CENTRO UNIVERSITÁRIO REDENTOR
ENGENHARIA DA COMPUTAÇÃO

IGOR DUARTE DE OLIVEIRA ALMEIDA

METRÔNOMO BÁSICO EM PYTHON
PROJETO N1

Itaperuna-RJ
2025

1. INTRODUÇÃO

O tempo é um dos elementos fundamentais na música, sendo responsável por estabelecer a velocidade de execução da melodia e harmonia. Através dele, é possível desenvolver os ritmos, responsáveis por caracterizar estilos musicais e conferir identidade às composições.

Nesse contexto, o metrônomo desempenha um papel muito importante no desenvolvimento da percepção rítmica, no aperfeiçoamento técnico de músicos e em práticas individuais ou em conjunto, visto que é uma ferramenta que tem a capacidade de marcar o tempo musical emitindo um som de maneira contante e precisa, estabelecendo uma padronização do início ao fim das músicas.

Este trabalho aborda o desenvolvimento de um metrônomo digital. A aplicação conta com uma interface gráfica interativa, permitindo ao usuário configurar o andamento, dar início ao metrônomo e pará-lo. Além de sua funcionalidade prática, o projeto busca explorar, do ponto de vista computacional, a relação entre tempo, ritmo e tecnologia no contexto da música.

2. ORGANIZAÇÃO DO PROJETO

O programa foi desenvolvido em Python e possui uma interface gráfica. O código está dividido em 3 módulos, além do arquivo main.py, que executa o programa.

O primeiro módulo contém as funções referentes à interface gráfica. O segundo módulo trata do funcionamento do metrônomo. E o terceiro módulo carrega o som disparado pelo metrônomo no programa.

O código completo do programa está disponível em: https://github.com/IgorDuart3/trabalhometronomo.git

3. MÓDULOS

Este capítulo tem o intuito de explicar o que foi desenvolvido em cada módulo listado anteriormente.

3.1. INTERFACE GRÁFICA

O módulo que abrange a interface gráfica é denominado interface.py, representado a seguir pela Figura 1.

```
1  from tkinter import *
2  from tkinter import messagebox
3  import metronomo
4
5  def criar_interface():
6      app = Tk()
7      app.title("Metrônomo")
8      app.geometry("500x300")
9
10     bpm_label = Label(app, text="BPM:")
11     bpm_label.pack(pady=5)
12
13     bpm_entry = Entry(app)
14     bpm_entry.insert(0, str(60))
15     bpm_entry.pack(pady=5)
16
17     play_pause_button = Button(app, text="Play")
18
19     def atualizar_bpm_callback():
20         valor = bpm_entry.get()
21         try:
22             valor_int = int(valor)
23             if valor_int < 30 or valor_int > 300:
24                 messagebox.showwarning("BPM inválido", "Por favor, insira um valor entre 30 e 300.")
25                 return False
26             metronomo.atualizar_bpm(valor_int)
27             return True
28         except ValueError:
29             messagebox.showerror("Erro", "Valor inválido. Digite um número inteiro.")
30             return False
31
32     def play_pause_callback():
33         if metronomo.esta_rodando():
34             metronomo.parar_metronomo()
35             play_pause_button.config(text="Play")
36         else:
37             if atualizar_bpm_callback():  # Só inicia se o BPM for válido
38                 metronomo.iniciar_metronomo()
39                 play_pause_button.config(text="Pause")
40
41     atualizar_bpm_button = Button(app, text="Atualizar BPM", command=atualizar_bpm_callback)
42     atualizar_bpm_button.pack(pady=10)
43
44     play_pause_button.config(command=play_pause_callback)
45     play_pause_button.pack(pady=20)
46
47     return app
```

