Pontifícia Universidade Católica do Rio de Janeiro

Departamento de Informática

INF2102 - Projeto Final de Programação

Gerador de Frames

Aluna: Debora Stuck

Professora: Clarisse Sieckenius de Souza

Orientador: Hélio Côrtes Vieira Lopes

04 de julho de 2022

Sumário

O projeto está disponível no link do github:
https://github.com/deborastuck/VideoFrame

I. Especificação do Programa

A. Escopo e requisitos

O escopo do projeto compreende a criação de uma ferramenta de pré-processamento de vídeos. Deve ser capaz de gerar frames e de gerar trechos específicos de vídeos. Deve ter uma interface gráfica para usuários utilizarem essas funções.

Por ser um projeto *open source*, o usuário poderá baixar o código e salvar em seu computador pelo *link* do *github*.

O executável estará disponível para *download* no *github* também, então, usuários que não tenham conhecimento em TI e não tenham interesse em visualizar o código, poderão utilizar a ferramenta para seus vídeos.

B. Especificação dos requisitos

Requisitos Funcionais:

RF1: O módulo deve ser capaz de gerar frames de um vídeo;

RF2: O módulo deve ser capaz de encontrar arquivos de vídeos e arquivos .json em pastas sem necessidade do usuário listar todas as pastas internas;

RF3: O módulo deve ser capaz de fazer leitura de arquivos .json;

RF4: O módulo deve ser capaz de gerar *frames* e trechos menores de vídeo, denominados *gifs* neste trabalho, de acordo com as anotações dos momentos importantes;

RF5: O módulo deve ser capaz de criar pastas com nomes pré-definidos e salvar os frames e trechos de vídeos nelas;

RF6: O usuário poderá especificar a pasta onde encontram-se os arquivos de vídeos e suas anotações;

RF7: O usuário poderá especificar quantos segundos antes e depois do tempo anotado para os eventos importantes ele deseja, para que sejam gerados os *gifs*;

Requisitos Não Funcionais:

RNF1: O módulo deve suportar arquivos de texto apenas na extensão .json;

RNF2: Os arquivos .json com as anotações dos momentos importantes dos vídeos devem possuir a tag ‘annotations’;

RNF3: O desenvolvimento será realizado na linguagem de programação *Python*;

RNF4: Os vídeos e arquivos de anotações devem estar em subpastas abaixo da pasta onde o arquivo do módulo ou o executável estiver salvo, caso contrário, o usuário deverá especificar (RF6);

RNF5: Os gifs serão gerados, por padrão, com 0 segundos antes e 5 segundos depois do tempo anotado para os eventos importantes, caso o usuário queira outra especificação deverá informar nos campos específicos (RF7)

II. Projeto do programa

A. Arquitetura e projeto

A ferramenta foi desenvolvida em linguagem *Python* e utiliza a biblioteca *OpenCV*, que é *open source* e muito utilizada na área de visão computacional. Também usa o framework *unittest* para realização dos testes de unidade automatizados, o pacote *tkinter* para criação da interface gráfica de interação com o usuário e o pacote *PyInstaller* que é responsável pela compactação do projeto em um executável.

Foi dividido em 3 pacotes: o primeiro contém o código para criação da **interface gráfica** (gui), o segundo com as classes que possuem a **lógica do negócio** (src) e a última que possui os **testes automatizados** (tests). Contendo a seguinte estrutura:

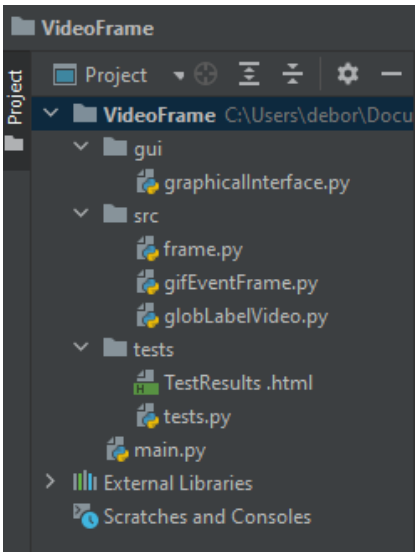

Figura 1: Estrutura de pacotes e classes

No pacote src estão presentes 3 classes, cada uma com sua responsabilidade:

Frame - responsável por toda a lógica de criação dos frames a partir de vídeos existentes nas subpastas do usuário. Também salva os arquivos criados na pastas específicas;

GifEventFrame - responsável pela geração dos *gifs* e dos *frames* dos momentos importantes anotados nos arquivos .json. Também salva estes arquivos nas pastas específicas. O usuário escolhe se executa os *gifs* ou os *frames* importantes, podendo ser um após o outro, mas não as duas ações juntas.

GlobLabelVideo - é uma classe de apoio que contém a lógica de pesquisar quais subpastas contém vídeos ou arquivos .json. Por ser chamada nas outras duas classes, esse código ficou separado.

