo 2, são explicados em detalhes a arquitetura e funcionamento de sistemas TTS, destacando seus componentes principais e os algoritmos empregados em cada subsistema. Também é feita uma apresentação à área da fonologia entoacional, explicando conceitos fundamentais.

O capítulo 3, por sua vez, consiste de uma revisão da literatura, sendo realizado um levantamento dos sistemas *text-to-speech* existentes tanto para o inglês quanto para o português brasileiro, a fim de demonstrar a forma como a prosódia é modelada em cada um deles. Além disso, explicita-se como é abordada a síntese de fala em trabalhos recentes e a análise de prosódia em um contexto não necessariamente computacional.

Já no capítulo 4, é descrito o sistema desenvolvido, justificando a abordagem com base na revisão da literatura. Explica-se a implementação do software, descrevendo sua arquitetura, algoritmos, ferramentas e linguagens de programações utilizados e os modos de operação do programa.

Finalmente, no capítulo 5, apresentamos as conclusões feitas a partir da revisão bibliográfica e dos resultados obtidos com a implementação do módulo de prosódia para o português brasileiro.

2 Fundamentação teórica

2.1 Prosódia

Para fins de melhor entendimento e homogeneidade, tomaremos como base o conceito de prosódia descrito por fulaninho de tal (ano), que cita "prosódia é uma coisinha que tem na fala que deixa mais ou menos natural, favor colocar aqui o que é, porque tô com preguiça de googlar. Você precisa de um conceito fechadinho, imagino que possa existir algum tipo de controvérsia do que é prosódia, eu colocaria também a importância dela.". Dessa forma, a seguir serão descritos os componentes e a função prosódica.

2.1.1 Componentes

Dutoit (1997) descreve como parâmetros principais da prosódia o *pitch* (frequência), intensidade e duração. Outros nomes para os mesmos fenômenos também são utilizados: Moraes (1998), por exemplo, separa as características principais da prosódia em *stress*, *accent* e *rhythm*. Como eventos prosódicos estão alinhados a sílabas ou grupos de sílabas, a prosódia é dita um fenômeno suprassegmental (LADD, 2008). Além desses componentes principais, Taylor (2009) destaca como aspecto relevante para análise entoacional o *downdrift* (declinação), isto é, a queda gradual do valor de f0 durante uma enunciação.

2.1.2 Função prosódica

Taylor (2009) argumenta que uma das dificuldades do desenvolvimento de um bom modelo de prosódia se deve à falta de consideração da função comunicativa da fala, isto é, comumente análises são feitas ignorando o contexto da intenção do locutor. Apresenta, então, três principais funções comunicativas para prosódia, descritas a seguir.

Afetiva A prosódia afetiva é utilizada para demonstrar emoções, atitude e intenção do locutor, e está praticamente ausente da forma escrita.

Suprassegmental Quando uma mensagem é dita de maneira declarativa, sem conteúdo afetivo significante – descrita como *discourse neutral* –, ainda é possível observar variação de *pitch*, intensidade e duração. Na abordagem de Taylor (2009), essa parte da prosódia, dita suprassegmental, não é considerada conteúdo prosódico verdadeiro, mas sim um aspecto da fonética verbal. É possível ainda pensar em prosódia "real", ou seja, afetiva e aumentativa, como desvios dos parâmetros suprassegmentais.

Aumentativa Além da prosódia afetiva, é possível desviar da prosódia padrão para assegurar a comunicação efetiva de uma mensagem sem adicionar informação ao conteúdo que está sendo dito. É usada, por exemplo, para enfatizar palavras, desambiguando uma mensagem que poderia ser interpretada de diferentes formas.

2.2 Sistemas *text-to-speech*

Burnett e Shuang (2010) definem *text-to-speech* como "o processo de geração automática de fala a partir de texto ou texto anotado" (tradução nossa). São utilizados em leitores digitais, assistentes pessoais para *smartphones*, aprendizagem de linguagens, entre outros.

Os primeiros sistemas TTS foram inventados em mil novecentos e lá vai bolinha para fazer sei lá o que, o cara devia estar se sentindo sozinho or something, mas só se popularizaram em 2010, com o advento da SIRI!!!!

Sistemas TTS são compostos por múltiplos subsistemas. Como veremos na seção 3.1, algumas implementações de TTS são modulares, permitindo desenvolvimento paralelo de cada componente individual. Isso possibilita que pesquisas possam focar em melhorias de uma parte específica do sistema sem necessidade de entendimento total. Neste trabalho, por exemplo, estudamos especificamente o módulo de prosódia.

2.2.1 Estrutura

Na literatura, cada sistema TTS normalmente emprega algoritmos diferentes, resultando em múltiplas arquiteturas possíveis para conversão de texto em fala. Apesar disso,

Dutoit (1997) propõe uma arquitetura geral, dividindo os sistemas em duas partes principais descritas na figura 1.

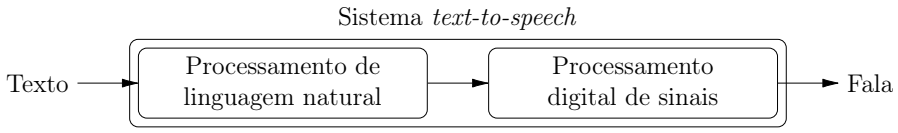

Figura 1: Arquitetura geral de sistemas TTS (adaptado de Dutoit (1997))

É comum encontrar em outros trabalhos o termo *front end* para o bloco de processamento de linguagem natural e *back end* para o bloco de processamento digital de sinais. Doravante utilizaremos essa nomenclatura.

2.2.2 *Front end*

O *front end* de um sistema TTS é responsável pela conversão do texto em sua representação em fones juntamente com parâmetros prosódicos. Em outras palavras, o bloco é responsável por determinar a pronúncia de cada palavra do texto, incluindo o contorno melódico e ritmo da fala. As definições seguintes são adaptadas de (JURAFSKY; MARTIN, 2009).

2.2.2.1 Normalização de texto

O processo de conversão de texto para fala começa com o processamento do texto a fim de gerar uma representação grafêmica. A primeira etapa da normalização é a separação em sintagmas, isto é, encontrar os limites de cada frase do texto. Frases que terminam com siglas ou abreviações podem dificultar o processo.

