UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE GRAMÁTICA REGULAR

APUCARANA – PR
2023

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE GRAMÁTICA REGULAR

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 *RegularGrammarSimulator.exe*

Linux

Abra o terminal na pasta que contém o *RegularGrammarSimulator*

Conceda permissão de execução para o arquivo através do comando:

```
chmod +x RegularGrammarSimulator
```

Execute o programa através do comando:

```
./RegularGrammarSimulator
```

Ou duplo clique.

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, QTimer, QEvent
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QStackedLayout, QSpacerItem, QSizePolicy
from PyQt6.QtGui import QGuiApplication, QIcon
from manual_input_view import ManualInputView
from file_input_view import FileInputView
```

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.

Qt: é uma enumeração dentro do módulo **PyQt6.QtCore** que contém constantes para vários eventos de teclado, mouse e outros eventos relacionados à entrada.

QTimer: é uma classe que fornece temporizadores repetitivos e de disparo único.

QEvent: é a classe base para todos os objetos de evento em PyQt6.

QApplication, **QMainWindow**, **QWidget**, **QLabel**, **QPushButton**, **QVBoxLayout**, **QHBoxLayout**, **QStackedLayout**, **QSpacerItem, QSizePolicy**: 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 SpacerItem e SizePolicy controlam espaços em brancos e como um widget pode ser redimensionado.

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

ManualInputView, **FileInputView**: 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.manual_input_view = None
        self.file_input_view = None

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

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

        layout = QVBoxLayout(self.central_widget)
        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 variáveis auxiliares inicialmente vazias para as outras telas do aplicativo, que serão utilizadas conforme necessário no layout pilha.

Recuperamos as dimensões da tela para ajustes de layout.

Definimos o título e o ícone da janela.

Define o widget central da janela principal.

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

- **Cabeçalho da aplicação e inicialização do layout em pilha:**

```python
        # Application Header
        title_label = QLabel("<h1>Simulador de Gramatica Regular</h1>")
        title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        layout.addWidget(title_label)

        # Stacked layout
        self.stacked_layout = QStackedLayout()
```

Criamos um rótulo de título e o adicionamos ao layout principal, logo estará presente em todas as telas da aplicação.

Definimos o alinhamento do rótulo do título para o centro.

Inicializamos o layout em pilha que será incrementado com conteúdo posteriormente.

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

```
44        # First stacked view: Welcome screen
45        self.welcome_layout = QVBoxLayout()
46        self.welcome_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
47
48        self.choices_layout_label = QLabel("<h2>Bem vindo, escolha a forma de entrada</h2>")
49        self.choices_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
50        self.welcome_layout.addWidget(self.choices_layout_label)
51
52        self.buttons_layout = QHBoxLayout()
53        self.buttons_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
54
55        self.button_manual_input = (QPushButton("Manual"))
56        self.button_manual_input.clicked.connect(self.show_manual_input_view)
57        self.button_file_input = (QPushButton("Arquivo"))
58        self.button_file_input.clicked.connect(self.show_file_input_view)
59
60        self.buttons_layout.addWidget(self.button_manual_input)
61        self.buttons_layout.addWidget(self.button_file_input)
62
63        self.welcome_layout.addLayout(self.buttons_layout)
64
65        ## Finalizing welcome screen layout into a widget
66        self.welcome_container = QWidget()
67        self.welcome_container.setLayout(self.welcome_layout)
68        self.welcome_container.setFixedWidth(int(self.screen_size.width() * 0.80))
```

Criamos um layout vertical (welcome_layout) com alinhamento 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 (buttons_layout).

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

Estabelecemos as dimensões do layout para 80% da largura da tela.

- **Finalizando o layout em pilha:**

```
        ## Adding all views to stacked layout
        self.stacked_layout.addWidget(self.welcome_container)
        self.stacked_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        ## Finalizing stacked layout into a widget
        self.stacked_layout_widget = QWidget()
        self.stacked_layout_widget.setLayout(self.stacked_layout)

