



UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

**DOCUMENTAÇÃO SOBRE O SIMULADOR DE
MÁQUINA DE TURING**

APUCARANA – PR
2023

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

**DOCUMENTAÇÃO SOBRE O SIMULADOR DE MÁQUINA DE
TURING**

Trabalho apresentado à disciplina de
Linguagens Formais Autômatos e
Computabilidade, do curso de Bacharelado
em Ciência da Computação.

Professor: Guilherme Nakahata

**APUCARANA – PR
2023**

SUMÁRIO

| | |
|-------------------------------|----|
| INSTRUÇÕES DE EXECUÇÃO | 4 |
| INSTRUÇÕES DE COMPILAÇÃO | 5 |
| DOCUMENTAÇÃO | 7 |
| main_window.py | 7 |
| manual_input_screen.py | 14 |
| file_input_screen.py | 25 |
| instruction.py | 31 |
| instruction_list_processor.py | 32 |

INSTRUÇÕES DE EXECUÇÃO

Windows

Execute *turing_machine_simulator.exe*

Linux

Abra o terminal na pasta que contém o *turing_machine_simulator*

Conceda permissão de execução para o arquivo através do comando

```
chmod +x turing_machine_simulator
```

Execute o programa através do comando

```
./turing_machine_simulator
```

INSTRUÇÕES DE COMPILAÇÃO

Windows

Instale o Python encontrado em:

<https://www.python.org/downloads/>

Abra o terminal para instalar as dependências:

```
pip install PyQt6 PyInstaller
```

Abra o terminal na pasta com o código fonte

Utilize o comando para gerar o executável na pasta dist

```
pyInstaller main_window.py --onefile --noconsole  
--icon=logo.ico --add-data "resources;resources"
```

Após isso siga as instruções de execução

Linux

Abra o terminal e instale o Python

Ubuntu/Debian

```
sudo apt install python3
```

Fedora

```
sudo dnf install python3
```

CentOS

```
sudo yum install centos-release-scl  
sudo yum install rh-python36  
scl enable rh-python36 bash
```

Arch

```
sudo pacman -S python
```

Instale o Package Installer for Python (pip)

Ubuntu/Debian

```
sudo apt install python3-pip
```

Fedora

```
sudo dnf install python3-pip
```

CentOS

```
sudo yum install python3-pip
```

Arch

```
sudo pacman -S python-pip
```

Ou utilizando o próprio Python

```
python3 get-pip.py
```

Instale as dependências:

```
sudo pip3 install pyinstaller pyqt6
```

Abra o terminal na pasta com o código fonte

Utilize o comando para gerar o arquivo binário na pasta dist

```
python3 -m PyInstaller main_window.py --onefile --noconsole  
--icon=logo.ico --add-data "resources:resources"
```

Após isso siga as instruções de execução

DOCUMENTAÇÃO

main_window.py

- Primeiramente os módulos e classes necessários são importados:

```
1 import os
2 import sys
3 from PyQt6.QtCore import Qt, QEvent
4 from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QStackedLayout, QMessageBox
5 from PyQt6.QtGui import QtGuiApplication, QIcon
6 from manual_input_screen import ManualInputScreen
7 from file_input_screen import FileInputScreen
8 from transition_table_screen import TransitionTableScreen
```

os e **sys**: Usado para manipulação de caminhos de arquivos e operações relacionadas ao sistema.

PyQt6.QtCore, **PyQt6.QtWidgets**, **PyQt6.QtGui**: Módulos do PyQt6 para criação de GUI.

QEvent: Uma classe do PyQt6 usada para lidar com eventos.

QApplication, **QMainWindow**, **QWidget**, **QLabel**, **QPushButton**, **QVBoxLayout**, **HBoxLayout**, **QStackedLayout**, **QMessageBox**: Classes PyQt6 para criação de elementos GUI, Application gerencia a instanciação ou execução da aplicação da além de ser utilizada dentro da própria MainWindow para obter as dimensões da tela do usuário, MainWindow para definir a janela principal, Widgets são estruturas genéricas do Qt estilo divs do HTML, labels são pequenos textos de títulos ou nomes, button é um botão, VBox é o layout vertical, HBox horizontal e Stacked é o layout em pilha, finalmente MessageBox é uma caixa de pop up utilizada para alertas.

QtGuiApplication, **QIcon**: Classes PyQt6 para aplicações GUI e gerenciamento de ícones.

ManualInputScreen, **FileInputScreen**, **TransitionTableScreen**: Classes personalizadas de módulos externos usados como parte da aplicação, elas direcionam o usuário para as telas de entrada manual ou por arquivo e são dinamicamente adicionadas ou retiradas do layout em pilha caso estão em foco ou não.

Nota-se que devido a estrutura de layouts e widgets do PyQt, muitas vezes temos que criar widgets que estão contidos em layouts e posteriormente

criamos widgets contêineres para segurar esses layouts para serem incluídos em outros layouts de hierarquia maior.

Isso inicialmente pode parecer bem confuso, mas é a maneira de conseguir resultados mais previsíveis e definidos para a estruturação e posicionamento de cada elemento de uma aplicação.

Novamente remete-se a analogia com o HTML que também segue uma grande estrutura hierárquica de vários componentes para popular o conteúdo de uma página.

- **Manipulação de caminhos do PyInstaller:**

```
9  ## PyInstaller file path handler
10 if getattr(sys, 'frozen', False):
11     # Running as a PyInstaller executable
12     base_path = sys._MEIPASS
13 else:
14     # Running as a script
15     base_path = os.path.abspath(".")
```

Verifica se o código está sendo executado como um executável PyInstaller ou como um script. Define o `base_path` de acordo para lidar com caminhos de arquivo, isso previne possíveis erros na execução do arquivo buildado.

- **Criando a janela principal do aplicativo (classe `MainWindow`):**


```

18 class MainWindow(QMainWindow):
19     def __init__(self):
20         super().__init__()
21
22         self.input_widget = None
23         self.transition_table_widget = None
24
25         self.setWindowTitle("Simulador Maquina de Turing")
26         self.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
27         self.central_widget = QWidget()
28         self.setCentralWidget(self.central_widget)
29
30         ## User screen's dimenions
31         self.screen = QGuiApplication.primaryScreen()
32         self.screen_size = self.screen.availableSize()
33
34         ## Master Layout
35         self.main_layout = QVBoxLayout(self.central_widget)
36         self.main_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

```

É uma subclasse do módulo QMainWindow do PyQt6, ou seja, essa será nossa janela “mestre” onde o layout em pilha meramente vai fazer a adição, remoção e troca do elemento ativo que será mostrado ao usuário.

Definimos o título e o ícone da janela.

Define o widget central da janela principal.

Recuperamos as dimensões da tela para ajustes de layout.

Criamos o layout principal como um layout vertical (QVBoxLayout) e definimos seu alinhamento.

- **Cabeçalho da aplicação:**

```

39         ## Application Header
40         title_label = QLabel("<h1>Simulador Maquina de Turing</h1>")
41         title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
42         self.main_layout.addWidget(title_label)

```

Criamos um rótulo de título e o adicionamos ao layout principal.

Definimos o alinhamento do rótulo do título para o centro.

- **Layout em pilha:**

```

43     ## Stacked Layout
44     self.stacked_layout_container = QWidget()
45     self.stacked_layout = QStackedLayout()
46     self.stacked_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
47

```

Configuramos um widget de contêiner (stacked_layout_container) e um layout em pilha (stacked_layout) para gerenciar múltiplas visualizações.

Garantimos que o alinhamento do layout empilhado esteja centralizado.

