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 das regras de produção:**

```
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 às 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.

file_input_screen.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, QTextEdit
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, QTextEdit: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py e manual_input_view.py, a única novidade aqui é o TextEdit que é similar ao QLineEdit, porém é uma caixa maior que possibilita a inserção de várias linhas de texto.

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 por arquivo (classe FileInputScreen):**

```
5   class FileInputView(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.starting_key = ""
11          self.grammar_map = {}
```

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, porém mais simplificadas comparados a entrada manual, aqui temos apenas a representação da gramática e o variável inicial.

Nota-se a utilização de set/dicionário, 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:**

```
13          self.file_input_layout = QVBoxLayout()
14          self.file_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
15
16          self.file_input_layout_label = QLabel("<h2>Descricao Formal</h2>")
17          self.file_input_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
18          self.file_input_layout.addWidget(self.file_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 da gramática regular:**

```
20          self.text_input = QTextEdit()
21          self.text_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
22          self.text_input.setPlaceholderText("Cole aqui as regras de construcao")
23          self.file_input_layout.addWidget(self.text_input)
24
25          self.test_word_layout = QHBoxLayout()
26          self.test_word_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Criamos um layout como contêiner deste item.

Criamos um QTextEdit para este item, onde o usuário vai colar sua gramática regular.

O formato de inserção por arquivo é bem simples:

[VARIÁVEL]>[REGRAS DE PRODUÇÃO]

Onde cada linha corresponde a uma variável e suas respectivas regras de produção, cada uma separada através de vírgulas.

Nota-se que a primeira variável inserida é considerada a inicial neste padrão.

- **Entrada do palavra de teste:**

```
25        self.test_word_layout = QHBoxLayout()
26        self.test_word_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
27
28        self.test_word_layout_label = QLabel("<h3>Palavra</h3>")
29        self.test_word_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
30        self.test_word_layout.addWidget(self.test_word_layout_label)
31
32        self.test_word_layout_input = QLineEdit()
33        self.test_word_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
34        self.test_word_layout_input.setPlaceholderText("Insira a palavra a ser testada")
35        self.test_word_layout.addWidget(self.test_word_layout_input)
36
37        self.file_input_layout.addLayout(self.test_word_layout)
```

Criamos um layout como contêiner deste item.

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

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

```
39          self.output_layout = QVBoxLayout()
40          self.output_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
41
42          self.output_layout_label = QLabel("<h2>Regras de Construcao Utilizadas</h2>")
43          self.output_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
44          self.output_layout.addWidget(self.output_layout_label)
45
46          self.output_contents_layout = QHBoxLayout()
47          self.output_contents_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
48
49          self.output_contents_layout_label = QLabel("")
50          self.output_contents_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
51          self.output_contents_layout.addWidget(self.output_contents_layout_label)
52
53          self.output_layout.addLayout(self.output_contents_layout)
54
55
56          self.file_input_layout.addLayout(self.output_layout)
```

Criamos um layout como contêiner deste item.

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:**

```
58        self.continue_button_layout = QHBoxLayout()
59        self.continue_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
60        self.button_script_validate_input = (QPushButton("Testar"))
61        self.button_script_validate_input.clicked.connect(self.validate_input)
62        self.continue_button_layout.addWidget(self.button_script_validate_input)
63        self.file_input_layout.addLayout(self.continue_button_layout)
64
65        self.choices_bottom = QHBoxLayout()
66        self.choices_bottom.setAlignment(Qt.AlignmentFlag.AlignCenter)
67        self.button_script_show_options_menu = (QPushButton("Resetar"))
68        self.button_script_show_options_menu.clicked.connect(self.resetar)
69        self.choices_bottom.addWidget(self.button_script_show_options_menu)
70        self.file_input_layout.addLayout(self.choices_bottom)
71
72        self.back_button_layout = QHBoxLayout()
73        self.back_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
74        self.file_input_layout_back_button = (QPushButton("Voltar"))
75        self.file_input_layout_back_button.clicked.connect(self.show_main_menu_callback)
76        self.back_button_layout.addWidget(self.file_input_layout_back_button)
77        self.file_input_layout.addLayout(self.back_button_layout)
78
```

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

Testar verifica a inserção de dados, da palavra e caso positivo realiza a tentativa de construção.

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

Voltar retorna a tela de boas vindas.

Conectamos os botões às suas respectivas funções.

- **ScrollArea e finalização do layout:**