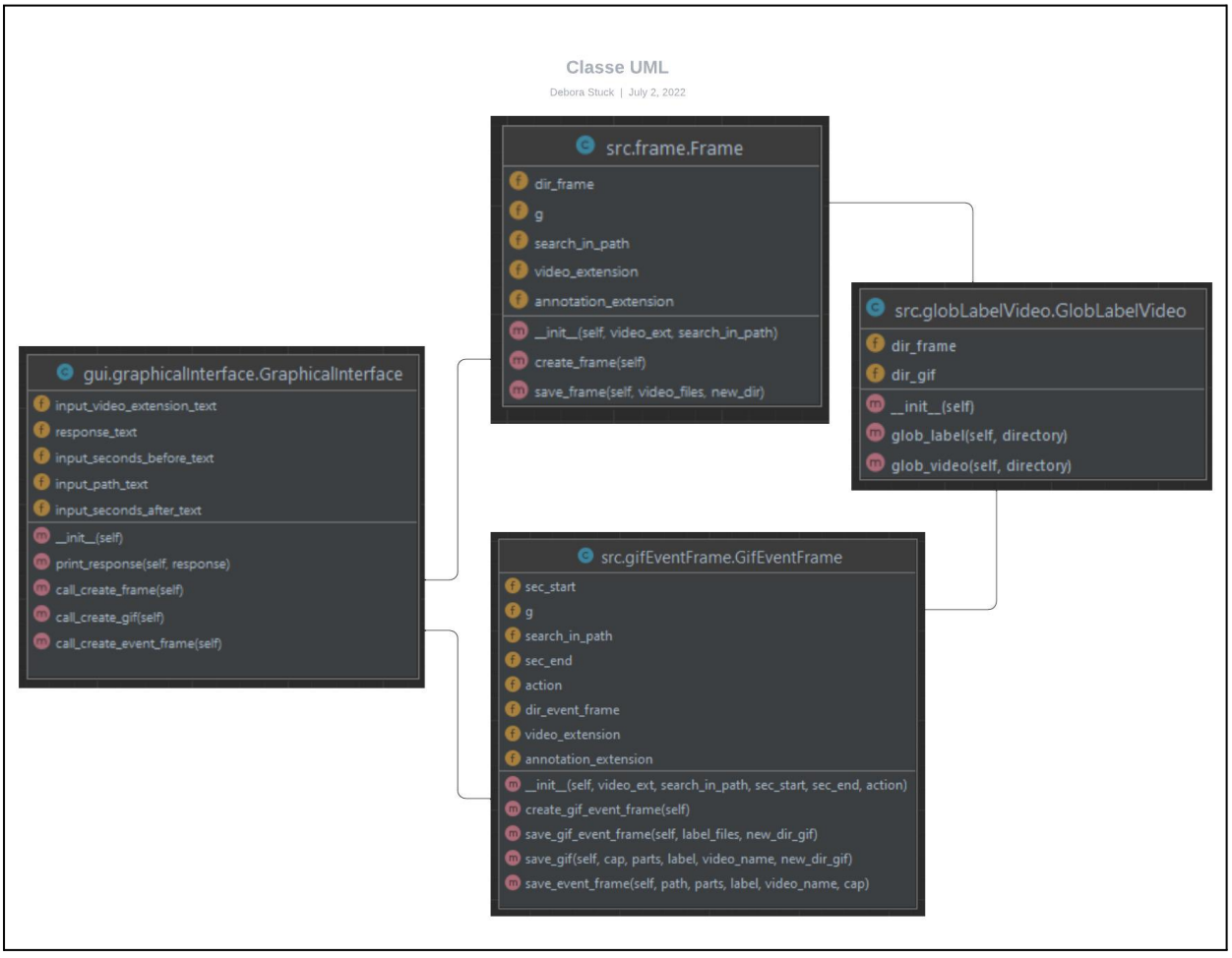

Figura 2: Diagrama de Classes

As interações e relacionamentos entre ator e sistema podem ser visualizadas no diagrama a seguir:

