UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE PIPELINE DE INSTRUÇÕES

APUCARANA – PR
2023

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE PIPELINE DE INSTRUÇÕES

Trabalho apresentado à disciplina de Arquitetura e Organização de Computadores, do curso de Bacharelado em Ciência da Computação.

Professor: Guilherme Nakahata

APUCARANA – PR
2023

SUMÁRIO

INSTRUÇÕES DE EXECUÇÃO

Windows

Execute *ProcessPipelineSimulator.exe*

Linux

Abra o terminal na pasta que contém o *ProcessPipelineSimulator*

Conceda permissão de execução para o arquivo através do comando:

```
chmod +x ProcessPipelineSimulator
```

Execute o programa através do comando:

```
./ProcessPipelineSimulator
```

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 Pipeline de Instrucoes")
        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 Pipeline de Instrucoes</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, QRadioButton
3  from logic_simple import SimpleInstruction, run_instructions_simple
4  from logic_no_overlap import NoOverlapInstruction, run_instructions_no_overlap
```

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.

SimpleInstruction, run_instructions_simple, NoOverlapInstruction, run_instructions_no_overlap: Funções e classes auxiliares que contém a lógica da criação do diagrama de tempo do pipeline de instruções, simple ou no overlap são duas variações que serão explicadas posteriormente.

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

```
6  class ManualInputView(QWidget):
7      def __init__(self, show_main_menu_callback):
8          super().__init__()
9
10         self.show_main_menu_callback = show_main_menu_callback
11         self.state = 0
12         self.simple_logic_array = []
13         self.no_overlap_logic_array = []
14         self.longest_input_string = 0
15         self.instruction_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 das instruções que serão inseridas no pipeline, as instruções são representadas por uma array de classe customizada com as variações simple e no overlap. A variável state controla a progressão do usuário no preenchimento dos campos, longest_input_string define o espaçamento entre as instruções no diagrama e o dicionário instruction_inputs armazena referências aos elementos de entrada gerados dinamicamente para o usuário dependendo de quantas instruções desejar inserir.

- **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>Dados das Instrucoes</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 da quantidade de instruções:**

```
24          self.number_of_instructions_layout = QHBoxLayout()
25          self.number_of_instructions_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
26
27          self.number_of_instructions_layout_label = QLabel("<h3>Quantidade de instrucoes</h3>")
28          self.number_of_instructions_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
29          self.number_of_instructions_layout.addWidget(self.number_of_instructions_layout_label)
30
31          self.number_of_instructions_layout_input = QLineEdit()
32          self.number_of_instructions_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
33          self.number_of_instructions_layout_input.setPlaceholderText("Insira a quantidade de instrucoes")
34          self.number_of_instructions_layout.addWidget(self.number_of_instructions_layout_input)
35
36          self.manual_input_layout.addLayout(self.number_of_instructions_layout)
```

Criamos um layout como contêiner deste item de entrada da quantidade de instruções a serem inseridas, inicialmente é o único item de entrada visível.

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

- **Entrada das instruções:**

```
39          self.instructions_layout = QVBoxLayout()
40          self.instructions_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
41
42          self.instructions_layout_label = QLabel("<h2>Instrucoes</h2>")
43          self.instructions_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
44          self.instructions_layout.addWidget(self.instructions_layout_label)
45          self.manual_input_layout.addLayout(self.instructions_layout)
46
47          # rules_layout items start invisible
48          for i in range(self.instructions_layout.count()):
49              item = self.instructions_layout.itemAt(i)
50
51              if isinstance(item.widget(), QWidget):
52                  item.widget().setVisible(False)
```

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 a quantidade de instruções inserida pelo usuário como base para que o preenchimento das instruções seja realizado.

- **Entrada da quantidade de tempo:**

```
55          self.time_limit_layout = QHBoxLayout()
56          self.time_limit_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
57
58          self.time_limit_layout_label = QLabel("<h3>Quantidade de tempo</h3>")
59          self.time_limit_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
60          self.time_limit_layout.addWidget(self.time_limit_layout_label)
61
62          self.time_limit_layout_input = QLineEdit()
63          self.time_limit_layout_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
64          self.time_limit_layout_input.setPlaceholderText("Insira a quantidade de tempo")
65          self.time_limit_layout.addWidget(self.time_limit_layout_input)
66
67          self.manual_input_layout.addLayout(self.time_limit_layout)
68
69          # alphabet_layout items start invisible
70          for i in range(self.time_limit_layout.count()):
71              item = self.time_limit_layout.itemAt(i)
72
73              if isinstance(item.widget(), QWidget):
74                  item.widget().setVisible(False)
```

Criamos um layout como contêiner deste item de entrada da quantidade de tempo, essencialmente o número de colunas do diagrama, 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.