        ## Finalizing widgets into main application layout
        self.main_application_layout = QHBoxLayout()

        self.left_spacer = QSpacerItem(0, 0, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
        self.main_application_layout.addItem(self.left_spacer)

        self.main_application_layout.addWidget(self.stacked_layout_widget)

        self.right_spacer = QSpacerItem(0, 0, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
        self.main_application_layout.addItem(self.right_spacer)
        self.main_application_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        layout.addLayout(self.main_application_layout)

        self.showMaximized()
```

Adicionamos o widget de boas vindas ao layout empilhado.

Definimos o widget de layout empilhado e colocamos na layout principal da aplicação.

Estabelecemos alguns itens de espaçamento para centralizar o layout empilhado.

Inicializamos a aplicação maximizada.

- **Funções dessa classe principal:**

```
    def changeEvent(self, event):
        if event.type() == QEvent.Type.WindowStateChange:
            if self.windowState() & Qt.WindowState.WindowMaximized:
                self.isMaximized = True
            else:
                if self.isMaximized:
                    self.center_on_screen()
                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.

```
118     def show_manual_input_view(self):
119         self.manual_input_view = ManualInputView(self.show_main_menu)
120         self.manual_input_view.setFixedWidth(int(self.screen_size.width() * 0.80))
121
122         self.stacked_layout.addWidget(self.manual_input_view)
123         self.stacked_layout.setCurrentWidget(self.manual_input_view)
124
125         if not self.isMaximized:
126             QTimer.singleShot(0, self.center_on_screen)
127
128     def show_file_input_view(self):
129         self.file_input_view = FileInputView(self.show_main_menu)
130         self.file_input_view.setFixedWidth(int(self.screen_size.width() * 0.80))
131
132         self.stacked_layout.addWidget(self.file_input_view)
133         self.stacked_layout.setCurrentWidget(self.file_input_view)
134
135         if not self.isMaximized:
136             QTimer.singleShot(0, self.center_on_screen)
```

Definimos funções para mostrar a tela de boas-vindas e passar para as telas de entrada manual e de arquivos.

Essencialmente criamos uma nova instância do layout desejado, e colocamos ele no topo da pilha.

```
138     ## Clean up functions
139     def destroy_manual_input_view(self):
140         if self.manual_input_view:
141             self.manual_input_view.deleteLater()
142             self.stacked_layout.removeWidget(self.manual_input_view)
143             self.manual_input_view.deleteLater()
144             self.manual_input_view = None
145
146     def destroy_file_input_view(self):
147         if self.file_input_view:
148             self.file_input_view.deleteLater()
149             self.stacked_layout.removeWidget(self.file_input_view)
150             self.file_input_view.deleteLater()
151             self.file_input_view = None
```

Definimos funções auxiliares destroy_manual_input_widget e destroy_file_input_widget para remover e excluir os widgets atuais ao alternar entre telas, poupando memória.

Essencialmente tiramos o layout da pilha e o deletamos quando o botão de voltar nas outras telas é clicado.

- **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 centralizar a janela na tela.

manual_input_view.py

- **Primeiramente os módulos e classes necessários são importados:**

```
1  from PyQt6.QtCore import Qt
2  from PyQt6.QtWidgets import QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QLineEdit, QMessageBox, QScrollArea
3  from logic import howConstruct
```

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

Qt: é uma enumeração dentro do módulo **PyQt6.QtCore** que contém constantes para vários eventos de teclado, mouse e outros eventos relacionados à entrada.

QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea, QMessageBox: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py, as novidades aqui são a ScrollArea que como o nome indica cria uma área de espaço definido que é expandida através do uso de barras de rolagem, LineEdit que possibilita o input do usuário através de uma linha como o nome implica, e finalmente MessageBox que é uma caixa de pop up utilizada para alertas ou mensagens de erro.

howConstruct: Função auxiliar que contém a lógica de execução da simulação de uma gramática regular, ela é importada pois também é utilizada na entrada por arquivo.

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

```
5  class ManualInputView(QWidget):
6      def __init__(self, show_main_menu_callback):
7          super().__init__()
8
9          self.show_main_menu_callback = show_main_menu_callback
10         self.state = 0
11         self.variables_set = {}
12         self.alphabet_set = {}
13         self.starting_key = ""
14         self.grammar_map = {}
15         self.derivation_inputs = {}
```

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

Iniciamos a classe passando uma função da classe principal como argumento na forma de “callback” para que a mesma função seja invocada. Essa função serve para voltar a tela de boas vindas.