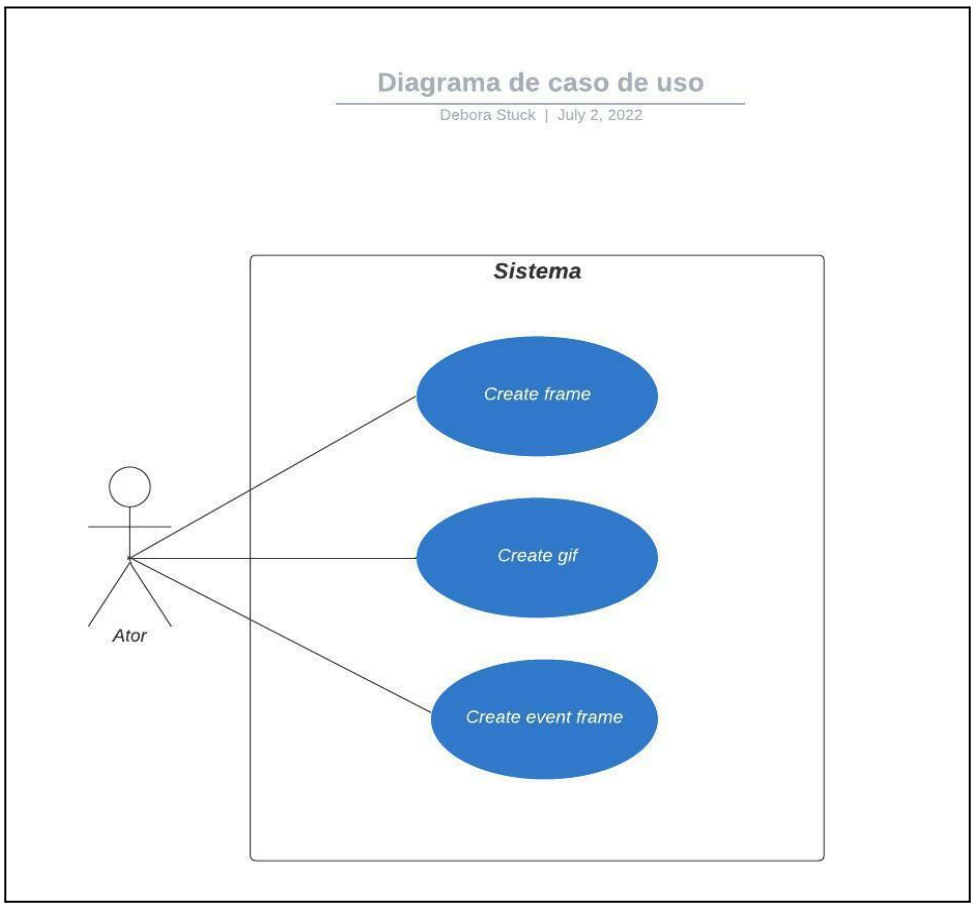

Figura 3 : Diagrama de Casos de Uso

Ao clicar no executável, a seguinte interface gráfica será exibida ao usuário:

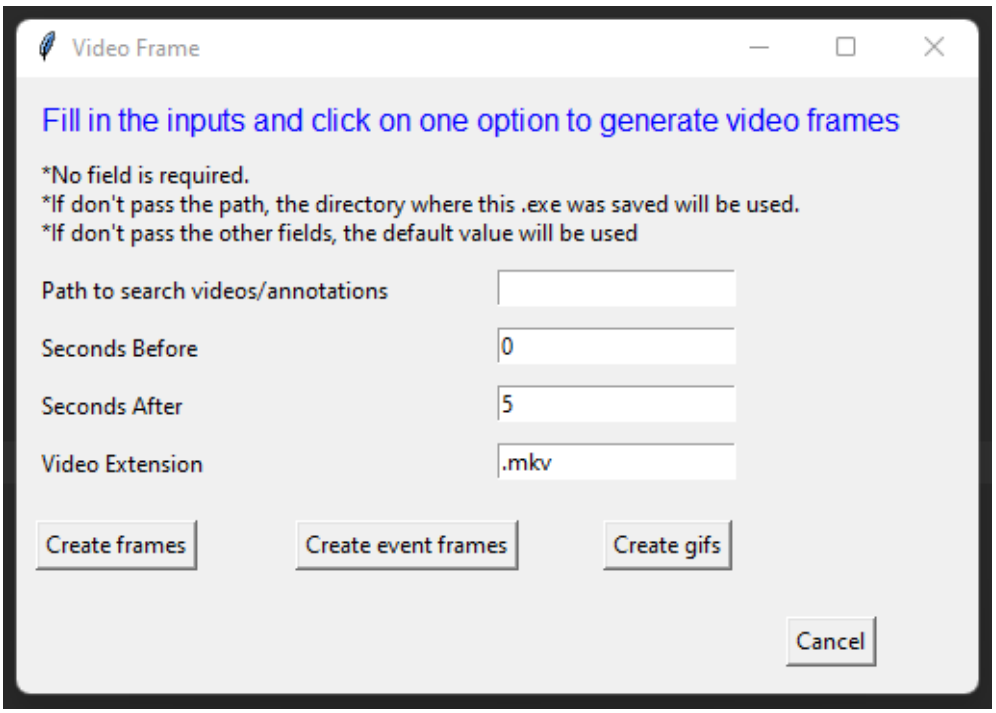

Figura 4: Interface gráfica

O usuário poderá especificar uma pasta para a busca de arquivos .json e de vídeos, quantos segundos antes e depois do tempo anotado ele quer para gerar os *gifs* e a extensão do vídeo que ele possui. Os dados visíveis nesta tela são os valores adotados como padrão neste trabalho. O usuário escolhe qual opção ele quer executar: Criar todos os frames a partir dos vídeos (*Create Frame*), gerar frames a partir das anotações (*Create Event Frames*) e gerar os trechos menores de vídeo a partir das anotações (*Create Gifs*). O botão *Cancel* fecha a tela.