- **Display do diagrama de tempo conforme as instruções e parâmetros inseridos:**

```
77          self.output_layout = QVBoxLayout()
78          self.output_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
79
80          self.output_layout_label = QLabel("<h2>Pipeline de Instrucoes Gerado</h2>")
81          self.output_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
82          self.output_layout.addWidget(self.output_layout_label)
83
84          self.output_contents_layout = QHBoxLayout()
85          self.output_contents_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
86
87          self.output_contents_layout_label = QLabel("")
88          self.output_contents_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
89          self.output_contents_layout.addWidget(self.output_contents_layout_label)
90
91          self.output_layout.addLayout(self.output_contents_layout)
92
93
94          self.manual_input_layout.addLayout(self.output_layout)
95
96          # output_layout items start invisible
97          for i in range(self.output_layout.count()):
98              item = self.output_layout.itemAt(i)
99
100             if isinstance(item.widget(), QWidget):
101                 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 a criação do diagrama de tempo com as instruções e parâmetros inseridos, o conteúdo do QLabel é alterado para mostrar o resultado ao usuário.

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

```
103         ## Footer
104         self.continue_button_layout = QHBoxLayout()
105         self.continue_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
106         self.button_script_validate_input = (QPushButton("Continuar"))
107         self.button_script_validate_input.clicked.connect(self.validate_input)
108         self.continue_button_layout.addWidget(self.button_script_validate_input)
109         self.manual_input_layout.addLayout(self.continue_button_layout)
110 
111         self.simple_or_complex_radio_layout = QHBoxLayout()
112         self.simple_or_complex_radio_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
113         self.simple_or_complex_radio = QRadioButton("Permitir Overlap?")
114         self.simple_or_complex_radio.setChecked(True)
115         self.simple_or_complex_radio.setVisible(False)
116         self.simple_or_complex_radio_layout.addWidget(self.simple_or_complex_radio)
117         self.manual_input_layout.addLayout(self.simple_or_complex_radio_layout)
118 
119         self.choices_bottom = QHBoxLayout()
120         self.choices_bottom.setAlignment(Qt.AlignmentFlag.AlignCenter)
121         self.button_script_show_options_menu = (QPushButton("Resetar"))
122         self.button_script_show_options_menu.clicked.connect(self.reset)
123         self.choices_bottom.addWidget(self.button_script_show_options_menu)
124         self.manual_input_layout.addLayout(self.choices_bottom)
125 
126         self.back_button_layout = QHBoxLayout()
127         self.back_button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
128         self.manual_input_layout_back_button = (QPushButton("Voltar"))
129         self.manual_input_layout_back_button.clicked.connect(self.show_main_menu_callback)
130         self.back_button_layout.addWidget(self.manual_input_layout_back_button)
131         self.manual_input_layout.addLayout(self.back_button_layout)
132 
```

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

Continuar avança a inserção de dados para o próximo passo.

Permitir overlap altera a lógica de geração de diagrama utilizada.

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:**

```
149     def is_integer(self, value):
150         try:
151             int(value)
152             return True
153         except ValueError:
154             return False
155
156     def is_integer_float(self, value):
157         try:
158             float_value = float(value)
159             int_value = int(float_value)
160             return float_value == int_value
161         except (ValueError, TypeError):
162             return False
163
164     def validate_value(self, value, instruction_number):
165         if not self.is_integer(value):
166             QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor numerico para a instrucao {str(instruction_number+1)}.")
167             return False
168         elif not self.is_integer_float(value):
169             QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor inteiro para a instrucao {str(instruction_number+1)}.")
170             return False
171         else:
172             return True
```

Definimos algumas funções auxiliares is_integer, is_integer_float e validade_value para verificar se o usuário inseriu um valor numérico válido para a quantidade de instruções, tempo de cada instrução e tempo total do diagrama.

As funções verificam se o conteúdo do QLineEdits são de fato um número, e se este número é um inteiro e não um float, gerando alertas que informam o usuário de possíveis erros na inserção dos dados.

