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

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:**

```
import os
import sys
from PyQt6.QtCore import Qt, QEvent
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QStackedLayout, QMessageBox
from PyQt6.QtGui import QGuiApplication, QIcon
from manual_input_screen import ManualInputScreen
from file_input_screen import FileInputScreen
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**, **QHBoxLayout**, **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.

QGuiApplication, **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):**

```python
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.input_widget = None
        self.transition_table_widget = None

        self.setWindowTitle("Simulador Maquina de Turing")
        self.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        ## User screen's dimenions
        self.screen = QGuiApplication.primaryScreen()
        self.screen_size = self.screen.availableSize()

        ## Master Layout
        self.main_layout = QVBoxLayout(self.central_widget)
        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:**

```python
        ## Application Header
        title_label = QLabel("<h1>Simulador Maquina de Turing</h1>")
        title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        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)
```

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.

```
    def center_on_screen(self):
        screen_geometry = QApplication.primaryScreen().availableGeometry()

        center_x = int((screen_geometry.width() - self.width()) / 2)
        center_y = int((screen_geometry.height() - self.height()) / 2.5)

        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.

```
def show_alert_box(self, title, text):
    alert=QMessageBox()
    alert.setIcon(QMessageBox.Icon.Information)
    alert.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
    alert.setWindowTitle(title)
    alert.setText(text)
    alert.setStandardButtons(QMessageBox.StandardButton.Ok)
    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.

```
    def show_welcome_screen(self):
        self.stacked_layout.setCurrentWidget(self.welcome_layout_container)
        if not self.isMaximized:
            self.center_on_screen()
        self.destroy_input_widget()
        self.destroy_transition_table_widget()

    def show_manual_input_screen(self):
        self.input_widget = ManualInputScreen(self.show_welcome_screen, self.
        self.stacked_layout.addWidget(self.input_widget)
        self.stacked_layout.setCurrentWidget(self.input_widget)

    def show_file_input_screen(self):
        self.input_widget = FileInputScreen(self.show_welcome_screen, self.ce
        self.stacked_layout.addWidget(self.input_widget)
        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:**

```
from PyQt6.QtCore import Qt, QTimer
from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea
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):**

```
class ManualInputScreen(QWidget):
    def __init__(self, show_welcome_screen_callback, center_on_screen_callbac
        super().__init__()

        self.show_welcome_screen_callback = show_welcome_screen_callback
        self.center_on_screen_callback = center_on_screen_callback
        self.show_alert_box_callback = show_alert_box_callback
        self.show_transition_table_callback = show_transition_table_callback

        self.number_of_states = 0
        self.main_alphabet_size = 0
        self.main_alphabet = set()
        self.main_alphabet_list = []
        self.aux_alphabet_size = 0
        self.aux_alphabet = set()
        self.aux_alphabet_list = []
        self.start_symbol = ''
        self.blank_symbol = ''
```

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

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)
```

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
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:**

```
        ## Main alphabet values input
        self.manual_input_main_alphabet_values_container = QWidget()
        self.manual_input_main_alphabet_values_container.setVisible(False)
        self.manual_input_main_alphabet_values_container.setMaximumHeight(80)
        self.manual_input_main_alphabet_values_layout = QHBoxLayout()

        self.main_alphabet_values_label = QLabel("Alfabeto principal: ")
        self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_label)

        self.main_alphabet_values_input_scroll_area = QScrollArea()

        self.main_alphabet_values_input_container = QWidget()
        self.main_alphabet_values_input_layout = QHBoxLayout()

