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

Leonardo Pereira Macedo

Desenvolvimento de um módulo para a *game engine* Godot

São Paulo
29 de julho de 2017

Desenvolvimento de um módulo para a *game engine* Godot

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo
29 de julho de 2017

Agradecimentos

Resumo

A área de *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*. 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.

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*. 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.

Sumário

Lista de Figuras

Lista de Tabelas

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, 2017a), 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, 2017b), 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, 2017). 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 3 contém pesquisas e buscas por uma biblioteca de código aberto que faça reconhecimento de voz eficientemente. O capítulo **??** consiste em integrar a biblioteca encontrada ao *Godot*, expandindo suas funcionalidades. No capítulo **??**, mostram-se os passos realizados para criar um jogo que demonstre a capacidade do novo módulo. 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

Conceitos básicos

Neste capítulo, abordaremos a parte teórica necessária para melhor entendimento e implementação do módulo no restante do trabalho. Em particular, veremos conceitos básicos de reconhecimento automático de voz e um pouco de sua implementação através do *modelo oculto de Markov*.

2.1 Definição de reconhecimento de voz

Reconhecimento automático de voz (ou da fala), ou *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.

A figura 2.1 apresenta os três componentes de um programa genérico STT: o usuário, o dispositivo que realiza o reconhecimento de voz e um *software* de aplicação. O usuário codifica um comando através de sua voz; o dispositivo converte a mensagem falada para um formato interpretável; e o *software* de aplicação aceita esta saída e realiza uma ação apropriada.

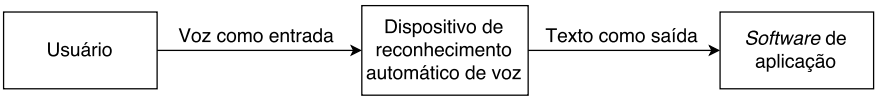

Figura 2.1: *Sistema genérico de reconhecimento automático de voz*

2.1.1 História e aplicações

O primeiro sistema de reconhecimento de voz conhecido foi o *Audrey*, construído em 1952 por três pesquisadores do *Bell Labs* para reconhecer dígitos falados por um único usuário. 10 anos depois, a IBM apresentou a máquina *Shoebox*, que reconhecia 16 palavras em inglês.

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* (HMM, sigla para *Hidden Markov Model*). Ao invés de procurar por modelos de palavras em padrões de som, HMM considerava a probabilidade de um som desconhecido possuir palavras, o que acelerou o processo e tornou possível usar um vocabulário maior nos computadores. Outro modelo que ganhou bastante popularidade na 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.

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. Muitas empresas utilizam máquinas para receber ligações de seus clientes; de acordo com o que interpretam, a chamada é redirecionada para um funcionário mais adequado. 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.

2.2 Parâmetros de reconhecimento de voz

Há diversos tipos de parâmetros que caracterizam as capacidades de um sistema de reconhecimento de voz. Eles se subdividem nos três tipos a seguir.

2.2.1 Específicos ao aplicativo

Relacionados a como o aplicativo em si realiza o reconhecimento de voz. Inclui dois parâmetros:

- A **forma de falar**, que pode ser 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.
- **Existência de treinamento**, que subdivide aplicativos 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.
  - 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. Sua desvantagem é a acurácia menor se comparado ao reconhecimento dependente; para contornar isso, costuma-se limitar o vocabulário reconhecido pelo sistema.

2.2.2 Específicos à tarefa

Dependendo do objetivo a ser alcançado com o reconhecimento de voz, alguns parâmetros podem ser melhor ajustados para obter maior velocidade ou acurácia. São eles:

- O **vocabulário**, referente a quantas palavras são reconhecidas pelo sistema. O 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.

- A **sintaxe**, isto é, a gramática artificial que o sistema aceita para uma determinada tarefa. O exemplo mais simples seria uma máquina de estados finita, onde as palavras permitidas após um estado ou nó são definidas explicitamente.

- O **fator de ramificação**, que é uma forma de medir a complexidade da sintaxe. É definido como o número médio de palavras permitidas em cada nó da gramática, e possui grande impacto sobre o desempenho do sistema.

2.2.3 Ambientais

Dentre os vários parâmetros externos ao sistema e que podem interferir no reconhecimento de voz, destacam-se:

- A taxa **sinal-para-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).

2.3 Estrutura da fala

Popularmente, imagina-se que a fala é composta por palavras, que por sua vez são formadas por fonemas (menores unidades sonoras do sistema fonológico de uma língua). Na prática, existe um processo dinâmico, sem partes claramente definidas.

2.4 Modelo oculto de Markov

Capítulo 3

Biblioteca para Reconhecimento de Voz

3.1 Considerações iniciais

Se o objetivo do projeto é criar um módulo de reconhecimento de voz, primeiramente é necessário chegar a uma biblioteca que implemente tal funcionalidade. Uma implementação do zero fugiria do tema deste trabalho, pois seria necessário aprender sobre reconhecimento de padrões voltado a sons, assunto relacionado a Inteligência Artificial.

A outra opção existente, e a que seguiremos, é procurar por uma biblioteca existente e aprender a manejá-la. Idealmente, gostaríamos que a biblioteca possuísse as seguintes características:

1. *Ter código aberto e licença permissiva:* Provavelmente o único aspecto essencial, uma vez que integraremos a biblioteca em uma *game engine*. A importância é ainda maior se levarmos em conta que jogos com fins comerciais podem ser produzidos em *Godot*.