Definimos várias variáveis auxiliares que são os parâmetros da gramática regular, as variáveis, terminais, a gramática em si, regras de derivação, variável de início, e um simples controle de estados para controlar a progressão do usuário no preenchimento dos campos..

Nota-se a utilização de sets, que podem servir ambos como HashSet ou HashMap de outras linguagens, para verificar se os símbolos são únicos e proporcionar uma busca mais rápida e eficaz.

- **Cabeçalho:**

```
17          self.manual_input_layout = QVBoxLayout()
18          self.manual_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
19
20          self.manual_input_layout_label = QLabel("<h2>Descricao Formal</h2>")
21          self.manual_input_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
22          self.manual_input_layout.addWidget(self.manual_input_layout_label)
```

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

Definimos o alinhamento do rótulo do título para o centro.

- **Entrada das variáveis:**

```
24          self.variables_layout = QHBoxLayout()
25          self.variables_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
26
27          self.variables_layout_label = QLabel("<h3>V</h3>")
28          self.variables_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
29          self.variables_layout.addWidget(self.variables_layout_label)
30
31          self.variables_layout_input = QLineEdit()
32          self.variables_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
33          self.variables_layout_input.setPlaceholderText("Insira as variáveis separadas por virgulas")
34          self.variables_layout.addWidget(self.variables_layout_input)
35
36          self.manual_input_layout.addLayout(self.variables_layout)
```

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

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

Indicamos ao usuário que as variáveis devem ser separadas por vírgula com um texto placeholder.

- **Entrada dos terminais:**

```
39          self.alphabet_layout = QHBoxLayout()
40          self.alphabet_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
41
42          self.alphabet_layout_label = QLabel("<h3>T</h3>")
43          self.alphabet_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
44          self.alphabet_layout.addWidget(self.alphabet_layout_label)
45
46          self.alphabet_layout_input = QLineEdit()
47          self.alphabet_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
48          self.alphabet_layout_input.setPlaceholderText("Insira o alfabeto da linguagem separado por virgulas")
49          self.alphabet_layout.addWidget(self.alphabet_layout_input)
50
51          self.manual_input_layout.addLayout(self.alphabet_layout)
52
53          # alphabet_layout items start invisible
54          for i in range(self.alphabet_layout.count()):
55              item = self.alphabet_layout.itemAt(i)
56
57              if isinstance(item.widget(), QWidget):
58                  item.widget().setVisible(False)
```

Criamos um layout como contêiner deste item, inicialmente é invisível com o loop for abaixo.

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

Indicamos ao usuário que os terminais devem ser separados por vírgula com um texto placeholder.

- **Entrada da variável inicial:**

```
61        self.start_symbol_layout = QHBoxLayout()
62        self.start_symbol_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
63
64        self.start_symbol_layout_label = QLabel("<h3>S</h3>")
65        self.start_symbol_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
66        self.start_symbol_layout.addWidget(self.start_symbol_layout_label)
67
68        self.start_symbol_layout_input = QLineEdit()
69        self.start_symbol_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
70        self.start_symbol_layout_input.setPlaceholderText("Insira o simbolo de partida para as derivacoes")
71        self.start_symbol_layout.addWidget(self.start_symbol_layout_input)
72
73        self.manual_input_layout.addLayout(self.start_symbol_layout)
74
75        # start_symbol_layout items start invisible
76        for i in range(self.start_symbol_layout.count()):
77            item = self.start_symbol_layout.itemAt(i)
78
79            if isinstance(item.widget(), QWidget):
80                item.widget().setVisible(False)
```

Criamos um layout como contêiner deste item, inicialmente é invisível com o loop for abaixo.

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

Indicamos ao usuário a inserção correta com um texto placeholder.

Verificações posteriores certificam que o usuário pode inserir apenas um valor que já foi inserido para as variáveis.

- **Entrada do alfabeto auxiliar:**

```
83        self.rules_layout = QVBoxLayout()
84        self.rules_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
85
86        self.rules_layout_label = QLabel("<h2>Derivacoes</h2>")
87        self.rules_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
88        self.rules_layout.addWidget(self.rules_layout_label)
89        self.manual_input_layout.addLayout(self.rules_layout)
90
91        # rules_layout items start invisible
92        for i in range(self.rules_layout.count()):
93            item = self.rules_layout.itemAt(i)
94
95            if isinstance(item.widget(), QWidget):
96                item.widget().setVisible(False)
97
```

Criamos um layout como contêiner deste item, inicialmente é invisível com o loop for abaixo.