        ### show_main_alphabet_values_input has the logic for adding QLineEdit widgets

        self.main_alphabet_values_input_container.setLayout(self.main_alphabet_values_input_layout)

        self.main_alphabet_values_input_scroll_area.setWidget(self.main_alphabet_values_input_container)
        self.main_alphabet_values_input_scroll_area.setWidgetResizable(True)
        self.main_alphabet_values_input_scroll_area.setMaximumHeight(80)

        self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_input_scroll_area)

        self.main_alphabet_values_confirm_button = QPushButton("Confirmar")
        self.main_alphabet_values_confirm_button.clicked.connect(self.show_aux_alphabet_size_input)
        self.manual_input_main_alphabet_values_layout.addWidget(self.main_alphabet_values_confirm_button)

        self.manual_input_main_alphabet_values_container.setLayout(self.manual_input_main_alphabet_values_layout)
        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:**

```
        ## Aux alphabet size input
        self.manual_input_aux_alphabet_size_container = QWidget()
        self.manual_input_aux_alphabet_size_container.setVisible(False)
        self.manual_input_aux_alphabet_size_container.setMaximumHeight(60)
        self.manual_input_aux_alphabet_size_layout = QHBoxLayout()

        self.aux_alphabet_size_label = QLabel("Tamanho do alfabeto auxiliar: ")
        self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_label)

        self.aux_alphabet_size_input = QLineEdit()
        self.aux_alphabet_size_input.setPlaceholderText(f"Insira um valor inteiro")
        self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_input)
```

```
121
122         self.aux_alphabet_size_confirm_button = QPushButton("Confirmar")
123         self.aux_alphabet_size_confirm_button.clicked.connect(self.show_aux_alphabet_values_input)
124         self.manual_input_aux_alphabet_size_layout.addWidget(self.aux_alphabet_size_confirm_button)
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("Simbolo 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("Simbolo 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.

```python
    def show_main_alphabet_values_input(self):
        value, type = input_parsing(self.main_alphabet_size_input.text())

        if(type == "float"):
            self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Numero deve ser inteiro")
        elif(type == "NaN"):
            self.show_alert_box_callback("Alerta!", f"Valor inserido '{value}' invalido! Entrada deve ser um numero")
        elif(type == "int"):
            self.main_alphabet_size = value
            self.main_alphabet_size_input.setDisabled(True)
            self.main_alphabet_size_confirm_button.setVisible(False)

            for i in range(self.main_alphabet_size):
                main_alphabet_values_input = QLineEdit()
                main_alphabet_values_input.setMaxLength(1)
                self.main_alphabet_values_input_layout.addWidget(main_alphabet_values_input)

            self.manual_input_main_alphabet_values_container.setVisible(True)
```

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 aceito, 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 exi
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 exi
340         elif(self.start_symbol_input.text().lower() == 'x' or self.start_symbol_input.text().lower() ==
341             self.show_alert_box_callback("Alerta!", f"Simbolo de inicio nao pode ser o mesmo que os res
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.
356         elif(self.blank_symbol_input.text() == self.start_symbol):
357             symbol_valid = False
358             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.
359         elif(self.main_alphabet.__contains__(self.blank_symbol_input.
360             symbol_valid = False
361             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.
362         elif(self.aux_alphabet.__contains__(self.blank_symbol_input.te
363             symbol_valid = False
364             self.show_alert_box_callback("Alerta!", f"Simbolo '{self.
365         elif(self.blank_symbol_input.text().lower() == 'x' or self.bla
366             self.show_alert_box_callback("Alerta!", f"Simbolo de bran
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 moda 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:**

```
from PyQt6.QtCore import Qt, QTimer
from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QGridLayout, QCheckBox, QScrollArea
from PyQt6.QtGui import QGuiApplication
from transition import Transition
from input_verification import verify_test_word_input, replacement_letter_valid, direction_letter_valid
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   ass 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  ass 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_siz
33        self.transition_inputs = {}
34        self.initial_state = None
35
36        ## Screen dimensions
37        screen = QGuiApplication.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 QLineEdits 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 transion 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
```

```
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.

```
    ## Filling the table with input fields where applicable
    for i in range(self.number_of_states):
        for j in range(self.main_alphabet_size + self.aux_alphabet_size + 2):
            transition_table_input = CustomLineEdit(self,i,j)
            transition_table_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
            transition_table_input.setPlaceholderText(f"{i},{j}")
            self.transition_table_content_layout.addWidget(transition_table_input, i+1, j+1)
            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.

