Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Leonardo Pereira Macedo

Desenvolvimento de um módulo de reconhecimento de voz para a *game engine* Godot

São Paulo
22 de outubro de 2017

Desenvolvimento de um módulo de reconhecimento de voz para a *game engine* Godot

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo
22 de outubro de 2017

Agradecimentos

Resumo

A área de jogos eletrônicos (*video games*) evoluiu muito desde o início da década da 70, quando começaram a ser comercializados. As principais causas estão relacionadas aos avanços em diferentes áreas da Computação.

Com o passar do tempo, surgiram as *game engines*: *frameworks* voltados especificamente para a criação de jogos, visando a facilitar o desenvolvimento e/ou algumas de suas etapas.

Focaremos em uma *game engine* em particular, *Godot* (Juan Linietsky, Ariel Manzur, 2017a). Por possuir código aberto, este *software* permite a extensão de suas funcionalidades através da criação de novos módulos.

Este projeto busca implementar um módulo de reconhecimento de voz para *Godot*, depois demonstrando a nova capacidade em um jogo simples desenvolvido na própria plataforma.

Palavras-chave: *software*, *game engine*, *Godot*, desenvolvimento de módulo, extensão de funcionalidade, reconhecimento de voz.

Abstract

Video games have evolved considerably since the beginning of the 70's, when they started to be commercialized. The main reasons are related to several advances in different fields of Computer Science.

Over time, *game engines* started appearing: *frameworks* designed specifically to assist on game creation, simplifying the process and/or some of its steps.

We will focus on a specific game engine, *Godot* (Juan Linietsky, Ariel Manzur, 2017a). Since it is an open source project, it is possible to extend its funcionalities by creating new modules.

This project's goal is to implement a speech recognition module for *Godot*, then showing the new feature in a simple game developed on the engine itself.

Keywords: software, game engine, *Godot*, module development, functionality extension, speech recognition.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Capítulo 1

Introdução

1.1 Motivação e objetivo

Hoje em dia, não há como negar que o mercado de *games* é um fenômeno mundial, gerando mais de US$ 91 bilhões em 2016 (SuperData Research, 2016). Comparado aos primeiros jogos, comercializados no início da década de 1970 (Wikipedia, 2017b), a evolução em diversas áreas da computação permitiu grandes avanços nos jogos criados. Inclui-se nisso a evolução dos computadores por conta da *Lei de Moore* (Wikipedia, 2017c), permitindo processamento mais rápido; *games* em 3D e gráficos cada vez mais sofisticados e realistas devido à Computação Gráfica; e adversários sofisticados e de raciocínio rápido com a Inteligência Artificial.

Junto aos próprios jogos, as tecnologias usadas para desenvolvê-los também tiveram progressos. Em especial, temos as *game engines*, que podem ser descritas como "*frameworks* voltados especificamente para a criação de jogos" (Enger, 2013). Elas oferecem diversas ferramentas para acelerar o desenvolvimento de um jogo, como maior facilidade na manipulação gráfica e bibliotecas prontas para tratar colisões entre objetos. Além disso, como eficiência é um fator essencial para manter um bom valor de FPS (*Frames per Second*), as *engines* costumam ter sua base construída em linguagens rápidas e compiladas, como C e C++.

Focaremos em uma *game engine* em particular, ***Godot*** (Juan Linietsky, Ariel Manzur, 2017a). O principal motivo de ter sido escolhida é por ser um *software* de código aberto, o que permite a qualquer pessoa baixar seu código fonte e fazer modificações. Em especial, a *engine* permite a criação de novos módulos para adicionar a ele novas funcionalidades.

Este trabalho visa a criar um novo módulo para *Godot*. Tal extensão adicionará funções simples de reconhecimento de voz, algo ainda inexistente no *software*. Feito isso, a nova funcionalidade será demonstrada em um jogo simples criado nessa *engine*.

1.2 Organização do trabalho

O capítulo 2 aborda resumidamente reconhecimento de voz através de um olhar teórico, seguido pelo capítulo **??**, onde estuda-se um pouco de uma forma de realizar reconhecimento de voz por meio de *Modelos Ocultos de Markov*.