```
174     def validate_input(self):
175         match self.state:
176             case 0:
177                 user_input = self.number_of_instructions_layout_input.text()
178 
179                 if not self.is_integer(user_input):
180                     QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor numerico.")
181                     return
182                 elif not self.is_integer_float(user_input):
183                     QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor inteiro.")
184                     return
185                 else:
186                     user_input = int(user_input)
187 
188                 self.number_of_instructions_layout_input.setDisabled(True)
189                 self.simple_logic_array = [SimpleInstruction() for _ in range(user_input)]
190                 self.no_overlap_logic_array = [NoOverlapInstruction() for _ in range(user_input)]
191                 self.state = 1
192 
193                 for i in range(self.instructions_layout.count()):
194                     item = self.instructions_layout.itemAt(i)
195 
196                     if isinstance(item.widget(), QWidget):
197                         item.widget().setVisible(True)
```

```
199             for i in range(user_input):
200                 instruction_layout = QHBoxLayout()
201                 instruction_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
202 
203                 instruction_layout_label = QLabel("<h3>Instrucao " + str(i+1) + " - </h3>")
204                 instruction_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
205                 instruction_layout.addWidget(instruction_layout_label)
206 
207                 instruction_layout_name_label = QLabel("Nome - ")
208                 instruction_layout_name_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
209                 instruction_layout.addWidget(instruction_layout_name_label)
210 
211                 instruction_layout_name_input = QLineEdit()
212                 instruction_layout_name_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
213                 instruction_layout_name_input.setPlaceholderText(f"Insira o nome da instrucao {str(i+1)}")
214                 instruction_layout.addWidget(instruction_layout_name_input)
215 
216                 instruction_layout_time_label = QLabel("Unidades de Tempo - ")
217                 instruction_layout_time_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
218                 instruction_layout.addWidget(instruction_layout_time_label)
219 
220                 instruction_layout_time_input = QLineEdit()
221                 instruction_layout_time_input.setAlignment(Qt.AlignmentFlag.AlignCenter)
222                 instruction_layout_time_input.setPlaceholderText(f"Insira o tempo necessario para executar a instrucao {str(i+1)}")
223                 instruction_layout.addWidget(instruction_layout_time_input)
224 
225                 self.instructions_layout.addLayout(instruction_layout)
226 
227                 self.instruction_inputs[i] = (instruction_layout_name_input, instruction_layout_time_input)
```

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 do número de instruções com as funções auxiliares descritas anteriormente. Caso o usuário inserir um valor válido, este campo será travado para que este valor não seja alterado novamente e a progressão do estado habilita a entrada dos próximos parâmetros. Ademais arrays com as classes customizadas de instruções são inicializadas com o tamanho do valor inserido pelo usuário.

Por fim, cria-se dinamicamente os campos de entrada de cada instrução para o usuário, que solicitam o nome de cada instrução e a quantidade de tempo necessário para ser executada.

```
229             case 1:
230                 valid_input = True
231                 self.longest_input_string = 0
232                 current_index = 0
233 
234                 for i, (instruction_layout_name_input, instruction_layout_time_input) in self.instruction_inputs.items():
235                     instruction_name = instruction_layout_name_input.text()
236                     instruction_time = instruction_layout_time_input.text()
237 
238                     if(not instruction_name):
239                         QMessageBox.warning(self, "Campo Vazio", f"Campo vazio detectado para a instrucao {str(i+1)}.")
240                         valid_input = False
241                         break
242 
243                     if (not self.validate_value(instruction_time, i)):
244                         valid_input = False
245                         break
246                     else:
247                         instruction_time = int(instruction_time)
248 
249                     if(len(instruction_name) > self.longest_input_string):
250                         self.longest_input_string = len(instruction_name)
251 
252                     self.simple_logic_array[i].set_name(instruction_name)
253                     self.simple_logic_array[i].set_size(instruction_time)
```

```
255                     self.no_overlap_logic_array[i].set_name(instruction_name)
256                     tempSize = instruction_time
257                     self.no_overlap_logic_array[i].set_size(tempSize)
258                     self.no_overlap_logic_array[i].set_starting_index(current_index)
259                     current_index += (tempSize - 1)
260                     self.no_overlap_logic_array[i].set_ending_index(current_index)
261                     current_index += 1
262 
263                 if valid_input:
264                     self.state = 2
265 
266                     for i, (instruction_layout_name_input, instruction_layout_time_input) in self.instruction_inputs.items():
267                         instruction_layout_name_input.setDisabled(True)
268                         instruction_layout_time_input.setDisabled(True)
269 
270                     for i in range(self.time_limit_layout.count()):
271                         item = self.time_limit_layout.itemAt(i)
272 
273                         if isinstance(item.widget(), QWidget):
274                             item.widget().setVisible(True)
```