Os QLabels para este item e os seus respectivos campos de entrada serão criados posteriormente através de uma função auxiliar que utiliza as variáveis inseridas pelo usuário como base para que o preenchimento das regras de produção seja realizado.

- **Entrada do palavra de teste:**

```
99        self.test_word_layout = QHBoxLayout()
100       self.test_word_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
101
102       self.test_word_layout_label = QLabel("<h3>Palavra</h3>")
103       self.test_word_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
104       self.test_word_layout.addWidget(self.test_word_layout_label)
105
106       self.test_word_layout_input = QLineEdit()
107       self.test_word_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
108       self.test_word_layout_input.setPlaceholderText("Insira a palavra a ser testada")
109       self.test_word_layout.addWidget(self.test_word_layout_input)
110
111       self.manual_input_layout.addLayout(self.test_word_layout)
112
113       # teste_word_layout items start invisible
114       for i in range(self.test_word_layout.count()):
115           item = self.test_word_layout.itemAt(i)
116
117           if isinstance(item.widget(), QWidget):
118               item.widget().setVisible(False)
```

Criamos um layout como contêiner deste item, inicialmente é invisível com o loop for abaixo.

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

Quando a aplicação chega neste ponto o botão de “continuar” se transforma em “testar".

- **Display do resultado do teste com a gramática inserida:**

```
121         self.output_layout = QVBoxLayout()
122         self.output_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
123 
124         self.output_layout_label = QLabel("<h2>Regras de Construcao Utilizadas</h2>")
125         self.output_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
126         self.output_layout.addWidget(self.output_layout_label)
127 
128         self.output_contents_layout = QHBoxLayout()
129         self.output_contents_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
130 
131         self.output_contents_layout_label = QLabel("")
132         self.output_contents_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
133         self.output_contents_layout.addWidget(self.output_contents_layout_label)
134 
135         self.output_layout.addLayout(self.output_contents_layout)
136 
137 
138         self.manual_input_layout.addLayout(self.output_layout)
139 
140         # output_layout items start invisible
141         for i in range(self.output_layout.count()):
142             item = self.output_layout.itemAt(i)
143 
144             if isinstance(item.widget(), QWidget):
145                 item.widget().setVisible(False)
```

Criamos um layout como contêiner deste item, inicialmente é invisível com o loop for abaixo.

Criamos apenas um QLabel para este item em um layout horizontal.

Quando a aplicação realiza o teste de uma palavra com a gramática inserida, o conteúdo do QLabel é alterado para mostrar o resultado ao usuário.

- **Rodapé com botões de continuar, resetar e voltar:**

```
147         ## Footer
148         self.continue_button_layout = QHBoxLayout()
149         self.continue_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
150         self.button_script_validate_input = (QPushButton("Continuar"))
151         self.button_script_validate_input.clicked.connect(self.validate_input)
152         self.continue_button_layout.addWidget(self.button_script_validate_input)
153         self.manual_input_layout.addLayout(self.continue_button_layout)
154 
155         self.choices_bottom = QHBoxLayout()
156         self.choices_bottom.setAlignment(Qt.AlignmentFlag.AlignCenter)
157         self.button_script_show_options_menu = (QPushButton("Resetar"))
158         self.button_script_show_options_menu.clicked.connect(self.resetar)
159         self.choices_bottom.addWidget(self.button_script_show_options_menu)
160         self.manual_input_layout.addLayout(self.choices_bottom)
161 
162         self.back_button_layout = QHBoxLayout()
163         self.back_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
164         self.manual_input_layout_back_button = (QPushButton("Voltar"))
165         self.manual_input_layout_back_button.clicked.connect(self.show_main_menu_callback)
166         self.back_button_layout.addWidget(self.manual_input_layout_back_button)
167         self.manual_input_layout.addLayout(self.back_button_layout)
168 
```

Criamos alguns botões sempre visíveis para fornecer diversas funcionalidades ao usuário.

Continuar avança a inserção de dados para o próximo passo.

Resetar volta a tela ao estado padrão, para inserir uma nova gramática.

Voltar retorna a tela de boas vindas.

Conectamos os botões as suas respectivas funções.