- **Primeira tela da pilha: Boas-vindas:**

```

49     ## First stacked view: Welcome Screen
50     self.welcome_layout_container = QWidget()
51     self.welcome_layout_container.setMinimumWidth(int(self.screen_size.width() * 0.70))
52     self.welcome_layout_container.setMinimumHeight(int(self.screen_size.height() * 0.70))
53
54     self.welcome_layout = QVBoxLayout()
55     self.welcome_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
56
57

```

```

58     welcome_label = QLabel("<h2>Bem vindo!</h2>")
59     welcome_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
60     self.welcome_layout.addWidget(welcome_label)
61
62     self.welcome_buttons_container = QWidget()
63     self.welcome_buttons_layout = QHBoxLayout()
64
65     self.button_script_manual = QPushButton("Manual")
66     self.button_script_manual.clicked.connect(self.show_manual_input_screen)
67     self.welcome_buttons_layout.addWidget(self.button_script_manual)
68     self.button_script_arquivo = QPushButton("Arquivo")
69     self.button_script_arquivo.clicked.connect(self.show_file_input_screen)
70     self.welcome_buttons_layout.addWidget(self.button_script_arquivo)
71
72     self.welcome_buttons_container.setLayout(self.welcome_buttons_layout)
73     self.welcome_layout.addWidget(self.welcome_buttons_container)
74
75     self.welcome_layout_container.setLayout(self.welcome_layout)

```

Criamos um widget de contêiner (welcome_layout_container) com dimensões mínimas.

Configuramos um layout vertical (welcome_layout) para esta tela e garantimos que seu alinhamento esteja centralizado.

Criamos um rótulo de boas-vindas e o adicionamos ao layout de boas-vindas.

Criamos botões para telas de entrada manual e de arquivos e os conecta às suas respectivas funções.

Adicionamos os botões a um layout horizontal (welcome_buttons_layout) dentro de um contêiner (welcome_buttons_container).

Adicionamos o contêiner ao layout de boas-vindas.

Definimos o layout de boas-vindas para o contêiner de layout de boas-vindas.

Adicionamos o contêiner de layout de boas-vindas ao layout em pilha.

- **Finalizando o layout em pilha:**

```
78     ## Finalizing the stacked layout
79     self.stacked_layout.addWidget(self.welcome_layout_container)
80
81     self.stacked_layout_container.setLayout(self.stacked_layout)
82
83     self.main_layout.addWidget(self.stacked_layout_container)
```

Definimos o layout empilhado para o contêiner de layout empilhado.

Adicionamos o contêiner de layout empilhado ao layout principal.

- **Personalização de layout:**

```
85     self.main_layout.setSpacing(20)
86     self.main_layout.setContentsMargins(30, 30, 30, 30)
87
88     self.showMaximized()
```

Definimos espaçamento e margens para o layout principal.

Definimos que a aplicação será inicializada de forma maximizada.

- **Funções dessa classe principal:**

```
90     def changeEvent(self, event):
91         if event.type() == QEvent.Type.WindowStateChange:
92             if self.windowState() & Qt.WindowState.WindowMaximized:
93                 self.isMaximized = True
94             else:
95                 if self.isMaximized:
96                     self.center_on_screen()
97                 self.isMaximized = False
```

Definimos uma função `changeEvent` para lidar com a alteração no estado da janela de maximizado para janela.

```
99     def center_on_screen(self):
100         screen_geometry = QApplication.primaryScreen().availableGeometry()
101
102         center_x = int((screen_geometry.width() - self.width()) / 2)
103         center_y = int((screen_geometry.height() - self.height()) / 2.5)
104
105         self.move(center_x, center_y)
```

Definimos uma função `center_on_screen` para centralizar a janela na tela calculando a diferença entre o espaço total da tela do usuário e o ocupado pela janela da aplicação.

```
107 def show_alert_box(self, title, text):
108     alert=QMessageBox()
109     alert.setIcon(QMessageBox.Icon.Information)
110     alert.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
111     alert.setWindowTitle(title)
112     alert.setText(text)
113     alert.setStandardButtons(QMessageBox.StandardButton.Ok)
114     alert.exec()
```

Definimos uma função `show_alert_box` para exibir uma caixa de mensagem informativa com título e texto que são passados como argumentos, desta forma toda vez que precisarmos de uma mensagem customizada de alerta na aplicação podemos invocar esta função.

```
119     def show_welcome_screen(self):
120         self.stacked_layout.setCurrentWidget(self.welcome_layout_container)
121         if not self.isMaximized:
122             self.center_on_screen()
123         self.destroy_input_widget()
124         self.destroy_transition_table_widget()
125
126     def show_manual_input_screen(self):
127         self.input_widget = ManualInputScreen(self.show_welcome_screen, self)
128         self.stacked_layout.addWidget(self.input_widget)
129         self.stacked_layout.setCurrentWidget(self.input_widget)
130
131     def show_file_input_screen(self):
132         self.input_widget = FileInputScreen(self.show_welcome_screen, self)
133         self.stacked_layout.addWidget(self.input_widget)
134         self.stacked_layout.setCurrentWidget(self.input_widget)
```

```

135
136     def show_transition_table_screen(self):
137         number_of_states, main_alphabet_size, main_alphabet, main_alphabet_li
138
139         self.transition_table_widget = TransitionTableScreen(number_of_states
140 self.stacked_layout.addWidget(self.transition_table_widget)
141 self.stacked_layout.setCurrentWidget(self.transition_table_widget)

```

Definimos funções para mostrar a tela de boas-vindas e passar para as telas de entrada manual, do preenchimento da tabela de transição e de arquivos.

```

143     def destroy_input_widget(self):
144         if self.input_widget:
145             self.input_widget.deleteLater()
146             self.stacked_layout.removeWidget(self.input_widget)
147             self.input_widget.deleteLater()
148             self.input_widget = None
149
150     def destroy_transition_table_widget(self):
151         if self.transition_table_widget:
152             self.transition_table_widget.deleteLater()
153             self.stacked_layout.removeWidget(self.transition_table_widget)
154             self.transition_table_widget.deleteLater()
155             self.transition_table_widget = None

```

Definimos funções `destroy_input_widget` e `destroy_transition_table_widget` para remover e excluir os widgets atuais ao alternar entre telas, poupando memória.

- Finalmente, o bloco `if __name__ == "__main__":`:

```

139 if __name__ == "__main__":
140     app = QApplication(sys.argv)
141     window = MainWindow()
142     sys.exit(app.exec())

```

Inicializa a aplicação PyQt6 (`app`).

Cria uma instância da classe `MainWindow` (janela).

Inicia o loop de eventos da aplicação com `app.exec()`.

- Em resumo, `main_window.py`:

Cria um aplicativo GUI com um layout empilhado que alterna entre uma tela de boas-vindas, uma tela de entrada manual e uma tela de entrada de

arquivo. Ele também lida com alterações de estado da janela e fornece funções para exibir mensagens de alerta e centralizar a janela na tela.

manual_input_screen.py

- Primeiramente os módulos e classes necessários são importados:

```
1 from PyQt6.QtCore import Qt, QTimer
2 from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea
3 from input_verification import input_parsing
```

PyQt6.QtCore, PyQt6.QtWidgets: Módulos do PyQt6 para criação de GUI.

QTimer: Uma classe do PyQt6 usada para lidar com a contagem de tempo.

QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py, a única novidade aqui é a ScrollArea que como o nome indica cria uma área de espaço definido que é expandida através do uso de barras de rolagem.

input_verification, input_parsing: Arquivo com funções auxiliares na verificação dos valores de entrada, ele é importado pois também é reutilizado na entrada por arquivo.

- Criando o widget da tela de entrada manual (classe ManualInputScreen):

```
5 class ManualInputScreen(QWidget):
6     def __init__(self, show_welcome_screen_callback, center_on_screen_callback):
7         super().__init__()
8
9         self.show_welcome_screen_callback = show_welcome_screen_callback
10        self.center_on_screen_callback = center_on_screen_callback
11        self.show_alert_box_callback = show_alert_box_callback
12        self.show_transition_table_callback = show_transition_table_callback
13
14        self.number_of_states = 0
15        self.main_alphabet_size = 0
16        self.main_alphabet = set()
17        self.main_alphabet_list = []
18        self.aux_alphabet_size = 0
19        self.aux_alphabet = set()
20        self.aux_alphabet_list = []
21        self.start_symbol = ''
22        self.blank_symbol = ''
```

```

24     ## Master Layout
25     self.manual_input_layout = QVBoxLayout()
26     self.manual_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
27

```

É uma subclasse do módulo widget do PyQt, como se fosse uma div do HTML, extremamente customizável.