Figura 1: interface.py

Nas linhas 1 e 2 temos os imports usados para a criação da interface gráfica, utilizando a biblioteca TKInter nativa do Python. Na linha 3, são importadas as

informações contidas no módulo metrônomo.py, que será abordado mais a frente. Da linha 5 adiante, teremos a função “criar_interface” que trata todos os elementos da interface gráfica do programa e sua complexidade é O(1). Nas linhas 6 declaramos “app” como o nome da variável da interface e iniciamos o uso da biblioteca com o termo “Tk()”, as linhas 7 e 8 tratam do título e tamanho da mesma. Nas linhas 10 e 11, introduzem BPM à tela. As linhas 13 a 15 são referentes à caixa de texto “bpm_entry” que receberá o valor do BPM. E finalmente, a linha 17 cria o botão Play, que inicia as batidas. As linhas descritas acima constroem a interface representada pela Figura 2.

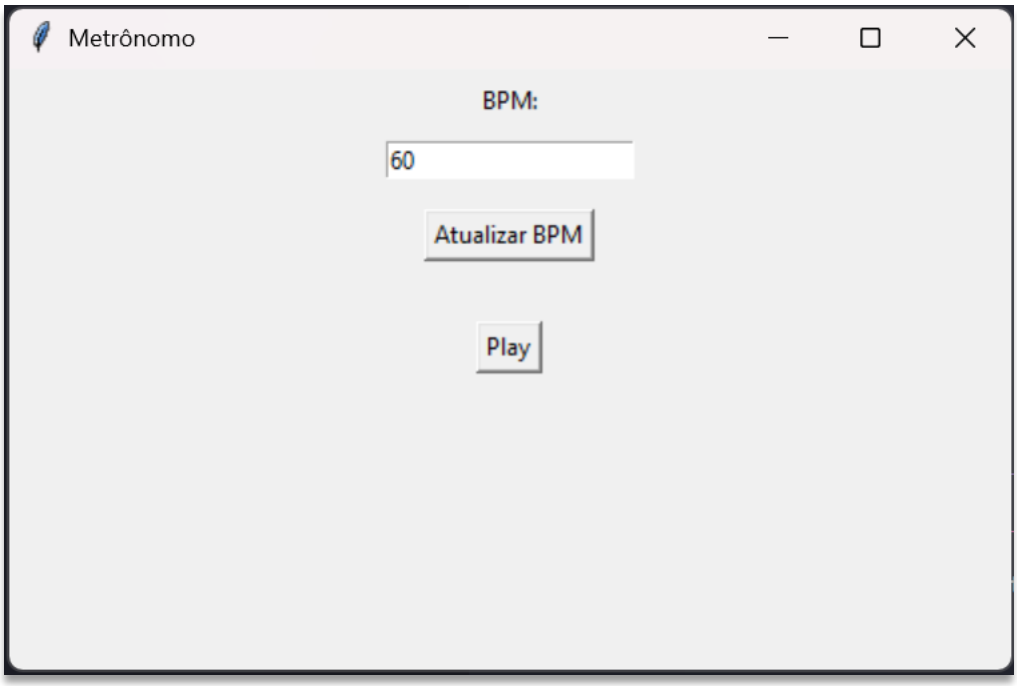

Figura 2: Interface criada com TKInter

A partir da linha 19 até a linha 30, temos a função “atualizar_bpm_callback”, de complexidade O(1), responsável por atualizar o valor do BPM no programa, introduzido pelo usuário e armazenado na variável “bpm_entry” em forma de str e convertido para int na variável “valor_int”. Essa função verifica se o input é menor que 30 ou maior que 300. Caso esteja fora do limite, é exibida uma caixa de erro, ilustrada pelas Figuras 3 e 4.