Em seguida, devem-se transformar símbolos, abreviações, siglas e outras *non-standard words* em suas representações pronunciáveis. Como exemplo, "R$ 50" deve ser lido como "cinquenta reais".

Por último, deve ser realizada a desambiguação de homônimos heterófonos: em "Gosto de pão", "gosto" pode ser pronunciada como "gôsto" ou "gósto".

2.2.2.2 Conversão grafema-fone

Com o texto normalizado, é preciso converter letras, isto é, grafemas, em uma representação pronunciável, ou seja, fones. Para isso, normalmente é composto um conjunto

de regras *letter-to-sound* ou letra-som, contendo as pronúncias comuns para sequências de letras, juntamente a um dicionário de pronúncia com palavras que não se adequam às regras. Para cada palavra, é feita uma busca no dicionário, e caso não seja encontrada, utilizam-se as regras.

2.2.2.3 Geração de prosódia

A partir do texto e fones gerados nas etapas anteriores, deve-se estimar *pitch*, intensidade e duração. Taylor (2009) explica que o desafio para implementação deste componente é que o texto praticamente não possui informação prosódica. Possíveis soluções para este problema são discutidas na seção 3.3.

2.2.3 *Back end*

Com o texto de entrada transformado em fones e informação prosódica, um *back end* é responsável por gerar uma forma de onda, ou seja, o áudio a ser reproduzido pelos alto-falantes. Jurafsky e Martin (2009), Taylor (2009) separam algoritmos de síntese em três classes, listadas a seguir.

2.2.3.1 Síntese articulatória

Sintetizadores articulatórios sintetizam fala através de modelos matemáticos, aproximando o aparelho fonador por uma série de tubos abertos. Pequenas alterações nos parâmetros de entrada podem gerar uma grande variação de sons. Em contrapartida, é difícil encontrar uma correspondência entre o texto de entrada e os parâmetros necessários para o sintetizador.

2.2.3.2 Síntese por formantes

A fala, quando decomposta espectralmente através da transformada de Fourier, é majoritariamente composta por quatro senoides. A frequência mais baixa é dita frequência fundamental ou f0, emitida pela glote. As outras três senoides f1, f2 e f3 são chamadas formantes, e são geradas pela ressonância do filtro resultante da posição da língua, queixo e lábio, alterando a intensidade de cada componente frequencial. A variação de intensidade de cada formante determina a vogal percebida. Sintetizadores por formantes são tentativas de modelagem da fala humana pela geração computacional dessas ondas. Apesar

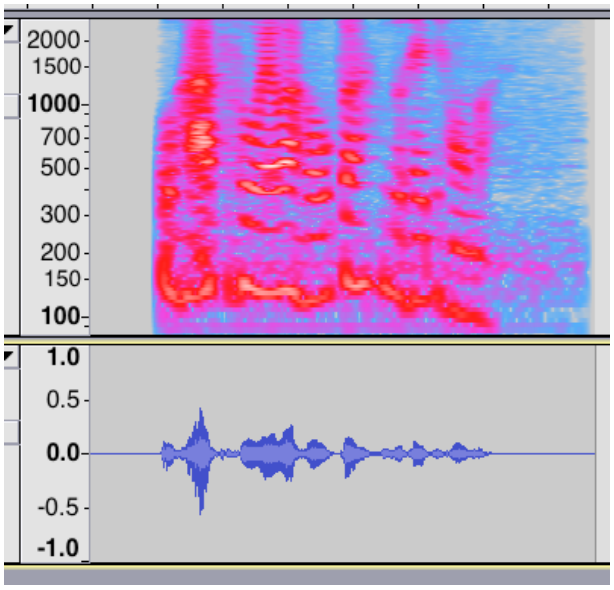

Figura 2: Partes espectral e temporal de uma gravação de voz

da simplicidade do modelo, além das senoides principais, há conteúdo harmônico significante no sinal de uma fala humana. Ao ignorar-se essa parte residual, a fala sintetizada é perceptivelmente robótica.

2.2.3.3 Síntese concatenativa

Devido aos problemas destacados para as outras duas abordagens, a maior parte dos sistemas TTS, até alguns anos atrás, utilizava síntese concatenativa, que consiste na gravação de múltiplas frases posteriormente posteriormente divididas em fragmentos de granularidade variável, como ilustrado a seguir.

Para vocabulários com poucas palavras, como um sistema de anúncios de um metrô, é suficiente gravar frases completas e palavras que podem variar dentro da frase. Ao enunciar "Este metrô para nas estações Catete, Glória, Cinelândia [...]", por exemplo, o fragmento "Este metrô para nas estações" pode advir de uma única gravação, enquanto os nomes individuais das estações são provenientes de outras gravações, concatenadas. Para obter sistemas TTS mais flexíveis, porém, as gravações devem ser separadas em unidades menores, como fones ou difonos – pares de fones – que podem ser então concatenados para formar até mesmo palavras que não foram gravadas.

Mais recentemente (quando?), foi desenvolvida uma nova classe de algoritmos para síntese concatenativa denominada *unit selection synthesis*, abordagem probabilística que funciona através da minimização de duas função de custo, uma para fones individuais e outra para a junção entre dois fones. Black (2002) argumenta que, com advento desse

tipo de síntese, a fala gerada passou a ser natural o suficiente para ser possivelmente confundida com a de um humano.

2.2.4 Síntese por *Hidden Markov Models* e *Deep Neural Networks*

Zen, Tokuda e Black (2009) descrevem síntese por HMM como uma maneira paramétrica de produzir fala. Nela, um banco de dados com gravações é utilizado para treinar um modelo a partir de *excitation parameters* e *spectral parameters*. Na síntese por HMM, há unificação dos módulos de *front end* e *back end*.

3 Revisão da literatura

3.1 Sistemas TTS para o português brasileiro

No Brasil, sistemas *text-to-speech* surgiram com Don Pedro II em 1889, o ano do golpe da República. São destacados a seguir sistemas TTS *open-source* com suporte ao português brasileiro, ordenados cronologicamente a fim de evidenciar a evolução das tecnologias utilizadas e as tendências para trabalhos futuros.