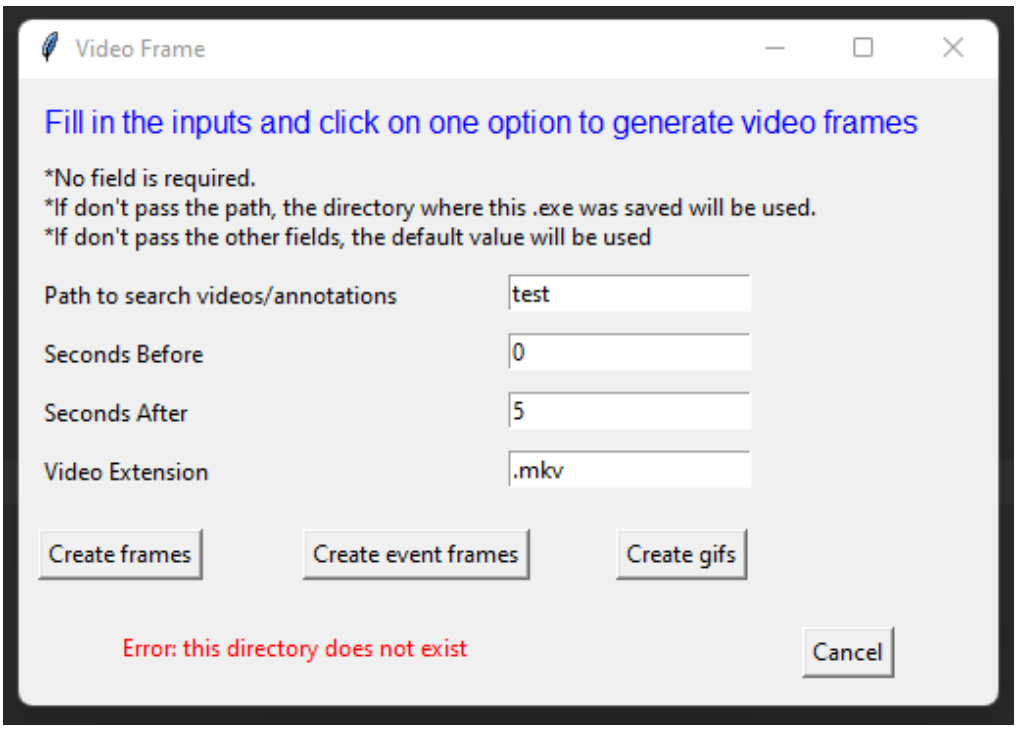

Figura 5: Interface gráfica - exemplo de mensagem de retorno com erro

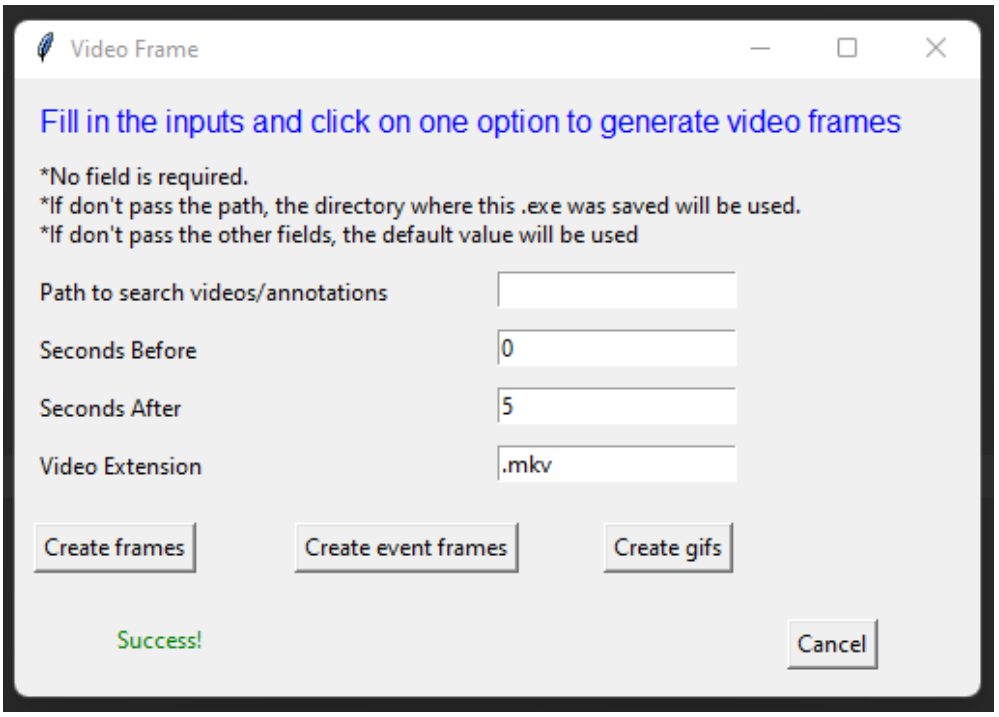

Figura 6: Interface gráfica - exemplo de mensagem de retorno com sucesso

B. Diagrama de modelagem de dados

O módulo não utiliza Banco de Dados, ele interage e salva arquivos diretamente no computador do usuário.