```
    ## Filling the first column with state name labels and the final two columns with check boxes
    count = 1
    for i in range(self.number_of_states):
        transition_table_states_label = QLabel(f"S[{count-1}]")
        transition_table_states_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.transition_table_content_layout.addWidget(transition_table_states_label, count, 0)

        self.transition_table_content_layout.addWidget(QCheckBox(), count, (self.main_alphabet_size
        self.transition_table_content_layout.addWidget(QCheckBox(), count, (self.main_alphabet_size
        count += 1

    self.transition_table_content_container.setLayout(self.transition_table_content_layout)
    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.

- **Botões de ações:**

```
    ## Confirm button
    self.transition_table_confirm_button = QPushButton("Confirmar")
    self.transition_table_confirm_button.clicked.connect(self.verify_table_integrity)
    self.transition_table_screen_layout.addWidget(self.transition_table_confirm_button)


    ## Back button
    self.transition_table_back_button = QPushButton("Voltar")
    self.transition_table_back_button.clicked.connect(self.show_welcome_screen_callback)
    self.transition_table_screen_layout.addWidget(self.transition_table_back_button)
```

Definimos botões de confirmação da entrada da tabela de transições e de voltar para retornar a tela de boas vindas abaixo do Grid.

- **Entrada da palavra a ser testada:**

```
138     ## Test word input
139     self.test_word_input_container = QWidget()
140     self.test_word_input_container.setVisible(False)
141     self.test_word_input_layout = QHBoxLayout()
142 
143     self.test_word_input_label = QLabel("Palavra: ")
144     self.test_word_input_layout.addWidget(self.test_word_input_label)
145 
146     self.test_word_input_input = QLineEdit()
147     self.test_word_input_input.setPlaceholderText(f"Insira uma palavra para ser testada")
148     self.test_word_input_layout.addWidget(self.test_word_input_input)
149 
150     self.test_word_input_confirm_button = QPushButton("Confirmar")
151     self.test_word_input_confirm_button.clicked.connect(self.check_word_and_run_machine)
152     self.test_word_input_layout.addWidget(self.test_word_input_confirm_button)
153 
154     self.test_word_input_container.setLayout(self.test_word_input_layout)
155     self.transition_table_screen_layout.addWidget(self.test_word_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.

- **Área de saída ou resultado:**

```
157         ## Result scroll area
158         self.tape_result_scroll_area = QScrollArea()
159         self.tape_result_scroll_area.setVisible(False)
160         self.tape_result_scroll_area.setWidgetResizable(True)
161         self.tape_result_scroll_area.setMinimumHeight(int(screen_size.height() * 0.50))
162 
163         self.tape_result_layout_container = QWidget()
164         self.tape_result_layout = QVBoxLayout()
165         self.tape_result_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Criamos uma ScrollArea para mostrar o resultado do teste da palavra com a Máquina de Turing inserida.

```
        ## Test word tape result
        self.tape_result_label = QLabel("<h3>Resultado da fita</h3>")
        self.tape_result_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tape_result_layout.addWidget(self.tape_result_label)
        self.tape_result_layout.addSpacing(10)

        self.tape_result_value = QLabel()
        self.tape_result_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tape_result_layout.addWidget(self.tape_result_value)
        self.tape_result_layout.addSpacing(20)


        ## Test word output
        self.test_word_output_label = QLabel("<h3>Transicoes realizadas</h3>")
        self.test_word_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tape_result_layout.addWidget(self.test_word_output_label)
        self.tape_result_layout.addSpacing(10)

        self.test_word_output_value = QLabel()
        self.test_word_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tape_result_layout.addWidget(self.test_word_output_value)


        self.tape_result_layout_container.setLayout(self.tape_result_layout)
        self.tape_result_scroll_area.setWidget(self.tape_result_layout_container)
        self.transition_table_screen_layout.addWidget(self.tape_result_scroll_area)
```

Inicializamos os campos que vão ser preenchidos com as informações do resultado, a fita final e o log das transições realizadas.

