

Pedro Thiago Cutrim dos Santos  
Matrícula: 2120601

**Projeto Final de Programação: Um ferramenta  
para criação de bases de dados de imagens de  
satélite apropriadas para GANs**

Rio de Janeiro, Brasil

2023

Pedro Thiago Cutrim dos Santos  
Matrícula: 2120601

**Projeto Final de Programação: Um ferramenta para  
criação de bases de dados de imagens de satélite  
apropriadas para GANs**

Trabalho apresentado ao coordenador do programa de pós-graduação em informática da PUC-Rio como requisito para obtenção de nota na disciplina INF2102-Projeto Final de Programação

Pontifícia Universidade Católica do Rio de Janeiro  
Departamento de Informática  
Programa de Pós-Graduação em Informática

Orientador: Sérgio Colcher

Rio de Janeiro, Brasil  
2023

# Sumário

<b>Sumário</b>	<b>2</b>
<b>1 ESPECIFICAÇÃO DO PROGRAMA</b>	<b>3</b>
1.1 <b>Objetivo</b>	<b>3</b>
1.2 <b>Escopo</b>	<b>3</b>
1.3 <b>Requisitos</b>	<b>3</b>
1.3.1 Requisitos Funcionais	3
1.3.2 Requisitos Não-Funcionais	3
<b>2 ARQUITETURA</b>	<b>5</b>
<b>3 TESTES</b>	<b>8</b>
<b>4 DOCUMENTAÇÃO PARA O USUÁRIO</b>	<b>9</b>
4.1 <b>Telas de navegação</b>	<b>9</b>
<b>5 CÓDIGO FONTE</b>	<b>12</b>
5.1 <b><i>controller</i></b>	<b>12</b>
5.1.1 ./src/controller/process_controller.py	12
5.1.2 ./src/controller/thread_controller.py	15
5.2 <b><i>imgproc</i></b>	<b>17</b>
5.2.1 ./src/imgproc/aux.py	17
5.2.2 ./src/imgproc/generator.py	20
5.2.3 ./src/imgproc/intersection.py	21
5.3 <b><i>interface</i></b>	<b>23</b>
5.3.1 ./src/interface/main_window.py	23
5.3.2 ./src/interface/processing <sub>tem</sub> .py	28
5.3.3 ./src/tests/test.py	33

# 1 Especificação do Programa

## 1.1 Objetivo

Existe um campo envolvendo redes de super-resolução na qual aplica-se elas em imagens de satélite de baixa resolução espacial com o intuito de melhoramento na qualidade da cena. Uma das complicações deste processo é o de preparação dos dados para ficarem apropriados para uso de treinamento nas GANs, é necessário fazer um janelamento apropriado e georreferenciado com o seu par de alta resolução, já que é inviável o uso da cena completa como entrada da rede devido ao alto consumo de memória que isso geraria. Nesta ferramenta, a partir do envio de dois *rasters* pares, o usuário poderá gerar uma base de dados janelada e georreferenciada e salvar o resultado em disco.

## 1.2 Escopo

O escopo deste projeto é a produção de uma interface gráfica para produção de bases de dados de imagens de satélite apropriadas para a aplicação em redes de super-resolução. Dispondo de fácil uso e agilidade na criação de múltiplas bases. Otimizando o tempo do processo de geração dos arquivos e reduzindo ao máximo o contato direto do usuário com código.

## 1.3 Requisitos

### 1.3.1 Requisitos Funcionais

Foram estabelecidos três Requisitos Funcionais (RF):

- **RF-01 Operar em larga escala:** a ferramenta deve ser capaz de gerenciar diretórios de imagens densos.
- **RF-02 Interface gráfica:** construir um meio de interação gráfico com a ferramenta.
- **RF-03 Histórico de resultados:** a ferramenta deve prover por meio da interface o acesso ao histórico dos elementos anteriormente processados.

### 1.3.2 Requisitos Não-Funcionais

Os Requisitos Não Funcionais (RNF) desse projeto são:

- **RNF-01 Consumo de memória:** processar imagens de satélites (*rasters*) que possuem altas resoluções sem utilizar uma grande quantidade de memória (carregar parcialmente sempre que possível).
- **RNF-02 Usabilidade:** simplificar a interação do usuário com o código, permitindo que eles iniciem o processo e avaliem de maneiras simples o resultado. Toda interação do usuário deve ser respondida com algum *feedback* de confirmação visual.
- **RNF-03 Confiabilidade:** o processamento de cada *raster* deve resultar sempre no mesmo resultado, mesmo que a ordem dos *rasters* na geração da intersecção seja repetida.
- **RNF-04 Estabilidade:** a ferramenta deve ser eficiente em termos de uso de memória quando lidando com grandes bases de imagens, sem prejudicar o desempenho por longos períodos.

## 2 Arquitetura

A ferramenta foi desenvolvida utilizando o *Qt* no python através da biblioteca *PySide2* para produção da aplicação da interface desktop.

Para o backend, o programa utiliza as bibliotecas python *rasterio*, *Pillow* e *numpy*, responsáveis por fazer o processamento em cima dos *rasters*, como intersecção georreferenciada e o recorte das imagens.

O projeto está dividido em três diretórios: *misc*, *resources* e *src*. O diretório *misc* é responsável por manter arquivos variados não vitais para o projeto, *resources* possui todos os ícones e imagens utilizadas pela interface do programa. A pasta *src* possui o código fonte do programa, está dividida em 4 outras subpastas. O detalhamento da estrutura de diretórios do programa pode ser visto na Figura 1.

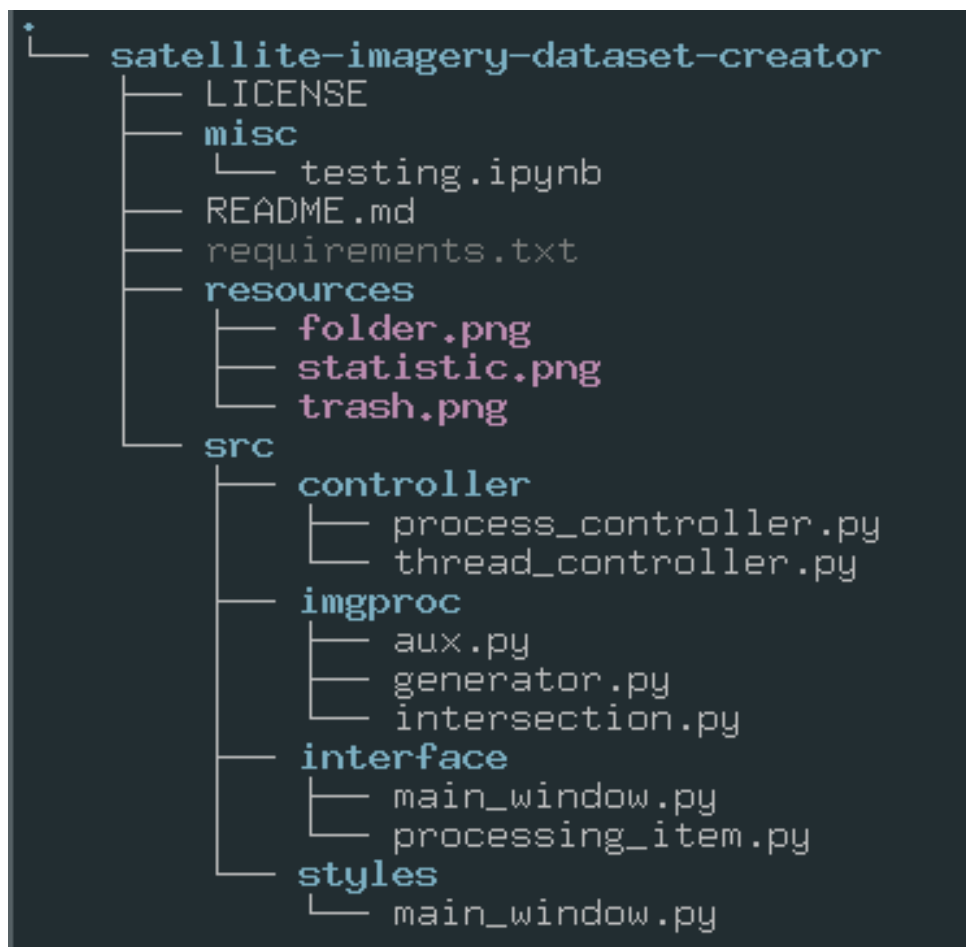


Figura 1 – Estrutura de diretórios do programa.