Iniciamos a classe passando algumas funções da classe principal como argumento na forma de “callbacks” para que a mesma função seja invocada. Essas funções são para voltar a tela de boas vindas, criar uma mensagem de alerta, centralizar a aplicação na tela e prosseguir para a tela de preencher a tabela de transição.

Definimos várias variáveis auxiliares que serão os parâmetros da máquina de turing, como número de estados e os símbolos utilizados.

Nota-se a utilização de lists e sets, a primeira para preservar a ordem de entrada do usuário e a segunda para verificar se os símbolos são únicos.

Criamos o layout principal como um layout vertical (QVBoxLayout) e definimos seu alinhamento.

- **Cabeçalho:**

```

29     ## Manual Input Screen Header
30     self.manual_input_label = QLabel("<h2>Entrada manual de valores</h2>")
31     self.manual_input_label.setMaximumHeight(120)
32     self.manual_input_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
33     self.manual_input_layout.addWidget(self.manual_input_label)
34

```

Criamos um rótulo de título e o adicionamos ao layout principal.

Definimos o alinhamento do rótulo do título para o centro.

- **Entrada do número de estados:**

```

36     ## Number of states input
37     self.manual_input_number_of_states_container = QWidget()
38     self.manual_input_number_of_states_container.setMaximumHeight(60)
39     self.manual_input_number_of_states_layout = QHBoxLayout()
40
41     self.number_of_states_label = QLabel("Numero de estados: ")
42     self.manual_input_number_of_states_layout.addWidget(self.number_of_states_label)
43
44     self.number_of_states_input = QLineEdit()
45     self.number_of_states_input.setPlaceholderText(f"Insira um valor inteiro")
46     self.manual_input_number_of_states_layout.addWidget(self.number_of_states_input)
47

```



```

47
48 self.number_of_states_input_confirm_button = QPushButton("Confirmar")
49 self.number_of_states_input_confirm_button.clicked.connect(self.show_main_alphabet_size_input)
50 self.manual_input_number_of_states_layout.addWidget(self.number_of_states_input_confirm_button)
51
52 self.manual_input_number_of_states_container.setLayout(self.manual_input_number_of_states_layout)
53 self.manual_input_layout.addWidget(self.manual_input_number_of_states_container)

```

Criamos um layout como contêiner deste item, inicialmente é o único item de entrada visível.

Criamos um QLabel para este item, um campo de entrada e um botão de confirmação em um layout horizontal.

Indicamos ao usuário que o valor deve ser inteiro com um texto placeholder.

- **Entrada do tamanho do alfabeto principal:**

```

56 ## Main alphabet size input
57 self.manual_input_main_alphabet_size_container = QWidget()
58 self.manual_input_main_alphabet_size_container.setVisible(False)
59 self.manual_input_main_alphabet_size_container.setMaximumHeight(60)
60 self.manual_input_main_alphabet_size_layout = QHBoxLayout()
61
62 self.main_alphabet_size_label = QLabel("Tamanho do alfabeto principal: ")
63 self.manual_input_main_alphabet_size_layout.addWidget(self.main_alphabet_size_label)
64
65 self.main_alphabet_size_input = QLineEdit()
66 self.main_alphabet_size_input.setPlaceholderText(f"Insira um valor inteiro")
67 self.manual_input_main_alphabet_size_layout.addWidget(self.main_alphabet_size_input)
68
69 self.main_alphabet_size_confirm_button = QPushButton("Confirmar")
70 self.main_alphabet_size_confirm_button.clicked.connect(self.show_main_alphabet_values_input)
71 self.manual_input_main_alphabet_size_layout.addWidget(self.main_alphabet_size_confirm_button)
72
73 self.manual_input_main_alphabet_size_container.setLayout(self.manual_input_main_alphabet_size_layout)
74 self.manual_input_layout.addWidget(self.manual_input_main_alphabet_size_container)

```

Criamos um layout como contêiner deste item, inicialmente é invisível.

Criamos um QLabel para este item, um campo de entrada e um botão de confirmação em um layout horizontal.

Indicamos ao usuário que o valor deve ser inteiro com um texto placeholder.

- **Entrada do alfabeto principal:**

```

77  ## Main alphabet values input
78  self.manual_input_main_alphabet_values_container = QWidget()
79  self.manual_input_main_alphabet_values_container.setVisible(False)
80  self.manual_input_main_alphabet_values_container.setMaximumHeight(80)
81  self.manual_input_main_alphabet_values_layout = QHBoxLayout()
82
83  self.main_alphabet_values_label = QLabel("Alfabeto principal: ")
84  self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_label)
85
86  self.main_alphabet_values_input_scroll_area = QScrollArea()
87
88  self.main_alphabet_values_input_container = QWidget()
89  self.main_alphabet_values_input_layout = QHBoxLayout()
90
91  ### show_main_alphabet_values_input has the logic for adding QLineEdit widgets
92
93  self.main_alphabet_values_input_container.setLayout(self.main_alphabet_values_input_layout)
94
95  self.main_alphabet_values_input_scroll_area.setWidget(self.main_alphabet_values_input_container)
96  self.main_alphabet_values_input_scroll_area.setWidgetResizable(True)
97  self.main_alphabet_values_input_scroll_area.setMaximumHeight(80)
98
99  self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_input_scroll_area)
100
101  self.main_alphabet_values_confirm_button = QPushButton("Confirmar")
102  self.main_alphabet_values_confirm_button.clicked.connect(self.show_aux_alphabet_size_input)
103  self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_confirm_button)
104
105  self.manual_input_main_alphabet_values_container.setLayout(self.manual_input_main_alphabet_values_layout)
106  self.manual_input_layout.addWidget(self.manual_input_main_alphabet_values_container)

```

Criamos um layout como contêiner deste item, inicialmente é invisível.

Criamos um QLabel para este item, N campos de entrada onde N é o tamanho do alfabeto principal dentro de uma ScrollArea e um botão de confirmação em um layout horizontal.

- **Entrada do alfabeto auxiliar:**

```

109  ## Aux alphabet size input
110  self.manual_input_aux_alphabet_size_container = QWidget()
111  self.manual_input_aux_alphabet_size_container.setVisible(False)
112  self.manual_input_aux_alphabet_size_container.setMaximumHeight(60)
113  self.manual_input_aux_alphabet_size_layout = QHBoxLayout()
114
115  self.aux_alphabet_size_label = QLabel("Tamanho do alfabeto auxiliar: ")
116  self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_label)
117
118  self.aux_alphabet_size_input = QLineEdit()
119  self.aux_alphabet_size_input.setPlaceholderText(f"Insira um valor inteiro")
120  self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_input)

```

```

121 self.aux_alphabet_size_confirm_button = QPushButton("Confirmar")
122 self.aux_alphabet_size_confirm_button.clicked.connect(self.show_aux_alphabet_values_input)
123 self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_confirm_button)
124
125
126 self.manual_input_aux_alphabet_size_container.setLayout(self.manual_input_aux_alphabet_size_layout)
127 self.manual_input_layout.addWidget(self.manual_input_aux_alphabet_size_container)