No capítulo 3, representa-se os primeiros passos para a concretização do trabalho, pois envolve a busca da melhor biblioteca de reconhecimento de voz que possa ser usada no módulo. A biblioteca escolhida, *Pocketsphinx*, é estudada no capítulo 4.

A arquitetura do *Godot* é apresentada no capítulo **??** a fim de se entender a lógica por trás da construção do módulo de reconhecimento de voz no capítulo **??**. O capítulo **??** apresenta a criação de jogo simples, feito na própria *game engine*, para demonstrar o módulo em funcionamento e suas capacidades.

O capítulo **??** apresenta as conclusões do trabalho. Por fim, há uma parte subjetiva contendo a apreciação pessoal do TCC e uma descrição das matérias que mais ajudaram no desenvolvimento do projeto.

Capítulo 2

Reconhecimento de voz

Neste capítulo, abordaremos a parte teórica do reconhecimento de voz, sem nos preocuparmos com a forma de implementação ou sua aplicação no contexto deste trabalho. Em particular, analisaremos brevemente os principais parâmetros que influenciam seu uso.

2.1 Definição

Reconhecimento automático de voz (ou da fala), muitas vezes referido como *speech to text* (**STT**), é um campo multidisciplinar que envolve as áreas de Inteligência Artificial, Estatística e Linguística. Busca-se desenvolver metodologias e tecnologias para que computadores sejam capazes de captar, reconhecer e traduzir a linguagem falada para texto (Wikipedia, 2017d).

2.2 História

Apresentamos uma breve visão histórica de sistemas de reconhecimento de voz, baseado principalmente em (Melanie Pinola, 2011), desde seu início até os dias atuais.

2.2.1 Décadas de 50 e 60: Primeiros passos

O primeiro sistema de reconhecimento de voz conhecido foi o *Audrey*, construído em 1952 por três pesquisadores do *Bell Labs*. A máquina conseguia reconhecer apenas dígitos falados por um único usuário.

10 anos depois, a IBM apresentou o *Shoebox*, que reconhecia 16 palavras em inglês, entre elas os dígitos de 0 a 9. Quando captava palavras como *plus*, *minus* ou *total*, *Shoebox* instruía outra máquina de adições a realizar cálculos ou imprimir o resultado.

A entrada era feita por um microfone (figura 2.1), que convertia a voz do usuário em impulsos elétricos, classificados internamente por um circuito de medição (IBM).

Figura 2.1: *Máquina Shoebox sendo operada (Cassiopedia)*

Laboratórios nos EUA, URSS, Inglaterra e Japão começaram a desenvolver hardware para reconhecer uma maior variedade de sons. Conseguiu-se suporte para quatro vogais e nove consoantes; um avanço notável, considerando a tecnologia da época.

2.2.2 Décadas de 70 e 80: Grandes avanços

Na década de 70, o departamento de defesa dos EUA mostrou grande interesse em financiar a tecnologia de reconhecimento de voz. Tal impulso ajudou no desenvolvimento do sistema *Harpy* de reconhecimento de voz pela Universidade Carnegie Mellon.

Usava-se um grafo para representar o domínio das palavras reconhecíveis. Um algoritmo de busca heurística, *Beam Search*, era aplicado para procurar a melhor interpretação para a voz de entrada. Este algoritmo assemelha-se ao *Best-First Search* (BFS), que explora um grafo através da expansão do estado mais promissor ao sair do estado presente. No entanto, sua otimização consiste em ordenar os próximos possíveis estados, através de uma heurística, antes de realizar uma expansão, o que permite prever o quão longe o estado presente está em relação ao estado meta. Com isso, o *Beam Search* é caracterizado como um algoritmo guloso, que gasta menos memória quando comparado ao BFS (Wikipedia, 2017a).

Através de uma forma de busca mais eficiente, *Harpy* conseguia entender 1011 palavras, aproximadamente o vocabulário de uma criança típica de três anos.