III. Código fonte

O código fonte foi comentado com o objetivo de facilitar a utilização da ferramenta e seu entendimento. Está disponível no link do *github* já citado anteriormente e no final desta documentação, no Apêndice A.

IV. Roteiro de testes

Foram feitos testes automatizados utilizando o *framework* do *Python* chamado ***unittest***. Ele suporta estruturas para testes unitários, que foram os testes realizados no programa, já que o módulo não é composto de muitas classes e componentes.

A. Critério de testes

CT1: Verificar se o sistema retorna mensagem de erro caso o diretório para pesquisar por arquivos de vídeo e os .json não existam;

CT2: Verificar se o sistema gera e salva todos os *frames* dos vídeos;

CT3: Verificar se o sistema gera e salva trechos de vídeos corretamente de acordo com as anotações de tempo do usuário;

CT4: Verificar se o sistema gera e salva *frames* de momentos especificados no arquivo .json.

B. Descrição dos casos de testes e resultado

CT1: O usuário preenche o campo '*path*' com o nome de um diretório para que o sistema pesquise por arquivos de anotações ou de vídeos. Passando um diretório inexistente no seu computador, o sistema retorna a mensagem de erro: 'Error: this directory does not exist'.

CT2: O usuário clica no botão '*create event frame*', informando uma pasta existente em seu computador. O teste verifica a quantidade de arquivos no pasta 'event frame' antes de criar os frames de eventos importantes e a quantidade após salvar os arquivos. O teste constata que não existia arquivo antes e passam a existir 190 após. E são exatamente 190 momentos importantes anotados no arquivo .json.

CT3: O usuário clica no botão '*create frame*', informando uma pasta existente em seu computador. O teste verifica a quantidade de arquivos com extensão .jpg

existentes na pasta destinada aos frames antes e depois de realizar a geração dos frames. O teste constata que antes do método ser executado não há arquivos .jpg e que após a sua execução existem 135.000. O que está correto, pois um vídeo contém 45 minutos (2.700 segundos) e são 25 frames por segundo. 2.700 * 25 = 67.500, e neste teste existiam dois arquivos de vídeos.

CT4: O usuário clica no botão '*create gif*', informando uma pasta existente em seu computador. O teste verifica a quantidade de arquivos existentes na pasta destinada aos *gifs* antes e depois de realizar a geração deles. O teste constata que antes do método ser executado não há arquivos na pasta e que após a sua execução existem 190. O que está correto, pois são 190 momentos importantes anotados no arquivo .json.

O relatório dos testes realizados foi exportado para um arquivo .html no pacote *tests* com o nome *TestsResults.html,* com o seguinte template.

