

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

Leonardo Pereira Macedo

Orientador: Prof. Dr. Marco Dimas Gubitoso

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

14 de novembro de 2017

- Evolução e sofisticação de jogos eletrônicos (*games*)
- Surgimento das ***game engines***: *frameworks* voltados para facilitar o desenvolvimento total ou parcial de jogos
  - Exemplos: *Unreal Engine*, *Unity*, *Godot*

- Evolução e sofisticação de jogos eletrônicos (*games*)
- Surgimento das ***game engines***: *frameworks* voltados para facilitar o desenvolvimento total ou parcial de jogos
  - Exemplos: *Unreal Engine*, *Unity*, *Godot*
- **Reconhecimento de voz** vem ficando cada vez mais integrado em nosso dia a dia
  - Autenticação de usuário, realização de buscas na Internet, etc.

- Evolução e sofisticação de jogos eletrônicos (*games*)
- Surgimento das ***game engines***: *frameworks* voltados para facilitar o desenvolvimento total ou parcial de jogos
  - Exemplos: *Unreal Engine*, *Unity*, *Godot*
- **Reconhecimento de voz** vem ficando cada vez mais integrado em nosso dia a dia
  - Autenticação de usuário, realização de buscas na Internet, etc.

Por que não fazer um trabalho que junte ambos os temas?

# Introdução

## Objetivo do trabalho

### Objetivo

Desenvolver um módulo (“*plugin*”) de reconhecimento de voz para a *game engine Godot*, demonstrando depois seu uso com um jogo simples

# Introdução

## Objetivo do trabalho

### Objetivo

Desenvolver um módulo (“*plugin*”) de reconhecimento de voz para a *game engine Godot*, demonstrando depois seu uso com um jogo simples

Pergunta: Por que escolher *Godot*?

# Introdução

## Objetivo do trabalho

### Objetivo

Desenvolver um módulo (“*plugin*”) de reconhecimento de voz para a *game engine Godot*, demonstrando depois seu uso com um jogo simples

Pergunta: Por que escolher *Godot*?

Resposta: Porque *Godot* é uma *game engine* de **código aberto**!

# Reconhecimento de Voz

## Definição e componentes

### Definição

**Reconhecimento automático de voz** é um campo que desenvolve técnicas para computadores captarem, reconhecerem e traduzirem a linguagem falada para texto; por isso também o nome *speech to text* (STT)



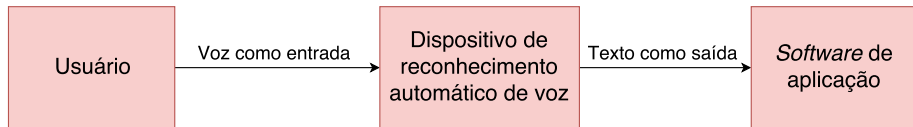
# Reconhecimento de Voz

## Definição e componentes

### Definição

**Reconhecimento automático de voz** é um campo que desenvolve técnicas para computadores captarem, reconhecerem e traduzirem a linguagem falada para texto; por isso também o nome *speech to text* (STT)

- Um sistema genérico STT possui três componentes:



# Reconhecimento de Voz

## Principais termos

- **Fluência:** Forma de comunicação com o sistema
  - Palavras isoladas
  - Palavras conectadas
  - Fala contínua

# Reconhecimento de Voz

## Principais termos

- **Fluência:** Forma de comunicação com o sistema
  - Palavras isoladas
  - Palavras conectadas
  - Fala contínua
- **Dependência do usuário:** Há treinamento?
  - Sistemas dependentes
  - Sistemas independentes

# Reconhecimento de Voz

## Principais termos

- **Fluência:** Forma de comunicação com o sistema
  - Palavras isoladas
  - Palavras conectadas
  - Fala contínua
- **Dependência do usuário:** Há treinamento?
  - Sistemas dependentes
  - Sistemas independentes
- **Vocabulário:** Palavras reconhecidas pelo sistema

# Reconhecimento de Voz

## Principais termos

- **Fluência:** Forma de comunicação com o sistema
  - Palavras isoladas
  - Palavras conectadas
  - Fala contínua
- **Dependência do usuário:** Há treinamento?
  - Sistemas dependentes
  - Sistemas independentes
- **Vocabulário:** Palavras reconhecidas pelo sistema
- ***Utterance*:** Vocalização de palavras

# Bibliotecas de reconhecimento de voz

Qual utilizar?