Sistemas de reconhecimento de voz só tiveram um avanço realmente significativo na década de 80, devido a um método estatístico denominado **Modelo Oculto de Markov** (ou **HMM**, sigla para *Hidden Markov Model*). Ao invés de procurar por modelos

de palavras em padrões de som, considera-se a probabilidade de um som desconhecido possuir palavras, o que acelerou o processo e tornou possível usar um vocabulário maior nos computadores. Veremos HMM com um pouco mais de detalhe na seção **??**.

Outro modelo que ganhou bastante popularidade na mesma época foi o de redes neurais, que é efetivo para classificar palavras isoladas e fonemas individuais mas encontra problemas em tarefas envolvendo reconhecimento contínuo. Ao contrário do HMM, este método não consegue modelar bem dependências temporais. No entanto, em ambos os casos, existia a necessidade de falar pausadamente para o sistema poder melhor interpretar o usuário.

Os progressos em sistemas de reconhecimento de voz começaram a se refletir no meio comercial. Destacamos a boneca *Julie* (figura 2.2), comercializada em 1987 como *"Finalmente, a boneca que te entende"*, pois era capaz de ser treinada para responder à voz de uma criança.

Figura 2.2: *Caixa da boneca Julie; note, na parte inferior, a frase "Ela entende o que você diz" (rrisner, 2016)*

2.2.3 Década de 90 até hoje: Popularização

Na década de 90, a popularização de computadores para uso pessoal e o desenvolvimento de processadores mais rápidos permitiu que o reconhecimento de voz ficasse viável para uma quantidade maior de pessoas.

Em 1996, surgiu o primeiro portal de voz, *VAL*, criado pela empresa de telecomunicações norte-americana BellSouth. O sistema atendia chamadas telefônicas e respondia de acordo com a informação proferida pelo cliente.

Até o final dos anos 2000, sistemas de reconhecimento de voz pareciam ter ficado estagnados em uma acurácia de aproximadamente 80%, e muitas aplicações eram ca-

racterizadas pela complexidade ou dificuldade de uso se comparadas ao tradicional *mouse* e teclado.

A popularidade do conceito ressurgiu com força através do aplicativo de Busca por Voz, feito pela Google para iPhone. As duas razões para o sucesso dessa forma de busca eram a facilidade de entrada de dados, se comparado ao teclado da plataforma, e o uso de *data centers* em nuvem da Google, o que retirava a necessidade de um poderoso processamento nos iPhones em si. Com isso, mostrava-se que era possível contornar duas das principais limitações: a disponibilidade de dados e a dificuldade de processá-los eficientemente.

A evolução na tecnologia de reconhecimento de voz foi tamanha que, atualmente, é inegável seu impacto em nosso dia a dia. Um celular moderno consegue captar palavras ou pequenas frases de seu usuário dentre um enorme vocabulário para fazer buscas na Internet, tocar uma música ou fazer uma ligação. Alguns países chegam até a usar reconhecimento de voz para autenticar a identidade de alguém por telefone, com o objetivo de evitar fornecer dados pessoais pelo mesmo. Também há usos em transportes, na área médica e para fins educativos, muitas vezes acentuados pela maior facilidade em se falar um comando comparado ao uso de um teclado ou interface gráfica.

2.3 Componentes de um sistema genérico

A figura 2.3 apresenta os três componentes de um sistema genérico envolvendo STT (National Research Council, 1984):

- O **usuário** do sistema, que codifica um comando através de sua voz;
- O **dispositivo** de STT, que converte a mensagem falada para um formato interpretável;
- O **software de aplicação**, que recebe a saída do dispositivo e realiza uma ação apropriada.

