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

Introdução

Contextualização

- 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*

Introdução

Contextualização

- 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?

Objetivo

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

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*?

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**!

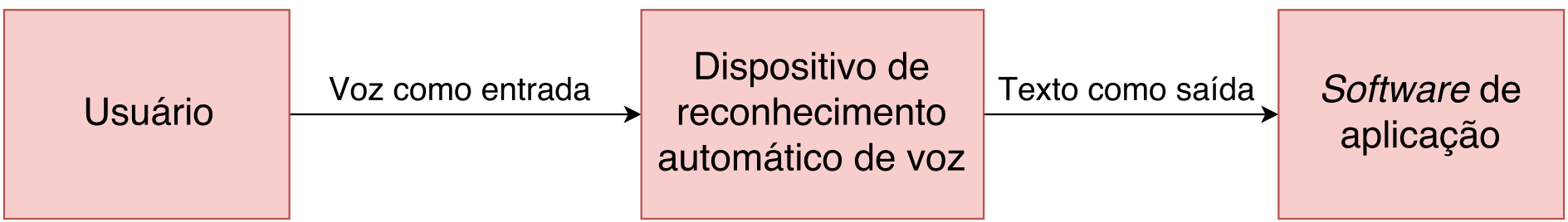
Definição

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)

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:



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

- **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 (%)**
Kaldi	6,5
Pocketsphinx v0.8	21,4
HTK 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 (%)**
Kaldi	6,5
Pocketsphinx v0.8	21,4
HTK 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 $\rightarrow$ 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/
```

Pocketsphinx

Implementação do reconhecimento de voz

1: **function recognize_speech**(*recorder*, *decoder*)
2: ▷ *recorder*: Tipo usado para gravar voz
3: ▷ *decoder*: Tipo usado para decodificar áudio

Pocketsphinx

Implementação do reconhecimento de voz

```
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
```

Pocketsphinx

Implementação do reconhecimento de voz

```
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)
```

Pocketsphinx

Implementação do reconhecimento de voz

```
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)
```

Pocketsphinx

Implementação do reconhecimento de voz

```
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
```

Pocketsphinx

Implementação do reconhecimento de voz

```
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)
```

Pocketsphinx

Implementação do reconhecimento de voz