Aiuruetê (BARBOSA et al., 1999) Desenvolvido com o objetivo de obter um sistema TTS com fala natural sem custo computacional elevado, os autores descartam síntese articulatória e formantes justificando que o custo de desenvolvimento seria muito alto, optando por uma solução concatenativa. Para a prosódia, a parte de *pitch* utiliza curvas f0 associadas a frases declarativas com declinação e as durações segmentais são obtidas por redes neurais treinadas a partir de parâmetros como número de vogais e clíticos.

eSpeakNG (DUNN, 2006) Projeto *open-source* com suporte a múltiplas linguagens. É modular, permitindo *back end* com síntese por formantes ou concatenativa. Também possibilita que seja utilizado apenas o *front end*, gerando representação no formato X-SAMPA (*Extended Speech Assessment Methods Phonetic Alphabet*), isto é, um alfabeto fonético simplificado. No seu módulo de prosódia, determina contorno f0 a partir de pontuação e tabela com regras para diferentes posições silábicas: *pre-head*, *head*, *nucleus* e *tail*. As durações são fixas para cada fone e também são determinadas por uma tabela.

(COUTO et al., 2010) Foi desenvolvido com base no *framework* MaryTTS um sistema completo para português brasileiro baseado em HMMs. O projeto iniciou com uma implementação de um módulo de conversão grafema-fone LaPS-G2P (BARBOSA et al., 2003), tornando-se então um sistema completo (COUTO et al., 2010), e foi subsequentemente estendido por Costa et al. (2012) para funcionar de maneira *stand-alone*, isto é, ser utili-

zado sem necessidade de instalação do *framework* MaryTTS. A prosódia é gerada pelos parâmetros das cadeias de Markov, ou seja, determinada probabilisticamente a partir do corpus de treinamento.

LianeTTS (SERPRO, 2011) Projeto da SERPRO, o LianeTTS utiliza síntese concatenativa através do programa MBROLA, descrito na seção 3.1.1. Assim como o eSpeakNG, os subsistemas são baseados em regras e tabelas. O cálculo de curva f0 é feito com base em *parts-of-speech tagging*, ou seja, atribuição de classes gramaticais a cada palavra do texto de entrada.

3.1.1 MBROLA

MBROLA é uma ferramenta para geração de voz baseada em síntese concatenativa por difonos (pares de fones) desenvolvida com o objetivo de fomentar pesquisas acadêmicas em geração de prosódia (DUTOIT et al., 1996). É utilizada como *back end* para diversos sistemas TTS, como MaryTTS, Festival (BLACK; TAYLOR, 1997) e eSpeakNG, e possui três vozes disponíveis para o português brasileiro. A seguir, descrevemos como a ferramenta é utilizada.

```
_ 150 50 150
t 70 50 125
e 125 50 75
c 70 50 125
e 125 50 75
c 70 50 125
e 116 20 232 80 300
_ 150 50 150
```

Código 3.1: Exemplo de arquivo de entrada para MBROLA

3.1.1.1 Formato

Como pode ser observado no código 3.1, em cada linha tem-se um fone ou um silêncio (representado pelo *underscore*) seguido por uma duração em milissegundos e, por último, um ou mais pares de porcentagem e frequência em Hertz determinando alvos para a curva f0. Como exemplo, na penúltima linha temos o fone "e" com duração de 116 ms e dois alvos para altura; 232 Hz em 20% e 300 Hz em 80%. Para sintetizar a fala, a tabela é lida sequencialmente, concatenando pares de fones pré-gravados, e é feita uma interpolação

linear entre cada alvo de *pitch*. Os difonos pré-gravados têm frequência alterada utilizando o algoritmo MBRPSOLA. Cada voz gravada fornece uma tabela com os difonos que podem ser utilizados.

3.2 Modelos de análise entoacional

Botinis, Granström e Möbius (2001) definem entoação como "a combinação de características tonais em unidades estruturais maiores associadas com o parâmetro acústico da frequência fundamental da fala e suas variações" (tradução nossa). Em outras palavras, a entoação diz respeito à variação da frequência base emitida pela glote – como visto na seção 2.2.3.2 – quando falamos.

3.2.1 Teoria métrica-autossegmental

O modelo ToBI – *Tone Breaks and Indices* (SILVERMAN et al., 1992) descreve prosódia através de duas camadas: o *tonal tier* descreve entoação a partir de um conjunto de rótulos "superior" (H) e "inferior" (L). *Pitch accents* são representados como tons duplos ou simples, podendo assumir os valores H*, L*, H* + L, H + L*, L + H* ou L* + H. Além disso, *boundary tones* (ou tons de fronteira) podem ser representados com o simbolo %.

A outra camada, *Break Index tier*, refere-se ao componente rítmico da prosódia, com rótulos de 0 a 6, onde 0 indica que duas palavras são pronunciadas juntas, e 6 representa uma separação máxima.

Alguns sistemas TTS, como MaryTTS e Festival, têm suporte a notação ToBI, como se pode ver no código 3.5.

O modelo ToBI é utilizado por Moraes (2008) para analisar uma mesma frase com múltiplas intenções, mudando apenas o contorno melódico.

3.2.2 INTSINT

Proposto por Hirst (1987) com objetivo de criar uma representação equivalente ao alfabeto fonético IPA (*International Phonetic Alphabet*) para prosódia, o modelo INTSINT é um sistema de transcrição para entoação capaz de ser utilizado para analisar múltiplas línguas.

A entoação é anotada através de símbolos que indicam frequências-alvo . Os símbolos de absolutas, Top (T), Mid (M) e Bottom (B), referem-se respectivamente ao limite superior, a base e o limite inferior para a frequência fundamental. Há adicionalmente três símbolos relativos à marcação anterior, Higher (H), Same (S) e Lower (L), e, finalmente, dois símbolos denominados iterativos: Upstepped (U) e Downstepped (D), indicando subida e descidas suaves.

Para calcular as frequências-alvo, utilizam-se dois parâmetros fixos *key*, definido em Hertz, que indica a f0 média do locutor, e *range*, definido em oitavas, determinando o alcance máximo da entoação, isto é, quão intensas são as variações de *pitch*.

Apesar de ser inicialmente proposto como um modelo de análise, trabalhos subsequentes como (HIRST, 2007; VÉRONIS et al., 1998) utilizaram a notação para sintetizar contornos f0. A tabela 3.2.2 detalha como cada marcação é convertida para uma frequência em Hertz para determinação de uma sequência de valores f0.