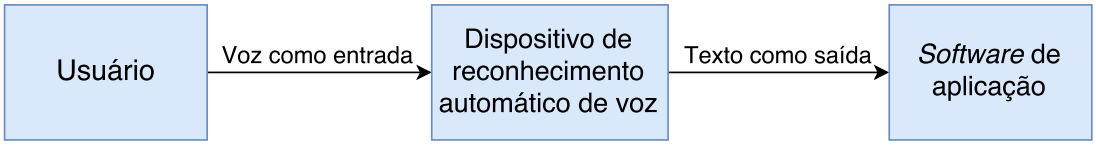

Figura 2.3: *Sistema genérico de reconhecimento automático de voz (National Research Council, 1984)*

2.4 Principais termos

De acordo com (National Research Council, 1984) e (Stephen Cook, 2002), apresentamos, a seguir, os termos mais recorrentes em sistemas de reconhecimento de voz. Também entramos em detalhes nos tipos de parâmetros que caracterizam as capacidades de um sistema de reconhecimento de voz, influenciando sua forma de funcionamento, eficiência e acurácia. A influência destes fatores varia de acordo com o tipo de aplicação que se deseja construir.

2.4.1 Fluência

A fluência está relacionada à forma de se comunicar com o sistema. Tipicamente, a fala do usuário pode ser feita através de *palavras isoladas*, com pausas entre elas; *palavras conectadas*, que são concatenadas sem pausas; ou *fala contínua*, onde o fluxo de palavras é semelhante a uma fala natural.

2.4.2 Dependência do usuário

A dependência ou não do usuário classifica os sistemas em dois grupos:

- Os sistemas **dependentes** (*speaker-dependent*), caracterizados pelo *treinamento* feito pelo usuário. Isto é, são computadores que analisam e se adaptam aos padrões particulares da fala captada, resultando em uma maior acurácia. Geralmente, o usuário deve ler algumas páginas de texto para a máquina antes de começar a usar o sistema. Esta variante é comumente usada em casos particulares, onde um número limitado de palavras deve ser reconhecido com bastante precisão (SpeechAngel, 2016).

- Os sistemas **independentes** (*speaker-independent*), que são desenvolvidos para reconhecer a voz de qualquer pessoa e não requerem treinamento. É a melhor opção para aplicações interativas que usam voz, já que não é viável fazer com que os usuários leiam páginas de texto antes do uso, ou para sistemas usados por diferentes pessoas. Sua desvantagem é a acurácia menor se comparado ao reconhecimento dependente; para contornar isso, costuma-se limitar o vocabulário reconhecido pelo sistema (SpeechAngel, 2016).

2.4.3 Vocabulário

O vocabulário representa as palavras reconhecidas pelo sistema. Seu tamanho pode ser pequeno (menor que 20 palavras) até muito grande (mais de 20 mil palavras), sendo diretamente proporcional à velocidade do reconhecimento. Além disso, a similaridade

entre a pronúncia de algumas palavras pode afetar a acurácia, uma vez que a distinção entre elas torna-se mais complicada.

2.4.4 *Utterance*

O termo *utterance* não possui uma tradução exata no contexto de reconhecimento de voz, embora possa ser interpretado como *"pronunciamento, elocução"*. Refere-se à vocalização (fala) de uma ou mais palavras, prounciadas de forma contínua e terminando com uma pausa clara, que possuem um significado único ao computador. Em outras palavras, *utterances* são o conteúdo entendido pelo sistema após receber a fala do usuário.

Ao voltarmos para o sistema genérico de reconhecimento de voz apresentado na seção 2.3, notaremos que a interpretação de *utterances* representa a saída produzida pelo dispositivo de STT.

2.4.5 Parâmetros ambientais

Parâmetros ambientais referem-se a fatores externos ao sistema que podem interferir no reconhecimento de voz. Destacam-se:

- A **relação sinal/ruído**, que avalia a intensidade média do sinal recebido em relação ao ruído de fundo, tipicamente medido em decibéis (dB). Quanto menor a taxa, maior a dificuldade no reconhecimento de voz.
- O **próprio usuário**, o que inclui o volume de sua voz, a velocidade com que fala e até mesmo sua condição psicológica: o nível de estresse de um piloto sob ataque em uma aeronave é diferente de alguém simplesmente querendo ouvir uma música, por exemplo.