- Busca por uma biblioteca que implemente STT; o que priorizar?

# Bibliotecas de reconhecimento de voz

Qual utilizar?

- Busca por uma biblioteca que implemente STT; o que priorizar?
  - Código aberto e licença permissiva
  - Rápida, leve
  - Escrita em *C/C++*
  - Flexível/configurável

# Bibliotecas de reconhecimento de voz

## Qual utilizar?

- Busca por uma biblioteca que implemente STT; o que priorizar?
  - Código aberto e licença permissiva
  - Rápida, leve
  - Escrita em *C/C++*
  - Flexível/configurável
- Três opções viáveis:
  - *Kaldi*
  - *Pocketsphinx*
  - *HTK*



# Bibliotecas de reconhecimento de voz

## Qual utilizar?

- Busca por uma biblioteca que implemente STT; o que priorizar?
  - Código aberto e licença permissiva
  - Rápida, leve
  - Escrita em *C/C++*
  - Flexível/configurável
- Três opções viáveis:
  - *Kaldi*
  - *Pocketsphinx*
  - *HTK*

Biblioteca	WER (%)
<i>Kaldi</i>	6,5
<i>Pocketsphinx</i> v0.8	21,4
<i>HTK</i> v3.4.1	19,8

Word error rate (WER) de bibliotecas STT  
para entradas em inglês

# Bibliotecas de reconhecimento de voz

## Qual utilizar?

- Busca por uma biblioteca que implemente STT; o que priorizar?
  - Código aberto e licença permissiva
  - Rápida, leve
  - Escrita em *C/C++*
  - Flexível/configurável
- Três opções viáveis:
  - *Kaldi*
  - ***Pocketsphinx*** ✓
  - *HTK*

Biblioteca	WER (%)
<i>Kaldi</i>	6,5
<i>Pocketsphinx</i> v0.8	21,4
<i>HTK</i> v3.4.1	19,8

Word error rate (WER) de bibliotecas STT  
para entradas em inglês

- Desenvolvida pela *Carnegie Mellon University* (projeto *CMUSphinx*)
- Define que palavras são formadas por **fonemas**

- Desenvolvida pela *Carnegie Mellon University* (projeto *CMUSphinx*)
- Define que palavras são formadas por **fonemas**
- Voz  $\rightarrow$  *Utterances*  $\rightarrow$  Vetores de características

- Desenvolvida pela *Carnegie Mellon University* (projeto *CMUSphinx*)
- Define que palavras são formadas por **fonemas**
- Voz  $\rightarrow$  *Utterances*  $\rightarrow$  Vetores de características
- Utiliza o **Modelo Oculto de Markov** na interpretação
  - Trata a fala gravada como uma sequência de estados, que transitam entre si com certa **probabilidade**

- Desenvolvida pela *Carnegie Mellon University* (projeto *CMUSphinx*)
- Define que palavras são formadas por **fonemas**
- Voz  $\rightarrow$  *Utterances*  $\rightarrow$  Vetores de características
- Utiliza o **Modelo Oculto de Markov** na interpretação
  - Trata a fala gravada como uma sequência de estados, que transitam entre si com certa **probabilidade**