Os subdiretórios da pasta *src* podem ser descritos da seguinte maneira:

- *controller*: contém os scripts responsável pelo controlador do programa, que faz a

ligação da interface do programa com os dados do modelo e realiza os processamentos necessários.

- *imgproc*: contém as funções responsáveis pelo processamento dos *rasters* e outras finalidades auxiliares.
- *interface*: contém os scripts que implementa a interface Qt do programa.
- *styles*: contém arquivos que descrevem a estilização utilizadas nas janelas da interface.

A primeira interação do usuário com o programa é a de envio dos arquivos de *rasters*. Essa atividade é descrita no diagrama mostrado na Figura 2.

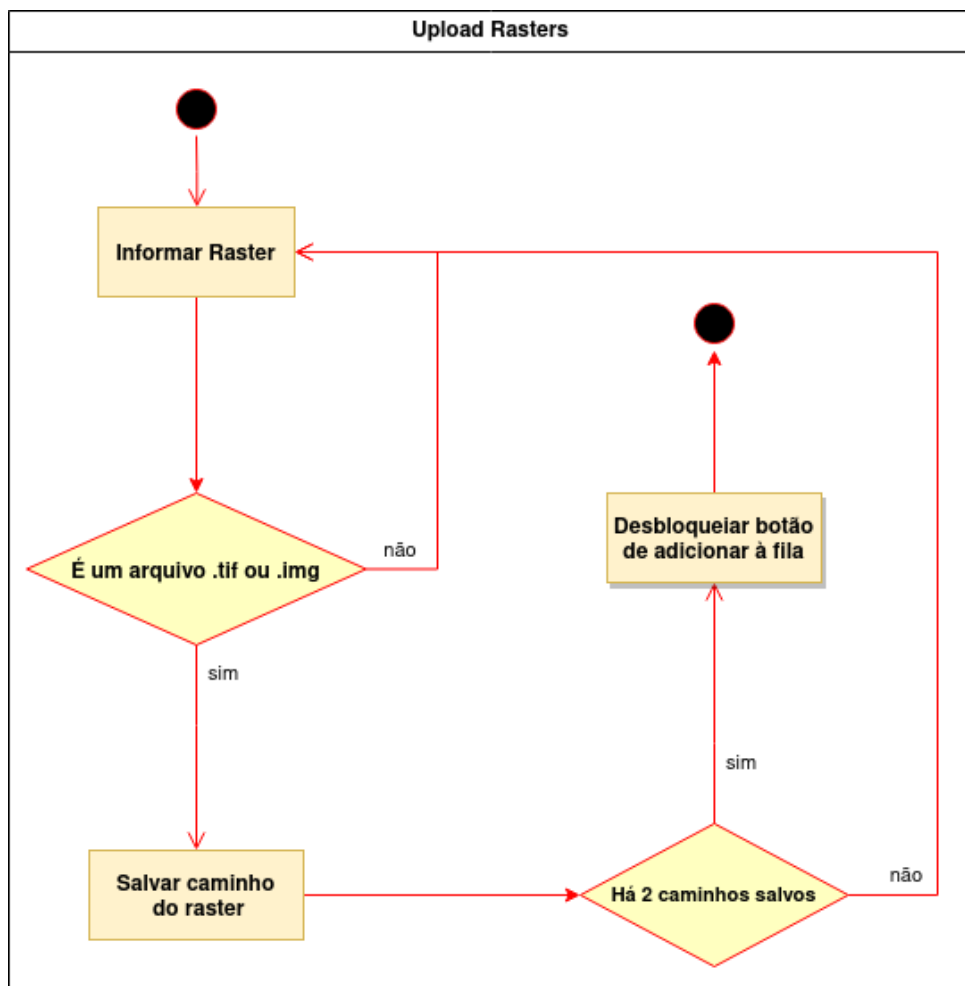


Figura 2 – Diagrama de atividade para *upload* de arquivos.

Após o envio dos arquivos para o programa, é criado um item de processamento no programa, onde o processamento é realizado da seguinte forma: (1) se necessário realiza-se a conversão do sistema de referência de coordenadas; (2) é feita a intersecção georreferenciada dos arquivos; (3) por fim o janelamento destas intersecções é gerado e salvo no sistema de arquivos.

O processamento de um item segue uma fila de prioridade, onde itens são processados pela ordem de criação. Após um item ser processado, ele pode ser deletado ou ter seus resultados acessados. A Figura 3 detalha este processo.