```

```

130 ## Aux alphabet values input
131 self.manual_input_aux_alphabet_values_container = QWidget()
132 self.manual_input_aux_alphabet_values_container.setVisible(False)
133 self.manual_input_aux_alphabet_values_container.setMaximumHeight(80)
134 self.manual_input_aux_alphabet_values_layout = QHBoxLayout()
135
136 self.aux_alphabet_values_label = QLabel("Alfabeto auxiliar: ")
137 self.manual_input_aux_alphabet_values_layout.addWidget(self.aux_alphabet_values_label)
138
139 self.aux_alphabet_values_input_scroll_area = QScrollArea()
140
141 self.aux_alphabet_values_input_container = QWidget()
142 self.aux_alphabet_values_input_layout = QHBoxLayout()
143
144 ### show_aux_alphabet_values_input has the logic for adding QLineEdit widgets
145
146 self.aux_alphabet_values_input_container.setLayout(self.aux_alphabet_values_input_layout)
147
148 self.aux_alphabet_values_input_scroll_area.setWidget(self.aux_alphabet_values_input_container)
149 self.aux_alphabet_values_input_scroll_area.setWidgetResizable(True)
150 self.aux_alphabet_values_input_scroll_area.setMaximumHeight(80)
151
152 self.manual_input_aux_alphabet_values_layout.addWidget(self.aux_alphabet_values_input_scroll_area)
153
154 self.aux_alphabet_values_confirm_button = QPushButton("Confirmar")
155 self.aux_alphabet_values_confirm_button.clicked.connect(self.show_start_symbol_input)
156 self.manual_input_aux_alphabet_values_layout.addWidget(self.aux_alphabet_values_confirm_button)
157
158 self.manual_input_aux_alphabet_values_container.setLayout(self.manual_input_aux_alphabet_values_layout)
159 self.manual_input_layout.addWidget(self.manual_input_aux_alphabet_values_container)

```

Idêntico ao alfabeto principal.

- **Entrada do símbolo de início:**

```

162 ## Start symbol input
163 self.manual_input_start_symbol_input_container = QWidget()
164 self.manual_input_start_symbol_input_container.setVisible(False)
165 self.manual_input_start_symbol_input_container.setMaximumHeight(60)
166 self.manual_input_start_symbol_input_layout = QHBoxLayout()
167
168 self.start_symbol_input_label = QLabel("Símbolo marcador de inicio: ")
169 self.manual_input_start_symbol_input_layout.addWidget(self.start_symbol_input_label)

```

```

170
171 self.start_symbol_input = QLineEdit()
172 self.start_symbol_input.setPlaceholderText(f"Insira um simbolo")
173 self.start_symbol_input.setMaxLength(1)
174 self.manual_input_start_symbol_input_layout.addWidget(self.start_symbol_input)
175
176 self.start_symbol_input_confirm_button = QPushButton("Confirmar")
177 self.start_symbol_input_confirm_button.clicked.connect(self.show_blank_symbol_input)
178 self.manual_input_start_symbol_input_layout.addWidget(self.start_symbol_input_confirm_button)
179
180 self.manual_input_start_symbol_input_container.setLayout(self.manual_input_start_symbol_input_layout)
181 self.manual_input_layout.addWidget(self.manual_input_start_symbol_input_container)

```

Criamos um layout como contêiner deste item, inicialmente é invisível.

Criamos um QLabel para este item, um campo de entrada e um botão de confirmação em um layout horizontal.

- **Entrada do símbolo de branco:**

```

184 ## Blank symbol input
185 self.manual_input_blank_symbol_input_container = QWidget()
186 self.manual_input_blank_symbol_input_container.setVisible(False)
187 self.manual_input_blank_symbol_input_container.setMaximumHeight(60)
188 self.manual_input_blank_symbol_input_layout = QHBoxLayout()
189
190 self.blank_symbol_input_label = QLabel("Símbolo marcador de branco: ")
191 self.manual_input_blank_symbol_input_layout.addWidget(self.blank_symbol_input_label)
192
193 self.blank_symbol_input = QLineEdit()
194 self.blank_symbol_input.setPlaceholderText(f"Insira um simbolo")
195 self.blank_symbol_input.setMaxLength(1)
196 self.manual_input_blank_symbol_input_layout.addWidget(self.blank_symbol_input)
197
198 self.blank_symbol_input_confirm_button = QPushButton("Confirmar")
199 self.blank_symbol_input_confirm_button.clicked.connect(self.verify_data)
200 self.manual_input_blank_symbol_input_layout.addWidget(self.blank_symbol_input_confirm_button)
201
202 self.manual_input_blank_symbol_input_container.setLayout(self.manual_input_blank_symbol_input_layout)
203 self.manual_input_layout.addWidget(self.manual_input_blank_symbol_input_container)

```

Similar ao símbolo de início com verificação adicional que será detalhada nas funções da classe.

- **Entrada do símbolo de branco:**

```

206 ## Back button
207 self.manual_input_back_button = QPushButton("Voltar")
208 self.manual_input_back_button.clicked.connect(self.show_welcome_screen_callback)
209 self.manual_input_layout.addWidget(self.manual_input_back_button)

```

Criamos um simples botão para retornar a tela de boas-vindas.

Criamos um QLabel para este item, um campo de entrada e um botão de confirmação em um layout horizontal.

- **Bloco if not self.isMaximized:**

```
195         if not self.isMaximized:
196             QTimer.singleShot(0, self.center_on_screen_callback)
```

Centralizamos a aplicação caso esteja minimizada para melhor visualização.

- **Funções dessa classe de entrada manual:**

```
216     def show_main_alphabet_size_input(self):
217         value, type = input_parsing(self.number_of_states_input.text())
218
219         if(type == "float"):
220             self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Nu
221         elif(type == "NaN"):
222             self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! En
223         elif(type == "int"):
224             self.number_of_states = value
225             self.number_of_states_input.setDisabled(True)
226             self.number_of_states_input_confirm_button.setVisible(False)
227             self.manual_input_main_alphabet_size_container.setVisible(True)
```

Definimos uma função show_main_alphabet_size_input para verificar a entrada da quantidade de estados antes de mostrar o campo de entrada do tamanho do alfabeto principal.

Esta função verifica o tipo da entrada e apenas aceita caso for inteiro e retorna uma mensagem de erro ao usuário caso não seja um número ou seja float.

Caso aceito, o valor desta entrada é congelado, não permitindo a modificação pelo usuário.

Esta função utiliza o input_parsing importado do arquivo input_verification.py.

```

229 def show_main_alphabet_values_input(self):
230     value, type = input_parsing(self.main_alphabet_size_input.text())
231
232     if(type == "float"):
233         self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Numero deve ser inteiro")
234     elif(type == "NaN"):
235         self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Entrada deve ser um numero")
236     elif(type == "int"):
237         self.main_alphabet_size = value
238         self.main_alphabet_size_input.setDisabled(True)
239         self.main_alphabet_size_confirm_button.setVisible(False)
240
241         for i in range(self.main_alphabet_size):
242             main_alphabet_values_input = QLineEdit()
243             main_alphabet_values_input.setMaxLength(1)
244             self.main_alphabet_values_input_layout.addWidget(main_alphabet_values_input)
245
246         self.manual_input_main_alphabet_values_container.setVisible(True)
247

```

Definimos uma função `show_main_alphabet_values_input` para verificar a entrada do tamanho do alfabeto principal antes de mostrar o campo de preencher seus símbolos.

Esta função verifica o tipo da entrada e apenas aceita caso for inteiro e retorna uma mensagem de erro ao usuário caso não seja um número ou seja float.

Caso aceite, o valor desta entrada é congelado, não permitindo a modificação pelo usuário.

Após isso ela cria N caixas de entrada em seu layout onde N é o tamanho do alfabeto, ou seja, uma caixa de entrada por símbolo.