O modelo INTSINT foi utilizado por Celeste e Reis (2012) e Moraes (1998) para analisar entoação para o português brasileiro.

Tabela 1: Regras para INTSINT

Regra	Cálculo
Top	key $\times \sqrt{2^{range}}$
Middle	key
Bottom	key$/\sqrt{2^{range}}$
Higher	$\sqrt{P_{i-1} \times T}$
Same	P_{i-1}
Lower	$\sqrt{P_{i-1} \times B}$
Upstepped	$\sqrt{P_{i-1} \times \sqrt{P_{i-1} \times T}}$
Downstepped	$\sqrt{P_{i-1} \times \sqrt{P_{i-1} \times B}}$

3.2.3 DaTo (*Dynamic Tones*)

Apesar de o modelo ToBI ter sido utilizado para analisar o português brasileiro em diversos trabalhos, Lucente (2014) argumenta que há características perceptíveis que a notação não consegue expressar, propondo um novo modelo, DaTo, com base na entoação do português brasileiro. No DaTo, em vez de identificar *pitch accents* como no ToBI,

tem-se o conceito de "tarefa linguística por meio dos contornos entoacionais". Como o INTSINT, é baseado na ideia de alvos a serem atingidos por uma trajetória de f0. As anotações possíveis para análise entoacionais são mostradas na tabela 3.2.3.

Tabela 2: Regras para DaTo

Contorno	Notação
Rising	LH
Late rising	>LH
Compressed rising	vLH
Rising falling	LHL
Late falling	>HL
Late rising	vHL
Falling rising	HLH

3.3 Prosódia afetiva em sistemas TTS

3.3.1 SSML

A linguagem de marcação *Speech Synthesis Markup Language* foi criada motivada pela dificuldade de predição computacional de pronúncia (TAYLOR; ISARD, 1997). Quando originalmente proposta, alguns sistemas TTS permitiam anotações extra-textuais para auxiliar a estimação de parâmetros, mas cada sistema tinha sua própria notação. Consequentemente, usuários tinham que aprender um sistema de anotação para cada programa diferente. A proposta da SSML é padronizar notações, permitindo que sistemas TTS recebam o texto com marcações adicionais feitas utilizando uma linguagem unificada. A linguagem foi adotada por soluções *open-source* como MaryTTS e eSpeakNG, além dos sistemas proprietários encontrados em Alexa, Google Assistant e Cortana. A especificação é mantida pela W3C (BURNETT; SHUANG, 2010) e cita os elementos *emphasis*, *break* e *prosody* como marcadores que podem auxiliar o processador de linguagem natural a gerar parâmetros prosódicos apropriados.

```
<speak>
    Siga <emphasis level="strong">aquele</emphasis> carro.
</speak>
```

Código 3.2: Exemplo de texto anotado com SSML

3.3.2 EmotionML

A EmotionML (SCHRÖDER; BURKHARDT, 2014) foi criada para estender textos e outros documentos, adicionando marcações que indicam uma emoção desejada ou identificada. Uma das aplicações da EmotionML é auxiliar algoritmos de TTS a determinarem prosódia.

Como menciona Taylor (2009), não há um acordo quanto ao sistema mais apropriado para representar emoções. A linguagem de marcação tem suporte a múltiplos sistemas descritivos, como categorias, dimensões, *appraisals* e *action tendencies*. As categorias podem receber valores discretos (como pode ser visto no código 3.3), determinando se uma emoção está presente, ou valores contínuos, determinando a intensidade de uma emoção específica (como pode ser visto no código 3.4), permitindo múltiplas categorias simultaneamente.

Na literatura, Charfuelan e Steiner (2013) descrevem um algoritmo para implementação de cálculo de parâmetros prosódicos a partir de anotações em EmotionML utilizando o *framework* MaryTTS.

```
<emotionml version="1.0" xmlns="http://www.w3.org/2009/10/emotionml">
  <emotion category-set="http://www.w3.org/TR/emotion-voc/xml#everyday-categories">
  <emotion>
    <category name="happy" />
    Que bom te ver!
  </emotion>
</emotionml>
```

Código 3.3: Exemplo de texto anotado com EmotionML com parâmetros discretos

```
<speak version="1.1" xmlns="http://www.w3.org/2001/10/synthesis"
        xmlns:emo="http://www.w3.org/2009/10/emotionml"
        xml:lang="en-US">
    <s>
        <emo:emotion
            category-set="http://www.w3.org/TR/emotion-voc/xml#everyday-categories">
           <emo:category name="worried" value="0.4"/>
        </emo:emotion>
        Precisa de ajuda?
    </s>
</speak>
```

Código 3.4: Exemplo de texto anotado com SSML e EmotionML (adaptado de (SCHRÖDER; BURKHARDT, 2014))

3.3.3 Anotação manual

Uma solução mais simples para geração de contornos f0 para prosódia efetiva é permitir que o usuário especifique na entrada do programa TTS marcações prosódicas utilizando um dos modelos de análise entoacional vistos anteriormente na seção 3.2. Apesar de requerer conhecimento de fonologia, é uma opção viável enquanto não são desenvolvidos algoritmos para determinação de prosódia a partir de marcação emocional. O sistema Festival (BLACK; TAYLOR, 1997) disponibiliza uma maneira de especificar entoação seguindo o modelo ToBI, como pode ser visto no código 3.5.

```
(Utterance Words
 (The
  (boy ((accent L*)))
  saw
  the
  (girl ((accent H*) (tone L-)))
  with
  the
  (telescope ((accent H*) (tone H-H%))))))
```

Código 3.5: Anotações no modelo ToBI para o sistema TTS Festival

4 Módulo de prosódia

Baseado na revisão da literatura, decidiu-se criar um módulo de prosódia que permite anotações prosódicas manuais, tornando possível variar a prosódia afetiva e aumentativa para um sistema TTS.

4.1 Implementação

4.1.1 espeak-ng

Foi utilizado o programa *open-source* eSpeakNG (DUNN, 2006) para realizar as primeiras etapas do *back end*: normalização de texto e conversão grafema-fone, ou seja, obter a partir do texto de entrada uma representação em fones. A saída gerada pelo programa é subsequentemente passada para o programa desenvolvido neste trabalho através da biblioteca `subprocess` da linguagem Python. O comando utilizado para obter os grafemas pode ser visto no código 4.1.