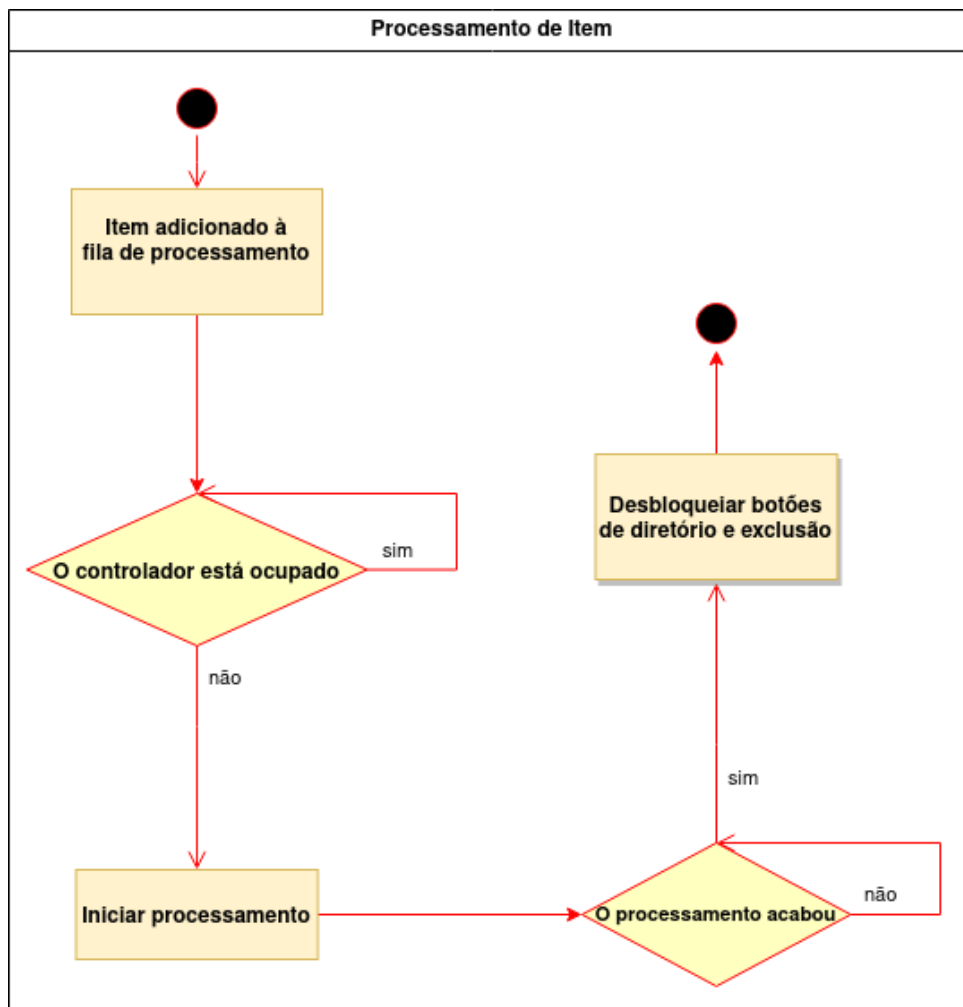


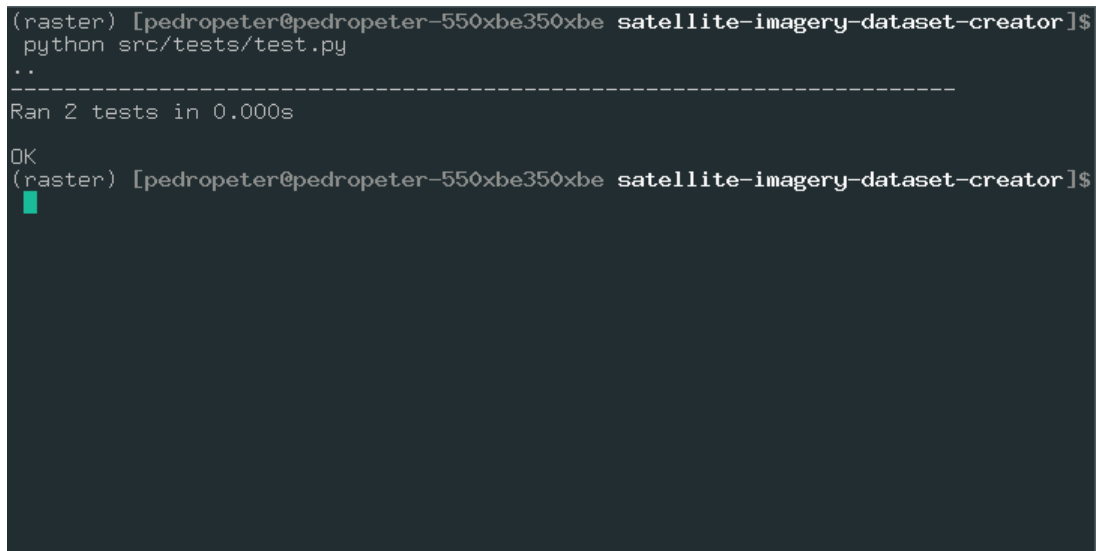
Figura 3 – Diagrama de atividade processamento de item.



## 3 Testes

Foram realizados dois testes para comprovar a integridade dos resultados gerados pela interface do programa. Estes testes de unidades consistem em verificar se os pares do janelamento dos *rasters* foram gerados na quantidade correta e também se cada par está completo.

A biblioteca utilizada para a realização dos testes foi a *unittest*.<sup>1</sup> O detalhamento do primeiro teste consiste em verificar se para cada item processado as pastas *lr* e *hr* foram geradas e possuem a mesma quantidade de arquivos. O segundo teste verifica se para cada um destes arquivos existentes em uma das pastas, o seu par está disponível na outra. A Figura 4 mostra o resultado do teste de unidade.

A terminal window with a dark background. The prompt is '(raster) [pedropeter@pedropeter-550xbe350xbe satellite-imagery-dataset-creator]\$'. The user enters 'python src/tests/test.py'. The output shows 'python src/tests/test.py' followed by '..' on the next line. A dashed line separates the command from the output. Below the dashed line, it says 'Ran 2 tests in 0.000s'. On the next line, it says 'OK'. The prompt is repeated at the bottom: '(raster) [pedropeter@pedropeter-550xbe350xbe satellite-imagery-dataset-creator]\$'. There is a small green cursor on the line following the second prompt.

```
(raster) [pedropeter@pedropeter-550xbe350xbe satellite-imagery-dataset-creator]$  
python src/tests/test.py  
..  
-----  
Ran 2 tests in 0.000s  
OK  
(raster) [pedropeter@pedropeter-550xbe350xbe satellite-imagery-dataset-creator]$
```

Figura 4 – Execução dos testes de unidade.

O código dos testes de unidade está disponível no repositório do programa disponibilizado pela documentação na Subseção 5.3.3.

---

<sup>1</sup> <https://docs.python.org/3/library/unittest.html>

## 4 Documentação para o Usuário

Antes de utilizar a aplicação, orienta-se a instalação, por parte do usuário, do Python e Git.<sup>1,2</sup> Após isso, o usuário deve realizar o download do código-fonte do projeto, contido em um repositório GitHub através do comando:<sup>3</sup>

```
$ git clone git@github.com:elheremes/satellite-imagery-dataset-creator.git
```

Ao executar esse comando, uma pasta com o nome *satellite-imagery-dataset-creator* no diretório de trabalho usado no terminal. O usuário deve entrar nessa pasta e instalar todas as dependências de bibliotecas do Python para o programa:

```
$ pip install -r requirements.txt
```

Ao final, nesta mesma pasta o usuário pode executar o programa utilizando o comando:

```
$ python -m src
```

### 4.1 Telas de navegação

O programa conta com apenas uma tela principal dividida em duas seções importantes, a de carregamento e envio dos *rasters* e a de histórico e progresso de processamento dos resultados. Inicialmente o usuário deve fazer o envio dos arquivos e adicioná-los a fila de processamento, este processo de abertura de programa até adição do item pode ser visto nas Figuras 5, 6 e 7.

Não é possível acessar o resultado do processamento até que seu progresso termine, a possibilidade de exclusão também só fica disponível depois das mesmas condições. Quando o item termina de processar, os botões de acesso ao diretório de resultado e exclusão deixam de ser transparentes e ficam clicáveis, como pode ser visto na Figura 8.

O programa disponibiliza um histórico de resultados de todos os itens que já foram processados em sua tela principal. Este histórico permanece mesmo se o programa for fechado e reaberto em uma outra ocasião. Este caso pode ser visto na Figura 9.

---

<sup>1</sup> <https://www.python.org/downloads/>

<sup>2</sup> <https://git-scm.com/downloads>

<sup>3</sup> <https://github.com/>

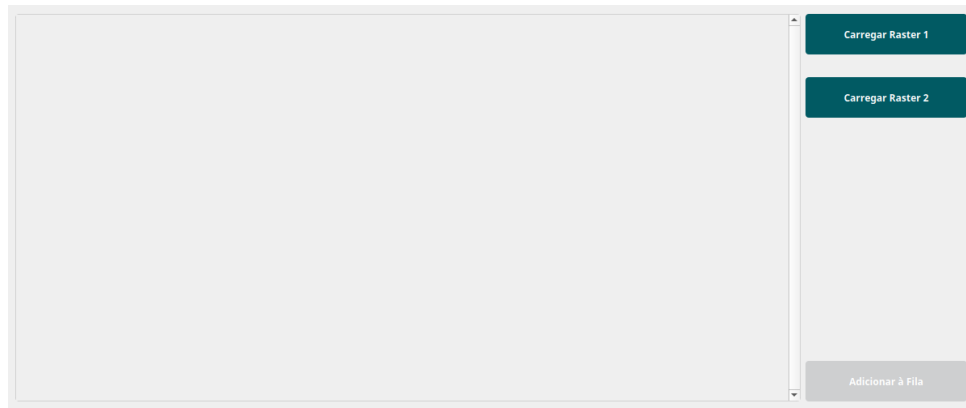


Figura 5 – Visão inicial do programa.