Capítulo 3

Bibliotecas para Reconhecimento de Voz

A seguir, veremos o primeiro item necessário para atingirmos o objetivo final: uma biblioteca que fará o reconhecimento de voz dentro do módulo.

Uma implementação do zero fugiria do tema deste trabalho, pois seria necessário aprender sobre reconhecimento de padrões voltado a sons e outros tópicos relacionados a Inteligência Artificial. A outra opção existente, e a que seguiremos, é procurar por uma biblioteca existente e aprender a manejá-la.

Analisaremos quais as características necessárias e desejáveis na biblioteca ideal, e estudaremos a que melhor se adequa ao nosso objetivo dentre as opções existentes.

3.1 Considerações iniciais

Recordemos os principais componentes para reconhecimento de voz, apresentados na seção 2.3. No contexto do módulo de reconhecimento de voz para *Godot*, as seguintes associações surgem naturalmente:

- O **usuário** representa tipicamente o **jogador**, que interage parcialmente ou totalmente com o jogo por meio de comandos de voz.
- O **dispositivo de STT** corresponde ao **módulo de reconhecimento de voz**, objetivo principal deste trabalho. Esta componente é usada pelo jogo para converter a fala do jogador em texto.
- O **software de aplicação** é o **jogo** em si, feito em *Godot*, que recebe indiretamente os comandos do usuário e realiza ações apropriadas.

3.2 A biblioteca ideal

Realçamos novamente que o módulo de reconhecimento de voz será usado diretamente em jogos. Tal contexto automaticamente nos leva a pensar em diversas características que a biblioteca ideal deve possuir.

3.2.1 Características obrigatórias

Em ordem decrescente de importância, temos:

1. **Ter código aberto e licença permissiva:** Justifica-se pela integração da biblioteca em uma *game engine* de código aberto. A importância é ainda maior se levarmos em conta que jogos com fins comerciais podem ser produzidos em *Godot*.

2. **Ser eficiente (rápida):** Já foi mencionado que o módulo de reconhecimento de voz será usado em uma *game engine*. Um jogo é um *software* onde tipicamente a eficiência é de extrema importância, pois costuma envolver a renderização de cenas várias vezes por segundo. Devido a isso, surge a necessidade da biblioteca ser *rápida* para não afetar negativamente a experiência do jogador.

3. **Reconhecer inglês:** O inglês possui presença constante em cenários de computação. Portanto, é a única língua que a biblioteca deve obrigatoriamente oferecer suporte.

4. **Não ser pesada:** Não é desejável ter uma biblioteca que ocupe muito espaço em disco (o que poderia aumentar o tamanho do jogo que a utiliza) e memória (aspecto relacionado diretamente à eficiência).

3.2.2 Características desejáveis

Em ordem decrescente de importância, temos:

1. **Ser multiplataforma:** *Godot* possibilita exportar jogos para diferentes plataformas, dentre elas Windows, MacOS, Unix, Android e iOS (Juan Linietsky, Ariel Manzur, 2017b). Uma biblioteca que possa ser compatível com o maior número possível destes sistemas operacionais tornaria o módulo de reconhecimento de voz mais flexível para a produção de jogos em diferentes ambientes.

2. **Reconhecer diferentes línguas:** Apesar da obrigatoriedade do inglês, a possibilidade de usar diferentes línguas aumentaria a versatilidade do módulo. Tal característica é acentuada ao notarmos que muitos jogos, hoje em dia, oferecem a possibilidade de alterar a língua.

3. **Ser implementada em C/C++:** Conforme veremos na seção **??**, *Godot* possui toda a sua base escrita em C++, linguagem também usada para a criação de módulos. A implementação da biblioteca na mesma linguagem ajudaria a simplificar problemas de compatibilidade. Eventualmente, C também é uma opção viável por ser aceita pela linguagem sucessora.