```
80        self.scroll_container_layout = QVBoxLayout()
81
82        scroll_area = QScrollArea()
83        scroll_area.setWidgetResizable(True)
84
85        scroll_content = QWidget()
86        scroll_content.setLayout(self.file_input_layout)
87
88        scroll_area.setWidget(scroll_content)
89
90        self.scroll_container_layout.addWidget(scroll_area)
91
```

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

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

```
95      def validate_input(self):
96          raw_text_input = self.text_input.toPlainText()
97          lines = raw_text_input.split('\n')
98
99          self.starting_key == ""
100         self.grammar_map.clear()
101         valid_input = True
102
103         try:
104             for line in lines:
105                 line = line.strip()
106                 key, value = line.split(">")
107                 if key in self.grammar_map:
108                     QMessageBox.warning(self, "Valores repetidos"
109                     valid_input = False
110                 values = value.split(",")
111                 if len(values) != len(set(values)):
112                     QMessageBox.warning(self, "Valores repetidos"
113                     valid_input = False
114                 self.grammar_map[key] = set(values)
115
116                 if self.starting_key == "":
117                     self.starting_key = key
118         except:
119             QMessageBox.warning(self, "Valores invalidos", "Valor
120             valid_input = False
```

Definimos uma função validate_input, que inicialmente vai criar a gramática do usuário conforme os padrões de inserção por texto.

O texto é dividido por cada linha, e as mesmas são divididas pelo “>”, caso não exista o try except certifica que o input é rejeitado.

Após isso, a segunda metade é dividida com as vírgulas que separam cada regra de produção.

Ademais, verifica-se se não existem regras de produção repetidas para a mesma variável.

Por fim, a primeira variável inserida é considerada a inicial.

```
122         if(valid_input):
123             test_word_input = self.test_word_layout_input.text()
124 
125             if test_word_input == "":
126                 QMessageBox.warning(self, "Teste Invalido", f"Por favor, insira uma palavra para ser testada.")
127             else:
128                 result = howConstruct(self.grammar_map, self.starting_key, test_word_input, "", 1)
129 
130                 if(result is not None):
131                     self.output_contents_layout_label.setText('\n'.join(result))
132                 else:
133                     self.output_contents_layout_label.setText(''.join("Impossivel contruir a palavra com essas regras."))
```

Caso a gramática for válida, verifica-se se o usuário isneriu uma palavra de teste e utiliza-se a função “howConstruct” para verificar se é possível ou não com a gramática inserida.

Caso positivo, o resultado detalhado dos passos é mostrado na tela para o usuário.

```
135     def resetar(self):
136         self.starting_key = ""
137         self.grammar_map.clear()
138 
139         self.text_input.setText("")
140         self.test_word_layout_input.setText("")
141         self.output_contents_layout_label.setText("")
```

Função bem simples de reset para retornar a tela ao estado inicial.

- **Em resumo, file_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 através de um texto padronizado, o que agiliza uma maior quantidade de testes.

logic.py

- **Criando a função auxiliar setToList**:

```python
def setToList(map, key):
    if key in map:
        return list(map[key])
    else:
        print(f"Key '{key}' not found in the map.")
        return []
```

Função auxiliar simples que transforma um Set Python em um List, para percorrer pelas regras de produção de uma forma mais simples.

- **Criando a função recursiva howConstruct**:

Esta função é a espinha dorsal de toda a aplicação e contém a lógica que executa o simulador de gramática regular.

```python
def howConstruct(map, startKey, targetWord, currentWord, iteration, result=None):
    if result is None:
        result = []

    if(currentWord == targetWord):
        result.append("Construcao completa!")
        return result

    if(len(currentWord) > (len(targetWord) + 1)):
        return None
```

Esta função recebe vários argumentos.

map: é o dicionário Python, equivalente a um HashMap em outras linguagens que representa a gramática regular, onde o par chave -> valor é uma variável -> regras de produção, que são representadas por um Set.

startKey: é a variável inicial que inicial o processo de produção.

targetWord: é a palavra a ser construída.

currentWord: é a palavra atual em cada iteração, inicia-se como uma string vazia.

iteration: é a profundidade de iteração atual.

result: é uma variável local criada na primeira iteração para armazenar a sequência de regras de produção utilizadas para cada ramo da árvore de possibilidades de construção.

Inicialmente temos os casos base que terminam a recursividade:

1. A palavra atual é igual a palavra desejada, então a construção foi bem sucedida, retornando a lista das regras de produção utilizadas.
2. O tamanho da palavra atual é maior que o da palavra desejada, ou seja, a construção não foi bem sucedida, retornando Nulo ao call stack deste ramo da árvore o que também esvazia a lista de resultado.