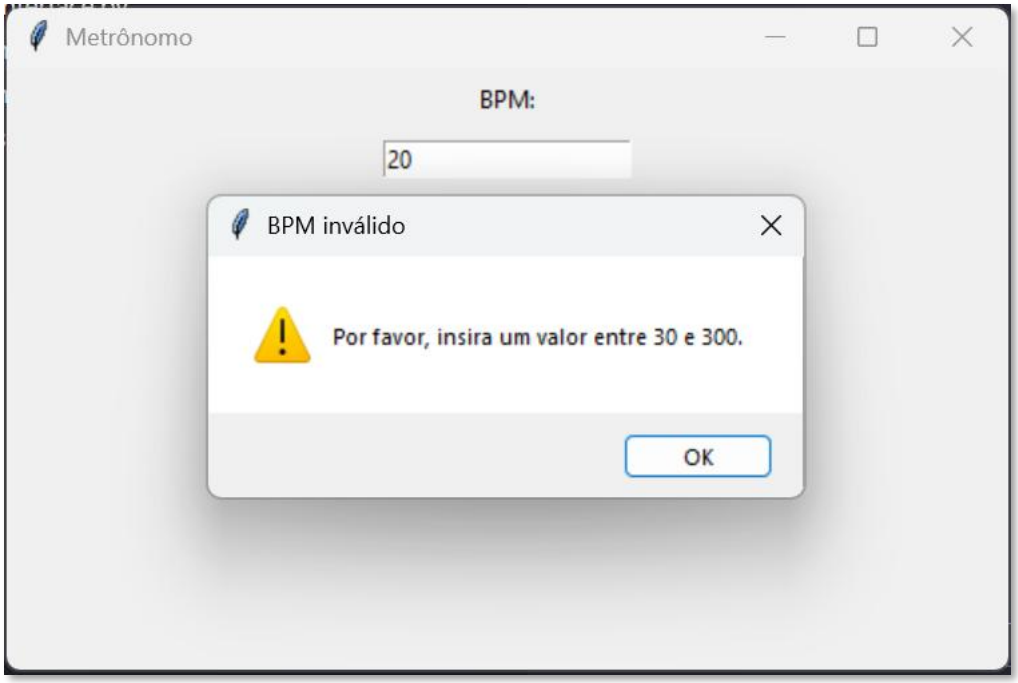

Figura 3: Erro menor que 30

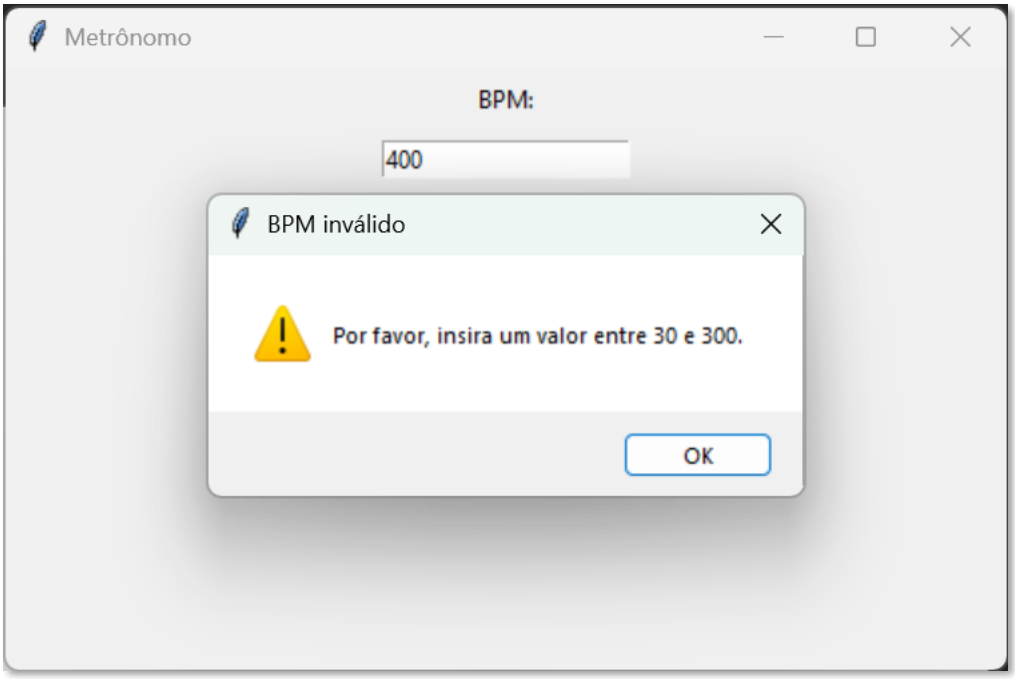

Figura 3: Erro maior que 300

Além disso o programa verifica se o usuário está inserindo letras ou símbolos, e caso não sejam números inteiros, é exibida outra caixa de erro, ilustrada nas Figuras 4 e 5.