```
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
```

- Criação de Juan Linietsky e Ariel Manzur em 2007
- Código aberto ao público em 2014

- Criação de Juan Linietsky e Ariel Manzur em 2007
- Código aberto ao público em 2014
- Código fonte escrito em ***C++***

- Criação de Juan Linietsky e Ariel Manzur em 2007
- Código aberto ao público em 2014
- Código fonte escrito em ***C++***
- Programação de jogos simplificada por meio de ***GDScript***

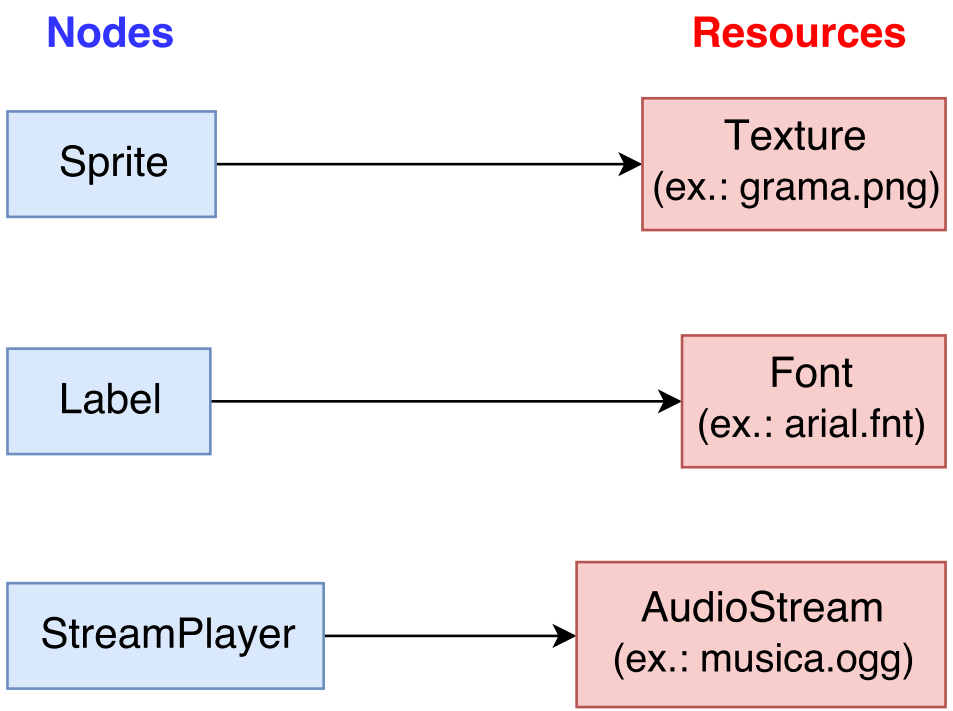
- `Object`: Classe base para todos os tipos não embutidos em *Godot*

- `Object`: Classe base para todos os tipos não embutidos em *Godot*
