# Trabalho Prático - Fase 1 - Requisitos

Rodrigo Guimarães Ourique, Ana de Jesus

Prof. Marcelo Soares Pimenta

## Introdução

De acordo com o enunciado enviado via moodle, "O objetivo do trabalho é a definição, a IMPLEMENTAÇÃO, teste e depuração de um GERADOR DE MÚSICA A PARTIR DE TEXTO, um software que recebe um TEXTO (a princípio, um texto livre, não estruturado, como um conto ou uma página de jornal) como entrada e gera (podemos dizer informalmente que o software ´toca´ via acionamento de funções de som) um conjunto de NOTAS CORRESPONDENTES AO TEXTO segundo alguns PARÂMETROS (timbre, ritmo – na forma de Beats por Minuto ou BPM, etc). Os parâmetros são definidos através de um mapeamento de texto para informações musicais. Parte do mapeamento já está definida abaixo. O restante deve ser definido pelo grupo e documentado para conhecimento do professor."

Para podermos iniciar o desenvolvimento de tal programa, enumeraremos aqui funcionalidades disponibilizadas ao usuário, estruturas e linguagem de programação que permitirão a execução de tais funcionalidades.

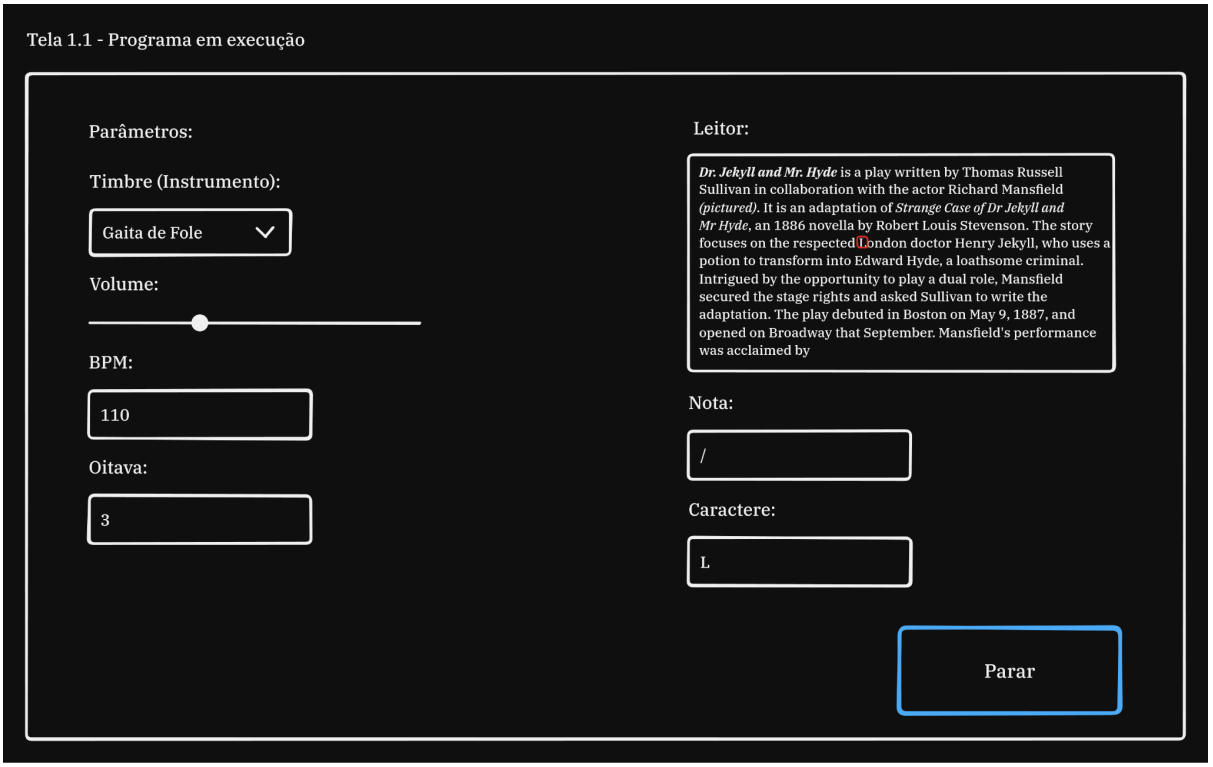
## Funcionalidades

- O programa deve suportar alguma forma de interface gráfica.
- O programa deve, ao ser inicializado, levar o usuário à um menu onde o mesmo informe o arquivo a ser lido e configure os parâmetros de funcionamento.
    - Todos os parâmetros possuem valores padrão, e o usuário pode simplesmente deixá-los como estão.
- Ao realizar a leitura de arquivo e parâmetros, o software deve avançar até uma tela onde o usuário pode acompanhar a evolução do programa à medida que lê o arquivo, podendo visualizar, por exemplo:
    - Instrumento MIDI atual
    - Volume atual
    - Nota atual
    - Oitava atual
- Ao finalizar, o software deve voltar à tela inicial, onde devem ser informados novos parâmetros para que uma nova execução seja realizada.

# Interface de Usuário (GUI)

Tela 1 (início)