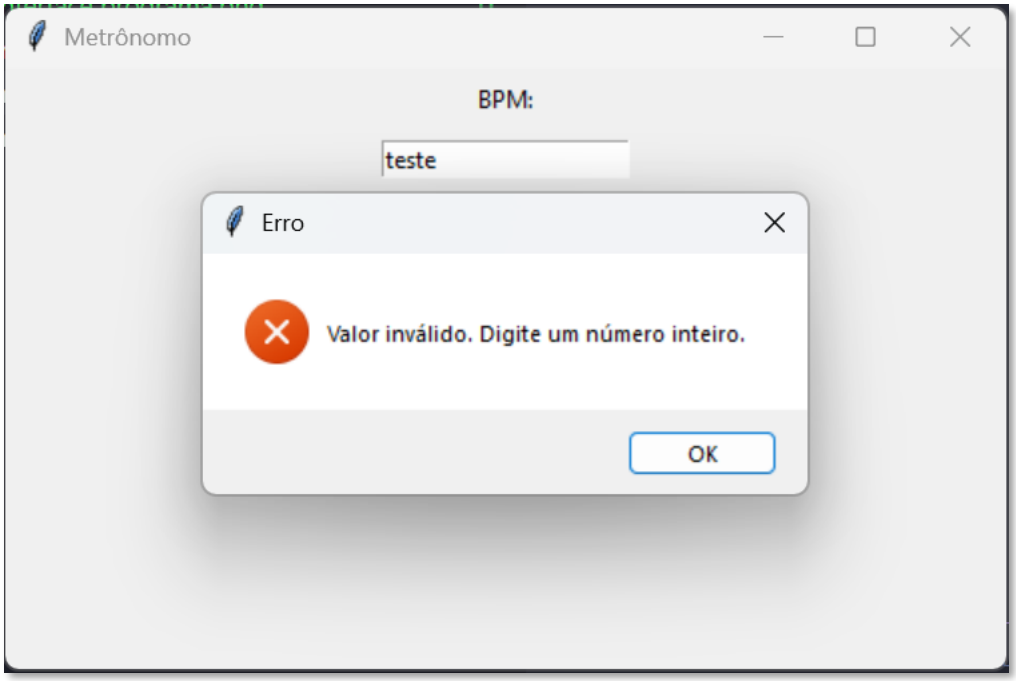

Figura 4: Erro input letras

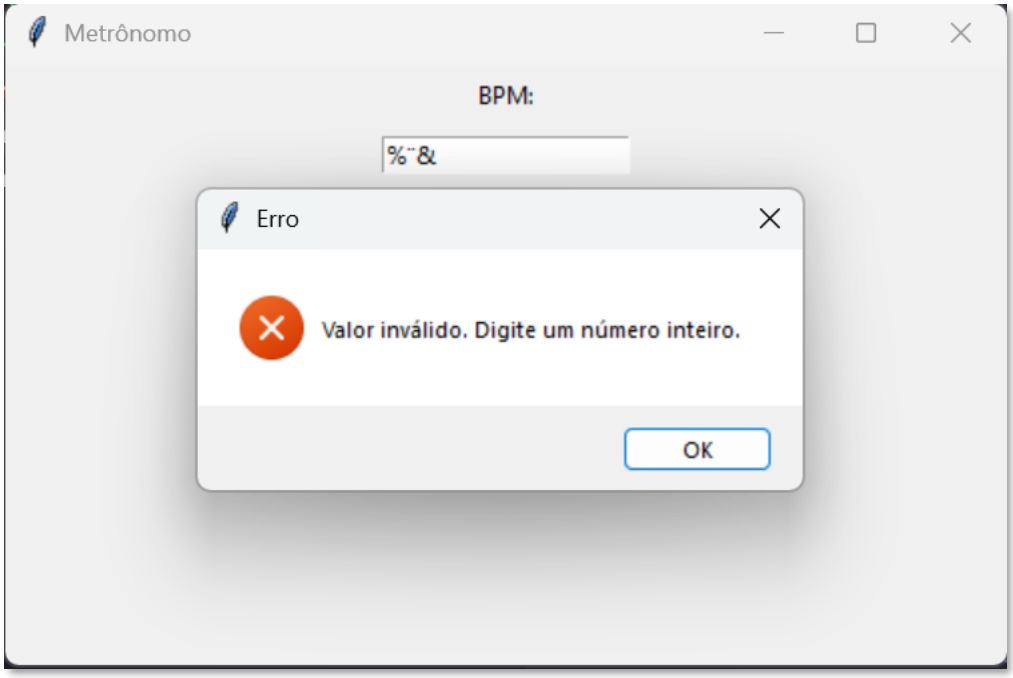

Figura 5: Erro input símbolos

A partir da linha 32 temos a função “play_pause_callback”, de complexidade O(1), que muda o texto do botão de “Play” para “Pause” quando o botão “Play” é acionado. O texto só muda se o input do usuário for válido. A interface com o botão “Pause” será ilustrada a seguir na Figura 6.