- **Finalizando o layout:**

```
        self.tape_result_layout_container.setLayout(self.tape_result_layout)
        self.tape_result_scroll_area.setWidget(self.tape_result_layout_container)
        self.transition_table_screen_layout.addWidget(self.tape_result_scroll_area)


        self.transition_table_screen_container.setLayout(self.transition_table_screen_layout)
        self.transition_table_screen_scroll_area.setWidget(self.transition_table_screen_container)

        self.master_layout.addWidget(self.transition_table_screen_scroll_area)

        self.setLayout(self.master_layout)

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

Definimos o layout principal da classe com a hierarquia dos contêineres, bem como utilizamos a mesma função para centralizar a janela caso não esteja maximizada.

- **Funções dessa classe:**

```
205     def find_transition_input(self, i, j):
206         return self.transition_inputs.get((i, j))
```

Definimos uma simples função auxiliar find_transition_input para buscarmos o valor de entrada de uma transição específica do layout.

```
208     def verify_table_integrity(self):
209         encountered_error = False
210         starting_state_exists = False
211         final_state_exists = False
```

Definimos uma longa e complexa função verify_table_integrity para validar as transições inseridas pelo usuário.

Definimos algumas flags locais de erro, uma genérica e duas pertinentes ao estado inicial e final.

```
for i in range(self.number_of_states):
    is_final = False

    if((self.transition_table_content_layout.itemAtPosition((i+1), (self.main_alphabet_size + se
        if(not starting_state_exists):
            starting_state_exists = True
            self.initial_state = i
        else:
            self.show_alert_box_callback("Alerta!", f"Apenas um estado inicial e permitido!")
            encountered_error = True
            self.initial_state = None
            break

    if((self.transition_table_content_layout.itemAtPosition((i+1), (self.main_alphabet_size + se
        final_state_exists = True
        is_final = True
```

Iniciamos um loop for para verificar todos os estados.

Verificamos se é um estado inicial, caso positivo mudamos a flag de existência para true e armazenamos o seu valor. Todavia se outro estado verificado for inicial um gatilho de erro é ativado, pois apenas um estado inicial é permitido.

Uma verificação similar porém mais simples é feita para o estado final, tendo em vista que mais de um deles é permitido.

```
for j in range(self.main_alphabet_size + self.aux_alphabet_size + 2):
    if(self.find_transition_input(i, j).text() == ''):
        self.show_alert_box_callback("Alerta!", f"Transicao '{i,j}' nao foi preenchida!")
        encountered_error = True
        break
    elif(self.find_transition_input(i, j).text().lower() == 'x'):
        self.transition_array[i][j] = Transition.simplified(is_final)
    else:
```

Iniciamos um for aninhado para verificar as colunas, ou seja, cada transição individual.

Primeiramente realizamos duas verificações triviais, se a transição não foi preenchida que aciona um gatilho de erro e se a verificação foi marcada como vazia, que utiliza um construtor alternativo naquele índice da array de transições.

```
else:
    parameters = []
    for parameter in self.find_transition_input(i, j).text():
        parameters.append(parameter)

    if(len(parameters)<3):
        self.show_alert_box_callback("Alerta!", f"Transicao '{i,j}' nao foi pree
        encountered_error = True
        break

    number_digits = 0
    number_string = ""
    for char in parameters:
        if char.isdigit():
            number_digits += 1
            number_string += char
        else:
            break

    if(number_digits==0):
        self.show_alert_box_callback("Alerta!", f"Notacao invalida na transicao
        encountered_error = True
        break

    next_state = int(number_string)
```

Após isso temos a verificação de uma transição normal.

Transformamos o texto da caixa em uma array de char chamado parâmetros.

Se parâmetros for menor que 3 um gatilho de erro é ativado, pois é o valor mínimo de parâmetros possíveis o que indica que o campo não foi preenchido corretamente.

Após isso temos a verificação dos dígitos, pois cada transição começa com um inteiro representando um estado futuro. Como estamos lidando com uma array de chars, são verificados dígito por dígito, isso é relevante pois em tabelas maiores uma transição pode ter um estado futuro com 2, 3 ou mais dígitos, o que deve ser considerado para os valores corretos serem armazenados.

Caso o primeiro valor não seja um número, um gatilho de erro é ativado afirmando que aquela transição não foi preenchida corretamente.