- **ScrollArea e finalização do layout:**

```
170         self.scroll_container_layout = QVBoxLayout()
171
172         scroll_area = QScrollArea()
173         scroll_area.setWidgetResizable(True)
174
175         scroll_content = QWidget()
176         scroll_content.setLayout(self.manual_input_layout)
177
178         scroll_area.setWidget(scroll_content)
179
180         self.scroll_container_layout.addWidget(scroll_area)
181
182
183         self.setLayout(self.scroll_container_layout)
```

Finalmente, encapsulamos o layout em uma ScrollArea para possibilitar visibilidade mesmo com resoluções pequenas ou outputs grandes.

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

```
185     def validate_input(self):
186         match self.state:
187             case 0:
188                 user_input = self.variables_layout_input.text()
189                 values_array = user_input.split(",")
190
191                 if len(values_array) != len(set(values_array)):
192                     QMessageBox.warning(self, "Valores repetidos", "Va
193                     self.variables_set.clear()
194                 elif any(len(variable) != 1 for variable in values_arr
195                     QMessageBox.warning(self, "Valores invalidos", "Pe
196                     self.variables_set.clear()
197                 else:
198                     self.variables_set = set(values_array)
199
200                     self.variables_layout_input.setDisabled(True)
201                     self.state = 1
202
203                     for i in range(self.alphabet_layout.count()):
204                         item = self.alphabet_layout.itemAt(i)
205
206                         if isinstance(item.widget(), QWidget):
207                             item.widget().setVisible(True)
```

Definimos uma função validate_input para verificar cada passo da inserção dos dados do usuário. O match case e a variável de estados é utilizado para realizar este controle.

Caso 0 verifica a entrada das variáveis, passando o texto de input para uma variável auxiliar, dividindo pela vírgula em uma list (array Python), verificando através de conversão para Set se todos os valores são únicos e se o tamanho de cada valor não passa de 1.

Caso positivo o estado progride, tranca a inserção de variáveis e deixa o usuário inserir os terminais.

```
209             case 1:
210                 user_input = self.alphabet_layout_input.text()
211                 values_array = user_input.split(",")
212 
213                 if len(values_array) != len(set(values_array)):
214                     QMessageBox.warning(self, "Valores repetidos", "Valor
215                     self.alphabet_set.clear()
216                 elif any(len(alphabet_symbol) != 1 for alphabet_symbol in
217                     QMessageBox.warning(self, "Valores invalidos", "Pelo
218                     self.alphabet_set.clear()
219                 elif any(variable in self.variables_set for variable in v
220                     QMessageBox.warning(self, "Valores ja existentes", "P
221                     self.alphabet_set.clear()
222                 else:
223                     self.alphabet_set = set(values_array)
224 
225                     self.alphabet_layout_input.setDisabled(True)
226                     self.state = 2
227 
228                     for i in range(self.start_symbol_layout.count()):
229                         item = self.start_symbol_layout.itemAt(i)
230 
231                         if isinstance(item.widget(), QWidget):
232                             item.widget().setVisible(True)
```

Caso 1 é similar ao 0, realizando as mesmas verificações para os terminais, com a verificação adicional de que eles não podem ser iguais a quaisquer variáveis inseridas anteriormente.