**Estados mais prováveis  $\rightarrow$  Melhor interpretação da voz**

- **Modelo acústico:** Arquivos que configuram detectores de fonemas

- **Modelo acústico:** Arquivos que configuram detectores de fonemas
- **Dicionário fonético:** Mapeamento {palavras → fonemas}

yellow Y EH L OW



- **Modelo acústico:** Arquivos que configuram detectores de fonemas
- **Dicionário fonético:** Mapeamento {palavras  $\rightarrow$  fonemas}

yellow Y EH L OW

- **Palavras-chave:** Palavras a serem detectadas, de acordo com limiar

yellow /1e-6/

- 1: **function recognize\_speech(recorder, decoder)**
- 2:     ▷ *recorder*: Tipo usado para gravar voz
- 3:     ▷ *decoder*: Tipo usado para decodificar áudio

```
1: function recognize_speech(recorder, decoder)  
2:     ▷ recorder: Tipo usado para gravar voz  
3:     ▷ decoder: Tipo usado para decodificar áudio  
4:     int buffer[SIZE], bytes  
5:     bool utt_started
```

```
1: function recognize_speech(recorder, decoder)
2:     ▷ recorder: Tipo usado para gravar voz
3:     ▷ decoder: Tipo usado para decodificar áudio
4:     int buffer[SIZE], bytes
5:     bool utt_started
6:     start_recording(recorder)
7:     start_utterance(decoder)
```

```
1: function recognize_speech(recorder, decoder)
2:     ▷ recorder: Tipo usado para gravar voz
3:     ▷ decoder: Tipo usado para decodificar áudio
4:     int buffer[SIZE], bytes
5:     bool utt_started
6:     start_recording(recorder)
7:     start_utterance(decoder)
8:     while true do
9:         bytes ← read_voice(recorder, buffer, SIZE)
10:        process_raw(decoder, buffer, bytes)
```

```
1: function recognize_speech(recorder, decoder)
2:     ▷ recorder: Tipo usado para gravar voz
3:     ▷ decoder: Tipo usado para decodificar áudio
4:     int buffer[SIZE], bytes
5:     bool utt_started
6:     start_recording(recorder)
7:     start_utterance(decoder)
8:     while true do
9:         bytes ← read_voice(recorder, buffer, SIZE)
10:        process_raw(decoder, buffer, bytes)
11:        if in_speech(decoder) and not utt_started then ▷ Usuário começou a falar
12:            utt_started ← true
```

```
1: function recognize_speech(recorder, decoder)
2:     ▷ recorder: Tipo usado para gravar voz
3:     ▷ decoder: Tipo usado para decodificar áudio
4:     int buffer[SIZE], bytes
5:     bool utt_started
6:     start_recording(recorder)
7:     start_utterance(decoder)
8:     while true do
9:         bytes ← read_voice(recorder, buffer, SIZE)
10:        process_raw(decoder, buffer, bytes)
11:        if in_speech(decoder) and not utt_started then ▷ Usuário começou a falar
12:            utt_started ← true
13:        if not in_speech(decoder) and utt_started then ▷ Usuário parou de falar
14:            end_utterance(decoder)
15:            get_hypothesis(decoder)
```

```
1: function recognize_speech(recorder, decoder)
2:     ▷ recorder: Tipo usado para gravar voz
3:     ▷ decoder: Tipo usado para decodificar áudio
4:     int buffer[SIZE], bytes
5:     bool utt_started
6:     start_recording(recorder)
7:     start_utterance(decoder)
8:     while true do
9:         bytes ← read_voice(recorder, buffer, SIZE)
10:        process_raw(decoder, buffer, bytes)
11:        if in_speech(decoder) and not utt_started then ▷ Usuário começou a falar
12:            utt_started ← true
13:        if not in_speech(decoder) and utt_started then ▷ Usuário parou de falar
14:            end_utterance(decoder)
15:            get_hypothesis(decoder)
16:            start_utt(decoder)
17:            utt_started ← false
```



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

Leonardo Pereira Macedo

Orientador: Prof. Dr. Marco Dimas Gubitoso

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

14 de novembro de 2017