2. *Ser implementada em C/C++:* Toda a base do *Godot* é construída em C++. A criação de módulos utiliza essa mesma linguagem, como será visto no capítulo 3.

3. *Reconhecer inglês:* Escolheremos inglês como uma linguagem obrigatória a se ter na biblioteca pelo caráter universal dessa linguagem.

4. *Ser multiplataforma:* Ao menos, oferecer suporte para sistemas Windows, MacOS e Linux.

5. *Implementar eficientemente reconhecimento de voz:* A biblioteca será usada em jogos, onde eficiência (velocidade) é uma questão central.

6. *Não ser pesada:* Embora tenha menos importância que os aspectos acima, não é desejável ter uma biblioteca que ocupe muito espaço.

Após uma pesquisa, que pode ser resumida pelo artigo em (NeoSpeech, 2016), cinco bibliotecas de reconhecimento de voz em geral se destacam por seu uso:

- *Kaldi* (Kaldi, 2017): É a biblioteca mais recente da lista, com seu código publicado em 2011. Escrita em C++.

- *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 3 de abril de 2017.

- *Julius* (Julius, 2014): Desenvolvida pela *Interactive Speech Technology Consortium* e escrita em C. Infelizmente, o suporte para inglês é limitado e não pode ser usado para propósitos comerciais, o que nos força a descartar esta biblioteca como uma possível opção.

Das cinco bibliotecas descritas, as mais viáveis são as três primeiras (*Kaldi*, *CMUSphinx* e *HTK*). *Simon* está em seus primeiros passos para MacOS, por isso a relutância em seu uso.

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.

No restante deste capítulo, analisaremos com mais cuidado as bibliotecas *Kaldi*, *PocketSphinx* e *HTK*, com o intuito de acompanhar mais de perto suas funcionalidades, vantagens e desvantagens.

O sistema operacional utilizado foi `ubuntu 16.04 LTS, 64 bits`.

3.2 *Pocketsphinx*

3.2.1 Instalação

Os passos abaixo foram baseados nas instruções em (CMUSphinx, 2016). Supõe-se que o usuário esteja usando um sistema *Unix*, com acesso a privilégios administrativos (*root*).

Antes de começar, é necessário instalar as seguintes dependências: `gcc, automake, autoconf, libtool, bison, swig, python-dev, pulseaudio`. Se estiver usando um sistema como *Ubuntu*, por exemplo, digite no terminal:

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

Primeiramente, baixaremos e instalaremos o pacote *Sphinxbase*, que oferece funcionalidades comuns a todos os projetos *CMUSphinx*:

1. Clone o repositório do *Sphinxbase*, o que resultará no diretório `sphinxbase`:
   ```
   $ git clone https://github.com/cmusphinx/sphinxbase
   ```

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

3. Execute o *script* `configure` que foi criado no último passo:
   ```
   $ ./configure
   ```

Se a plataforma utilizada não possui aritmética de ponto flutuante, deve-se rodar ao invés disso:

```
$ ./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. Construa a biblioteca através do `Makefile`:
   ```
   $ make
   ```

5. Instale a biblioteca *Sphinxbase* no sistema:
   ```
   $ sudo make install
   ```

   Observação: Para desinstalar a biblioteca *Sphinxbase* do sistema, basta digitar:
   ```
   $ sudo make uninstall
   ```

Os passos necessários para baixar e instalar a biblioteca *Pocketsphinx* são semelhantes:

1. Clone o repositório do *Pocketsphinx*, o que resultará no diretório `pocketsphinx`:
   ```
   $ git clone https://github.com/cmusphinx/pocketsphinx
   ```

2. Certifique-se que os diretórios `sphinxbase` e `pocketsphinx` estejam no mesmo diretório.

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

4. Execute o *script* `configure` que foi 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. Construa a biblioteca através do `Makefile`:
   ```
   $ make
   ```

6. Instale a biblioteca *Pocketsphinx* no sistema:
   ```
   $ sudo make install
   ```

   Observação: Para desinstalar a biblioteca *Pocketsphinx* do sistema, basta digitar:
   ```
   $ sudo make uninstall
   ```

Referências Bibliográficas

CMUSphinx(2015) CMUSphinx. *About the CMUSphinx*. http://cmusphinx.sourceforge.net/wiki/about, Fevereiro 2015. Acessado: 2017-04-03. 7

CMUSphinx(2016) CMUSphinx. Building application with pocketsphinx. http://cmusphinx.sourceforge.net/wiki/tutorialpocketsphinx, 2016. Acessado: 2017-04-17. 8

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. 8

HTK(2016) HTK. *What is HTK?* htk.eng.cam.ac.uk, 2016. Acessado: 2017-04-03. 8

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

Julius(2014) Julius. *Open-Source Large Vocabulary CSR Engine Julius*. http://julius.osdn.jp/en_index.php, 2014. Acessado: 2017-04-03. 8

Kaldi(2017) Kaldi. *About the Kaldi project*. http://kaldi-asr.org/doc/about.html, Março 2017. Acessado: 2017-04-03. 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. 7

Simon(2017) Simon. *About Simon*. https://simon.kde.org/, 2017. Acessado: 2017-04-03. 8

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

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