Caso positivo o estado progride, tranca a inserção de terminais e deixa o usuário a variável inicial.

```
234             case 2:
235                 user_input = self.start_symbol_layout_input.text()
236 
237                 if(len(user_input) > 1):
238                     QMessageBox.warning(self, "Valor invalido", "Por favor insira um valor existente das variaveis.")
239                 elif(user_input not in self.variables_set):
240                     QMessageBox.warning(self, "Valor invalido", "Por favor insira um valor existente das variaveis.")
241                 else:
242                     self.starting_key = user_input
243 
244                     self.start_symbol_layout_input.setDisabled(True)
245                     self.state = 3
246 
247                     for i in range(self.rules_layout.count()):
248                         item = self.rules_layout.itemAt(i)
249 
250                         if isinstance(item.widget(), QWidget):
251                             item.widget().setVisible(True)
```

```
253                     starting_derivacao_layout = QHBoxLayout()
254                     starting_derivacao_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
255 
256                     starting_derivacao_layout_label = QLabel("<h3>" + self.starting_key + " - </h3>")
257                     starting_derivacao_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
258                     starting_derivacao_layout.addWidget(starting_derivacao_layout_label)
259 
260                     starting_derivacao_layout_input = QLineEdit()
261                     starting_derivacao_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
262                     starting_derivacao_layout_input.setPlaceholderText("Insira as regras separadas por virgula")
263                     starting_derivacao_layout.addWidget(starting_derivacao_layout_input)
264 
265                     self.rules_layout.addLayout(starting_derivacao_layout)
266 
267                     self.derivation_inputs = {self.starting_key: starting_derivacao_layout_input}
268 
269                     variables_set_copy = self.variables_set.copy()
270                     variables_set_copy.discard(self.starting_key)
```

```
272                     for variable in variables_set_copy:
273                         derivacao_layout = QHBoxLayout()
274                         derivacao_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
275 
276                         derivacao_layout_label = QLabel("<h3>" + variable + " - </h3>")
277                         derivacao_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
278                         derivacao_layout.addWidget(derivacao_layout_label)
279 
280                         derivacao_layout_input = QLineEdit()
281                         derivacao_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
282                         derivacao_layout_input.setPlaceholderText("Insira as regras separadas por virgula")
283                         derivacao_layout.addWidget(derivacao_layout_input)
284 
285                         self.rules_layout.addLayout(derivacao_layout)
286 
287                         self.derivation_inputs[variable] = derivacao_layout_input
```

Caso 2 verifica se a variável inicial inserida pelo usuário faz parte das inseridas anteriormente.

Caso positivo, o estado progride e são gerados os QLabels e QLineEdits para a inserção das regras de produção para cada variável.

Nota-se que a variável inicial sempre será a primeira linha, mas as demais tem ordem aleatória por tratar de um Set.

Além disso nota-se o preenchimento do dicionário de “derivation_inputs” para controlar quais QLineEdits pertencem a cada variável pois são geradas dinamicamente.

```
289             case 3:
290                 valid_input = True
291                 self.grammar_map.clear()
292                 combined_set = self.variables_set.union(self.alphabet_set)
293 
294                 for variable, line_edit in self.derivation_inputs.items():
295                     inputs = line_edit.text().split(",")
296 
297                     if any(not input_str.strip() for input_str in inputs):
298                         QMessageBox.warning(self, "Campo Vazio", f"Campo vazio 
299                         valid_input = False
300                         break
301 
302                     if len(inputs) != len(set(inputs)):
303                         QMessageBox.warning(self, "Regras repetidas", f"Regras 
304                         valid_input = False
305                         break
306 
307                     for input in inputs:
308                         if not self.validate_string(input, combined_set):
309                             QMessageBox.warning(self, "Símbolo Invalido", f"Símb
310                             valid_input = False
311                             break
312 
313                     else:
314                         self.grammar_map[variable] = set(inputs)
315 
```

```
316                 if valid_input:
317                     print(self.grammar_map)
318 
319                     self.state = 4
320 
321                     for line_edit in self.derivation_inputs.values():
322                         line_edit.setDisabled(True)
323 
324                     for i in range(self.test_word_layout.count()):
325                         item = self.test_word_layout.itemAt(i)
326 
327                         if isinstance(item.widget(), QWidget):
328                             item.widget().setVisible(True)
329 
330                     self.button_script_validate_input.setText("Testar")
```

Caso 3 verifica as regras de produção inseridas pelo usuário seguindo algumas regras básicas.

Uma mesma variável não pode ter regras de produção repetidas.

Regras de produção devem utilizar as variáveis e terminais estabelecidos anteriormente pelo usuário.

O usuário não pode deixar uma variável sem regras de produção.

Caso positivo o estado progride, tranca a inserção de regras de produções, deixa o usuário inserir palavras de teste e modifica o botão de “continuar” para “testar”.

```
425     def validate_string(self, input_string, char_set):
426         if "*" not in input_string:
427             return all(char in char_set for char in input_string)
428         elif input_string == "**":
429             return True
430         else:
431             return False
```