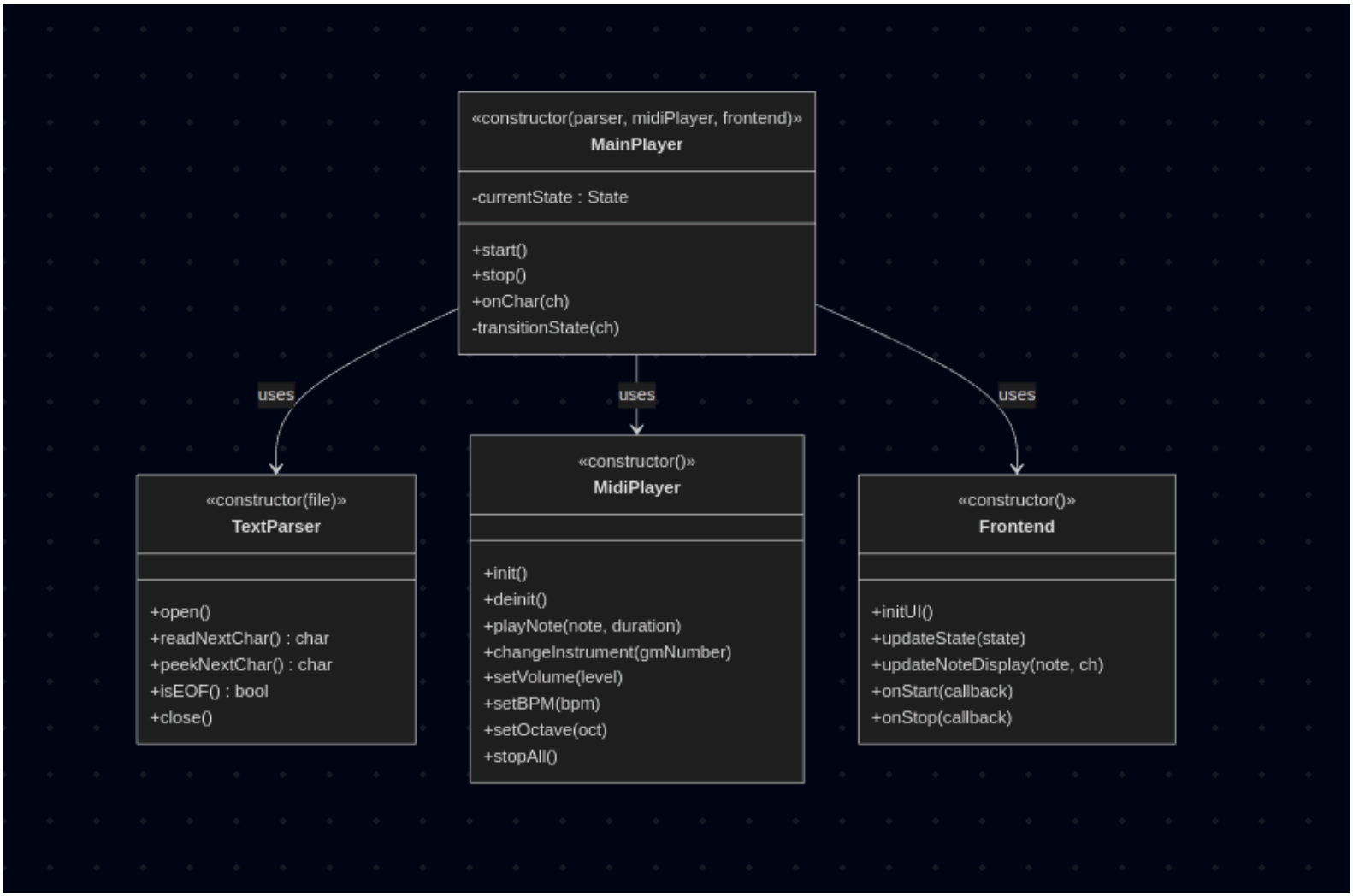

Tela 1.1 (Programa em Execução)



# Diagrama de Classes:

Imaginamos que o programa conterá as seguintes classes:

- [**MainPlayer**] Uma classe para representar a máquina de estados do “Player”
    - É quem irá interfacear com a classe do frontend
    - É quem iŕa definir para qual estado transitar em função do próximo char lido
- [**TextParser**] Uma classe para abstrair o leitor/parser de texto.
    - Classe simples. Provávelmente alocará buffer para ler o arquivo, lerá o próximo char quando pedido, e identificará EOF.
- [**MidiPlayer**] Uma classe para abstrair o player de MIDI
    - É quem irá nos fornecer métodos e interfaces simples de chamar para produção de sons
    - Init/Deinit do sistema MIDI
    - “Wrappeará” complexidade do sistema de MIDI
- [**Frontend**] Uma classe para abstrair o frontend
    - Irá manipular o front end em HTML, para atualização de estados de botões e outros componentes das telas.
    - Provavelmente será instanciada e chamada única e exclusivamente pela classe *MainPlayer*



**O Diagrama foi gerado pelo seguinte código Mermaid:**

```
classDiagram
    class MainPlayer {
        <<constructor(parser, midiPlayer, frontend)>>
        +start()
        +stop()
        +onChar(ch)
        -transitionState(ch)
        -currentState : State
    }

    class TextParser {