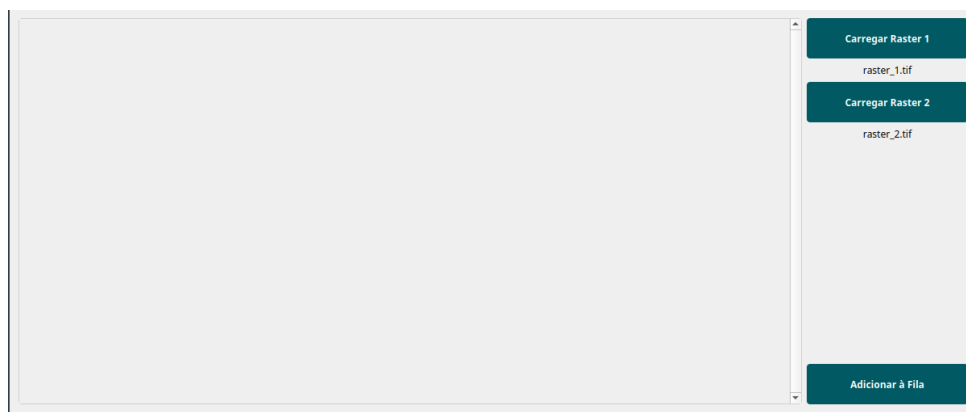


Figura 6 – Botão de adicionar à fila disponível após o upload de ambos os *rasters*.

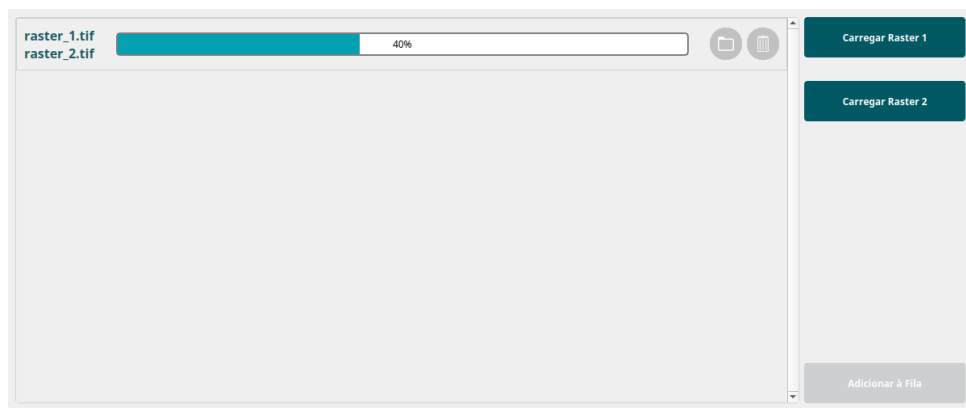


Figura 7 – Processamento de um item em progresso.

Por fim, o usuário pode excluir um item do histórico apertando o botão vermelho com símbolo de lixeira. Esta ação irá realizar a exclusão do item na interface e de seu conteúdo, a Figura 10 mostra um caso no qual o segundo item foi deletado pelo usuário.

Ao clicar no botão com símbolo de pasta do item, o programa irá abrir os resultados no sistema de arquivos padrão do sistema. Dentro desta pasta irá conter outras duas subpastas com nomes: *lr* e *hr*, onde cada uma dessas pastas contém respectivamente o janelamento dos *rasters* de menor e maior resolução espacial.

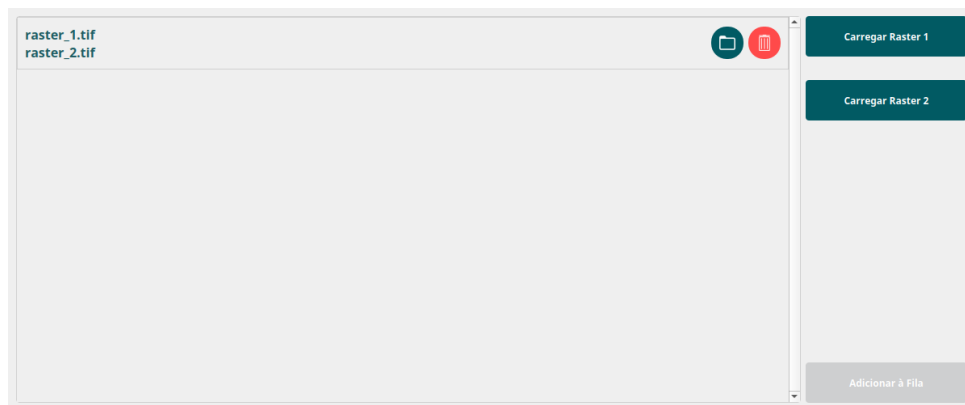


Figura 8 – Item finalizado, botões de acesso ao diretório e exclusão acessíveis.

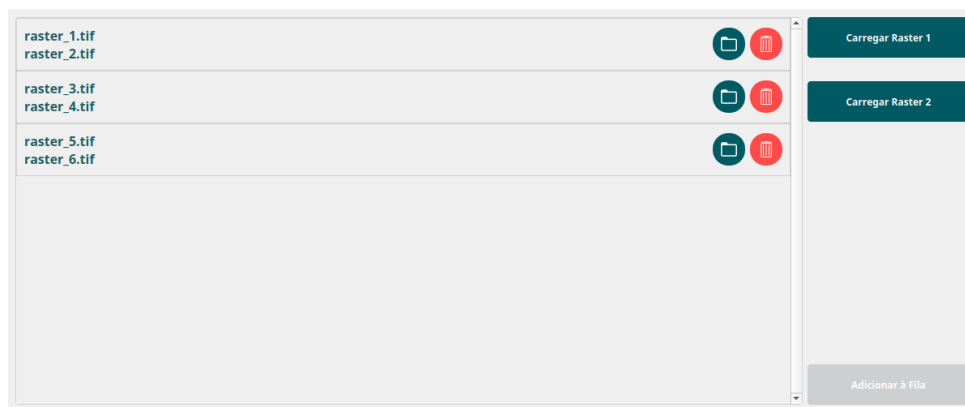


Figura 9 – Histórico de itens processados.

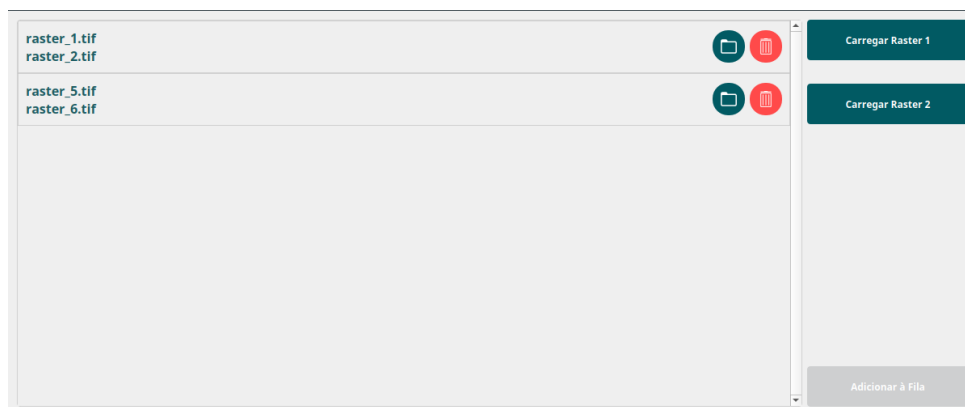


Figura 10 – Histórico após a exclusão do segundo item.

## 5 Código Fonte

O código completo deste projeto pode ser encontrado no GitHub.<sup>1</sup> Além disso, todas as funções incluídas foram detalhadamente descritas nas seções subsequentes.