Esta função utiliza o `input_parsing` importado do arquivo `input_verification.py`.


```

248 def show_aux_alphabet_size_input(self):
249     self.main_alphabet_list.clear()
250     self.main_alphabet.clear()
251     alphabet_valid = True
252
253     for i in range(self.main_alphabet_size):
254         widget = self.main_alphabet_values_input_layout.itemAt(i).widget()
255         letter = widget.text()
256
257         if(self.main_alphabet.__contains__(letter) or letter == ''):
258             if self.main_alphabet.__contains__(letter):
259                 self.show_alert_box_callback("Alerta!", f"Letra '{letter}' repetida no alfabeto!")
260             else:
261                 self.show_alert_box_callback("Alerta!", f"Alfabeto contem letra vazia!")
262             alphabet_valid = False
263             break
264         elif(letter.lower() == 'x' or letter.lower() == 'l' or letter.lower() == 'r'):
265             self.show_alert_box_callback("Alerta!", f"Alfabeto contem simbolo reservado! ('X', 'L', 'R')")
266             alphabet_valid = False
267             break
268         else:
269             self.main_alphabet.add(letter)
270             self.main_alphabet_list.append(letter)
271
272     if(alphabet_valid):
273         for i in range(self.main_alphabet_size):
274             self.main_alphabet_values_input_layout.itemAt(i).widget().setDisabled(True)
275         self.main_alphabet_values_confirm_button.setVisible(False)
276         self.manual_input_aux_alphabet_size_container.setVisible(True)

```

Definimos uma função `show_aux_alphabet_size_input` para verificar a entrada do alfabeto principal antes de mostrar o campo do tamanho do alfabeto auxiliar.

Esta função verifica se o alfabeto principal não contém símbolos repetidos ou reservados (l,r,x).

Esta função também rejeita entradas vazias.

Caso aceito, o valor desta entrada é congelado, não permitindo a modificação pelo usuário.

```

278 def show_aux_alphabet_values_input(self):
279     value, type = input_parsing(self.aux_alphabet_size_input.text())
280
281     if(type == "float"):
282         self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Nu
283     elif(type == "NaN"):
284         self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! En
285     elif(type == "int"):
286         self.aux_alphabet_size = value
287         self.aux_alphabet_size_input.setDisabled(True)
288         self.aux_alphabet_size_confirm_button.setVisible(False)
289
290         for i in range(self.aux_alphabet_size):
291             aux_alphabet_values_input = QLineEdit()
292             aux_alphabet_values_input.setMaxLength(1)
293             self.aux_alphabet_values_input_layout.addWidget(aux_alphabet_values_input)
294
295         self.manual_input_aux_alphabet_values_container.setVisible(True)

```

```

297 def show_start_symbol_input(self):
298     self.aux_alphabet_list.clear()
299     alphabet_valid = True
300
301     for i in range(self.aux_alphabet_size):
302         widget = self.aux_alphabet_values_input_layout.itemAt(i).widget()
303         letter = widget.text()
304
305         if(self.aux_alphabet.__contains__(letter) or letter == ''):
306             if self.aux_alphabet.__contains__(letter):
307                 self.show_alert_box_callback("Alerta!", f"Letra '{letter}' repetida no a
308             else:
309                 self.show_alert_box_callback("Alerta!", f"Alfabeto contem letra vazia!")
310             alphabet_valid = False
311             self.aux_alphabet.clear()
312             break
313         elif(letter.lower() == 'x' or letter.lower() == 'l' or letter.lower() == 'r'):
314             self.show_alert_box_callback("Alerta!", f"Alfabeto contem simbolo reservado!
315             alphabet_valid = False
316             self.aux_alphabet.clear()
317             break
318         else:
319             self.aux_alphabet.add(letter)
320             self.aux_alphabet_list.append(letter)
321
322     if(alphabet_valid):
323         for i in range(self.aux_alphabet_size):
324             self.aux_alphabet_values_input_layout.itemAt(i).widget().setDisabled(True)
325             self.aux_alphabet_values_confirm_button.setVisible(False)
326             self.manual_input_start_symbol_input_container.setVisible(True)

```


Definimos funções `show_aux_alphabet_values_input` e `show_start_symbol_input` que possuem o comportamento idêntico do alfabeto principal para o alfabeto auxiliar.

```
328 def show_blank_symbol_input(self):
329     symbol_valid = True
330
331     if(self.start_symbol_input.text() == ''):
332         symbol_valid = False
333         self.show_alert_box_callback("Alerta!", f"Simbolo '{self.start_symbol_input.text()}' vazio!")
334     elif(self.main_alphabet.__contains__(self.start_symbol_input.text())):
335         symbol_valid = False
336         self.show_alert_box_callback("Alerta!", f"Simbolo '{self.start_symbol_input.text()}' ja existe!")
337     elif(self.aux_alphabet.__contains__(self.start_symbol_input.text())):
338         symbol_valid = False
339         self.show_alert_box_callback("Alerta!", f"Simbolo '{self.start_symbol_input.text()}' ja existe!")
340     elif(self.start_symbol_input.text().lower() == 'x' or self.start_symbol_input.text().lower() == 'r'):
341         self.show_alert_box_callback("Alerta!", f"Simbolo de inicio nao pode ser o mesmo que os reservados")
342         symbol_valid = False
343
344     if(symbol_valid):
345         self.start_symbol = self.start_symbol_input.text()
346         self.start_symbol_input.setDisabled(True)
347         self.start_symbol_input_confirm_button.setVisible(False)
348         self.manual_input_blank_symbol_input_container.setVisible(True)
```

Definimos uma função `show_blank_symbol_input` para verificar a entrada do símbolo de início antes de mostrar o campo do tamanho do símbolo de branco.

Esta função verifica se o símbolo de início não é igual a qualquer símbolo contido nos alfabetos principal e auxiliar ou reservados (l,r,x).

Esta função também rejeita entradas vazias.

Caso aceito, o valor desta entrada é congelado, não permitindo a modificação pelo usuário.

```

350     def verify_data(self):
351         symbol_valid = True
352
353         if(self.blank_symbol_input.text() == ''):
354             symbol_valid = False
355             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.blank_symbol_input.text()}' não é válido.")
356         elif(self.blank_symbol_input.text() == self.start_symbol):
357             symbol_valid = False
358             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.blank_symbol_input.text()}' não é válido.")
359         elif(self.main_alphabet.__contains__(self.blank_symbol_input.text())):
360             symbol_valid = False
361             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.blank_symbol_input.text()}' não é válido.")
362         elif(self.aux_alphabet.__contains__(self.blank_symbol_input.text())):
363             symbol_valid = False
364             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.blank_symbol_input.text()}' não é válido.")
365         elif(self.blank_symbol_input.text().lower() == 'x' or self.blank_symbol_input.text() == self.blank_symbol):
366             self.show_alert_box_callback("Alerta!", f"Simbolo de branco não é permitido.")
367             symbol_valid = False
368
369         if(symbol_valid):
370             self.blank_symbol = self.blank_symbol_input.text()
371             self.blank_symbol_input.setDisabled(True)
372             self.blank_symbol_input_confirm_button.setVisible(False)
373             self.show_transition_table_callback()

```

Definimos uma função `verify_data` que possui o comportamento similar do símbolo de início para o símbolo de branco.

Esta função também verifica se o símbolo de branco não é igual ao de início.

Caso aceito, todos os valores armazenados serão enviados para uma instância da classe `TransitionTableScreen` e a tela do usuário muda para a tela do preenchimento da tabela de transições.

```

375     def send_values(self):
376         return self.number_of_states, self.main_alphabet_size,

```

Definimos uma função `send_values` para realizar esse papel de exportar os dados armazenados nesta classe.

- **Em resumo, `manual_input_screen.py`:**

Cria um widget que pode ser instanciado e adicionado para o layout em pilha da classe principal. Este widget recebe todos os valores necessários para criar uma tabela de transição, realizando a verificação e validação, e caso aceitos direciona o usuário para a tela de seu preenchimento.

transition_table_screen.py

- Primeiramente os módulos e classes necessários são importados:

```
1 from PyQt6.QtCore import Qt, QTimer
2 from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QGridLayout, QCheckBox, QScrollArea
3 from PyQt6.QtGui import QGuiApplication
4 from transition import Transition
5 from input_verification import verify_test_word_input, replacement_letter_valid, direction_letter_valid
6 from turing_machine_logic import run_turing_machine
```

PyQt6.QtCore, PyQt6.QtWidgets, PyQt6.QtGui: Módulos do PyQt6 para criação de GUI.