- `Reference`: Gerenciamento automático de memória

- `Object`: Classe base para todos os tipos não embutidos em *Godot*
- `Reference`: Gerenciamento automático de memória
- `Resource`: Armazenamento de dados

- `Object`: Classe base para todos os tipos não embutidos em *Godot*
- `Reference`: Gerenciamento automático de memória
- `Resource`: Armazenamento de dados
- `Node`: Define um comportamento

- `Object`: Classe base para todos os tipos não embutidos em *Godot*
- `Reference`: Gerenciamento automático de memória
- `Resource`: Armazenamento de dados
- `Node`: Define um comportamento



- Em relação a reconhecimento de voz:

- Em relação a reconhecimento de voz:
  - **Fluência**: Palavras conectadas

- Em relação a reconhecimento de voz:
  - **Fluência**: Palavras conectadas
  - **Dependência do usuário**: Sistema independente

- Em relação a reconhecimento de voz:
  - **Fluência**: Palavras conectadas
  - **Dependência do usuário**: Sistema independente
  - **Vocabulário**: Tipicamente pequeno

Módulo *Speech to Text*

Requisitos

- Em relação a reconhecimento de voz:
  - **Fluência**: Palavras conectadas
  - **Dependência do usuário**: Sistema independente
  - **Vocabulário**: Tipicamente pequeno
- Execução do módulo em **paralelo** com o restante do jogo $\rightarrow$ *thread*

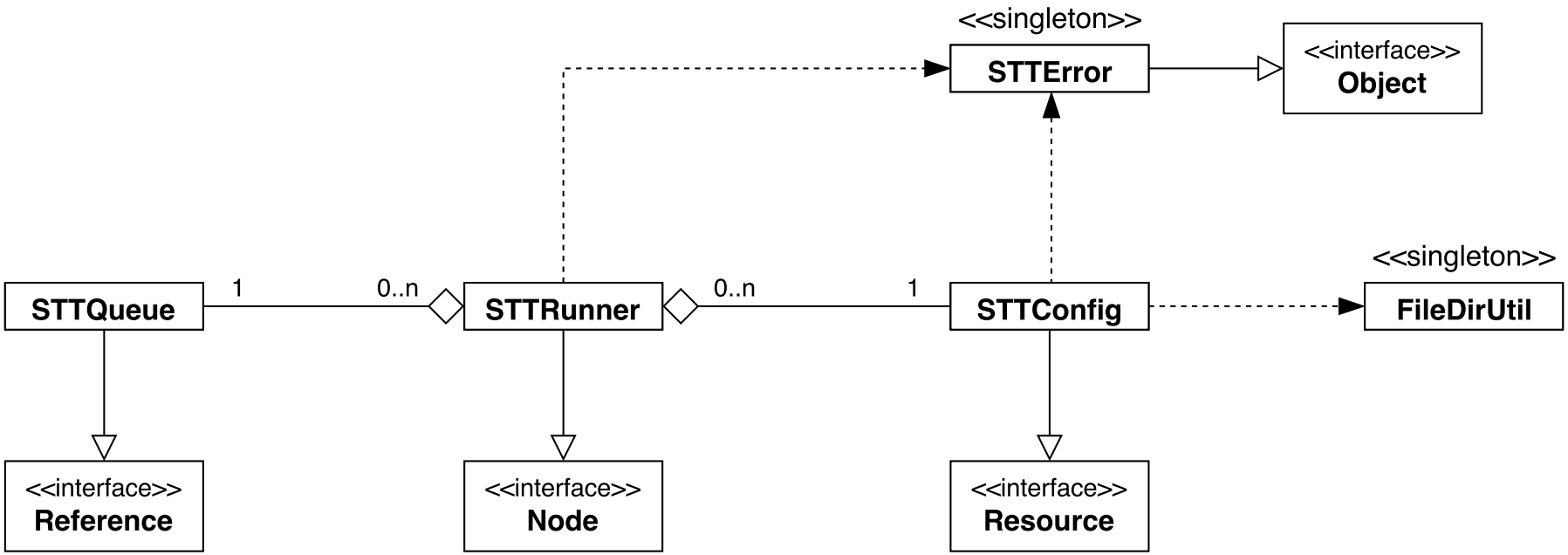