Função auxiliar de verificação dos strings das regras de produção, basicamente verifica se os símbolos utilizados fazem parte das variáveis ou terminais e também cuidam do caso de “**” vazio.

```
332             case 4:
333                 user_input = self.test_word_layout_input.text()
334 
335                 if user_input == "":
336                     QMessageBox.warning(self, "Teste Invalido", f"Por favor, insira uma palavra para ser testada.")
337                 else:
338                     for i in range(self.output_layout.count()):
339                         item = self.output_layout.itemAt(i)
340 
341                         if isinstance(item.widget(), QWidget):
342                             item.widget().setVisible(True)
343 
344                     result = howConstruct(self.grammar_map, self.starting_key, user_input, "", 1)
345 
346                     if(result is not None):
347                         self.output_contents_layout_label.setText('\n'.join(result))
348                     else:
349                         self.output_contents_layout_label.setText(''.join("Impossivel contruir a palavra com essas regras."))
```

Caso 4 verifica se o usuário inseriu alguma coisa no QLineEdit da palavra e utiliza a função da lógica principal da gramática regular para verificar se a construção da palavra é possível ou não.

Caso positivo as regras de produção utilizadas e sua ordem são mostradas para o usuário.

a entrada da quantidade de estados antes de mostrar o campo de entrada do tamanho do alfabeto principal.

```
352     def resetar(self):
353         self.state = 0
354         self.variables_set.clear()
355         self.alphabet_set.clear()
356         self.starting_key = ""
357         self.derivation_inputs.clear()
358         self.grammar_map.clear()
359 
360         self.variables_layout_input.setText("")
361         self.alphabet_layout_input.setText("")
362         self.start_symbol_layout_input.setText("")
363         self.test_word_layout_input.setText("")
364         self.output_contents_layout_label.setText("")
365 
366         self.variables_layout_input.setDisabled(False)
367         self.alphabet_layout_input.setDisabled(False)
368         self.start_symbol_layout_input.setDisabled(False)
369 
370         self.button_script_validate_input.setText("Continuar")
```

```
372         for i in range(self.alphabet_layout.count()):
373             item = self.alphabet_layout.itemAt(i)
374 
375             if isinstance(item.widget(), QWidget):
376                 item.widget().setVisible(False)
377 
378         for i in range(self.start_symbol_layout.count()):
379             item = self.start_symbol_layout.itemAt(i)
380 
381             if isinstance(item.widget(), QWidget):
382                 item.widget().setVisible(False)
```

```
384         while self.rules_layout.count():
385             layout_item = self.rules_layout.takeAt(0)
386             if layout_item:
387                 item_layout = layout_item.layout()
388                 if item_layout:
389                     while item_layout.count():
390                         item = item_layout.takeAt(0)
391                         widget = item.widget()
392                         if widget:
393                             widget.deleteLater()
394                         else:
395                             pass
396                     item_layout.deleteLater()
397                 else:
398                     widget = layout_item.widget()
399                     if widget:
400                         widget.deleteLater()
401
402         self.rules_layout_label = QLabel("<h3>Derivacoes</h3>")
403         self.rules_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
404         self.rules_layout.addWidget(self.rules_layout_label)
405         self.manual_input_layout.addLayout(self.rules_layout)
406
```

```
407         for i in range(self.rules_layout.count()):
408             item = self.rules_layout.itemAt(i)
409
410             if isinstance(item.widget(), QWidget):
411                 item.widget().setVisible(False)
412
413         for i in range(self.test_word_layout.count()):
414             item = self.test_word_layout.itemAt(i)
415
416             if isinstance(item.widget(), QWidget):
417                 item.widget().setVisible(False)
418
419         for i in range(self.output_layout.count()):
420             item = self.output_layout.itemAt(i)
421
422             if isinstance(item.widget(), QWidget):
423                 item.widget().setVisible(False)
```

Função reset retorna a tela ao estado inicial. Consiste em “esvaziar” ou “zerar” variáveis, loops for deixando elementos invisíveis novamente, loop while deletando os QLineEdits e QLabels de inserção de regras de produção e restaurando o QLabel inicial deste item, retornando o botão de “testar” para “continuar” novamente e habilitando a interação com itens previamente trancados para que uma nova gramática seja inserida.

- **Em resumo, manual_input_view.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 gramática regular, realizando a verificação e validação, e caso aceitos possibilita que o usuário teste palavras com as suas regras de produção.

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):**

```
ass CustomLineEdit(QLineEdit):
  def __init__(self, transition_table, i, j, parent=None):
      super().__init__(parent)
      self.transition_table = transition_table
      self.i = i
      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.