QTimer: Uma classe do PyQt6 usada para lidar com a contagem de tempo.

QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QGridLayout, QCheckBox, QScrollArea: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py, as únicas novidades aqui são o GridLayout, que utiliza um sistema de coordenadas para posicionar elementos e a CheckBox que é uma espécie de botão que o usuário pode ativar ou desativar.

QGuiApplication: Classe PyQt6 para aplicativos GUI.

transition: Classe personalizada para armazenar as informações de cada transição de uma forma mais organizada.

input_verification, verify_test_word_input, replacement_letter_valid, direction_letter_valid: Arquivo com funções auxiliares na verificação dos valores de entrada para cada transição, ele é importado pois também é reutilizado na entrada por arquivo.

turing_machine_logic, run_turing_machine: Arquivo com a lógica da execução da Máquina de Turing, ele é importado pois também é reutilizado na entrada manual.

- Criando uma implementação auxiliar de QLineEdit(classe CustomLineEdit):

```

8  class CustomLineEdit(QLineEdit):
9      def __init__(self, transition_table, i, j, parent=None):
10         super().__init__(parent)
11         self.transition_table = transition_table
12         self.i = i
13         self.j = j

```

Definimos uma implementação customizada de um QLineEdit que armazena sua posição na tabela de transição, desta forma podemos localizar cada campo de entrada mais facilmente.

- Criando o widget da tela do preenchimento da tabela de transições(classe TransitionTableScreen):

```

15  class TransitionTableScreen(QWidget):
16      def __init__(self, number_of_states, main_alphabet_size, main_alphabet, main_alphabet_list, aux_a
17         super().__init__()
18
19         self.show_welcome_screen_callback = show_welcome_screen_callback
20         self.center_on_screen_callback = center_on_screen_callback
21         self.show_alert_box_callback = show_alert_box_callback
22
23         self.number_of_states = number_of_states
24         self.main_alphabet_size = main_alphabet_size
25         self.main_alphabet = main_alphabet
26         self.main_alphabet_list = main_alphabet_list
27         self.aux_alphabet_size = aux_alphabet_size
28         self.aux_alphabet = aux_alphabet
29         self.aux_alphabet_list = aux_alphabet_list
30         self.start_symbol = start_symbol
31         self.blank_symbol = blank_symbol
32         self.transition_array = [[Transition.simplified(False) for _ in range((self.main_alphabet_size
33         self.transition_inputs = {}
34         self.initial_state = None
35
36         ## Screen dimensions
37         screen = QApplication.primaryScreen()
38         screen_size = screen.availableSize()
39
40         ## Master Layout
41         self.master_layout = QVBoxLayout()
42
43         self.transition_table_screen_scroll_area = QScrollArea()
44         self.transition_table_screen_scroll_area.setWidgetResizable(True)
45         self.transition_table_screen_scroll_area.setMinimumWidth(int(screen_size.width() * 0.70))
46         self.transition_table_screen_scroll_area.setMinimumHeight(int(screen_size.height() * 0.70))
47
48         self.transition_table_screen_container = QWidget()
49         self.transition_table_screen_layout = QVBoxLayout()
50         self.transition_table_screen_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
51

```

É uma subclasse do módulo widget do PyQt, como se fosse uma div do HTML, extremamente customizável.

Iniciamos a classe passando todos os parâmetros inseridos pelo usuário na classe de entrada manual, bem como algumas funções da classe principal como argumento na forma de “callbacks” para que a mesma função seja invocada. Essas funções são para voltar a tela de boas vindas, criar uma mensagem de alerta e centralizar a aplicação na tela.

Definimos variáveis locais para armazenar todos os parâmetros inseridos pelo usuário.

Definimos um array 2D de transições.

Definimos uma tupla que irá armazenar os QLineEdit customizados para termos fácil acesso às entradas da tabela de transição.

Definimos uma flag de erro para verificar se um estado foi selecionado como inicial pelo usuário. É obrigatório para a Máquina de Turing ter um estado inicial, algumas podem ter até vários, mas esta variação permite apenas 1.

Obtemos as dimensões da tela do usuário.

Criamos o layout principal como um layout vertical (QVBoxLayout) e definimos seu alinhamento.

Criamos uma ScrollArea caso a tabela seja muito grande para criar barras de rolagem e definimos um tamanho mínimo de 70% da altura e largura da tela do usuário.

- **Cabeçalho da tela de preenchimento da tabela de transições:**

```
53  ## Manual Input Screen Header
54  self.transition_table_screen_label = QLabel("<h2>Tabela de transicoes</h2>")
55  self.transition_table_screen_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
56  self.transition_table_screen_layout.addWidget(self.transition_table_screen_label)
```

Criamos um rótulo de título e o adicionamos ao layout principal.

Definimos o alinhamento do rótulo do título para o centro.

- **Grid da tabela de transições:**

```

59     ## Number of states input
60     self.transition_table_content_container = QWidget()
61
62     ## Setting up the transition table with a grid layout
63     self.transition_table_content_layout = QGridLayout()
64     self.transition_table_content_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
65
66     ## Setting up the header for the first column with the state names
67     transition_table_header = QLabel("Estados")
68     self.transition_table_content_layout.addWidget(transition_table_header, 0, 0)
69
70     ## Setting up the alphabet headers sequentially (this is why the lists were important)
71     count = 1
72     for i in range(len(self.main_alphabet_list)):
73         transition_table_header = QLabel(self.main_alphabet_list[i])
74         transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
75         self.transition_table_content_layout.addWidget(transition_table_header, 0, count)
76         count += 1
77
78     for i in range(len(self.aux_alphabet_list)):
79         transition_table_header = QLabel(self.aux_alphabet_list[i])
80         transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
81         self.transition_table_content_layout.addWidget(transition_table_header, 0, count)
82         count += 1
83
84     ## Setting up the headers for the starting and blank symbols
85     transition_table_header = QLabel(self.start_symbol)
86     transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
87     self.transition_table_content_layout.addWidget(transition_table_header, 0, count)
88
89     transition_table_header = QLabel(self.blank_symbol)
90     transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
91     self.transition_table_content_layout.addWidget(transition_table_header, 0, count+1)
92
93     ## Setting up the headers that will allow the user to pick which states are final and the initial state
94     transition_table_header = QLabel("Estado Inicial?")
95     transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
96     self.transition_table_content_layout.addWidget(transition_table_header, 0, count+2)
97
98     transition_table_header = QLabel("Estado Final?")
99     transition_table_header.setAlignment(Qt.AlignmentFlag.AlignCenter)
100    self.transition_table_content_layout.addWidget(transition_table_header, 0, count+3)

```

Definimos os headers da tabela com base nas informações fornecidas pelo usuário na seguinte forma:

[ESTADOS][ALF_P][ALF_AUX][INICIO][BRANCO][INICIAL][FINAL]

Considerando que [ALF_P] e [ALF_AUX] ocupam N e M colunas onde N e M são os seus tamanhos respectivamente.

```

102     ## Filling the table with input fields where applicable
103     for i in range(self.number_of_states):
104         for j in range(self.main_alphabet_size + self.aux_alphabet_size + 2):
105             transition_table_input = CustomLineEdit(self,i,j)
106             transition_table_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
107             transition_table_input.setPlaceholderText(f"{i},{j}")
108             self.transition_table_content_layout.addWidget(transition_table_input, i+1, j+1)
109             self.transition_inputs[(i, j)] = transition_table_input

```

Preenchemos a tabela com os campos de entrada de cada transição para que o usuário possa preenchê-la.