Módulo *Speech to Text*

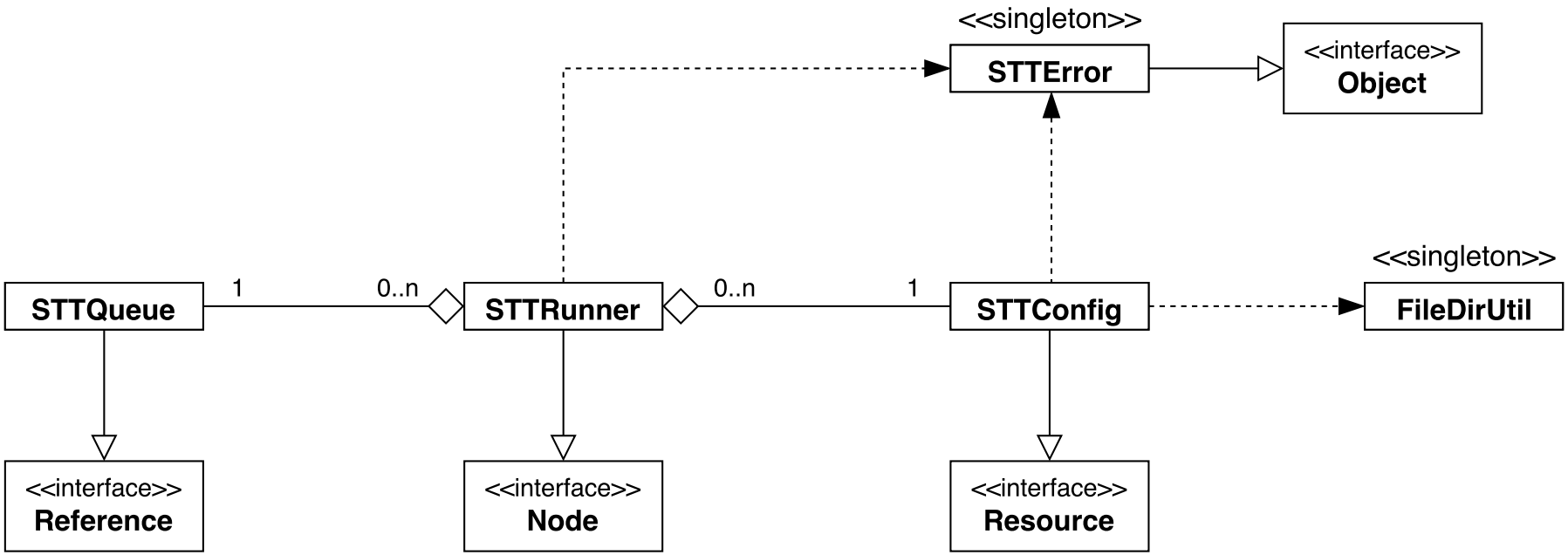
Requisitos

- Em relação a reconhecimento de voz:
  - **Fluência**: Palavras conectadas
  - **Dependência do usuário**: Sistema independente
  - **Vocabulário**: Tipicamente pequeno
- Execução do módulo em **paralelo** com o restante do jogo $\rightarrow$ *thread*
- Uso de um ***buffer*** para armazenar palavras conforme são reconhecidas

Módulo *Speech to Text*

Implementação

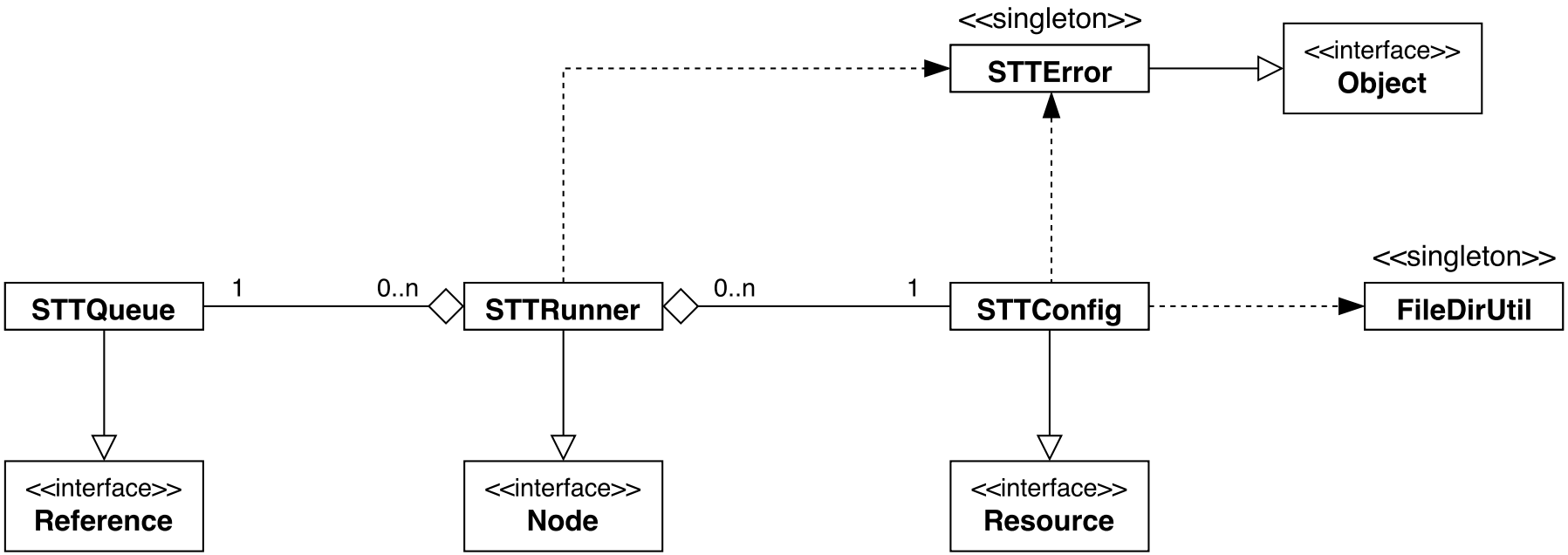




- `STTConfig`: Controle dos arquivos de configuração

Módulo *Speech to Text*

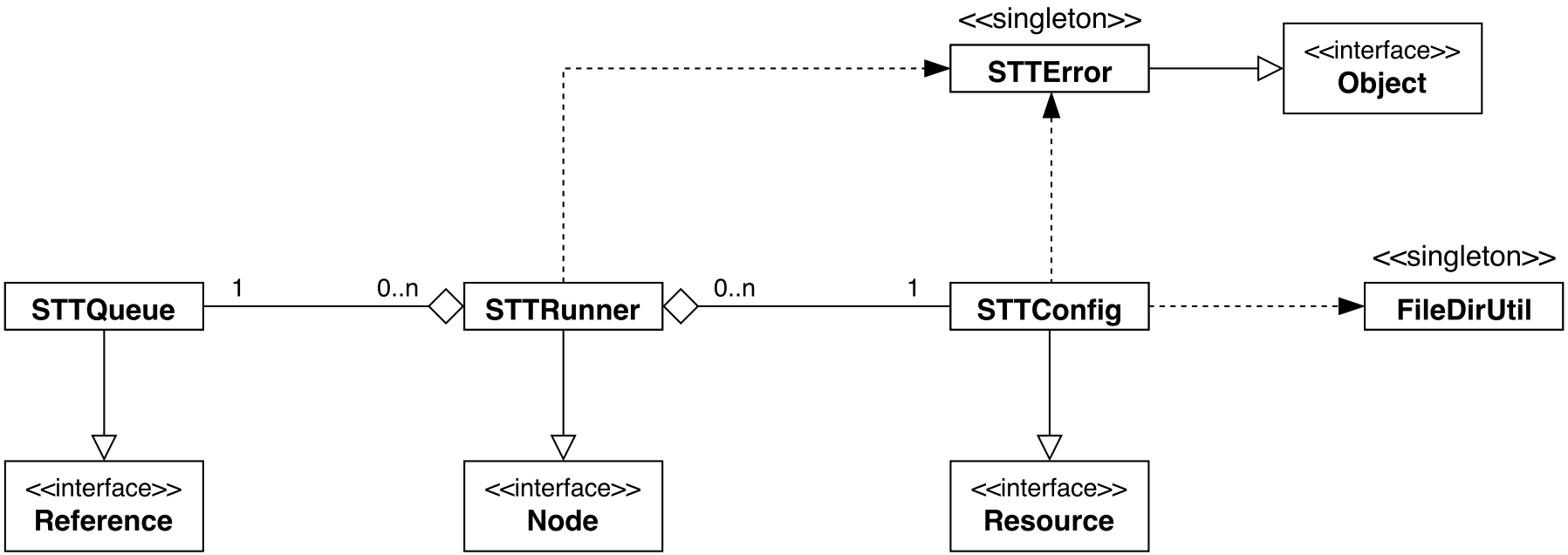
Implementação