- A *flag* `-v` seleciona uma voz. Neste caso, `pt-br`
- `-x` e `-q` escrevem fones na saída em vez da fala sintetizada.
- `-sep=/` separa fones utilizando o caractere `/`
- O caractere ’ indica *primary stress*, ou seja, o fone tônico da palavra

```
$ espeak-ng -v pt-br ’Bom dia’ -x -q --sep=/
$ b/’o/N dZ/’i/&
```

Código 4.1: Utilização do programa espeak e saída correspondente

Apesar da existência de outras ferramentas para *front end* para o português brasileiro, optamos pelo eSpeakNG pela disponibilidade e facilidade de instalação.

4.1.2 MBROLA

O programa `sampa_mbrola.py` foi desenvolvido para converter a saída gerada pelo eSpeakNG em fones suportados pela voz utilizada. O equivalente ao fone `/&/` gerado pelo espeak é `/a/` na voz do MBROLA, por exemplo.

A ferramenta de conversão lê a tabela do arquivo `sampa_mbrola.tbl` e substitui cada fone em notação XSAMPA pelo equivalente. A tabela possui três colunas: o fone que o espeak produz, o fone equivalente para o MBROLA e um exemplo numa palavra do português. No código 4.2 é possível ver algumas linhas do arquivo. Utilizamos neste trabalho a voz `br3` desenvolvida por Denis R. Costa disponível no site oficial do projeto MBROLA.

```
b | b | _b_arco
k | k | _c_om
d | d | _d_ose
& | a | v_a_le
6 | @ | tam_a_nho
n | n | _n_unca
```

Código 4.2: Extrato de linhas da tabela de conversão

Além da conversão XSAMPA-MBROLA, são determinadas as durações padrão para cada fone a partir da tabela `durations.tbl`, adaptada do código utilizado pelo LianeTTS (SERPRO, 2011). Cada linha contém o fone e uma duração em milissegundos.

4.1.3 INTSINT

Como visto na seção 3.2.2, o INTSINT serve não apenas para análise, mas também para síntese de contornos f0. Escolhemos utilizá-lo na implementação pela simplicidade da notação e flexibilidade.

Em `intsint.py` é feita a determinação de valores f0 a partir do texto de entrada anotado e das regras descritas na tabela 4.1.3. O pseudocódigo 1 ilustra o funcionamento do processo de conversão.

Algoritmo 1 Pseudocódigo para geração de frequências utilizando modelo INTSINT

freq_t $\leftarrow$ key $\times \sqrt{2^{range}}$
freq_m $\leftarrow$ key
freq_b $\leftarrow$ key$/\sqrt{2^{range}}$
para todo rótulo P_i **faça**
  se rótulo = T **então**
    $f_i \leftarrow$ freq_t
  senão se rótulo = M **então**
    $f_i \leftarrow$ freq_m
  senão se rótulo = B **então**
    $f_i \leftarrow$ freq_b
  senão se rótulo = H **então**
    $f_i \leftarrow \sqrt{f_{i-1} \times \text{freq\_t}}$
  senão se rótulo = S **então**
    $f_i \leftarrow f_{i-1}$
  senão se rótulo = L **então**
    $f_i \leftarrow \sqrt{f_{i-1} \times \text{freq\_b}}$
  senão se rótulo = U **então**
    $f_i \leftarrow \sqrt{f_{i-1} \times \sqrt{f_{i-1} \times \text{freq\_t}}}$
  senão se rótulo = D **então**
    $f_i \leftarrow \sqrt{f_{i-1} \times \sqrt{f_{i-1} \times \text{freq\_b}}}$
  fim se
fim para

4.1.4 Editor gráfico

Para permitir refinamento de prosódia após cálculo inicial de parâmetros, foi desenvolvido um editor gráfico para *web* utilizando HTML, CSS e JavaScript. A duração e altura de cada fone podem ser especificadas manualmente arrastando barras de controle, gerando um novo arquivo de áudio com a fala sintetizada a partir dos novos parâmetros.

4.1.4.1 Arquitetura

O editor se comunica com o espeak-ng e MBROLA através de um servidor programado em Python utilizando o *framework* Flask para prover *endpoints* de uma API *RESTful*. Foi utilizada uma arquitetura cliente-servidor (FIELDING, 2000) comumente observada em

aplicações *web* atuais. A maneira como os módulos se comunicam pode ser vista na figura 3.

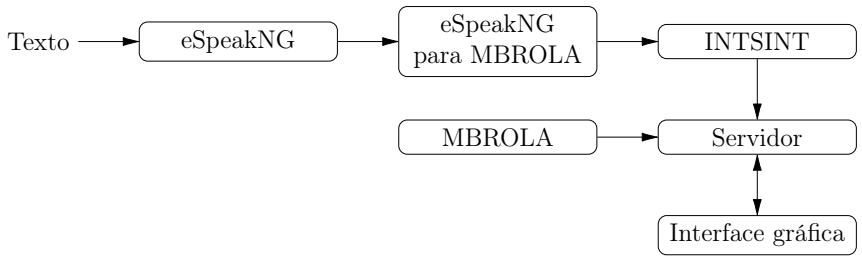

Figura 3: Arquitetura para o programa da interface gráfica

4.1.4.2 *Endpoints*

Foram criados dois *endpoints* para a API, possibilitando a comunicação entre interface gráfica e servidor.

[POST] /api/espeak Recebe um texto como entrada e gera uma resposta no formato JSON com lista de fones. Cada fone, por sua vez, possui campos `duration`, `phone_mbrola`, `phone_sampa` e `pitch_changes`. Um exemplo de resposta pode ser visto no código A.4.

[POST] /api/mbrola Recebe uma lista de fones no formato MBROLA descrito na seção 3.1.1, passa a lista para o programa para criar o arquivo de áudio final e gera uma resposta com o nome do arquivo de áudio contendo a fala sintetizada.

4.1.5 Utilização

O programa pode ser executado através de uma interface de linha de comandos, como ilustrado no código 4.3.

```
$ echo "Testando" | python3 sampa_mbrola.py > out.pho
$ mbrola ../br3/br3 out.pho out.wav; afplay out.wav
```

Código 4.3: Utilização por linha de comandos

Alternativamente, pode-se utilizar um navegador e ter acesso à interface gráfica, permitindo edição de parâmetros para fones individuais.