Nota-se que ambos loops for iniciam do 0 e estamos armazenando o i e j em nossa tupla de transition_inputs, isso torna os valores mais acessíveis de utilizar posteriormente, caso contrário teríamos que iniciar do 1,1, tendo em vista que 0,0 da Grid está ocupada pelos headers da tabela.

```

111     ## Filling the first column with state name labels and the final two columns with check boxes
112     count = 1
113     for i in range(self.number_of_states):
114         transition_table_states_label = QLabel(f"S[{count-1}]")
115         transition_table_states_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
116         self.transition_table_content_layout.addWidget(transition_table_states_label, count, 0)
117
118         self.transition_table_content_layout.addWidget(QCheckBox(), count, (self.main_alphabet_size
119         self.transition_table_content_layout.addWidget(QCheckBox(), count, (self.main_alphabet_size
120         count += 1
121
122     self.transition_table_content_container.setLayout(self.transition_table_content_layout)
123     self.transition_table_screen_layout.addWidget(self.transition_table_content_container)

```

Finalizamos a tabela preenchendo a coluna 0 com o nome de cada estado, S[0] até S[N] bem como as duas últimas colunas com CheckBoxes para o usuário marcar quais estados são finais, bem como o inicial.

file_input_screen.py

- Primeiramente os módulos e classes necessários são importados:

```
1 from PyQt6.QtCore import Qt, QTimer
2 from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QTextEdit, QScrollArea, QFileDialog
3 from PyQt6.QtGui import QApplication
4 from instruction import Instruction
5 from instruction_list_processor import run_instructions
```

PyQt6.QtCore, PyQt6.QtWidgets, PyQt6.QtGui: Módulos do PyQt6 para criação de GUI.

QTimer: Uma classe do PyQt6 usada para lidar com a contagem de tempo.

QWidget, QPushButton, QVBoxLayout, QLabel, QTextEdit, QScrollArea, QFileDialog: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py e manual_input_screen.py, as únicas novidades aqui são o QTextEdit que é basicamente um QLineEdit só que é uma caixa de texto ao invés de apenas uma linha de entrada e o QFileDialog

que permite o usuário explorar a estrutura de pastas de seu computador para abrir um arquivo de texto com instruções pré-definidas.

QGuiApplication, QIcon: Classes PyQt6 para aplicativos GUI e gerenciamento de ícones.

instruction: Classe personalizada para armazenar as informações de cada instrução de uma forma mais organizada, como se fosse um struct de outras linguagens.

instruction_list_processor, run_instructions: Arquivo com a lógica da execução do ciclo de instruções, ele é importado pois também é reutilizado na entrada manual.

- Criando o widget da tela de entrada por arquivo(classe FileInputScreen):

```
7 class FileInputScreen(QWidget):
8     def __init__(self, show_welcome_screen_callback, show_alert_box_callback, center_on_screen_callback):
9         super().__init__()
10
11         self.show_welcome_screen_callback = show_welcome_screen_callback
12         self.show_alert_box_callback = show_alert_box_callback
13         self.center_on_screen_callback = center_on_screen_callback
14
15         self.instructions_array = []
16
17         ## Screen dimensions
18         screen = QGuiApplication.primaryScreen()
19         screen_size = screen.availableSize()
20
21
22         ## Master Layout
23         self.master_layout = QVBoxLayout()
24
25         self.file_input_scroll_area = QScrollArea()
26         self.file_input_scroll_area.setWidgetResizable(True)
27         self.file_input_scroll_area.setMinimumWidth(int(screen_size.width() * 0.70))
28         self.file_input_scroll_area.setMinimumHeight(int(screen_size.height() * 0.70))
29
30         self.file_input_container = QWidget()
31         self.file_input_layout = QVBoxLayout()
32         self.file_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

```

35     ## Manual Input Screen Header
36     self.file_input_label = QLabel("<h2>Entrada de valores por arquivo</h2>")
37     self.file_input_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
38     self.file_input_layout.addWidget(self.file_input_label)
39
40
41     ## Text box input
42     self.file_input_text_box = QTextEdit()
43     self.file_input_text_box.setPlaceholderText(f"Cole o texto da tabela de transicao aqui...")
44     self.file_input_text_box.setMinimumHeight(int(screen_size.height() * 0.50))
45     self.file_input_layout.addWidget(self.file_input_text_box)
46
47
48     ## Confirm button
49     self.file_input_confirm_button = QPushButton("Confirmar")
50     self.file_input_confirm_button.clicked.connect(self.get_data)
51     self.file_input_layout.addWidget(self.file_input_confirm_button)
52
53     ## Open file button
54     self.file_input_open_file_button = QPushButton("Abrir arquivo")
55     self.file_input_open_file_button.clicked.connect(self.open_file_button)
56     self.file_input_layout.addWidget(self.file_input_open_file_button)
57
58     ## Back button
59     self.file_input_back_button = QPushButton("Voltar")
60     self.file_input_back_button.clicked.connect(lambda: self.show_welcome_screen_callback())
61
62     self.file_input_layout.addWidget(self.file_input_back_button)

```

```

65     ## Result scroll area
66     self.instructions_result_scroll_area = QScrollArea()
67     self.instructions_result_scroll_area.setVisible(False)
68     self.instructions_result_scroll_area.setWidgetResizable(True)
69     self.instructions_result_scroll_area.setMinimumHeight(int(screen_size.height() * 0.50))
70
71     self.instructions_result_layout_container = QWidget()
72     self.instructions_result_layout = QVBoxLayout()
73     self.instructions_result_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
74
75     ## MBR result
76     self.mbr_result_label = QLabel("<h2>MBR Final</h2>")
77     self.mbr_result_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
78     self.instructions_result_layout.addWidget(self.mbr_result_label)
79     self.instructions_result_layout.addSpacing(10)
80
81     self.mbr_result_value = QLabel()
82     self.mbr_result_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
83     self.instructions_result_layout.addWidget(self.mbr_result_value)
84     self.instructions_result_layout.addSpacing(20)

```

```

87     ## Instructions log output
88     self.instructions_log_output_label = QLabel("<h2>Instrucoes realizadas</h2>")
89     self.instructions_log_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
90     self.instructions_result_layout.addWidget(self.instructions_log_output_label)
91     self.instructions_result_layout.addSpacing(10)
92
93     self.instructions_log_output_value = QLabel()
94     self.instructions_log_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
95     self.instructions_result_layout.addWidget(self.instructions_log_output_value)
96     self.instructions_result_layout.addSpacing(20)
97
98
99     ## Tape result output
100    self.tape_result_output_label = QLabel("<h2>Fita final</h2>")
101    self.tape_result_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
102    self.instructions_result_layout.addWidget(self.tape_result_output_label)
103    self.instructions_result_layout.addSpacing(10)
104
105    self.tape_result_output_value = QLabel()
106    self.tape_result_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
107    self.instructions_result_layout.addWidget(self.tape_result_output_value)
108
109
110    self.instructions_result_layout_container.setLayout(self.instructions_result_layout)
111    self.instructions_result_scroll_area.setWidget(self.instructions_result_layout_container)
112    self.file_input_layout.addWidget(self.instructions_result_scroll_area)

```

```

115    self.file_input_container.setLayout(self.file_input_layout)
116    self.file_input_scroll_area.setWidget(self.file_input_container)
117
118    self.master_layout.addWidget(self.file_input_scroll_area)
119
120    self.setLayout(self.master_layout)
121
122    if not self.isMaximized:
123        QTimer.singleShot(0, self.center_on_screen_callback)

```

Estrutura extremamente similar a tela de resultado da classe de entrada manual, porém com a caixa de texto no começo do layout vertical para que o usuário insira várias linhas de texto rapidamente, agilizando a velocidade de teste de instruções diferentes.

Além disso, temos o botão de navegação da estrutura de pastas e arquivos do computador do usuário.

- **Funções dessa classe de entrada por arquivo:**

```

125     def open_file_button(self):
126         file_dialog = QFileDialog()
127         file_dialog.setNameFilter("Sequencias de instrucoes (*.txt)")
128         file_dialog.setFileMode(QFileDialog.FileMode.ExistingFile)
129
130         if file_dialog.exec() == QFileDialog.DialogCode.Accepted:
131             selected_file = file_dialog.selectedFiles()[0]
132
133             with open(selected_file, "r") as file:
134                 raw_text = file.read()
135
136             self.file_input_text_box.setText(raw_text)
137
138             lines = raw_text.split('\n')
139             self.parse_instructions(lines)

```

Definimos uma função `open_file_button` que utiliza um `FileDialog` para mostrar a tela de explorar os arquivos do computador do usuário com o filtro para arquivos do formato `.txt`. O conteúdo deste arquivo será escrito na caixa `TextEdit` da interface e pode ser editada pelo usuário a qualquer momento.