### 5.1 *controller*

#### 5.1.1 `./src/controller/process_controller.py`

```
1  """
2  [AUTOR]
3      Pedro Thiago Cutrim dos Santos
4      Github: @elheremes
5
6  [DESCRIÇÃO]
7      Controlador responsável pela comunicação dos
8      processos de backend e a interface.
9
10     Este script faz a gestão da fila de processos,
11     salvamento de arquivos e criação de threads para
12     o programa.
13 """
14
15 import os
16 import numpy as np
17 from PIL import Image
18
19 from PySide2.QtCore import QThreadPool
20
21 from src.imgproc import (
22     intersect_rasters,
23     reproject_raster,
24     get_spatial_resolution,
25     generate_image_slices
26 )
27
28 from src.controller import (
29     Worker
30 )
31
32
33 class ProcessController:
```

<sup>1</sup> <https://github.com/elheremes/satellite-imagery-dataset-creator>

```

34     """
35     Classe para o controlador de processos.
36
37     Gerencia a fila de processos enviados pelo usuário,
38     escolhe qual processo deve ser executado e salva os
39     arquivos de resultado.
40
41     [ATRIBUTOS]
42         busy: booleano que informa se o controlador está
43             executando um processo ou não.
44         queue: lista com a fila de processos não ainda
45             executados.
46         thread_pool: QThreadPool para criação de threads de
47             processo para cada elemento.
48     """
49
50     def __init__(self):
51         self.busy = False
52         self.queue = []
53         self.thread_pool = QThreadPool()
54
55     def append_to_queue(self, process_item):
56         """
57         Adiciona o item a fila processamento.
58
59         [ARGUMENTOS]
60             process_item: classe ProcessingItem
61                         com informações do processo.
62         """
63
64         self.queue.append(process_item)
65         self.check_queue()
66
67     def check_queue(self):
68         """
69         Verifica o status da fila, caso o controlador
70         já esteja com um processo em execução ou a fila
71         de processamento esteja vazia, nada acontece. Caso
72         contrário, uma thread de execução é criada e o status
73         do controlador muda para ocupado.
74         """
75
76         if not self.busy and len(self.queue) > 0:
77             self.busy = True
78             worker = Worker(self.process_video)
79

```

```

80     worker.signals.progress.connect(self.queue[0].update_progress)
81
82         self.thread_pool.start(worker)
83
84     def register_end_process(self):
85         """
86         Remove o processo da lista, muda o status do controlador
87         para disponível e faz uma nova verificação na fila.
88         """
89
90         self.queue.pop(0)
91         self.busy = False
92         self.check_queue()
93
94     def process_video(self, progress_callback):
95         """
96         Realiza a intersecção dos rasters e gera o janelamento
97         de imagens dos resultados.
98
99         [ARGUMENTOS]
100             progress_callback: função de callback para atualização
101                               de progresso na interface.
102         """
103
104         process = self.queue[0]
105
106         process.disable_trash_btn()
107
108         process.start_processing()
109
110         res1 = get_spatial_resolution(process.raster1_name)
111         res2 = get_spatial_resolution(process.raster2_name)
112
113         overlap_1, overlap_2, bands = intersect_rasters(
114             process.raster1_name, process.raster2_name)
115
116         os.remove('tmp/temporary.tif')
117         os.remove('tmp/temporary.tif.aux.xml')
118
119         progress_callback.emit(40)
120
121         if bands == 1:
122             overlap_1 = overlap_1.reshape(
123                 (overlap_1.shape[1], overlap_1.shape[2]))
124             overlap_2 = overlap_2.reshape(
125                 (overlap_2.shape[1], overlap_2.shape[2]))

```

```

126
127         overlap_1 = np.asarray(Image.fromarray(overlap_1 * 255),
'L')
128         overlap_2 = np.asarray(Image.fromarray(overlap_2 * 255),
'L')
129     elif bands == 3:
130         overlap_1 = overlap_1.reshape(
131             (overlap_1.shape[1], overlap_1.shape[2],
overlap_1.shape[0]))
132         overlap_2 = overlap_2.reshape(
133             (overlap_2.shape[1], overlap_2.shape[2],
overlap_2.shape[0]))
134
135         overlap_1 = np.asarray(Image.fromarray(overlap_1 * 255),
'RGB')
136         overlap_2 = np.asarray(Image.fromarray(overlap_2 * 255),
'RGB')
137
138         progress_callback.emit(50)
139
140         generate_image_slices(overlap_1, overlap_2, res1, res2,
141                               process.folder, lr_size=128)
142
143         with open('{}/names.txt'.format(process.folder), 'w') as f:
144             f.write(process.raster1_name + '\n')
145             f.write(process.raster2_name + '\n')
146
147         process.update_progress(100)
148
149         self.register_end_process()
150
151         process.end_processing(None)
152
153         process.enable_trash_btn()

```

### 5.1.2 ./src/controller/thread\_controller.py

```

1  """
2  [AUTOR]
3      Pedro Thiago Cutrim dos Santos
4      Github: @elheremes
5
6  [DESCRIÇÃO]
7      Controlador responsável pela definição das classes
8      de Threads na padronização do QT PySide2.
9  """
10

```



```

11 from PySide2.QtCore import (
12     QObject,
13     QRunnable,
14     Signal,
15     Slot
16 )
17
18 import traceback
19 import sys
20
21
22 class WorkerSignals(QObject):
23     """
24     Define os sinais disponíveis de um thread de trabalho em execução.
25     Os sinais suportados são:
26
27     Finalizado [finished]: sem dados.
28     Error [error]: tupla (exctype, valor, traceback.format_exc()).
29     Resultado [result]: objeto de dados retornado do processamento,
30         qualquer coisa.
31     Progresso [progress]: inteiro indicando % de progresso.
32     """
33
34     finished = Signal()
35     error = Signal(tuple)
36     result = Signal(object)
37     progress = Signal(int)
38
39
40 class Worker(QRunnable):
41     """
42     Thread do Worker;
43     Herda de QRunnable para lidar com a configuração de threads de
44     trabalho,
45     sinais e conclusão.
46
47     [ATRIBUTOS]
48     fn: função de retorno a ser executada na thread de trabalho.
49     args: argumentos para passa para a função de retorno.
50     kwargs: palavras-chave para passar para função de retorno.
51     signals: classe WorkerSignals para uso de sinais.
52     """
53
54     def __init__(self, fn, *args, **kwargs):
55         super(Worker, self).__init__()
56
57         self.fn = fn

```

```

57         self.args = args
58         self.kwargs = kwargs
59         self.signals = WorkerSignals()
60
61         self.kwargs['progress_callback'] = self.signals.progress
62
63     @Slot()
64     def run(self):
65         """
66         Inicializa a execução da função informada passando os
67         argumentos e palavras-chave.
68
69         Ao final emite um sinal para o programa principal
70         informando o status da thread.
71         """
72
73         try:
74             result = self.fn(*self.args, **self.kwargs)
75         except:
76             traceback.print_exc()
77             exctype, value = sys.exc_info()[:2]
78             self.signals.error.emit((exctype, value,
79                                     traceback.format_exc()))
80         else:
81             # Return the result of the processing
82             self.signals.result.emit(result)
83         finally:
84             self.signals.finished.emit() # Done

```

## 5.2 *imgproc*

### 5.2.1 `./src/imgproc/aux.py`

```

1  """
2  [AUTOR]
3      Pedro Thiago Cutrim dos Santos
4      Github: @elheremes
5
6  [DESCRIÇÃO]
7      Script responsável por conter funções auxiliares para
8      utilização de ambos backend e frontend do programa.
9  """
10
11 import rasterio
12 import os
13

```

```

14 from rasterio.warp import (
15     calculate_default_transform,
16     reproject,
17     Resampling
18 )
19
20
21 def reproject_raster(raster, dst_crs):
22     """
23     Reprojeta o CRS original do raster para o CRS informado.
24
25     [ARGUMENTOS]
26         raster: DatasetReader do raster carregado pelo rasterio.
27         dst_crs: string informando o sistema de coordenadas desejado.
28
29     [RETORNO]
30         Retorna um DatasetReader convertido para o CRS desejado.
31     """
32
33     transform, width, height = calculate_default_transform(
34         raster.crs, dst_crs, raster.width, raster.height,
35         *raster.bounds)
36
37     kwargs = raster.meta.copy()
38
39     kwargs.update({
40         'crs': dst_crs,
41         'transform': transform,
42         'width': width,
43         'height': height
44     })
45
46     with rasterio.open('tmp/temporary.tif', 'w', **kwargs) as dst:
47         for i in range(1, raster.count + 1):
48             reproject(
49                 source=rasterio.band(raster, i),
50                 destination=rasterio.band(dst, i),
51                 src_transform=raster.transform,
52                 src_crs=raster.crs,
53                 dst_transform=transform,
54                 dst_crs=dst_crs,
55                 resampling=Resampling.nearest)
56
57     raster.close()
58
59     return rasterio.open('tmp/temporary.tif')