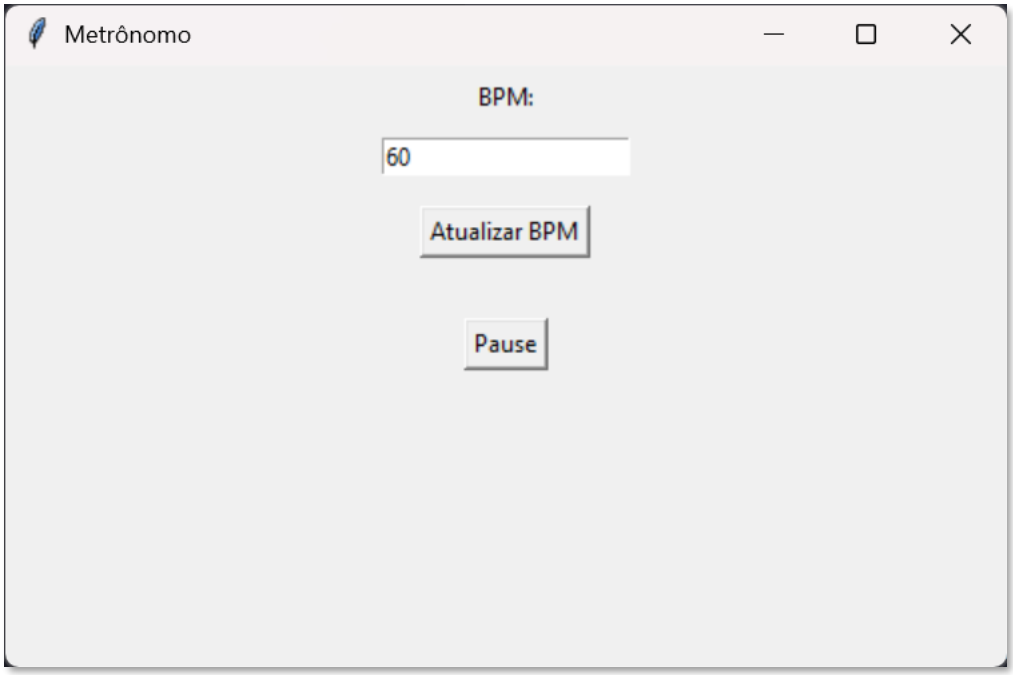

Figura 6: Interface botão pause

3.2. METRÔNOMO

O módulo metrônomo, cujo arquivo recebe o nome metronomo.py é responsável pela lógica do programa, ao intervalo entre as batidas e o loop do som. O código está representado pela Figura 7. Nas linhas 1 e 2, são importadas as bibliotecas time e threading. A biblioteca time vai se responsável pelo intervalo de tempo entre uma batida e outra do metrônomo. A biblioteca threading permitirá que o usuário troque o BPM sem precisar encerrar o programa. A linha 3 está importando a função “som_click” do módulo som que será abordado no próximo capítulo.

A linha 6 inicia um dicionário contendo as variáveis que fazem parte da parte lógica do programa.

A partir da linha 13 temos a função “atualizar_bpm”, de complexidade O(1) responsável por receber o valor de BPM e tratá-lo, assim como foi feito na função “atualizar_bpm_callback” no módulo anterior. Além disso, na linha 20 é feito o cálculo do intervalo de tempo em segundos.

```python
import time
import threading
from sons import som_click

# Variáveis de estado
estado = {
    "rodando": False,
    "bpm": 60,
    "intervalo": 1.0,
    "thread": None
}

def atualizar_bpm(valor_bpm):
    try:
        valor_bpm = int(valor_bpm)
        if valor_bpm < 30 or valor_bpm > 300:
            print("Por favor, informe um valor de BPM entre 30 e 300.")
            return
        estado["bpm"] = valor_bpm
        estado["intervalo"] = 60.0 / estado["bpm"]
        print(f'BPM alterado para {estado["bpm"]}.')
    except ValueError:
        print("Valor inválido. Informe um número inteiro.")

def _loop_metronomo():
    while estado["rodando"]:
        som_click.play()
        time.sleep(estado["intervalo"])

def iniciar_metronomo():