```
if(next_state>(self.number_of_states-1)):
    self.show_alert_box_callback("Alerta!", 
    encountered_error = True
    break
```

Verificamos se o estado futuro inserido de fato existe, caso contrário outro gatilho é ativado. Não verificamos se é menor que 0 pois é uma condição trivial, tendo em vista que o negativo como primeiro dígito ativaria o gatilho de erro anterior.

```
if(replacement_letter_valid(parameters[number_digits], sel
        replacement_letter = parameters[number_digits]
else:
    self.show_alert_box_callback("Alerta!", f"Notacao inva
    encountered_error = True
    break

if(direction_letter_valid(parameters[number_digits+1])):
    direction = parameters[number_digits+1]
else:
    self.show_alert_box_callback("Alerta!", f"Notacao inva
    encountered_error = True
    break
```

Utilizamos as funções auxiliares importadas para verificar se os símbolos inseridos são válidos ou não, ou seja, se estão nos alfabetos inseridos e no caso da direção se correspondem com “L” ou “R”.

Ademais, nota-se que utilizamos a variável local number_digits para selecionar os valores nas posições corretas baseado em quantos dígitos o primeiro número inteiro continha.

```
if(len(parameters)>number_digits+2):
    self.show_alert_box_callback("Alerta!", f"Notacao invalida na transicao '{i,j}'! Apenas 3 parametros sao aceitos, [NUMERO]
    encountered_error = True
    break

current_state = i
current_letter = (self.transition_table_content_layout.itemAtPosition(0, (j+1))).widget().text()

self.transition_array[i][j] = Transition(current_state, next_state, current_letter, replacement_letter, direction, is_final)
```

Verificamos se o usuário não inseriu parâmetros além dos necessários.

Criamos uma nova transição com as informações validadas no índice i,j da tabela.

```
292             if(encountered_error):
293                 break
294
295         if(not encountered_error):
296             if(not starting_state_exists):
297                 self.show_alert_box_callback("Alerta!", f"Tabela de transicao sem estado inicial!")
298                 encountered_error = True
299
300             if(not final_state_exists):
301                 self.show_alert_box_callback("Alerta!", f"Tabela de transicao sem estado final!")
302                 encountered_error = True
303
304             if(not encountered_error):
305                 self.transition_table_confirm_button.setVisible(False)
306                 for i in range(self.number_of_states):
307                     self.transition_table_content_layout.itemAtPosition((i+1), (self.main_alphabet_siz
308                     self.transition_table_content_layout.itemAtPosition((i+1), (self.main_alphabet_siz
309                     for j in range(self.main_alphabet_size + self.aux_alphabet_size + 2):
310                         self.find_transition_input(i, j).setDisabled(True)
311
312                 self.test_word_input_container.setVisible(True)
```

Verificamos se a tabela possui algum estado inicial e final.

Caso a tabela seja aceita, ela não poderá mais ser editada e o campo de inserção de uma palavra para teste se tornará visível.