```

```

60
61 def get_spatial_resolution(raster_path):
62     """
63     Obtém a resolução espacial do raster informado.
64
65     [ARGUMENTOS]
66         raster_path: caminho para o arquivo raster.
67
68     [RETORNO]
69         Retorna em inteiro a resolução espacial do
70         raster (em metros).
71     """
72
73     with rasterio.open(raster_path) as raster:
74         return int(raster.res[0])
75
76
77 def create_folder(path):
78     """
79     Verifica se a pasta existe e, caso não, realiza
80     a criação do diretório.
81
82     [ARGUMENTOS]
83         path: caminho do diretório a ser checado ou
84         criado.
85     """
86
87     if not os.path.exists(path):
88         os.makedirs(path)
89
90
91 def get_list_of_process_ids():
92     """
93     Lista todos os ids de processos já executados e
94     não excluídos pelos usuários na pasta temporária.
95
96     [RETORNO]
97         Uma lista com os ids em inteiro dos processos
98         salvos.
99     """
100
101     return [int(name)
102             for name in os.listdir("tmp/")]
103
104
105 def get_new_id():
106     """

```

```

107     Verifica o id do último processo salvo e retorna
108     o próximo disponível.
109
110     [RETORNO]
111         Inteiro com o valor do próximo id disponível.
112     """
113
114     created_ids = get_list_of_process_ids()
115
116     if len(created_ids) == 0:
117         return 1
118     else:
119         return max(created_ids) + 1

```

### 5.2.2 ./src/imgproc/generator.py

```

1     """
2     [AUTOR]
3         Pedro Thiago Cutrim dos Santos
4         Github: @elheremes
5
6     [DESCRIÇÃO]
7         Script responsável por realizar e salvar o janelamento
8         das interseções de alta e baixa resolução dos rasters.
9     """
10
11     from PIL import Image
12     import numpy as np
13
14     from src.imgproc import get_spatial_resolution, create_folder
15
16
17     def generate_image_slices(ar1, ar2, res1, res2, save_dir, lr_size=128):
18         """
19         Realiza o janelamento dos arrays numpy e salva em
20         formato de imagens png, separando-os em alta resolução
21         e baixa resolução através do fator calculado entre as
22         resoluções espaciais de ambos rasters.
23
24         [ARGUMENTOS]
25             ar1: numpy array contendo os valores da interseção do
26                 primeiro raster.
27             ar2: numpy array contendo os valores da interseção do
28                 segundo raster.
29             res1: resolução espacial em inteiro do primeiro raster.
30             res2: resolução espacial em inteiro do segundo raster.
31             lr_size: resolução (lr_size x lr_size) da imagem de baixa

```

```

32         qualidade.
33         save_dir: diretório onde serão salvos os resultados.
34     """
35
36     images_per_axis = int(ar1.shape[0] / lr_size)
37     resolution_factor = int(res1 / res2)
38     hr_size = lr_size * resolution_factor
39
40     count_slice = 1
41
42     create_folder(save_dir)
43     create_folder(save_dir + "/lr")
44     create_folder(save_dir + "/hr")
45
46     for i in range(images_per_axis):
47         for j in range(images_per_axis):
48             slice1 = Image.fromarray(
49                 np.uint8(ar1[i*lr_size:(i+1)*lr_size,
50 j*lr_size:(j+1)*lr_size]), 'L')
51
52             slice2 = Image.fromarray(
53                 np.uint8(ar2[i*hr_size:(i+1)*hr_size,
54 j*hr_size:(j+1)*hr_size]), 'L')
55
56             slice1.save("{}lr/slice_{}.png".format(save_dir,
57 count_slice))
58             slice2.save("{}hr/slice_{}.png".format(save_dir,
59 count_slice))
60
61             count_slice = count_slice + 1

```

### 5.2.3 ./src/imgproc/intersection.py

```

1     """
2     [AUTOR]
3         Pedro Thiago Cutrim dos Santos
4         Github: @elheremes
5
6     [DESCRIÇÃO]
7         Script responsável por gerar a interseção georreferenciada
8         entre dois rasters informados.
9
10        Os rasters podem ter 1 ou 3 bandas, sendo necessário a mesma
11        quantidade de bandas para ambos os rasters que irão ser comparados.
12    """
13
14

```

```

15 import rasterio
16 from shapely.geometry import box
17
18 from src.imgproc import reproject_raster
19
20
21 def intersect_rasters(raster1_path, raster2_path):
22     """
23     Calcula a interseção georreferenciada entre dois rasters de
24     1 ou 3 bandas.
25
26     Ambos os rasters precisam ter a mesma quantidade de bandas.
27
28     [ARGUMENTOS]
29         raster1_path: caminho para o arquivo do primeiro raster.
30         raster2_path: caminho para o arquivo do segundo raster.
31
32     [RETORNO]
33         Retorna uma tripla possuindo 2 arrays numpy (overlap_1 e
34         overlap_2) onde cada um equivale aos valores dos rasters
35         que caem na interseção georreferenciada. O último valor
36         da tripla (bands) trata-se da quantidade de bandas dos
37         rasters informados para a função.
38     """
39
40     ras1 = rasterio.open(raster1_path)
41     ras2 = rasterio.open(raster2_path)
42
43     if ras1.count != ras2.count:
44         raise Exception("Número de bandas entre os dois rasters é
45         diferente. ({0} != {1})".format(
46             ras1.count, ras2.count))
47
48     aux_ras = reproject_raster(ras2, ras1.crs)
49     ras2.close()
50     ras2 = aux_ras
51
52     ext1 = box(*ras1.bounds)
53     ext2 = box(*ras2.bounds)
54
55     intersection = ext1.intersection(ext2)
56
57     win1 = rasterio.windows.from_bounds(*intersection.bounds,
58     ras1.transform)
59     win2 = rasterio.windows.from_bounds(*intersection.bounds,
60     ras2.transform)

```

```

59     overlap_1 = ras1.read(window=win1)
60     overlap_2 = ras2.read(window=win2)
61
62     bands = ras1.count
63
64     ras1.close()
65     ras2.close()
66
67     return (overlap_1, overlap_2, bands)

```

## 5.3 *interface*

### 5.3.1 `./src/interface/main_window.py`

```

1  """
2  [AUTOR]
3      Pedro Thiago Cutrim dos Santos
4      Github: @elheremes
5
6  [DESCRIÇÃO]
7      Script responsável pela criação da tela principal do programa.
8
9      A tela é responsável pelo carregamento dos rasters, visualização
10     da fila e do status dos processos e acesso aos resultados.
11  """
12
13  from PySide2.QtWidgets import (
14      QWidget,
15      QPushButton,
16      QFileDialog,
17      QVBoxLayout,
18      QHBoxLayout,
19      QScrollArea,
20      QLabel
21  )
22
23  from PySide2.QtCore import (
24      Qt
25  )
26
27  from PySide2.QtGui import (
28      QCursor
29  )
30
31  from src.interface import (
32      ProcessingItem