```
275
276                 self.button_script_validate_input.setText("Gerar pipeline")
277                 self.simple_or_complex_radio.setVisible(True)
278
279             else:
280                 self.simple_logic_array = []
281                 self.no_overlap_logic_array = []
282                 self.simple_logic_array = [SimpleInstruction() for _ in range(user_input)]
283                 self.no_overlap_logic_array = [NoOverlapInstruction() for _ in range(user_input)]
```

Caso 1 verifica se o usuário preencheu todos os campos de cada instrução, e se os campos de quantidade contém valores numéricos válidos. Se a entrada for válida, as arrays que representam as instruções são preenchidas com os valores.

Nota-se que o preenchimento da variação no overlap é um pouco mais complexo, com o parâmetro de um índice de controle onde cada instrução começa e termina, para que seja possível realizar a manipulação do pipeline sem que duas instruções iguais não ocorram ao mesmo tempo.

Além disso, o estado progride, o que tranca todos os campos de cada instrução individual para que não sejam alteradas novamente e a entrada da quantidade total de tempo é habilitada ao usuário, por fim o botão de continuar tem seu texto modificado para “gerar pipeline”.

```
285         case 2:
286             user_input = self.time_limit_layout_input.text()
287
288             if not self.is_integer(user_input):
289                 QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor numerico.")
290                 return
291             elif not self.is_integer_float(user_input):
292                 QMessageBox.warning(self, "Valor invalido!", f"Por favor, insira um valor inteiro.")
293                 return
294             else:
295                 user_input = int(user_input)
296
297             if(self.simple_or_complex_radio.isChecked() == True):
298                 result, loops = run_instructions_simple(self.simple_logic_array, user_input, self.longest_input_string)
299             else:
300                 result, loops = run_instructions_no_overlap(self.no_overlap_logic_array, user_input, self.longest_input_string)
301
302             result.append(f"==================================================")
303             result.append(f"Total de instrucoes - {loops}")
304
305             for i in range(self.output_layout.count()):
306                 item = self.output_layout.itemAt(i)
307
308                 if isinstance(item.widget(), QWidget):
309                     item.widget().setVisible(True)
310
311             self.output_contents_layout_label.setText('\n'.join(result))
```

Caso 2 verifica se o usuário inseriu uma quantidade de tempo total numérica válida. Se a verificação for positiva, o diagrama de tempo do pipeline de instruções será gerado conforme o estado do botão “Permitir Overlap?” e o

resultado será mostrado na tela para o usuário através do preenchimento do QLabel de output.

```
313     def reset(self):
314         self.state = 0
315         self.simple_logic_array = []
316         self.no_overlap_logic_array = []
317         self.longest_input_string = 0
318         self.instruction_inputs.clear()
319 
320         self.number_of_instructions_layout_input.setText("")
321         self.time_limit_layout_input.setText("")
322         self.output_contents_layout_label.setText("")
323 
324         self.number_of_instructions_layout_input.setDisabled(False)
325         self.time_limit_layout_input.setDisabled(False)
326 
327         while self.instructions_layout.count():
328             layout_item = self.instructions_layout.takeAt(0)
329             if layout_item:
330                 item_layout = layout_item.layout()
331                 if item_layout:
332                     while item_layout.count():
333                         item = item_layout.takeAt(0)
334                         widget = item.widget()
335                         if widget:
336                             widget.deleteLater()
337                         else:
338                             pass
339                     item_layout.deleteLater()
```

```
340                 else:
341                     widget = layout_item.widget()
342                     if widget:
343                         widget.deleteLater()
344 
345         self.instructions_layout_label = QLabel("<h2>Instrucoes</h2>")
346         self.instructions_layout_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
347         self.instructions_layout.addWidget(self.instructions_layout_label)
348         self.manual_input_layout.addLayout(self.instructions_layout)
349 
350         for i in range(self.instructions_layout.count()):
351             item = self.instructions_layout.itemAt(i)
352 
353             if isinstance(item.widget(), QWidget):
354                 item.widget().setVisible(False)
355 
```