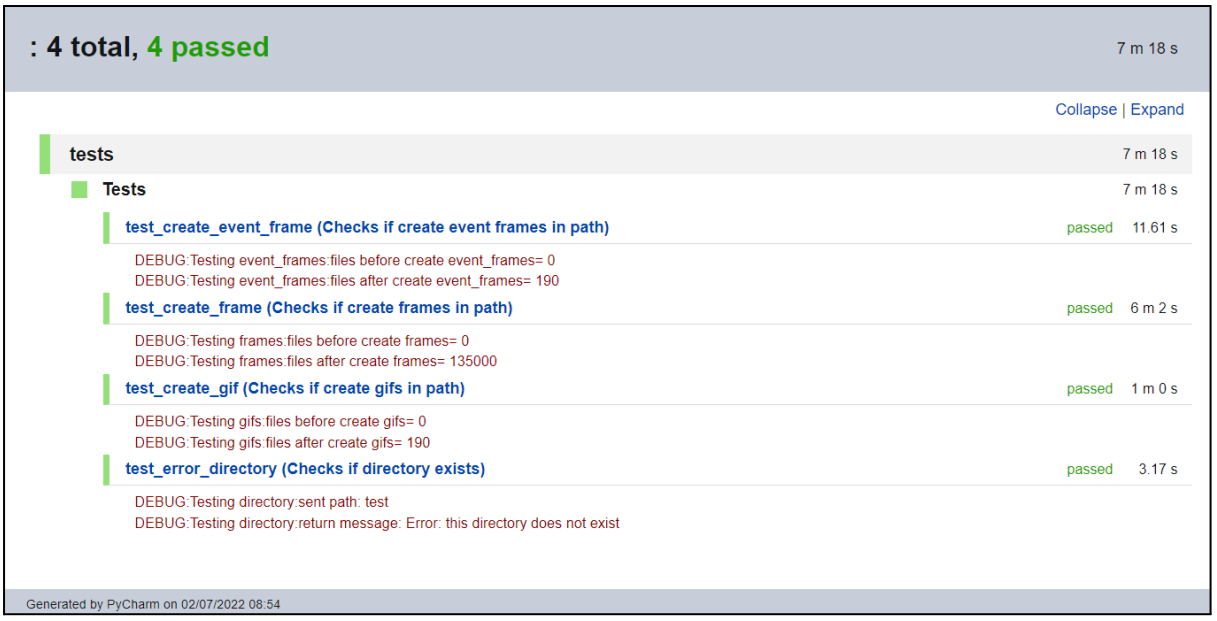

Figura 6: Relatório dos testes automatizados

Abaixo, uma imagem da pasta *eventFrames* após execução do teste de geração das imagens dos momentos importantes (CT2). O código utiliza o padrão de nome composto por: nome do evento (*Corner*), nome do vídeo (1_224p) e o número do frame correspondente (4549.0).

Figura 7: Arquivos .jpg gerados na pasta *eventFrames*

O exemplo citado acima, da primeira imagem da figura 7, corresponde à anotação no arquivo .json da figura abaixo. Fazendo os cálculos: 3:02 corresponde a 182 segundos, 182 * 25 (porque são 25 fps neste vídeo) resulta em 4550. Como a contagem dos frames inicia em 0, o frame correspondente a este momento importante anotado é o de número 4549.

```
33          {
34              "gameTime": "1 - 03:02",
35              "label": "Corner",
36              "position": "182775",
37              "team": "away",
38              "visibility": "visible"
39          },
```

Figura 8: Arquivo .json com anotação do *Corner*

V. Documentação para o usuário

A. Usuários visados pelo projeto

Os usuários do projeto são principalmente estudantes de mestrado e doutorado, que estejam fazendo pesquisa na área de visão computacional, mais especificamente com vídeos. Eles poderão acessar o código fonte no *link* do *github*, fazer alterações, melhorias e utilizar a ferramenta na etapa de pré-processamento de suas pesquisas.

O projeto pode ser útil também para estudantes de graduação que estejam estudando *Python,* querendo resolver problemas de visão computacional com vídeos e/ou estudando sobre testes unitários em *Python*.

Também poderá ser utilizado por pessoas que trabalham com vídeos e precisam de uma ferramenta que realize as ações desenvolvidas neste projeto. Isso por meio

do executável disponível, sem necessidade de conhecimento técnico de desenvolvimento de sistemas.

B. Contexto de atividade e como o programa pode ajudar

Os usuários poderão utilizar o módulo na fase de pré-processamento dos dados, conseguindo, de forma rápida e simples, obter *frames* e trechos de vídeos. O módulo também poderá servir como fonte de estudo e aprendizado de formas de implementação de testes unitários em um projeto *Python* e na utilização da biblioteca *OpenCV* para criação de *frames*.

Pessoas que trabalham com vídeos, como *youtubers* e *designers* gráficos, e precisam editá-los, gerando apenas imagens de alguns frames (*event frames*) ou trechos menores de vídeo (*gifs*) também se beneficiarão da ferramenta por meio do executável.

C. Instalação e execução do programa

O projeto está disponível no *github* no *link*: https://github.com/deborastuck/VideoFrame. A sua instalação é fácil, apenas sendo necessário baixar o código disponível, utilizando o comando *'clone'* que a plataforma disponibiliza. Com o código instalado no computador do usuário, ele poderá utilizar qualquer método implementado e obterá os resultados, conforme descrito nos requisitos e no diagrama de casos de uso.

A ferramenta também poderá ser utilizada sem acesso ao código fonte, bastando baixar e salvar o executável na máquina do usuário, clicar no executável e interagir com a interface, conforme descrito no item: II, figura 4.

Algumas especificações, como pacotes necessários para rodar o código *Python,* estão melhor descritas no arquivo '*readme'* que acompanha o projeto no *github*.

Em caso de insucesso, o *github* contém uma aba chamada *Issues*. Qualquer usuário pode descrever o ocorrido e eu receberei a notificação e poderei responder no mesmo local. Como fica visível para todos, ajuda toda a comunidade.

Exemplo de estrutura de diretório e pastas. O código do programa, que está na pasta *VideoFrame*, deve ficar na raiz da estrutura do usuário. Assim como o executável (*videoframe*). E deve existir uma pasta (ou mais pastas) que contenham os vídeos e arquivos de anotações (*dataset*).

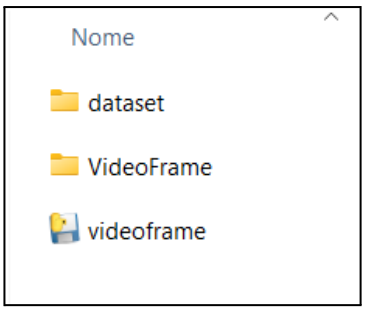

Figura 9: Exemplo de estrutura de pastas.