    if not estado["rodando"]:
        estado["rodando"] = True
        estado["thread"] = threading.Thread(target=_loop_metronomo, daemon=True)
        estado["thread"].start()

def parar_metronomo():
    estado["rodando"] = False
    print("Metrônomo parado.")
```

Figura 7: metronomo.py

A função “_loop_metronomo”, de complexidade O(n) mantem o loop do som importado do módulo sons, com o intervalo de tempo correto usando o método .sleep() da biblioteca time.

A função “iniciar_metronomo”, de complexidade O(1), dá início aos pulsos, e utilizando a biblioteca threading e o método .Thread permite que o usuário mude o valor de BPM sem que o programa precise ser encerrado.

A função “parar_metronomo”, de complexidade O(1), interrompe o funcionamento do programa atribuindo o valor Booleano “False” à variável “rodando” presente no dicionário “estado”.

3.3. SONS

O módulo sons utiliza a biblioteca pygame, importada na linha 1, que atribui o som do arquivo “cowbel.wav” à variável som_click através dos métodos .mixer.Sound().

O código do módulo sons, do arquivo sons.py, é representado pela Figura 8.

```
1  import pygame
2
3  pygame.mixer.init()
4  som_click = pygame.mixer.Sound('cowbel.wav')
5
```

Figura 8: sons.py

3.4. MAIN

O arquivo main.py deste programa importa a função “criar_interface” do módulo interface e a executa, utilizando o método .mainloop(), como pode ser observado na Figura 9.

```python
from interface import criar_interface

if __name__ == "__main__":
    app = criar_interface()
    app.mainloop()

```

Figura 9: main.py

4. CONCLUSÃO

Este projeto tem uma aplicação simples. Durante o desenvolvimento, o programa foi testado diversas vezes, tentando simular todos os erros possíveis. Até o momento, o programa apresenta um bom funcionamento.