- `STTConfig`: Controle dos arquivos de configuração
- `STTRunner`: *Thread* para realizar reconhecimento de voz

Módulo *Speech to Text*

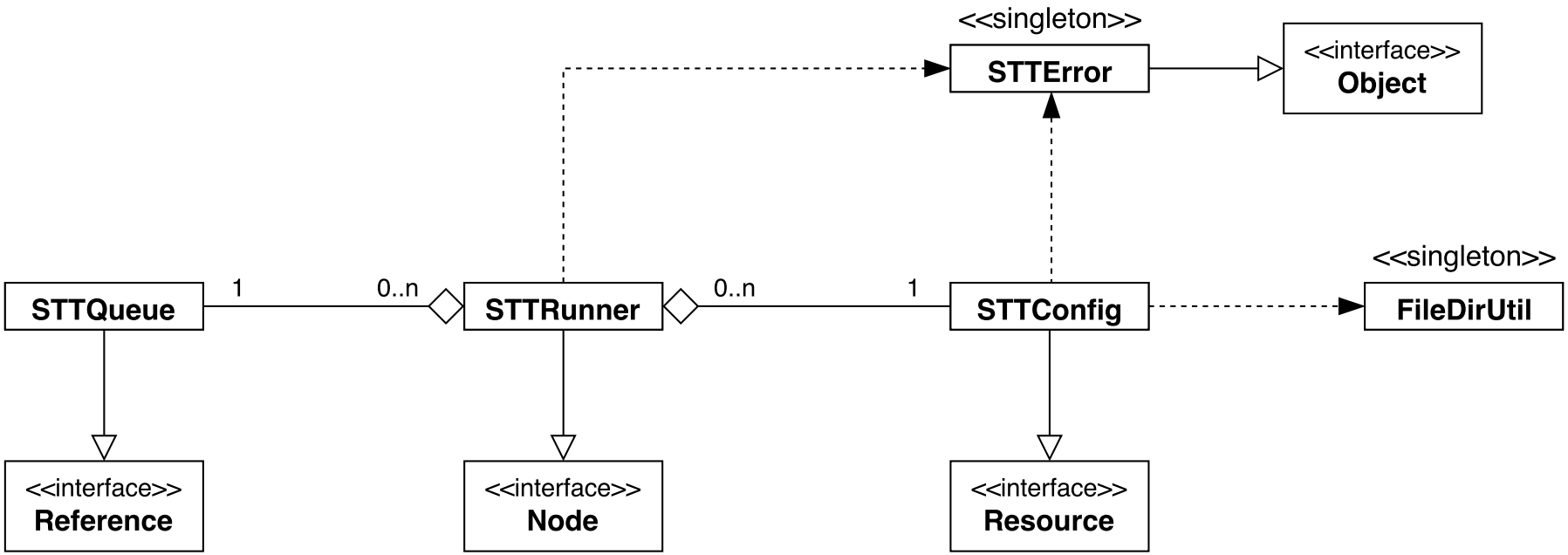
Implementação



- `STTConfig`: Controle dos arquivos de configuração
- `STTRunner`: *Thread* para realizar reconhecimento de voz
- `STTQueue`: Fila para guardar palavras reconhecidas pelo `STTRunner`

Módulo *Speech to Text*

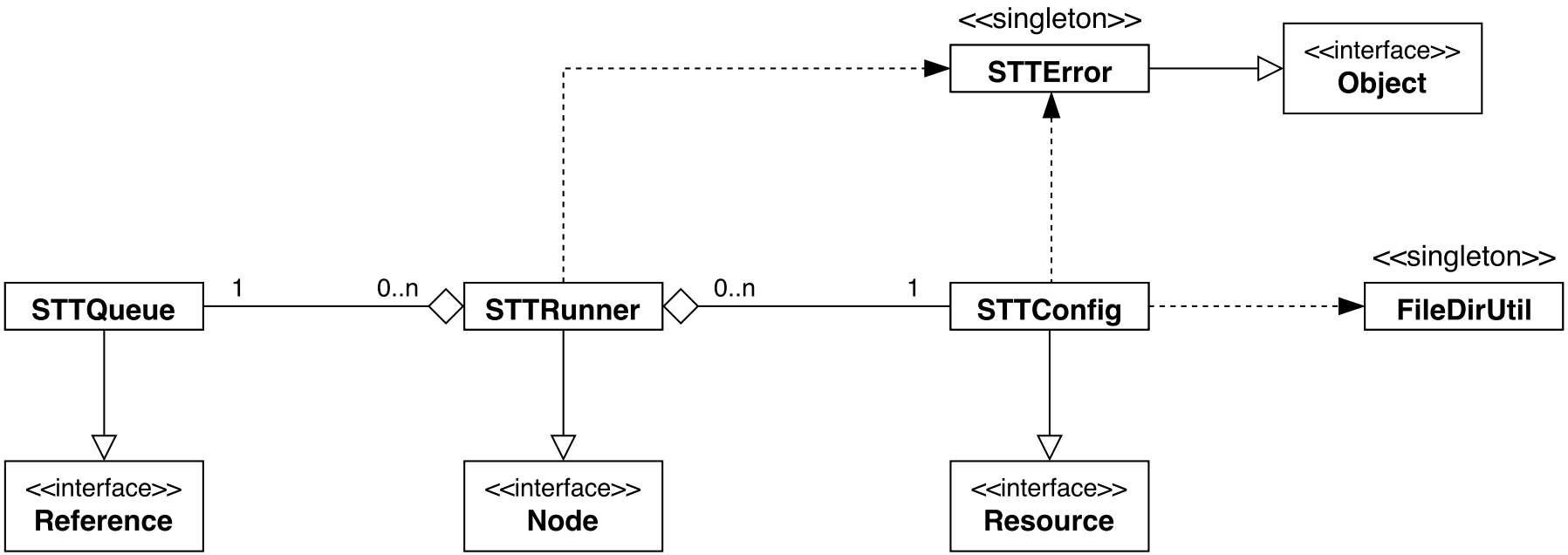
Implementação



- `STTConfig`: Controle dos arquivos de configuração
- `STTRunner`: *Thread* para realizar reconhecimento de voz
- `STTQueue`: Fila para guardar palavras reconhecidas pelo `STTRunner`