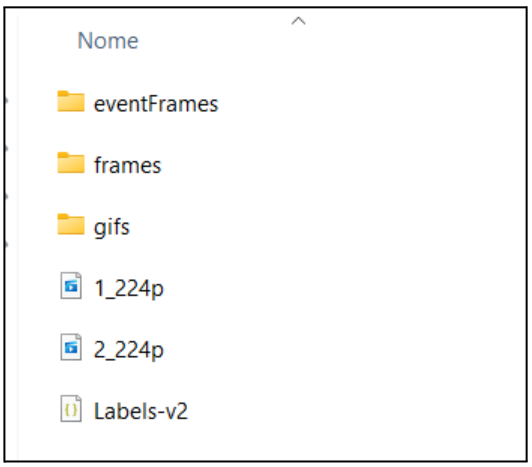

Figura 10: Exemplo de estrutura dentro da pasta ‘dataset’.

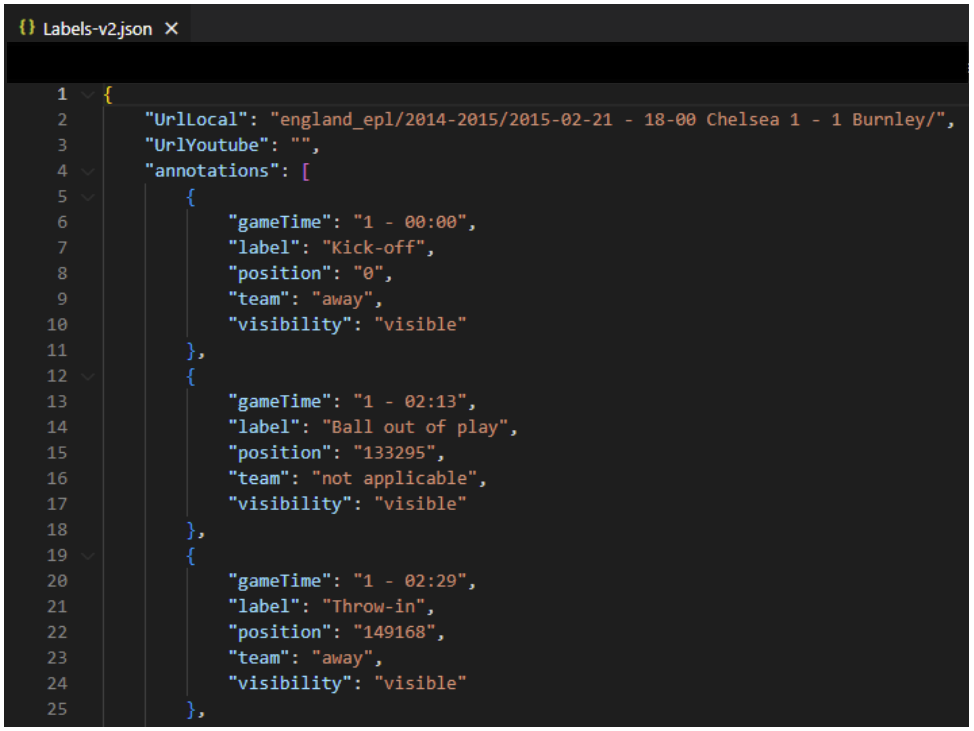

Figura 11: Exemplo de estrutura do arquivo de anotações .json’.

D. Tarefas que o programa apoia

O projeto apoia a fase de pré-processamento de vídeos para posterior análise e utilização em modelos de aprendizado de máquina. É possível dividir e salvar os frames dos vídeos e dividir e salvar trechos de vídeos de acordo com as anotações de tempo realizadas pelo usuário. Estas ações podem ser executadas como explicado no item II figura 4.

VI. Contexto da pesquisa e expectativas de colaboração científica

O projeto consiste em realizar ações fundamentais na etapa de pré-processamento de vídeos em pesquisas na área de visão computacional. Ele divide o vídeo em frames e em trechos de acordo com as anotações. O objetivo principal do projeto é facilitar a fase inicial do processamento de dados. No meu caso, por exemplo, os arquivos gerados por esse módulo serão usados no modelo de aprendizado de máquina que será treinado para resolver o problema de sumarização de vídeos de partidas de futebol, tema da minha dissertação de mestrado.

O principal benefício será tornar público um código que gera *frames* e trechos de vídeos de acordo com anotações dos usuários de forma automática, ou seja, o programa percorre as pastas e executa a ação desejada de forma iterativa em todos os arquivos de vídeo e de anotações encontrados.

Pelos teste realizados, podemos perceber que o processamento acontece em poucos minutos, o processo mais demorado foi de obter todos os *frames* dos vídeos com 6m2s. Esse processo gerou 135.000 arquivos .jpg, pois processou 2 vídeos, cada um com 45 minutos de duração e 25 fps.