```
def check_word_and_run_machine(self):
    self.tape = verify_test_word_input(self.test_word_input_input.text(), self.main_alphabet,

    if(self.tape is not None):
        result_tape, result_text, transition_log = run_turing_machine(self.transition_array, s
        self.tape_result_scroll_area.setVisible(True)
        self.tape_result_value.setText(''.join(result_tape) + "\n\n" + result_text)
        self.test_word_output_value.setText(''.join(transition_log))
    else:
        self.show_alert_box_callback("Alerta!", f"Palavra invalida! Simbolos inseridos que nao
```

Por fim, definimos uma função check_word_and_run_machine para verificar a palavra inserida, se não foi vazia ou contém caracteres que não estão nos alfabetos e transformá-la na fita, com o símbolo de início no índice 0 e símbolos de branco após a palavra.

Essa fita é utilizada para chamar a função que executa a Máquina de Turing com todos os parâmetros estipulados pelo usuário. O retorno da função é mostrado para o usuário na forma da fita final e o log das transições.

- **Em resumo, transition_table_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 inseridos pelo usuário para criar uma tabela de transição, e mostra ela vazia para que o usuário preencha todas as transições da forma que desejar. Depois disso, o widget verifica se a tabela é válida ou não, avisando o usuário com mensagens de erro detalhadas e específicas sobre o que houve de errado e caso positivo permite o teste de palavras, mostrando o resultado na tela.

file_input_screen.py

Esta classe apresenta uma lógica extremamente similar às da entrada manual e tabela de transições, com pequenas peculiaridades para lidar com uma array de linhas e vírgulas para separar as instruções.

Desta forma, como esta documentação já está relativamente extensa, esta parte será omitida para evitar muita repetição.

O formato de inserção por arquivo é:

[N_ESTADOS]
[ALF_P]
[ALF_AUX]
[SIMBOLO_INICIO]
[SIMBOLO_BRANCO]
[ESTADO_INICIAL]
[PALAVRA_DE_TESTE]
[TABELA DE TRANSICAO]
.

Onde a tabela de transição tem o seguinte formato:

[ESTADO_FINAL],[TRANSICAO_0_0],[TRANSICAO_0_N]
[ESTADO_FINAL],[TRANSICAO_N_0],[TRANSICAO_N_N]

Onde [ESTADO_FINAL] deve ser T ou F, e as transições devem corresponder uma tabela com as colunas dos símbolos na mesma ordem de inserção do alfabeto principal, auxiliar e símbolos início e branco.

Um exemplo de uma entrada válida:

6
a,b
A,B
<
>
0

aabb

F,1AR,X,0AR,3BR,X,X

F,1aR,2BL,X,1BR,X,X

F,2aL,X,2AL,2BL,0<R,X

F,X,X,X,3BR,X,4>L

F,X,X,4AL,4BL,5<R,X

T,X,X,X,X,X,X

transition.py

- **Criando a classe Instruction**:

```python
class Transition:
    def __init__(self, current_state, next_state, current_letter, replacement_letter, direction, is_final):
        self.current_state = current_state
        self.next_state = next_state
        self.current_letter = current_letter
        self.replacement_letter = replacement_letter
        self.direction = direction
        self.is_final = is_final

    @classmethod
    def simplified(cls, is_final):
        return cls(-1, None, None, None, None, is_final)
```

Classe simples para armazenar todos os parâmetros de uma transição de uma maneira mais fácil de trabalhar. Também possui um construtor alternativo caso a transição seja vazia.

input_verification.py

- **Arquivo com funções auxiliares:**

```python
def input_parsing(value):
    try:
        parsed_value = float(value)
        if parsed_value.is_integer():
            return int(parsed_value), "int"
        else:
            return parsed_value, "float"
    except ValueError:
        return value, "NaN"
```

Definimos a função input_parsing para verificar se uma entrada é inteiro, float ou não é um número.

```python
def verify_test_word_input(word, main_alphabet, aux_alphabet, start_symbol, blank_symbol):
    is_word_valid = True

    word = start_symbol + word
    while(len(word)<50):
        word = word + blank_symbol

    tape = []
    for letter in word:
        if(replacement_letter_valid(letter, main_alphabet, aux_alphabet, start_symbol, blank_symbol)):
            tape.append(letter)
        else:
            is_word_valid = False
            break

    if(is_word_valid):
        return tape
    else:
        return None
```

Definimos a função verify_test_word_input para transformar uma entrada de palavra de teste em uma fita, bem como verificar se não contém símbolos inválidos.