5 Resultados

Com o levantamento dos sistemas TTS desenvolvidos para o português brasileiro, é possível observar que há carência de suporte à síntese expressiva. Seguindo o modelo de funções prosódicas de Taylor (2009), os sistemas estudados concentram-se na geração de prosódia suprassegmental, mas não há suporte algum à geração de prosódia afetiva e aumentativa atualmente.

Linguagens de marcação como SSML e EmotionML já começaram a ser integradas a *frameworks open-source* e sistemas comerciais de TTS para múltiplas línguas, mas não foram encontrados trabalhos para o português brasileiro.

Estudando os modelos de anotação entoacional, percebe-se que ainda não há uma solução considerada mais apropriada para analisar o português brasileiro. Há, por outro lado, uma grande quantidade de trabalhos de análise de contornos melódicos que podem ser futuramente adaptados e convertidos em parâmetros para sistemas TTS expressivos, servindo como base para a geração manual ou automática de prosódia afetiva.

Identificamos lacunas que precisam ser preenchidas, como a criação de corpus anotados prosodicamente e algoritmos de compreensão textual para geração de marcação EmotionML.

Com o desenvolvimento dos programas para linha de comandos e para *web* neste trabalho, dá-se um passo inicial para a melhoria de prosódia afetiva e aumentativa em sistemas TTS para o português brasileiro.

6 Considerações finais

Neste trabalho foi apresentada a definição de um sistema TTS, bem como a descrição de seus componentes principais e possíveis implementações para cada módulo. Também foi realizada uma busca dos sistemas TTS *open-source* e comerciais existentes. Com o objetivo de melhorar a geração de prosódia, foi feita uma revisão do estado da arte, descrevendo os principais sistemas de anotação usados na fonologia entoacional, principalmente aplicados ao português brasileiro. Investigamos como a prosódia funciona nos sistemas encontrados e identificamos que, enquanto a geração de prosódia suprassegmental em trabalhos recentes já produz resultados satisfatórios, ainda há desafios quando à síntese de fala expressiva relacionada à prosódia afetiva e aumentativa.

6.1 Trabalhos futuros

Considerando o que foi estudado e concluído, próximos passos para melhorar a naturalidade da fala em sistemas TTS a partir do módulo de prosódia incluem:

- Expansão do sistema desenvolvido neste trabalho para adicionar suporte a mais modelos de notação entoacional, como ToBI e DaTo.
- Avaliação estatística da qualidade de falas geradas com a aplicação desenvolvida comparada a outros sistemas *open-source* e comerciais.
- Adicionar suporte a SSML e EmotionML ao sistema desenvolvido, gerando curvas f0 a partir de marcações.
- Uso técnicas de técnicas de *Natural Language Understanding* para gerar notação SSML e EmotionML automaticamente.
- Desenvolvimento de corpus anotados com prosódia para o português brasileiro.

Referências

Amazon.com, Inc. *Alexa*. 2014. Disponível em: <https://developer.amazon.com/alexa>. Acesso em 25 de setembro de 2017.

Apple Inc. *Siri*. 2011. Disponível em: <https://www.apple.com/ios/siri/>. Acesso em 25 de setembro de 2017.

BARBOSA, F. et al. Grapheme-phone transcription algorithm for a Brazilian Portuguese TTS. In: SPRINGER. *International Workshop on Computational Processing of the Portuguese Language*. [S.l.], 2003. p. 23–30.

BARBOSA, P. A. et al. Aiuruete: A high-quality concatenative text-to-speech system for brazilian portuguese with demisyllabic analysis-based units and a hierarchical model of rhythm production. In: *Sixth European Conference on Speech Communication and Technology*. [S.l.: s.n.], 1999.

BLACK, A. W. Perfect synthesis for all of the people all of the time. In: IEEE. *Speech Synthesis, 2002. Proceedings of 2002 IEEE Workshop on*. [S.l.], 2002. p. 167–170.

BLACK, A. W.; TAYLOR, P. A. *The Festival Speech Synthesis System: System Documentation*. 1.1. ed. Scotland, UK, 1997. Avaliable at http://www.cstr.ed.ac.uk/projects/festival.html.

BOTINIS, A.; GRANSTRÖM, B.; MÖBIUS, B. Developments and paradigms in intonation research. *Speech communication*, Elsevier, v. 33, n. 4, p. 263–296, 2001.

BURNETT, D.; SHUANG, Z. W. *Speech Synthesis Markup Language (SSML) Version 1.1*. [S.l.], 2010. Disponível em: <http://www.w3.org/TR/2010/REC-speech-synthesis11-20100907/>. Acesso em 5 de junho de 2018.

CELESTE, L.; REIS, C. Análise entonativa formal: Intsint aplicado ao português. *Journal of Speech*, v. 2, n. 2, p. 3–21, 2012.

CHARFUELAN, M.; STEINER, I. Expressive speech synthesis in MARY TTS using audiobook data and emotionML. In: *INTERSPEECH*. [S.l.: s.n.], 2013. p. 1564–1568.

COSTA, E. S. et al. Um sintetizador de voz baseado em hmms livre: Dando novas vozes para aplicaç oes livres no português do brasil. In: *Workshop de Software Livre*. [S.l.: s.n.], 2012.

COUTO, I. et al. An open source HMM-based text-to-speech system for Brazilian Portuguese. In: *7th international telecommunications symposium*. [S.l.: s.n.], 2010.

DUNN, R. H. *espeak-ng*. 2006. Disponível em: <https://github.com/espeak-ng/espeak-ng>. Acesso em 29 de outubro de 2017.

DUTOIT, T. *An introduction to text-to-speech synthesis*. [S.l.: s.n.], 1997. (Text, Speech and Language Technology 3).

DUTOIT, T. et al. The MBROLA project: Towards a set of high quality speech synthesizers free of use for non commercial purposes. In: IEEE. *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*. [S.l.], 1996. v. 3, p. 1393–1396.

FIELDING, R. T. REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000.

HIRSCHBERG, J. Speech synthesis, prosody. *Encyclopedia of language & linguistics*, v. 7, p. 49–55, 2006.