3.3 Bibliotecas viáveis

Realizou-se uma pesquisa por bibliotecas de reconhecimento de voz que sigam o máximo de características possíveis propostas na seção 3.2. O artigo (NeoSpeech, 2016) sintetiza razoavelmente bem os resultados da busca. A seguir, destacamos as quatro bibliotecas mais notáveis encontradas:

- **Kaldi** (Kaldi, 2017): É a biblioteca mais recente da lista, com seu código publicado em 2011. Escrita em C++, é tida como uma biblioteca para pesquisadores de reconhecimento de voz.
- **CMUSphinx** (CMUSphinx, 2015): Desenvolvida pela *Carnegie Mellon University*, possui diversos pacotes para diferentes tarefas e aplicações. O pacote principal é escrito em Java. Existe também a variante *Pocketsphinx*, com características interessantes para este trabalho: é escrita em C, possuindo maior velocidade e portabilidade que a biblioteca original.
- **HTK** (HTK, 2016): Desenvolvida pela *Cambridge University Engineering Department*, HTK é uma sigla para *Hidden Markov Model Toolkit*. É escrita em C, com novas versões sendo lançadas consistentemente.
- **Simon** (Simon, 2017): Popular para Linux e escrita em C++, Simon utiliza *CMUSphinx*, *HTK* e *Julius* internamente. Não havia suporte para *MacOS* até abril de 2017.

Um artigo de 2014 comparou *Kaldi*, *CMUSphinx* e *HTK* em relação a precisão e tempo gasto (Gaida, 2014). *Kaldi* obteve resultados vastamente superiores; *CMUSphinx* obteve bons resultados em pouco tempo; *HTK* precisou de muito mais tempo e treino para conseguir resultados na ordem dos outros dois.

Capítulo 4

Pocketsphinx

Neste capítulo, analisaremos mais a fundo a biblioteca *Pocketsphinx*, incluindo seu funcionamento, instruções para usá-la de forma básica e passos para compilação a partir do código fonte.

Supõe-se que o usuário esteja usando um sistema operacional *Unix*, e que possua acesso a privilégios administrativos para a realização de alguns passos. Recomenda-se que o leitor possua um microfone à disposição no computador, podendo ser embutido ou externo, para melhor aproveitamento.

Todas as instruções e comandos apresentados foram originalmente realizados no sistema `Ubuntu 16.04 LTS, 64-bit` do autor.

4.1 Funcionamento

4.2 Compilação

Apresentamos instruções, em *Bash*, para baixar e compilar a biblioteca *Pocketsphinx*. Os passos foram baseados nas instruções em (CMUSphinx, 2016).

Antes de começar, instale as seguintes dependências em seu sistema:

```
gcc, automake, autoconf, libtool, bison, swig, python-dev, pulseaudio
```

Em um sistema *Ubuntu*, por exemplo, digitaria-se no terminal:

```
$ sudo apt-get install gcc automake autoconf libtool bison swig \
  python-dev pulseaudio
```

4.2.1 Pacote *Sphinxbase*

O pacote **Sphinxbase** oferece funcionalidades comuns a todos os projetos *CMUSphinx*. Siga as instruções abaixo para compilá-lo.

1. Clone o repositório do *Sphinxbase*.

```
$ git clone https://github.com/cmusphinx/sphinxbase
```

2. Dentro do diretório `sphinxbase/` criado pelo passo anterior, execute o *script* `autogen.sh` para gerar o arquivo `configure`:

```
$ ./autogen.sh
```

3. Execute o *script* `configure` criado no último passo:

```
# Padrão
$ ./configure

# Plataformas sem aritmética de ponto flutuante
$ ./configure --enable-fixed --without-lapack
```

   Note que qualquer dependência ausente no sistema (por exemplo, o pacote `swig`) será notificada ao usuário neste passo. Se a execução ocorrer sem problemas, um `Makefile` será gerado.

4. Compile o *Sphinxbase* através do `Makefile`:

```
$ make
```

4.2.2 Pacote *Pocketsphinx*

O pacote **Pocketsphinx** contém as funcionalidades de reconhecimento de voz em si que nos interessam para este trabalho. Siga as instruções abaixo para compilá-lo.