        <<constructor(file)>>
        +open()
        +readNextChar() char
        +peekNextChar() char
        +isEOF() bool
        +close()
    }

    class MidiPlayer {
        <<constructor()>>
        +init()
        +deinit()
        +playNote(note, duration)
        +changeInstrument(gmNumber)
        +setVolume(level)
        +setBPM(bpm)
        +setOctave(oct)
        +stopAll()
    }

    class Frontend {
        <<constructor()>>
        +initUI()
        +updateState(state)
        +updateNoteDisplay(note, ch)
        +onStart(callback)
        +onStop(callback)
    }

    MainPlayer --> TextParser : uses
    MainPlayer --> MidiPlayer : uses
```

```
MainPlayer --> Frontend : uses
```

## Observações Finais:

- Após certa deliberação, vemos que o caminho de menor esforço para uma implementação completa desses requisitos seria um WebApp baseado em HTML + JavaScript, em virtude de:
    - Facilidade de implementação de interface gráfica (HTML is an almost universal interface)
    - Facilidade de programação: Por se tratar de um programa simples, não temos necessidade de controle de baixo-nível e alta-performance sobre a máquina. Podemos trocar a complexidade do código (como a que teríamos em uma implementação em C++) pela baixa performance do JS nesse caso.
    - Portabilidade: Web Browsers run anywhere, quite literally.