```



```

33 )
34
35 from src.controller import (
36     ProcessController
37 )
38
39 from src.imgproc import (
40     get_list_of_process_ids,
41     get_new_id
42 )
43
44
45 class MainWindow(QWidget):
46     """
47     Tela principal da interface do programa.
48
49     Recebe a upload dos rasters e mostra a fila de processamento
50     e histórico de resultados.
51
52     [ATRIBUTOS]
53         raster1_name: caminho do primeiro raster enviado.
54         raster2_name: caminho do segundo raster enviado.
55         process_controller: Classe ProcessController responsável
56                             pela execução do backend.
57     """
58
59     def __init__(self, parent=None):
60         super(MainWindow, self).__init__(parent)
61
62         self.raster1_name = None
63         self.raster2_name = None
64
65         self.settings()
66         self.create_widgets()
67         self.set_layout()
68         self.add_widgets()
69
70         self.process_controller = ProcessController()
71
72         self.load_processed_videos()
73
74     def settings(self):
75         """
76         Definição de configurações básicas do Qt, como
77         tamanho e título da janela.
78         """
79

```

```

80         self.resize(1200, 500)
81         self.setWindowTitle("Raster GAN Dataset Creator")
82
83     def create_widgets(self):
84         """
85         Método responsável pela criação de botões, labels e
86         sinais presentes na interface da janela principal.
87         """
88
89         # Botões
90         self.btn_load_video_1 = QPushButton("Carregar Raster 1")
91         self.btn_load_video_1.setObjectName("loadVideoButton")
92         self.btn_load_video_1.setFixedWidth(200)
93         self.btn_load_video_1.setCursor(QCursor(Qt.PointingHandCursor))
94
95         self.btn_load_video_2 = QPushButton("Carregar Raster 2")
96         self.btn_load_video_2.setObjectName("loadVideoButton")
97         self.btn_load_video_2.setFixedWidth(200)
98         self.btn_load_video_2.setCursor(QCursor(Qt.PointingHandCursor))
99
100        self.btn_process = QPushButton("Adicionar à Fila")
101        self.btn_process.setObjectName("processVideoButton")
102        self.btn_process.setFixedWidth(200)
103        self.btn_process.setCursor(QCursor(Qt.PointingHandCursor))
104        self.btn_process.setDisabled(True)
105
106        # Labels
107        self.raster1_label = QLabel('')
108        self.raster1_label.setObjectName("regionLabel")
109        self.raster1_label.setAlignment(Qt.AlignCenter)
110        self.raster1_label.setDisabled(True)
111
112        self.raster2_label = QLabel('')
113        self.raster2_label.setObjectName("regionLabel")
114        self.raster2_label.setAlignment(Qt.AlignCenter)
115        self.raster2_label.setDisabled(True)
116
117        # Sinais
118        self.btn_load_video_1.clicked.connect(self.add_raster_1)
119        self.btn_load_video_2.clicked.connect(self.add_raster_2)
120        self.btn_process.clicked.connect(self.add_process_to_queue)
121
122    def set_layout(self):
123        """
124        Método responsável por definir o formato do layout
125        da janela principal.
126        """

```

```

127
128     self.scroll = QScrollArea()
129     self.scroll.setObjectName("videosContainer")
130     self.widget = QWidget()
131
132     self.process_layout = QVBoxLayout()
133     self.process_layout.setMargin(0)
134     self.process_layout.setSpacing(0)
135     self.process_layout.setContentsMargins(0, 0, 0, 0)
136     self.process_layout.setAlignment(Qt.AlignTop)
137
138     self.widget.setLayout(self.process_layout)
139     self.scroll.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
140     self.scroll.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
141     self.scroll.setWidgetResizable(True)
142     self.scroll.setWidget(self.widget)
143
144     self.buttons_layout = QVBoxLayout()
145     self.buttons_layout.setAlignment(Qt.AlignTop)
146
147     main_layout = QHBoxLayout()
148     main_layout.addWidget(self.scroll)
149     main_layout.addLayout(self.buttons_layout)
150
151     self.setLayout(main_layout)
152
153     def add_widgets(self):
154         """
155         Responsável por ligar os botões, sinais e caixas de
156         texto ao layout da tela principal.
157         """
158
159         self.buttons_layout.addWidget(self.btn_load_video_1)
160         self.buttons_layout.addWidget(self.raster1_label)
161         self.buttons_layout.addWidget(self.btn_load_video_2)
162         self.buttons_layout.addWidget(self.raster2_label)
163         self.buttons_layout.addStretch()
164         self.buttons_layout.addWidget(self.btn_process)
165
166     def add_raster_1(self):
167         """
168         Obtém o caminho em arquivo do primeiro raster informado
169         pelo usuário ao pressionar o botão.
170         """
171
172         filename, _ = QFileDialog.getOpenFileName(
173             self, "Selecione um raster", filter="TIFF(*.tiff *.tif)")

```

```

174
175         if filename == '':
176             return
177
178         self.raster1_name = filename
179
180         self.raster1_label.setText(filename.split('/')[-1])
181         self.raster1_label.setDisabled(False)
182
183         self.check_process_button()
184
185     def add_raster_2(self):
186         """
187         Obtém o caminho em arquivo do segundo raster informado
188         pelo usuário ao pressionar o botão.
189         """
190
191         filename, _ = QFileDialog.getOpenFileName(
192             self, "Selecione um raster", filter="TIFF(*.tiff *.tif)")
193
194         if filename == '':
195             return
196
197         self.raster2_name = filename
198
199         self.raster2_label.setText(filename.split('/')[-1])
200         self.raster2_label.setDisabled(False)
201
202         self.check_process_button()
203
204     def check_process_button(self):
205         """
206         Verifica se ambos os rasters já foram informados e habilita
207         o botão de enviar a fila caso verdadeiro.
208         """
209
210         if self.raster1_name is not None:
211             if self.raster2_name is not None:
212                 self.btn_process.setDisabled(False)
213
214     def add_process_to_queue(self):
215         """
216         Cria um objeto de processo e envia para a fila de processamento.
217         Além disso, limpa os dados de upload na interface e bloqueia o
218         botão
219         de envio para fila para evitar duplicações acidentais pelo
220         usuário.

```

```

219         """
220
221         process_id = get_new_id()
222
223         process = ProcessingItem(
224             raster1_name=self.raster1_name,
225             raster2_name=self.raster2_name,
226             folder='tmp/{}'.format(process_id))
227
228         self.raster1_name = None
229         self.raster2_name = None
230
231         self.raster1_label.setText('')
232         self.raster1_label.setDisabled(True)
233         self.raster2_label.setText('')
234         self.raster2_label.setDisabled(True)
235
236         self.btn_process.setDisabled(True)
237
238         self.process_layout.addWidget(process)
239         self.process_controller.append_to_queue(process)
240
241     def load_processed_videos(self):
242         """
243         Carrega o histórico de execuções presentes nos arquivos e os
244         manda
245         para interface.
246         """
247
248         ids = get_list_of_process_ids()
249
250         for i in ids:
251             with open('tmp/{}/names.txt'.format(i), 'r') as f:
252                 raster1_name = f.readline()[:-1]
253                 raster2_name = f.readline()[:-1]
254
255                 p = ProcessingItem(raster1_name=raster1_name,
256                                     raster2_name=raster2_name,
257                                     folder='tmp/{}'.format(i),
258                                     processed=True)
259
260                 self.process_layout.addWidget(p)

```

### 5.3.2 ./src/interface/processingitem.py

```

1     """
2     [AUTOR]

```

```

3     Pedro Thiago Cutrim dos Santos
4     Github: @elheremes
5
6     [DESCRIÇÃO]
7     Script responsável pela visualização do item de processamento
8     na interface.
9
10    Cada item de processamento mostra o status (%) do processamento
11    e se já finalizado o acesso a pasta de resultados. Também permite
12    a exclusão desse processo no histórico.
13    """
14
15    import shutil
16
17    from showinfm import show_in_file_manager
18
19    from PySide2.QtWidgets import (
20        QWidget,
21        QPushButton,
22        QLabel,
23        QProgressBar,
24        QGridLayout,
25        QVBoxLayout,
26        QFrame,
27    )
28
29    from PySide2.QtCore import (
30        Qt,
31        QSize
32    )
33
34    from PySide2.QtGui import (
35        QCursor,
36        QIcon
37    )
38
39
40    class ProcessingItem(QWidget):
41        """
42        Tela de item processado na interface.
43
44        Informa os nomes dos rasters presentes no processo,
45        progresso de processamento e atalhos para o usuário
46        poder acessar os arquivos e deletar o resultado.
47
48        [ATRIBUTOS]
49        processed_status: booleano informando se o item já foi