HIRST, D. J. *La représentation linguistique des systèmes prosodiques: une approche cognitive*. Tese (Doutorado) — Aix-Marseille 1, 1987.

HIRST, D. J. A praat plugin for momel and intsint with improved algorithms for modelling and coding intonation. In: *Proceedings of the XVIth International Conference of Phonetic Sciences*. [S.l.: s.n.], 2007. v. 12331236.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009. ISBN 0131873210.

LADD, D. R. *Intonational phonology*. [S.l.]: Cambridge University Press, 2008.

LUCENTE, L. Uma abordagem fonética na fonologia entoacional. *Fórum Linguístico*, v. 11, n. 1, p. 79–95, 2014.

Microsoft Corp. *Cortana*. 2014. Disponível em: <`https://www.microsoft.com/en-us/windows/cortana`>. Acesso em 25 de setembro de 2017.

MORAES, J. A. de. Brazilian portuguese. *Intonation systems: A survey of twenty languages*, Cambridge University Press, p. 179, 1998.

MORAES, J. A. de. The pitch accents in Brazilian Portuguese: analysis by synthesis. In: *Proc. Speech Prosody*. [S.l.: s.n.], 2008. p. 389–397.

RAJESWARI, K.; UMA, M. Prosody modeling techniques for text-to-speech synthesis systems–a survey. *International Journal of Computer Applications*, v. 39, n. 16, p. 8–11, 2012.

SCHRÖDER, M.; BURKHARDT, F. *Emotion Markup Language (EmotionML) 1.0*. [S.l.], maio 2014. Disponível em: <`http://www.w3.org/TR/2014/REC-emotionml-20140522/`>. Acesso em 8 de junho de 2018.

SERPRO. *LianeTTS*. 2011. Disponível em: <`http://intervox.nce.ufrj.br`>. Acesso em 7 de junho de 2018.

SILVERMAN, K. et al. Tobi: A standard for labeling english prosody. In: *Second international conference on spoken language processing*. [S.l.: s.n.], 1992.

TAYLOR, P. *Text-to-speech synthesis*. [S.l.]: Cambridge University Press, 2009.

TAYLOR, P.; ISARD, A. SSML: A speech synthesis markup language. *Speech communication*, Elsevier, v. 21, n. 1-2, p. 123–133, 1997.

VÉRONIS, J. et al. A stochastic model of intonation for text-to-speech synthesis1. *Speech Communication*, Elsevier, v. 26, n. 4, p. 233–244, 1998.

ZEN, H.; TOKUDA, K.; BLACK, A. W. Statistical parametric speech synthesis. *Speech Communication*, Elsevier, v. 51, n. 11, p. 1039–1064, 2009.

APÊNDICE A – Primeiro apêndice

```
import sqlite3
import sampa_mbrola
from flask import Flask, g, jsonify, render_template, abort, request
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/api/espeak', methods=['POST'])
def frontend():
    text = request.form['text']
    converter = sampa_mbrola.Converter()
    sentence = converter.convert_sentence(text)
    return jsonify(sentence.dictify())

@app.route('/api/mbrola', methods=['GET'])
def mbrola(page):
    pass

if __name__ == '__main__':
    app.run(debug=True)
```

Código A.1: Servidor

```
import sys
import re
import subprocess
import random
import json
from collections import OrderedDict
from typing import List

def flatten(lst: list):
    return [item for sublist in lst for item in sublist]
```

