Trabalho de Técnicas de Construção de Programas

Nomes: Ronald de Souza Maciel, Lucas Marques Silva
Prof. º: Marcelo Pimenta
Técnicas de Construção de Programas — 2018/2

Documentação do Trabalho Final

1) Requisitos:

1.1) Requisitos Funcionais

- Interpretador Python*:* Precisaremos do Interpretador Python para que seja possível ler o código que foi escrito e convertê-lo em bytecode (código de máquina).
- Debugger específico para Python: Permite a colocação de breakpoints para melhor visualização e entendimento do código, nos ajudando a entender o passo a passo do fluxo do código e garantindo a validação de trechos de código no software.
- Framework para teste do Software em Python: Necessário para testar as entradas e validar suas saídas. Ajuda e reforça a modularização do código, já que podemos testar cada módulo separadamente (respeitando o seguindo o "Modelo V" para testes)
- IDE com suporte à linguagem Python: Necessário para escrita e execução o software, conjuntamente com o Interpretador Python.
- GitHub: Repositório do código para auxiliar no desenvolvimento do software.
- Framework para Interface Python: Necessário para usabilidade, permitindo que o cliente utilize o software.
- Sistema Operacional Windows 7 (ou maior) e/ou Sistema Operacional Ubuntu 14.04 (ou maior): Necessário para o gerenciamento de recursos do programa e para fornecer suporte a entrada e saída do software.

1.2) Requisitos Não-Funcionais

- Computador: Hardware necessário para realização e concretização do software.
- Monitor: Necessário para a visualização da execução do software.
- Teclado: Necessário para o gerenciamento de entrada no software.
- Mouse: Necessário para o gerenciamento de controle no software.
- Interface Gráfica do Usuário: Existem boas razões para que usemos uma GUI, um programa de computador com uma GUI bem planejada pode ser usada por qualquer usuário, independente de quão complexo o programa seja.
- Caixa de Som: Necessário para avaliação do conteúdo do software.
- Drivers de som atualizados: Necessário para avaliação do conteúdo do software, além de ser o total responsável pela saída do software.

2) Cronograma

- **23/10**: Entrega da Fase 1: Entrega dos Requerimentos, definições de classes e módulos e interface.

- **09/11**: Entrega da Fase 2: Entrega da Implementação dos módulos, arquivos de testes.

- **28/11**: Entrega da Fase 3: Entrega final da construção do software com as técnicas vistas na disciplina ao longo do semestre.

3) Projeto de definição das Classes:

Abaixo estarão listadas as classes, atributos, métodos e módulos que são utilizados no trabalho final.

Classes:

- Classe de tratamento de texto:

  - *Class Texto:* Classe que gera o .txt com o texto digitado.

- Classes responsáveis pela música, som e atributos:

  - *Class GerenciaMusica:* Classe que se responsabiliza pelo gerenciamento da criação, alteração e manutenção da música

  - *Class Midi:* Classe que possui os atributos necessários para a reprodução de arquivos MIDI.

  - *Class Musica:* Classe que define os atributos iniciais da música que tocará para o usuário. Também responsável pela inicialização inicial padrão da música (default).

- Classe da Interface:

  - *Classe Window(QWidget):* Principal responsável pela criação e configuração da interface de interação com o usuário.

4) Projeto de definição dos Métodos:

Abaixo estão listados os métodos utilizados no trabalho final para a realização da especificação dada.

Métodos:

- *main():* Método inicial da aplicação; por onde o programa começará a ser executado.

- *mapeiaCaractere():* Método que mapeará cada caractere lido na string recebida pelo usuário em uma ação conforme o enunciado do trabalho.

- *inicializaEstruturas():* Método que inicializará as estruturas (variáveis) *varMusica* e *varMidi*.

- *geraTXT()*: Método que gera um arquivo de saída "saida.txt". Esse arquivo possui o texto que o usuário forneceu ao software.

- *load():* Método que carrega Antigo WAV e tranforma-o em um novo, de acordo com a leitura do texto fornecido pelo usuário

- *hz_to_MIDI():* Método que converte de HZ para notas MIDI.

- *criaWav():* Método que cria nova wav conforme as especificações lidas no texto fornecido pelo usuário.

- *tocaMusica():* Método que tocará a música já configurada, de acordo com o texto fornecido pelo usuário.

- *addNoteToMidi():* Método que adiciona cada nota *.midi* ao *.midi* que vai conter todas as notas.

- *criaNovoMIDI():* Método que cria o novo MIDI a ser salvo com as devidas configurações.

- *iniciaUi():* Método que criação e inicialização de partes da interface, como botões e barra de menu.