```

```

50         processado ou não.
51         raster1_name: caminho do primeiro raster.
52         raster2_name: caminho do segundo raster.
53         folder: caminho de salvamento dos dados gerados.
54     """
55
56     def __init__(self, raster1_name, raster2_name, folder,
57 processed=False, parent=None):
58
59         super(ProcessingItem, self).__init__(parent)
60
61         self.processed_status = processed
62         self.raster1_name = raster1_name
63         self.raster2_name = raster2_name
64         self.folder = folder
65
66         self.create_widgets()
67         self.set_layout()
68         self.add_widgets()
69
70         if processed:
71             self.processed_video()
72
73     def create_widgets(self):
74         """
75         Método responsável pela criação de botões, labels e
76         sinais presentes na interface do item de processamento.
77         """
78
79         # Labels
80         name_1 = self.raster1_name.split('/')[-1]
81         name_2 = self.raster2_name.split('/')[-1]
82
83         self.raster1_name_label = QLabel('{ }\n{ }'.format(name_1,
84 name_2))
85         self.raster1_name_label.setObjectName("videoLabel")
86
87         # Botões
88         self.statistic_btn = QPushButton()
89         self.statistic_btn.setIcon(QIcon("./resources/folder.png"))
90         self.statistic_btn.setIconSize(QSize(20, 20))
91         self.statistic_btn.setCursor(QCursor(Qt.PointingHandCursor))
92         self.statistic_btn.setDisabled(True)
93         self.statistic_btn.setFixedWidth(40)
94         self.statistic_btn.setFixedHeight(40)
95         self.statistic_btn.setObjectName("statisticButton")
96
97         self.trash_btn = QPushButton()

```

```

95     self.trash_btn.setIcon(QIcon("./resources/trash.png"))
96     self.trash_btn.setIconSize(QSize(20, 20))
97     self.trash_btn.setCursor(QCursor(Qt.PointingHandCursor))
98     self.trash_btn.setFixedWidth(40)
99     self.trash_btn.setFixedHeight(40)
100    self.trash_btn.setObjectName("trashButton")
101
102    self.progress_bar = QProgressBar()
103    self.progress_bar.hide()
104
105    # Sinais
106    self.trash_btn.clicked.connect(self.delete_element)
107    self.statistic_btn.clicked.connect(self.open_results)
108
109    def set_layout(self):
110        """
111        Método responsável por definir o formato do layout
112        do item de processamento.
113        """
114
115        container = QFrame()
116        container.setObjectName("videoContainer")
117
118        self.video_layout = QGridLayout()
119        container.setLayout(self.video_layout)
120
121        main_layout = QVBoxLayout()
122        main_layout.addWidget(container)
123        main_layout.setMargin(0)
124        main_layout.setContentsMargins(0, 0, 0, 0)
125
126        self.setLayout(main_layout)
127
128    def add_widgets(self):
129        """
130        Responsável por ligar os botões, sinais e caixas de
131        texto ao layout do item de processamento.
132        """
133
134        self.video_layout.addWidget(self.raster1_name_label, 0, 0)
135        self.video_layout.addWidget(self.progress_bar, 0, 1)
136        self.video_layout.addWidget(self.statistic_btn, 0, 3)
137        self.video_layout.addWidget(self.trash_btn, 0, 4)
138
139    def delete_element(self):
140        """
141        Deleta os arquivos em discos relacionado ao processo e

```



```

142         o exclui da interface.
143         """
144
145         shutil.rmtree(self.folder)
146
147         self.setParent(None)
148
149     def processed_video(self):
150         """
151         Habilita o botão para acessar os dados do resultado em
152         disco.
153         """
154
155         self.statistic_btn.setDisabled(False)
156
157     def open_results(self):
158         """
159         Abre os resultados no diretório do sistema.
160         """
161
162         show_in_file_manager(self.folder)
163
164     def start_processing(self):
165         """
166         Habilita a barra de progresso do processamento.
167         """
168
169         self.progress_bar.setValue(0)
170         self.progress_bar.show()
171
172     def end_processing(self, data):
173         """
174         Desabilita a barra de progresso do processamento.
175         """
176
177         self.progress_bar.hide()
178         self.statistic_btn.setDisabled(False)
179
180     def update_progress(self, value):
181         """
182         Atualiza o valor da barra de progresso do processamento.
183
184         [ARGUMENTOS]
185             value: inteiro que representa a porcentagem da
186                   barra de progresso.
187         """
188

```

```

189         self.progress_bar.setValue(value)
190
191     def enable_trash_btn(self):
192         """
193         Habilita o botão de exclusão do elemento.
194         """
195
196         self.trash_btn.setDisabled(False)
197
198     def disable_trash_btn(self):
199         """
200         Desabilita o botão de exclusão do elemento.
201         """
202
203         self.trash_btn.setDisabled(True)

```

### 5.3.3 ./src/tests/test.py

```

1  """
2  [AUTOR]
3      Pedro Thiago Cutrim dos Santos
4      Github: @elheremes
5
6  [DESCRIÇÃO]
7      Script responsável pelos testes de unidade do
8      programa.
9  """
10
11  import unittest
12  import os
13
14
15  class TestFilesQuantity(unittest.TestCase):
16      def test_results_size(self):
17          """
18          Este método realiza o teste verificando a quantidade
19          dos resultados de processamento salvos na pasta 'tmp'
20          do programa.
21
22          A verificação consiste em verificar se o número de
23          arquivos das pastas 'lr' e 'hr' coincidem, caso não
24          então o teste retorna um erro.
25          """
26
27          results_lr = []
28          results_hr = []
29

```

```

30         if os.path.isfile('tmp'):
31             ids = [int(name)
32                     for name in os.listdir("tmp/") if
os.path.isdir(os.path.join("tmp/", name))]
33
34             for i in ids:
35                 aux = len([name for name in
os.listdir("tmp/{}/lr/".format(i))
36                     if
os.path.isdir(os.path.join("tmp/{}/lr/".format(i), name))])
37
38                 results_lr.append(aux)
39
40                 aux = len([name for name in
os.listdir("tmp/{}/hr/".format(i))
41                     if
os.path.isdir(os.path.join("tmp/{}/hr/".format(i), name))])
42
43                 results_hr.append(aux)
44
45             self.assertEqual(results_lr, results_hr)
46
47
48 class TestFilesNames(unittest.TestCase):
49     def test_results_names(self):
50         """
51         Este método realiza o teste verificando se o nome dos
52         pares 'lr' e 'hr' coincidem em todos os itens de processamento.
53         """
54
55         results_lr = []
56         results_hr = []
57
58         if os.path.isfile('tmp'):
59             ids = [int(name)
60                     for name in os.listdir("tmp/") if
os.path.isdir(os.path.join("tmp/", name))]
61
62             for i in ids:
63                 aux = [name for name in
os.listdir("tmp/{}/lr/".format(i))
64                     if
os.path.isdir(os.path.join("tmp/{}/lr/".format(i), name))])
65
66                 results_lr.append(aux)
67
68                 aux = [name for name in

```

```
os.listdir("tmp/{}/hr/".format(i))
69         if
os.path.isdir(os.path.join("tmp/{}/hr/".format(i), name))]
70
71         results_hr.append(aux)
72
73         self.assertEqual(results_lr, results_hr)
74
75
76 if __name__ == '__main__':
77     unittest.main()
```