```
19      if(iteration == 1):
20          startingKeyRules = setToList(map, startKey)
21          updatedCurrentWord = ""
22
23          for constructionRule in startingKeyRules:
24              updatedCurrentWord = currentWord + constructionRule
25              result = howConstruct(map, startKey, targetWord, updatedCurrentWord, (iteration + 1))
26              if(result is not None):
27                  result.insert(0, "Aplicado - (S > " + constructionRule + ") - String atual = '" + updatedCurrentWord + "'")
28                  return result
```

Caso especial da primeira iteração, a construção deve se iniciar de algum lugar, neste caso será da variável determinada como a de início.

Este bloco de código cria uma lista temporária das regras de produção da variável inicial, e atualiza a palavra atual, respectivamente.

Em termos mais práticos imagine um S > aA, bB.

O loop for realizará duas chamadas recursivas diferentes:

Palavra atual = “aA” e Palavra atual = “bB”.

Porém, considerando a natureza do call stack, a primeira possibilidade será exausta primeiro em sua maior profundidade antes de entrarmos na possibilidade “bB”.

Como a iteração é incrementada, a próxima chamada recursiva entrará no bloco de código generalizado:

```
    if(iteration > 1):
        index = -1

        for i in range(len(currentWord)):
            potentialKey = currentWord[i]
            if(potentialKey in map):
                index = i
                keyValue = potentialKey
                break

        if(index != -1):
            currentKeyRules = setToList(map, keyValue)
            updatedCurrentWord = ""

            for constructionRule in currentKeyRules:
                if(constructionRule == "**"):
                    updatedCurrentWord = currentWord[:index] + currentWord[index + 1:]
                else:
                    updatedCurrentWord = currentWord[:index] + constructionRule + currentWord[index + 1:]

                result = howConstruct(map, startKey, targetWord, updatedCurrentWord, (iteration + 1))
                if(result is not None):
                    result.insert(0, "Aplicado - (" + keyValue + " > " + constructionRule + ") - String atual = '" + updatedCurrentWord + "'")
                    return result
        else:
            return None

    return None
```

Variável local de índice é inicializada como -1 para fins de controle, o que fará mais sentido posteriormente.

Loop for na palavra atual, lembrando que neste ponto ela já foi inicializada com alguma regra de produção da variável inicial.

Este loop busca variáveis no string da palavra atual, para o substituir com regras de produções adicionais, de forma similar que a produção inicial foi feita.

Voltando ao exemplo da palavra atual = “aA”.

Este loop for vai percorrer esta string até encontrar a variável “A”.

Quando for encontrada, atualizamos o índice com a sua localização e saímos do loop com um break para realizarmos o próximo passo das chamadas recursivas.

Ocorre que o próximo if verifica se o índice é -1 ou não, caso contrário retorna Nulo e termina este ramo da recursão.

Isso faz sentido porque se a palavra atual não caiu nos casos base estabelecidos no topo da função recursiva explicados anteriormente, e o loop for não conseguiu encontrar uma variável para realizar novas produções, isso significa que este ramo atual termina aqui, e não é a construção válida procurada.

Em contrapartida, caso uma variável seja encontrada, ele será “resolvida” através de chamadas recursivas, similares a variável inicial, por isso é importante guardar seu índice.

A lógica verifica todas as regras de produção da variável encontrada, e substitui elas no string atual para realizar novas chamadas recursivas.

Voltamos ao exemplo da palavra atual = “aA”.

A variável “A” foi encontrada no índice 1.

Verificamos as regras de produção desta variável, vamos dizer que são “a”, “aA”, “**”.

Desta forma, temos 3 chamadas recursivas a ser realizadas substituindo a variável com as regras de produção:

Palavra atual = “aa”, Palavra atual = “aaA”, Palavra atual = “a”.

Porém novamente lembrando que pela natureza do call stack, a primeira possibilidade será exausta primeiro em sua maior profundidade antes de entrarmos nas possibilidades “aaA” e “a”.

Isso é realizado na prática pela manipulação de substrings, por isso o armazenamento do índice da variável encontrada é fundamental.

Ademais, é importante notar que as chamadas recursivas somente retornam a sequência de passos se realmente teve sucesso em sua iteração, por isso o “if (result is not None)” caso contrário a iteração cai no último caso base que é o “return None” no final da função o que serve ambos para cortar as recursões sem sucesso, tanto para apresentar um retorno final se todas as possibilidades foram exaustas e nenhum resultado válido foi encontrado.

Desta forma, a função vai continuar chamando a si mesma até que um dos casos base seja encontrado.