1. Clone o repositório do *Pocketsphinx*, o que criará o diretório `pocketsphinx/`.

```
$ git clone https://github.com/cmusphinx/pocketsphinx
```

2. Certifique-se que as pastas `sphinxbase/` e `pocketsphinx/` estejam no mesmo diretório, pois *Pocketsphinx* usa o caminho `../` para procurar pelo pacote *Sphinxbase*.

3. Dentro do diretório `pocketsphinx/`, execute o *script* `autogen.sh` para gerar o arquivo `configure`:

```
$ ./autogen.sh
```

4. Execute o *script* `configure` criado no último passo:

```
$ ./configure
```

Note que qualquer dependência ausente no sistema será notificada ao usuário neste passo. Se a execução ocorrer sem problemas, um `Makefile` será gerado.

5. Compile o *Pocketsphinx* através do `Makefile`:

```
$ make
```

4.2.3 Teste de verificação

Para verificar se a compilação feita nas subseções 4.2.1 e 4.2.2 ocorreu corretamente, recomenda-se fazer um teste de reconhecimento de voz contínuo com o binário `pocketsphinx_continuous`, criado na compilação do *Pocketsphinx*. Nesta verificação, o usuário fala uma palavra ou uma frase curta, em inglês, em seu microfone. Quando um silêncio é detectado, o programa analisa o *utterance* obtido e imprime na tela o texto que calculou ser a melhor interpretação.

No diretório onde encontram-se as pastas `sphinxbase/` e `pocketsphinx/`, execute o conteúdo da listagem 4.1.

```
# Diretório contendo arquivos para reconhecimento de voz (modelos, etc.)
MODELDIR=pocketsphinx/model

./pocketsphinx/src/programs/pocketsphinx_continuous \
-inmic yes \                                # Acionar uso do microfone
-hmm   $MODELDIR/en-us/en-us/mdef \         # Diretório do modelo acústico
-dict  $MODELDIR/en-us/cmudict-en-us.dict \ # Arquivo do dicionário
-lm    $MODELDIR/en-us/en-us.lm.bin         # Arquivo do modelo da língua
```

Listagem 4.1: *Comandos para teste de reconhecimento de voz contínuo usando Pocketsphinx*

O programa imediatamente irá imprimir uma lista de seus parâmetros e seus respectivos valores. Depois, avisará ao usuário que está pronto para receber a entrada de voz por meio de uma linha terminada em `Ready....`

A listagem 4.2 representa uma saída resumida ao se falar "`one two three`" no microfone. Os caracteres `[..]` representam uma ou mais linhas omitidas.

```
INFO: continuous.c(275): Ready....
INFO: continuous.c(261): Listening...
[..]
INFO: ngram_search_fwdtree.c(1550):     3081 words recognized (15/fr)
INFO: ngram_search_fwdtree.c(1552):   703838 senones evaluated (3400/fr)
INFO: ngram_search_fwdtree.c(1556):  2241048 channels searched (10826/fr)
[..]
INFO: ngram_search_fwdflat.c(302): Utterance vocabulary contains 154 words
INFO: ngram_search_fwdflat.c(948):     2575 words recognized (12/fr)
```

```
INFO: ngram_search_fwdflat.c(950):    148022 senones evaluated (715/fr)
INFO: ngram_search_fwdflat.c(952):    209298 channels searched (1011/fr)
[..]
INFO: ngram_search.c(1381): Lattice has 317 nodes, 942 links
INFO: ps_lattice.c(1380): Bestpath score: -4833
[..]
one two three
```

Listagem 4.2: *Saída do* `pocketsphinx_continuous` *ao se falar* "`one two three`"

Referências Bibliográficas

Cassiopedia() Cassiopedia. *Computing History Timeline*. http://www.cassiopeia.it/resources-2/computing-history-timeline. Acessado: 2017-09-29. ix, 4

CMUSphinx(2015) CMUSphinx. *About the CMUSphinx*. http://cmusphinx.sourceforge.net/wiki/about, Fevereiro 2015. Acessado: 2017-04-03. 11