- `STTError`: Definição de constantes numéricas para possíveis erros

Módulo *Speech to Text*

Implementação



- `STTConfig`: Controle dos arquivos de configuração
- `STTRunner`: *Thread* para realizar reconhecimento de voz
- `STTQueue`: Fila para guardar palavras reconhecidas pelo `STTRunner`
- `STTError`: Definição de constantes numéricas para possíveis erros
- `FileDirUtil`: Classe auxiliar para manipular arquivos e diretórios

Color Clutter

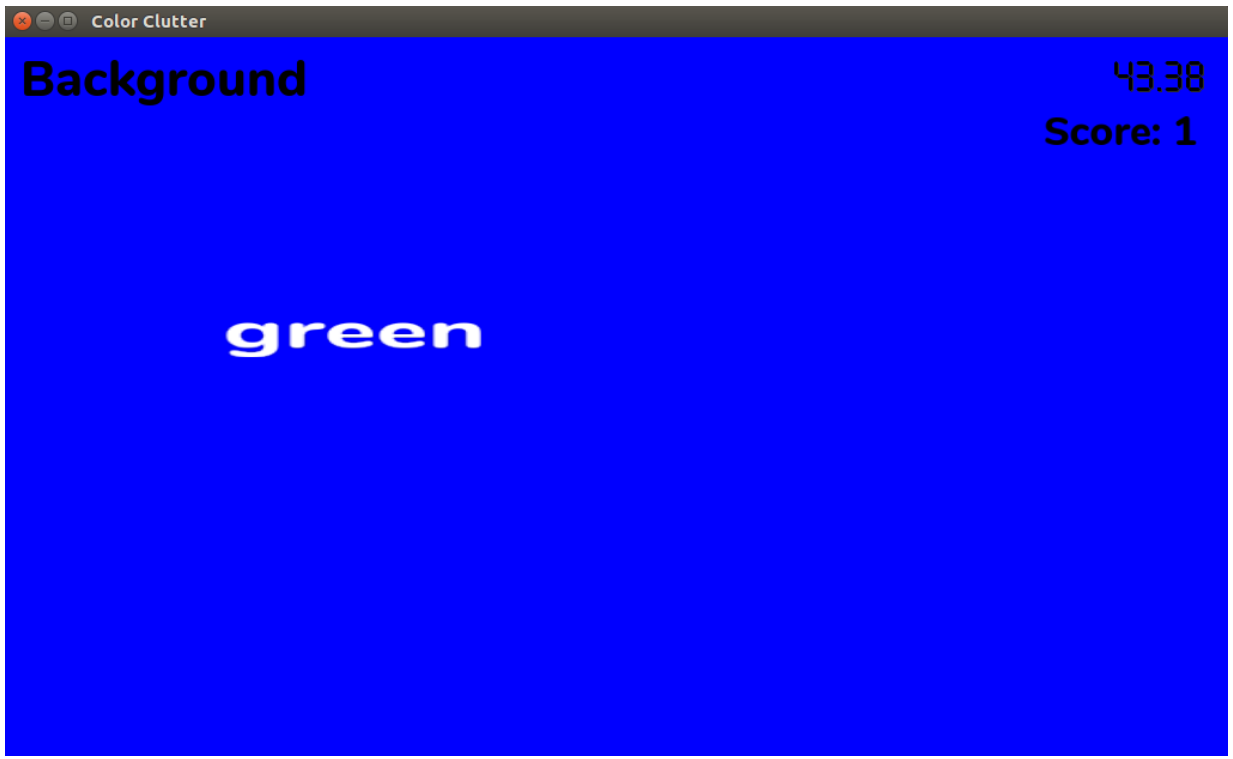
(Jogo demonstrativo)

- Jogo criado em *Godot* para demonstrar o módulo *Speech to Text*
- Todo o controle é feito por voz

Color Clutter

(Jogo demonstrativo)