```
356         for i in range(self.time_limit_layout.count()):
357             item = self.time_limit_layout.itemAt(i)
358 
359             if isinstance(item.widget(), QWidget):
360                 item.widget().setVisible(False)
361 
362         for i in range(self.output_layout.count()):
363             item = self.output_layout.itemAt(i)
364 
365             if isinstance(item.widget(), QWidget):
366                 item.widget().setVisible(False)
367 
368         self.button_script_validate_input.setText("Continuar")
369         self.simple_or_complex_radio.setVisible(False)
```

Por fim, a 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 instruções e restaurando o QLabel inicial deste item, retornando o botão de “gerar pipeline” para “continuar” novamente e habilitando a interação com itens previamente trancados para que novas instruções sejam inseridas.

- **Em resumo, multiplication_input_view.py:**

Cria um widget que pode ser instanciado e adicionado para o layout em pilha da classe principal. Este widget recebe os valores de entrada do usuário para estabelecer os parâmetros de quantidade de tempo total e instruções e após sua verificação executam a lógica para criar um diagrama de tempo do pipeline destas instruções.

file_input_screen.py

Lógica similar a tela de entrada manual, porém relativamente simplificada para a inserção através de arquivo. A maior parte do conteúdo visível é composta por um QTextEdit que deve ser preenchido no seguinte formato:

[NOME DA INSTRUCAO],[QUANTIDADE DE TEMPO]

…

[NOME DA INSTRUCAO],[QUANTIDADE DE TEMPO]

[TEMPO TOTAL]

Onde as primeiras N linhas correspondem ao nome de cada instrução e seu tempo de execução, separados por vírgula e a última linha corresponde ao tempo total do diagrama.

As demais diferenças estão na lógica de verificar e aceitar a entrada do usuário conforme o formato descrito acima. Os botões servem para a mesma finalidade.

logic_simple.py

A lógica simplificada assume que cada instrução é subdividida em sub instruções que são distintas entre si, desta forma apenas o deslocamento de uma unidade de tempo é necessário. Para que o exemplo seja mais visualmente claro:

A A B B C C X X X
X A A B B C C X X
X X A A B B C C X
X X X A A B B C C

- **Criando a classe SimpleInstruction:**

```
1   class SimpleInstruction:
2       def __init__(self):
3           self.name = ""
4           self.size = 0
5
6       def get_name(self):
7           return self.name
8
9       def get_size(self):
10          return self.size
11
12      def set_name(self, name):
13          self.name = name
14
15      def set_size(self, size):
16          self.size = size
```

Classe que armazena os parâmetros de uma instrução simples, nome e tamanho.

```
18  def run_instructions_simple(instructions_array, cycles, longest_instruction_string):
19      log = []
20      number_of_instructions = len(instructions_array)
21
22      spaces = ""
23      for i in range(longest_instruction_string):
24          spaces += " "
25
26      loops = 0
```

```
28      while True:
29          current_index = 0
30          times_printed = 0
31          iteration_string = ""
32
33          for i in range(cycles):
34              if(i < loops):
35                  iteration_string += "X" + spaces
36              else:
37                  if(current_index < number_of_instructions):
38                      if(times_printed < instructions_array[current_index].get_size()):
39                          iteration_string += (instructions_array[current_index].get_name() + spaces)
40                          times_printed += 1
41                      else:
42                          times_printed = 0
43                          current_index += 1
44                          if(current_index < number_of_instructions):
45                              iteration_string += (instructions_array[current_index].get_name() + spaces)
46                              times_printed += 1
47                          else:
48                              iteration_string += "X" + spaces
49                  else:
50                      iteration_string += "X" + spaces
```