CMUSphinx(2016) CMUSphinx. Building application with pocketsphinx. http://cmusphinx.sourceforge.net/wiki/tutorialpocketsphinx, 2016. Acessado: 2017-04-17. 13

Enger(2013) Michael Enger. *Game Engines: How do they work?* https://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/, Junho 2013. Acessado: 2017-04-03. 1

Gaida(2014) Christian Gaida. *Comparing Open-Source Speech Recognition Toolkits*. http://suendermann.com/su/pdf/oasis2014.pdf, 2014. Acessado: 2017-04-03. 11

HTK(2016) HTK. *What is HTK?* htk.eng.cam.ac.uk, 2016. Acessado: 2017-04-03. 11

IBM() IBM. *IBM Shoebox*. https://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html. Acessado: 2017-09-29. 4

Juan Linietsky, Ariel Manzur(2017a) Juan Linietsky, Ariel Manzur. *Godot Engine*. https://godotengine.org, 2017a. Acessado: 2017-04-03. iii, v, 1

Juan Linietsky, Ariel Manzur(2017b) Juan Linietsky, Ariel Manzur. *Godot Features*. https://godotengine.org/features#multiplatform-deploy, 2017b. Acessado: 2017-09-20. 10

Kaldi(2017) Kaldi. *About the Kaldi project*. http://kaldi-asr.org/doc/about.html, Março 2017. Acessado: 2017-04-03. 11

Melanie Pinola(2011) Melanie Pinola. *Speech Recognition Through the Decades: How We Ended Up With Siri*. http://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html, Novembro 2011. Acessado: 2017-09-30. 3

National Research Council(1984) National Research Council. Automatic Speech Recognition in Severe Environments. The National Academies Press. ix, 6, 7

NeoSpeech(2016) NeoSpeech. *Top 5 Open Source Speech Recognition Toolkits*. http://blog.neospeech.com/top-5-open-source-speech-recognition-toolkits, 2016. Acessado: 2017-04-03. 11

rrisner(2016) rrisner. http://www.ebay.com/itm/Vintage-1987-Worlds-of-Wonder-Muneca-Interactiva-Julie-mas-inteligente-hablando-/272259248680?_ul=BO, Junho 2016. Acessado: 2017-09-30. ix, 5

Simon(2017) Simon. *About Simon*. https://simon.kde.org/, 2017. Acessado: 2017-04-03. 11

SpeechAngel(2016) SpeechAngel. *The difference between speaker-dependent and speaker-independent recognition software*. https://speechangel.com/2016/05/04/difference-speaker-dependent-speaker-independent-recognition-software, Maio 2016. Acessado: 2017-09-29. 7

Stephen Cook(2002) Stephen Cook. Speech recognition howto. http://www.tldp.org/HOWTO/Speech-Recognition-HOWTO/introduction.html, Abril 2002. Acessado: 2017-09-30. 7

SuperData Research(2016) SuperData Research. *Worldwide game industry hits $91 billion in revenues in 2016, with mobile the clear leader*. https://venturebeat.com/2016/12/21/worldwide-game-industry-hits-91-billion-in-revenues-in-2016-with-mobile-the-clear-leader, 2016. Acessado: 2017-04-02. 1

Wikipedia(2017a) Wikipedia. *Beam Search*. https://en.wikipedia.org/wiki/Beam_search, Julho 2017a. Acessado 2017-09-29. 4

Wikipedia(2017b) Wikipedia. *The commercialization of video games*. https://en.wikipedia.org/wiki/History_of_video_games#The_commercialization_of_video_games, 2017b. Acessado: 2017-04-03. 1

Wikipedia(2017c) Wikipedia. *Moore's Law*. https://en.wikipedia.org/wiki/Moore%27s_law, 2017c. Acessado: 2017-04-03. 1

Wikipedia(2017d) Wikipedia. *Speech Recognition*. https://en.wikipedia.org/wiki/Speech_recognition, Setembro 2017d. Acessado: 2017-09-29. 3