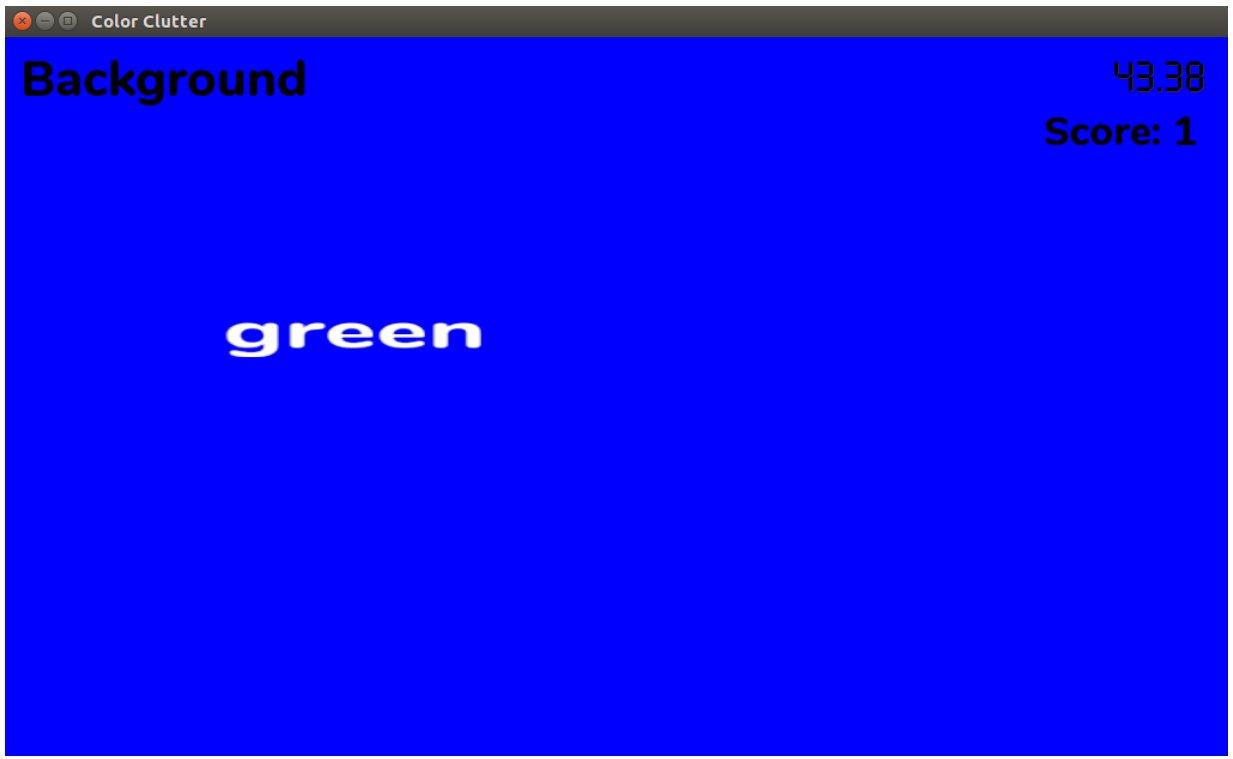
- Jogo criado em *Godot* para demonstrar o módulo *Speech to Text*
- Todo o controle é feito por voz



Color Clutter

(Jogo demonstrativo)

- Jogo criado em *Godot* para demonstrar o módulo *Speech to Text*
- Todo o controle é feito por voz

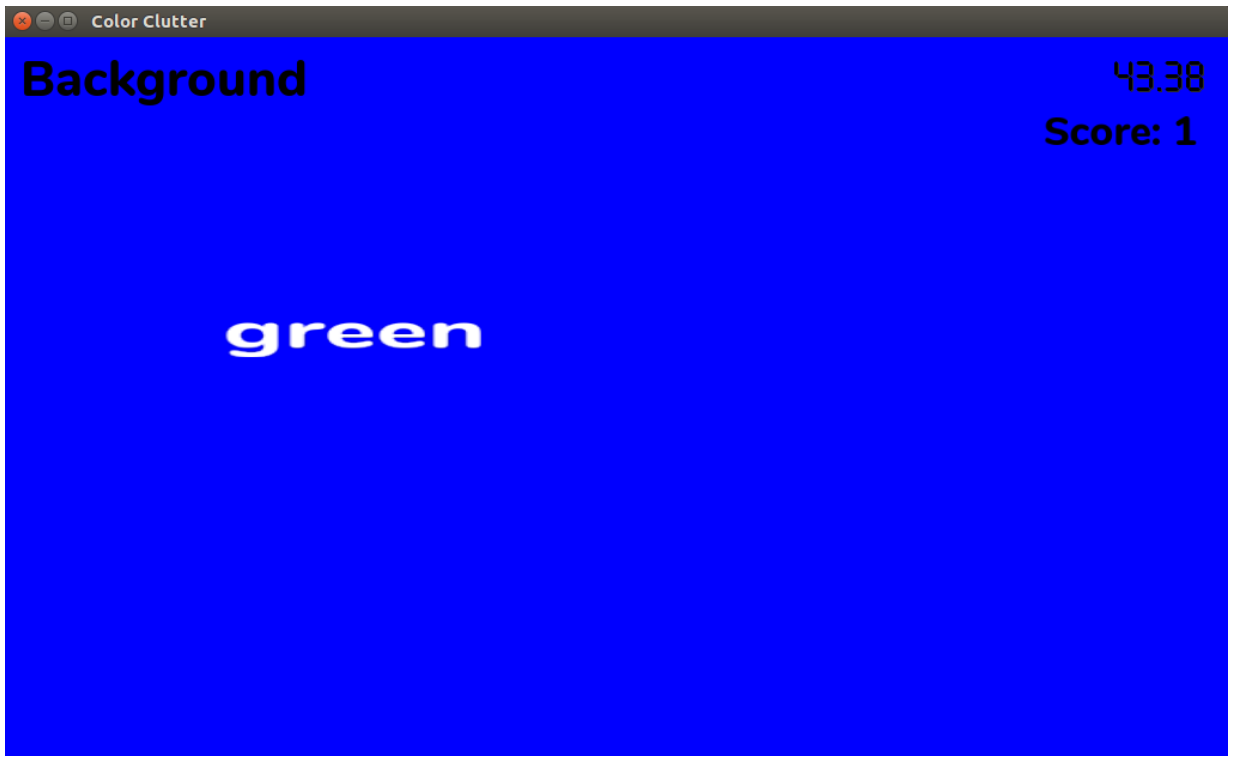


- Testes realizados com 10 usuários diferentes

Color Clutter

(Jogo demonstrativo)

- Jogo criado em *Godot* para demonstrar o módulo *Speech to Text*
- Todo o controle é feito por voz



- Testes realizados com 10 usuários diferentes
  - Rápido, mas rígido quanto à pronúncia de algumas cores

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