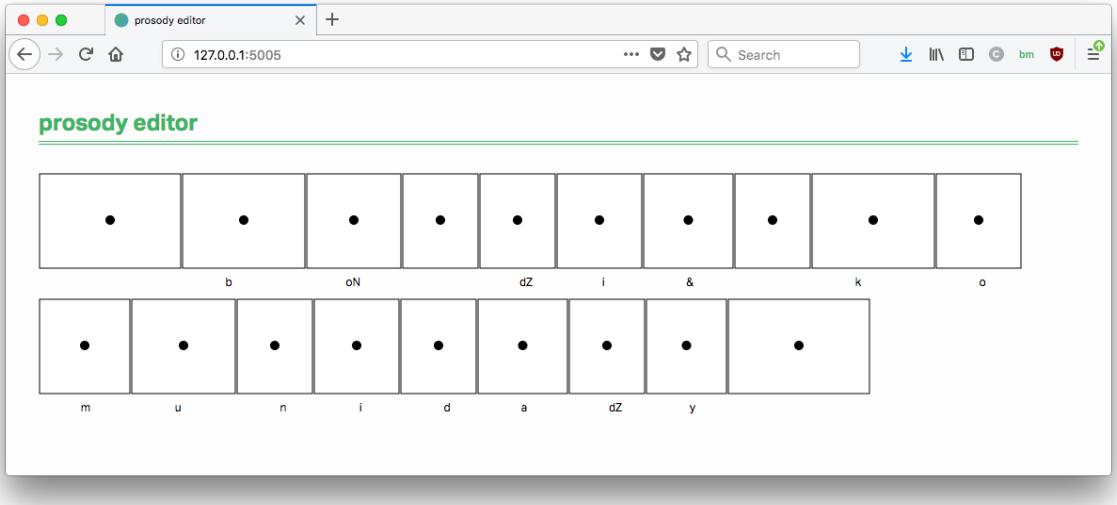

Figura 4: Editor gráfico

```
class Phone():
    def __init__(
            self,
            phone_sampa: str,
            phone_mbrola: str,
            duration: int,
            pitch_changes: list
    ):
        self.phone_sampa = phone_sampa
        self.phone_mbrola = phone_mbrola
        self.duration = duration
        self.pitch_changes = pitch_changes

    def as_line(self):
        return "{} {} {}".format(
            self.phone_mbrola,
            self.duration,
            " ".join([str(item) for item in flatten(self.pitch_changes)])
        )

class Sentence():
    def __init__(self, phones: List[Phone]=None):
        if phones is None:
            self.phones = []
        else:
            self.phones = phones
```

```
    def mbrola_phones(self):
        return [phone.phone_mbrola for phone in self.phones]

    def dictify(self):
        return [vars(phone) for phone in self.phones]

    def __repr__(self):
        return "\n".join([phone.as_line() for phone in self.phones])


class Converter():
    def __init__(self):
        self.load_sampa_mbrola()
        self.load_durations()

    def load_sampa_mbrola(self):
        equivs = {}
        with open("sampa_mbrola.tbl") as f:
            for line in f:
                k, v, _ = line.split()
                equivs[k] = v

        self.equivs = OrderedDict(
            sorted(equivs.items(), key=lambda t: -len(t[0])))

    def load_durations(self):
        durations = {}
        with open("durations.tbl") as f:
            for line in f:
                k, v = line.split()
                durations[k] = v

        self.durations = durations

    def convert_phoneme(self, sentence: str) -> tuple:
        """Returns first phone from the sentence"""

        if sentence[0] == " ":
            return ("_", 1)
        elif self.equivs:
            # s is a special case, needs to peek next
            if sentence[0] == "s":
```

```
            if sentence[1] in "aeiou&":
                return ("s", 1)

        for equiv in self.equivs.items():

            if re.match(re.escape(equiv[0]), sentence):
                # print("match:", equiv[0], phoneme, "=", equiv[1])
                return (equiv[1], len(equiv[0]))

    # print("didn't match", phoneme)
    return (phoneme[0], 1)

def get_duration(self, phoneme: str) -> int:
    if self.durations and phoneme in self.durations:
        return self.durations[phoneme]
    else:
        return 100

def convert_sentence(self, input_str: str) -> Sentence:
    sentence = Sentence()
    sentence.phones.append(Phone(" ", "_", 150, [[50, 150]]))

    ignored = ["@", "\n", ",", "'", "^", ";"]

    sampa = self.text_to_sampa(input_str)
    sampa = sampa.replace("'", "")
    print(";; ", sampa)

    while sampa:
        if sampa[0] not in ignored:
            converted = self.convert_phoneme(sampa)
            phone_sampa = sampa[:converted[1]]
            sampa = sampa[converted[1]:]

            duration = self.get_duration(converted[0])
            phone = Phone(
                phone_sampa = phone_sampa,
                phone_mbrola = converted[0],
                duration = int(duration),
                pitch_changes = [[50, 150]] # percentage, Hz
            )

            sentence.phones.append(phone)
```

```
            else:
                sampa = sampa[1:]

        sentence.phones.append(Phone(" ", "_", 150, [[50, 150]]))
        return sentence

    def text_to_sampa(self, sentence: str) -> str:
        espeak_str = "espeak-ng -v pt-br '{}' -x -q".format(sentence)

        p = subprocess.Popen(espeak_str, stdout=subprocess.PIPE, shell=True)
        (output, err) = p.communicate()
        p_status = p.wait()
        output = output.decode("utf-8")
        output = output.replace("\n", "").strip()
        return output


if __name__ == "__main__":
    converter = Converter()

    for line in sys.stdin:
        print(";;", line)
        print(converter.convert_sentence(line))
```

Código A.2: Conversor eSpeakNG-MBROLA

```
var app = new Vue({
  el: '#app',
  data: {
    phones: [],
    height: 300
  },
  mounted: function() {
    var self = this;
    const requestURL = "http://127.0.0.1:5000/api/espeak";
    let XHR = new XMLHttpRequest();
    let FD = new FormData();

    FD.append("text", "Bom dia, comunidade");

    XHR.open("POST", requestURL);
    XHR.send(FD);

    XHR.onreadystatechange = function() {
```

```
      if (XHR.readyState === XMLHttpRequest.DONE) {
        if (XHR.status === 200) {
          const results = JSON.parse(XHR.responseText);
          self.phones = results;
        }
      }
    };
  },
  computed: {
    totalDuration: function() {
      let durations = this.phones.map((phone) => phone.duration);
      console.log(durations.reduce((acc, val) => acc + val, 0));
      return durations.reduce((acc, val) => acc + val, 0);
    }
  }
});
```

Código A.3: Editor gráfico

```
[
  {
    "duration": 150,
    "phone_mbrola": "_",
    "phone_sampa": " ",
    "pitch_changes": [[50, 150]
    ]
  },
  {
    "duration": 80,
    "phone_mbrola": "n",
    "phone_sampa": "n",
    "pitch_changes": [[50, 150]]
  },
  {
    "duration": 110,
    "phone_mbrola": "u",
    "phone_sampa": "U",
    "pitch_changes": [[50, 150]]
  },
  ...
  {
    "duration": 150,
    "phone_mbrola": "_",
    "phone_sampa": " ",
```

```
    "pitch_changes": [[50, 150]]
  }
]
```

Código A.4: Exemplo de resposta para *endpoint* do eSpeakNG

```
{
  "mp3_file": "07acc5c4ba924294.mp3"
}
```

Código A.5: Exemplo de resposta para *endpoint* do MBROLA

```
import json

def get_duration(self, phoneme: str) -> int:
    if self.durations and phoneme in self.durations:
        return self.durations[phoneme]
    else:
        return 100

def convert_sentence(self, input_str: str) -> Sentence:
    sentence = Sentence()
    sentence.phones.append(Phone(" ", "_", 150, [[50, 150]]))

    ignored = ["@", "\n", ",", "'", "^", ";"]

    sampa = self.text_to_sampa(input_str)
    sampa = sampa.replace("'", "")
    print(";; ", sampa)

    while sampa:
        if sampa[0] not in ignored:
            converted = self.convert_phoneme(sampa)
            phone_sampa = sampa[:converted[1]]
            sampa = sampa[converted[1]:]

            duration = self.get_duration(converted[0])
            phone = Phone(
                phone_sampa = phone_sampa,
                phone_mbrola = converted[0],
                duration = int(duration),
                pitch_changes = [[50, 150]] # percentage, Hz
            )

            sentence.phones.append(phone)
```

```
        else:
            sampa = sampa[1:]

    sentence.phones.append(Phone(" ", "_", 150, [[50, 150]]))
    return sentence

def text_to_sampa(self, sentence: str) -> str:
    espeak_str = "espeak-ng -v pt-br '{}' -x -q".format(sentence)

    p = subprocess.Popen(espeak_str, stdout=subprocess.PIPE, shell=True)
    (output, err) = p.communicate()
    p_status = p.wait()
    output = output.decode("utf-8")
    output = output.replace("\n", "").strip()
    return output
```

Código A.6: Gerador de f0 a partir do modelo INTSINT