```
52          loops += 1
53
54          log.append(iteration_string)
55
56          if(current_index < number_of_instructions):
57              break
58
59      return log, loops
```

Lógica de execução de instruções na variação simples que recebe uma array da classe de instruções, o tempo total e o tamanho da maior instrução como argumentos.

O tempo total define o limite de colunas, enquanto o tamanho da maior instrução serve como parâmetro para criar o espaçamento entre cada instrução.

Loop while se repete até que a última instrução “encoste” na última coluna de tempo, de forma que a próxima linha estaria fora deste limite estabelecido pelo usuário.

Uma array de instruções pode ser imaginada como uma fita, por exemplo se o usuário inseriu 3 instruções, “A” com o tamanho 1, “B” com o tamanho 2 e “C” com o tamanho 3, podemos pensar em uma fita destas instruções como:

“ABBCCC”

current_index é uma variável local para controlar a posição desta “fita” e times_printed auxilia no controle de instruções com diferentes tamanhos.

Pois bem, a variável local “loops” controla o offset de colunas antes das instruções, que inicia-se em 0, e posteriormente adiciona a quantidade de X correspondente a seu valor, ou seja, na primeira linha de nosso exemplo:

A B B C C C

Enquanto na terceira:

X X X A B B C C C

O current_index verifica qual das 3 instruções do nosso exemplo está sendo “imprimida”, enquanto o times_printed controla o seu tamanho, por exemplo a instrução “B” de nosso exemplo deve ser imprimida 2 vezes, e a C 3 vezes antes de passar para a próxima instrução. Após todas as instruções serem imprimidas as demais colunas serão “X”.

logic_no_overlap.py

A lógica no overlap rejeita a ideia de subdivisão de instruções, ou seja, deve-se certificar que uma instrução é concluída em sua totalidade antes de adicionar outra na pipeline para evitar conflitos. Para que o exemplo seja mais visualmente claro:

A A B B B C X X X X X X X X X
X X A A X B B B C X X X X X X
X X X X A A X X B B B C X X X
X X X X X X A A X X X B B B C

- **Criando a classe NoOverlapInstruction**:

```
class NoOverlapInstruction:
    def __init__(self):
        self.name = ""
        self.size = 0
        self.starting_index = 0
        self.ending_index = 0

    def increment_indexes(self):
        self.starting_index += self.size
        self.ending_index += self.size

    def get_name(self):
        return self.name

    def get_size(self):
        return self.size

    def get_starting_index(self):
        return self.starting_index

    def get_ending_index(self):
        return self.ending_index

    def set_name(self, name):
        self.name = name

    def set_size(self, size):
        self.size = size

```

```
30      def set_starting_index(self, starting_index):
31          self.starting_index = starting_index
32
33      def set_ending_index(self, ending_index):
34          self.ending_index = ending_index
```

Classe que armazena os parâmetros de uma instrução no overlap, nome, tamanho, índice de início, índice de término e um método para incrementar os índices.

Retornando a analogia das instruções como uma “fita”, se o usuário inserir “A” com tamanho 2, “B” com tamanho 3 e “C” com tamanho 1, temos:

AABBBC

Onde A tem o índice de início 0 e término 1, B tem início 2 e término 4 e C tem início e término 5.

```
36  def run_instructions_no_overlap(instructions_array, cycles, longest_instruction_string):
37      log = []
38      number_of_instructions = len(instructions_array)
39
40      spaces = ""
41      for i in range(longest_instruction_string):
42          spaces += " "
43
44      loops = 0
45
46      while True:
47          current_index = 0
48          iteration_string = ""
49
50          for i in range(cycles):
51              if(current_index < number_of_instructions):
52                  if (instructions_array[current_index].get_starting_index() > cycles):
53                      break
54
55                  if(i < instructions_array[current_index].get_starting_index()):
56                      iteration_string += "X" + spaces
57                  else:
58                      if(i <= instructions_array[current_index].get_ending_index()):
59                          iteration_string += (instructions_array[current_index].get_name() + spaces)
60                      else:
61                          instructions_array[current_index].increment_indexes()
62                          current_index += 1
63
64                          if(current_index < number_of_instructions):
65                              if(instructions_array[current_index].get_starting_index() > cycles):
66                                  break
```

```
67
68                                  if(i < instructions_array[current_index].get_starting_index()):
69                                      iteration_string += "X" + spaces
70                                  else:
71                                      iteration_string += (instructions_array[current_index].get_name() + spaces)
72                              else:
73                                  iteration_string += "X" + spaces
74                      else:
75                          iteration_string += "X" + spaces
76
77              loops += 1
78
79              log.append(iteration_string)
80
81              if(current_index < number_of_instructions):
82                  break
83
84      return log, loops
```

Lógica de execução de instruções na variação no overlap, similar a execução simples, porém utiliza os índices de início e fim de cada instrução para verificar se será “imprimido” um X ou o nome da instrução atual, desta forma isso efetivamente cria o efeito de evitar instruções repetidas na mesma coluna.