```python
def replacement_letter_valid(letter, main_alphabet, aux_alphabet, start_symbol, blank_symbol):
    if(main_alphabet.__contains__(letter) or aux_alphabet.__contains__(letter) or letter == start_symbol or letter == blank_symbol):
        return True
    else:
        return False

def direction_letter_valid(letter):
    if(letter.lower() == 'l' or letter.lower() == 'r'):
        return True
    else:
        return False
```

Definimos as funções auxiliares replacement_letter_valid e direction_letter_valid para verificar se um símbolo existe nos alfabetos inseridos e se uma direção condiz com “L” ou “R”.

turing_machine_logic.py

- **Criando a função run_turing_machine**:

Esta função é a espinha dorsal de toda a aplicação e contém a lógica que executa a Máquina de Turing.

```
def run_turing_machine(transition_array, tape, initial_state, main_alphabet_size, aux_alphabet_size):
    tape_pointer = 1
    current_state = initial_state
    transition_log = []
    result_text = None
```

Esta função recebe todos os parâmetros da Máquina de Turing como a array 2D de transições como argumentos.

Lembrando que cada estado é uma linha e cada coluna é uma transição da array 2D de transições.

O ponteiro da fita é inicializado na primeira posição.

```
while True:
    is_stuck = True
    is_current_state_final = False
    out_of_bounds_error = False

    if(transition_array[current_state][0].is_final):
        is_current_state_final = True
```

Loop while pois o número de iterações é incerto.

Flags de erro se a máquina não tem mais transições possíveis, se o estado atual é final, e se houve alguma transição que foge do escopo da array da fita.

```
for j in range(main_alphabet_size + aux_alphabet_size + 2):
    if(transition_array[current_state][j].current_state != -1):
        if(tape[tape_pointer]==transition_array[current_state][j].current_letter):
            transition_log.append("Trocou '"+tape[tape_pointer]+"' com '"+transition_arra
            tape[tape_pointer]=transition_array[current_state][j].replacement_letter
```

```
        if(transition_array[current_state][j].direction.lower() == 'l'):
            if((tape_pointer - 1) < 0):
                out_of_bounds_error = True
                break
            else:
                tape_pointer -= 1
        else:
            if((tape_pointer + 1) >= len(tape)):
                out_of_bounds_error = True
                break
            else:
                tape_pointer += 1

        current_state = transition_array[current_state][j].next_state
        is_stuck = False
        break
```

Loop for para verificar cada transição de um estado, buscando alguma que seja igual o caractere atualmente apontado pelo ponteiro da fita.

Caso positivo, a transição é realizada conforme a regra estabelecida, troca de caracteres e movimento do ponteiro em uma direção. Além disso a array de log armazena a transição realizada.

O estado atual é sobrescrito pelo estado futuro da transição utilizada.

is_stuck se torna falsa para continuar para a próxima iteração.

break utilizado para sair do for pois uma transição válida foi encontrada.

```
38          if(out_of_bounds_error):
39              result_text = "Palavra nao aceita! Ponteiro fora do escopo da fita!"
40              break
41          elif(is_stuck and is_current_state_final and tape_pointer == 1):
42              result_text = "Palavra aceita!"
43              break
44          elif(is_stuck and is_current_state_final):
45              result_text = "Palavra nao aceita! Ponteiro nao esta na posicao inicial!"
46              break
47          elif(is_stuck):
48              result_text = "Palavra nao aceita! Estado sem saida!"
49              break
50
51      return tape, result_text, transition_log
```

Iterações continuam até a máquina não encontrar uma transição válida e is_stuck continuar true.

Caso a máquina termine em um estado final com o ponteiro da fita na primeira posição, a palavra é aceita.

Caso contrário, a palavra não é aceita, mas de qualquer forma os logs e a fita final são mostrados ao usuário pela aplicação.