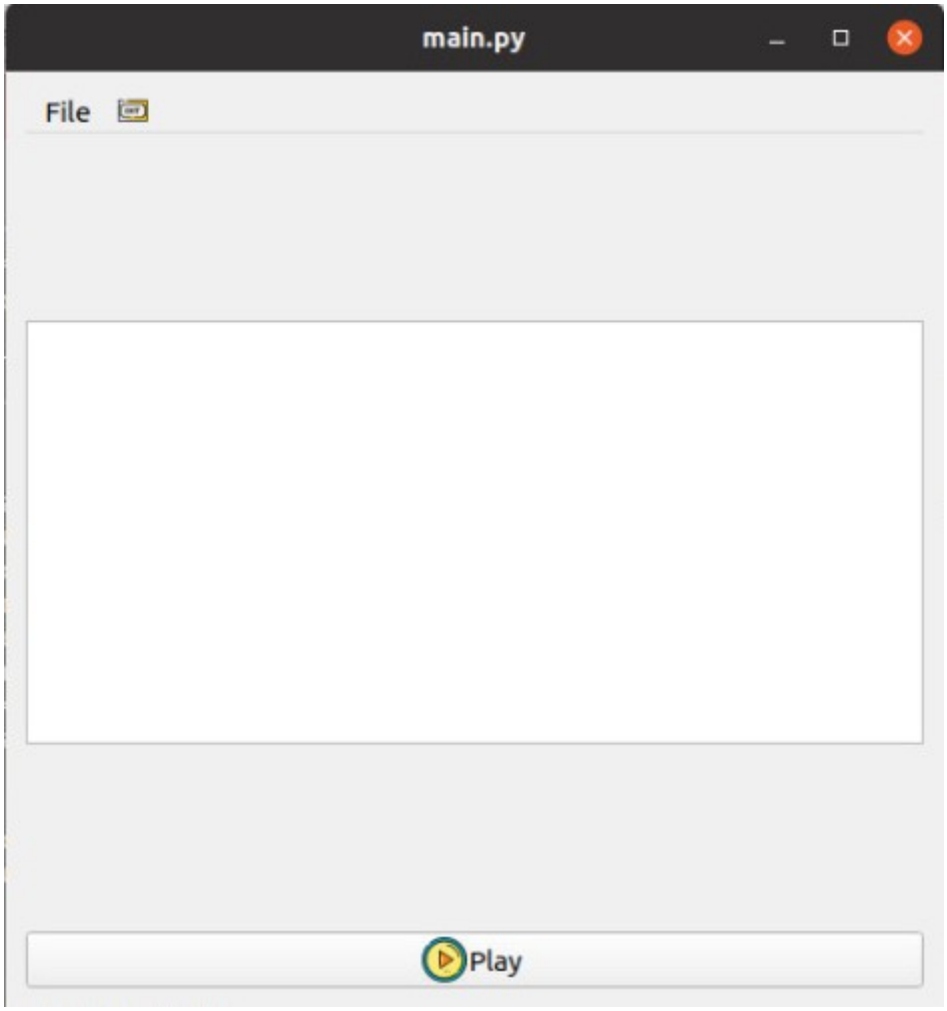
- *carregaTexto():* Método que carrega o texto digitado na interface e passa para o método que trata do mapeamento do texto fornecido pelo usuário.

- *openfiles():* Método que lê o que foi salvo em uma variável que continha o conteúdo digitado na interface e passa para o método de tratamento do texto fornecido pelo usuário.

- *exit():* Método de saída do software.

6) Projeto da Interface com o Usuário

Abaixo temos uma imagem de como é a interface do trabalho final.



Completamente diferente da interface que havíamos pensado, na qual havia duas telas de interface: uma primária, onde continha diversos botões, tais como um *link para o GitHub*, Um *botão de início,* um *botão de instrução (Leia-me).* E uma tela de interface secundária, onde seria aberta após o usuário clicar no *botão de* início. Essa segunda interface possuía - também – diversos botões, como *play, botão de volume, pause,* etc.

Entretanto, percebemos que esse projeto de interface feria uma das primeiras qualidades de softwares que estudamos: *a usabilidade.* Sendo assim, resolvemos reprojetar a interface. Essa interface reprojetada – e portanto, a interface final – possui maior usabilidade (é mais intuitiva para o usuário) e também maior robustez (justamente pelo seu design simplificado induz o usuário a cometer menos erros). Sendo assim, optamos pela interface que pudesse favorecer o maior número de usuários.

Nessa interface, temos as seguintes funcionalidades:

- *File:* Caso o usuário deseje inserir um arquivo de texto externo ao software para a finalidade dele, é possível.

- *Exit:* Botão de saída do software. Fica ao lado da funcionalidade *File* pois pensamos que o usuário precisava de mais uma indução para a saída do software; possui uma imagem de *exit* justamente para chamar mais a atenção do usuário.

- *Campo de texto:* Campo disponível ao usuário para digitar o texto para a transformação do mesmo em música. Fizemos esse campo grande e amplo por dois motivos: chamar a atenção do usuário e maior liberdade em relação ao tamanho do texto.

- *Botão Play:* Botão que deverá ser pressionado após a inserção do texto – seja por arquivo externo ou digitação no campo de texto. Colocamos ele largo e com uma figura pelos mesmos motivos: chamar atenção e indução de ação do usuário.