```

141     def get_data(self):
142         raw_text = self.file_input_text_box.toPlainText()
143         lines = raw_text.split('\n')
144         self.parse_instructions(lines)

```

Definimos uma função `get_data` que recebe o texto da `QTextEdit` para variáveis mais “amigáveis” ao processamento de dados, dividindo o conteúdo em uma array de strings, onde cada índice corresponde a uma linha de texto.

```

146     def parse_instructions(self, lines):
147         self.instructions_array.clear()
148         error_triggered = False
149

```

```

150     for i in range(len(lines)):
151         line_value = lines[i].split(",")
152         num_a = 0
153         num_b = 0
154
155         if(line_value[0] not in ["000001", "000010", "000011", "000100", "000101", "000110", "000111", "001000", "001001", "001010", "001011", "001111", "001100"]):
156             self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui codigo valido!")
157             error_triggered = True
158             break
159
160         if(line_value[0] in ["000001", "000010", "000011", "000100", "000101", "000110", "000111", "001000", "001001", "001111"]):
161             try:
162                 num_a = int(line_value[1])
163             except ValueError:
164                 self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui valor valido!")
165                 error_triggered = True
166                 break
167
168             if(line_value[0] == "000010"):
169                 try:
170                     num_b = float(line_value[2])
171                 except ValueError:
172                     self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui valor valido!")
173                     error_triggered = True
174                     break
175
176             self.instructions_array.append(Instruction(line_value[0], num_a, num_b))
177         else:
178             self.instructions_array.append(Instruction(line_value[0], num_a))
179     else:
180         self.instructions_array.append(Instruction(line_value[0]))

```

```

182     if(not error_triggered):
183         mbr, log, tape_display, out_of_bounds_error = run_instructions(self.instructions_array)
184
185         if(out_of_bounds_error):
186             self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes parada antes de sua conclusao, jump realizado para linha inexistente!")
187
188         self.mbr_result_value.setText(str(mbr))
189         self.tape_result_output_value.setText(''.join(tape_display))
190
191         if(len(log) == 0):
192             self.instructions_log_output_value.setText("VAZIO")
193         else:
194             self.instructions_log_output_value.setText(''.join(log))
195
196         self.instructions_result_scroll_area.setVisible(True)
197     else:
198         self.instructions_result_scroll_area.setVisible(False)

```

Definimos uma função `parse_instructions` para verificar se as instruções foram inseridas com a formatação correta, "[CODE],[VALUE_A],[VALUE_B]" em uma linha separada, dividindo cada valor com uma vírgula, com a mesma verificação de número, inteiro ou float da entrada manual bem como uma verificação adicional se o código em questão existe ou não. Caso nenhum problema seja encontrado, as instruções são executadas com a lógica importada da função `run_instructions` do arquivo `instruction_list_processor.py` e o log resultante é mostrado ao usuário.

- **Em resumo, `file_input_screen.py`:**

Cria um widget que pode ser instanciado e adicionado para o layout em pilha da classe principal. Este widget é bem similar ao `manual_input_screen.py` porém com algumas nuances para lidar com a entrada de um grande string de

texto ao invés de uma entrada mais controlada como a manual. Nota-se que ambas as classes utilizam a mesma lógica importada de outro arquivo.

instruction.py

- Criando a classe Instruction:

```
1  class Instruction:
2      def __init__(self, code, value_a=None, value_b=None):
3          self.code = code
4          self.value_a = value_a
5          self.value_b = value_b
```

Esta classe é extremamente simples, quase como um struct de outras linguagens, definindo alguns valores a serem armazenados para uma conveniência um pouco maior do desenvolvimento da aplicação.

Em síntese, o código da instrução é sempre armazenado e os valores a e b são opcionais, tendo em vista que dependem do próprio código de instrução. Desta forma a lógica para a adição de uma instrução cuida desta verificação e inserção correta na array de sequência de instruções.

instruction_list_processor.py

- Primeiramente os módulos e classes necessários são importados:

```
1 from math import sqrt
```

math com a função sqrt: Realiza a operação de raiz quadrada da instrução "001010".

- Criando a função run_instructions:

Esta função é a espinha dorsal de toda a aplicação e contém a lógica que executa a sequência de instruções.

```
3 def run_instructions(instructions_array):
4     greatest_pos_value = 0
5
6     for i in range(len(instructions_array)):
7         if(instructions_array[i].code in ["000001", "000010", "000011", "000100", "000101", "000110", "001111"]):
8             if(instructions_array[i].value_a > greatest_pos_value):
9                 greatest_pos_value = instructions_array[i].value_a
```

De início, ela verifica qual é a maior posição de memória para retornar uma fita de endereços de memória mais “verídica”.

```
11     mbr = 0
12     tape = [0.0] * (greatest_pos_value + 1)
13     log = []
14     tape_display = []
15     current_instruction = 0
16
17     out_of_bounds_error = False
```

Todas as variáveis auxiliares são inicializadas em zero, vazias ou falsas.

```
19 while(current_instruction < len(instructions_array)):
20     current_instruction_code = instructions_array[current_instruction].code
```

Um loop while é utilizado para executar toda a sequência de instruções devido a possibilidade de existir JUMPs que modificam a ordem de execução.

```
22 match current_instruction_code:
23     case "000001":
24         mbr = tape[instructions_array[current_instruction].value_a]
25         log.append(f"Operacao {instructions_array[current_instruction].code} realizada\n")
26         log.append(f"MBR recebeu o valor {tape[instructions_array[current_instruction].value_a]:.1f} da posicao {instructions_array[current_instruction].value_a}\n")
27         log.append(f"MBR atual e {mbr:.1f}\n\n")
28     case "000010":
29         tape[instructions_array[current_instruction].value_a] = instructions_array[current_instruction].value_b
30         log.append(f"Operacao {instructions_array[current_instruction].code} realizada\n")
31         log.append(f"A posicao {instructions_array[current_instruction].value_a} recebeu o valor {instructions_array[current_instruction].value_b:.1f}\n\n")
32     case "000011":
```

Cada iteração cai em um switch case, ou match case em Python que verifica o código da instrução e executa as operações apropriadas, bem como armazena o que aconteceu no log.

```
96     if(current_instruction < 0 or current_instruction > len(instructions_array)):
97         out_of_bounds_error = True
98         break
```

Como uma estrutura de array é utilizada, existe a possibilidade de um JUMP ser executado para um índice fora de seu escopo, desta forma a booleana out_of_bounds_error serve como um gatilho para interromper a execução e informar o usuário caso isso aconteça.

```
100     for i in range(len(tape)):
101         tape_display.append(str(str(i) + ": " + str(tape[i]) + " - "))
102         if(i % 15 == 0):
103             tape_display.append("\n")
104
105     if(out_of_bounds_error):
106         log.append("Operacao parada neste ponto. Linha do jump inexistente\n")
107         return mbr, log, tape_display, True
108
109     return mbr, log, tape_display, False
```

Por fim, o MBR, a fita e o log são retornados pela função, dados que são utilizados nas classes `ManualInputScreen` e `FileInputScreen` para serem mostrados na tela da aplicação.