Caso os usuários tentem utilizar o código sem atender corretamente às premissas de utilização, o código poderá perder crédito diante da comunidade e ser inutilizado. O que não traria ganhos às pesquisas e também não ajudaria em seu avanço e melhorias. As premissas estão listadas no arquivo *readme* no *github*, juntamente com o código e são elas:

- As anotações de tempos para geração dos trechos devem estar em arquivos .json, com a adequada formatação;
- As anotações também devem conter necessariamente a tag ‘annotations’;

- O código ou o executável deve ser salvo no computador do usuário no diretório acima da pasta ou das pastas que contenham os arquivos de vídeos e de anotações;
- Caso o código ou executável não seja salvo no diretório corretamente, o usuário deve passar como parâmetro o diretório onde deseja que o código pesquise pelos vídeos e anotações.

Apêndice A - Código fonte

Arquivo main:

main.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

from gui.graphicalInterface import GraphicalInterface


def main():
    """
    Initializes the program by running the graphical interface
    """
    GraphicalInterface()


if __name__ == '__main__':
    main()

```

Classe Frame

frame.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

import cv2
import os
import sys
from pathlib import Path
from src.globLabelVideo import GlobLabelVideo


class Frame:

    def __init__(self, video_ext, search_in_path):
        """
        Initializes the class with the specific params, but the user can send other values in the graphical interface
        :param video_ext: String
        :param search_in_path: String (the default value is where the executable was saved)
        """
        self.dir_frame = 'frames/'
        self.annotation_extension = '.json'
        if video_ext == "":
            self.video_extension = '.mkv'
        else:
            self.video_extension = video_ext
        if search_in_path != "":
            self.search_in_path = search_in_path
        elif getattr(sys, 'frozen', False):
            self.search_in_path = os.path.dirname(sys.executable)
        else:
            self.search_in_path = str(Path(__file__).parents[2])
        self.g = GlobLabelVideo()
```

frame.py

```python
def create_frame(self):
    """
    Base method, calls the other methods to create the video frames
    The glob will search for files with video extension in the subfolders of the base directory
    Call the save_frame method
    :return String (success or error)
    """
    directory = self.search_in_path + '/**/*' + self.video_extension
    if os.path.exists(self.search_in_path):
        video_files, new_dir_video = self.g.glob_video(directory)
    else:
        return 'Error: this directory does not exist'

    response = self.save_frame(video_files, new_dir_video)
    return response
```

frame.py

```python
def save_frame(self, video_files, new_dir):
    """
    For all video files found, get the file name
    Capture the video with OpenCV
    While ret is true, read the video getting the boolean value (ret) and the frame image
    Save the frame image in new directory with the name consisting of:
    frame + file name + number of the current frame sequence + extension jpg
    :param video_files: list
    :param new_dir: String
    :return String (success or error)
    """
    for path in video_files:
        filename = os.path.basename(Path(path).stem)
        cam = cv2.VideoCapture(path)
        currentframe = 0

        while True:
            ret, frame = cam.read()
            if ret:
                name = new_dir + '/frame' + filename + str(currentframe) + '.jpg'
                cv2.imwrite(name, frame)
                currentframe += 1
            else:
                break

        cam.release()
        cv2.destroyAllWindows()

    return "success"
```

Classe GitEventFrame:

gifEventFrame.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

import cv2
import json
import os
import sys
from pathlib import Path
from src.globLabelVideo import GlobLabelVideo

class GifEventFrame:

    def __init__(self, video_ext, search_in_path, sec_start, sec_end, action):
        """
        Initializes the class with the specific params, but the user can send other values in the graphical interface
        :param video_ext: String
        :param search_in_path: String
        """
        self.dir_event_frame = 'eventFrames/'
        self.annotation_extension = '.json'
        if video_ext == "":
            self.video_extension = ".mkv"
        else:
            self.video_extension = video_ext
        if search_in_path != "":
            self.search_in_path = search_in_path
        elif getattr(sys, 'frozen', False):
            self.search_in_path = os.path.dirname(sys.executable)
        else:
            self.search_in_path = str(Path(__file__).parents[2])
        self.action = action
        if sec_start == "":
            self.sec_start = 0
        else:
            self.sec_start = sec_start
        if sec_end == "":
            self.sec_end = 5
        else:
            self.sec_end = sec_end
        self.g = GlobLabelVideo()
```

gifEventFrame.py

```python
    def create_gif_event_frame(self):
        """
        Base method, calls the other methods to create the parts of the video (gifs) or the event frames
        The glob will search for files with annotation extension in the subfolders of the base directory
        Checks if the annotation extension is .json before continue
        Call the globLabel method to get the list of directories with annotations files and
        the new directory to save the frames
        Call the save_gif_event_frame method
        :return String (success or error)
        """
        directory = self.search_in_path + '/**/*' + self.annotation_extension
        if os.path.exists(self.search_in_path):
            if self.annotation_extension == '.json':
                label_files, new_dir_gif = self.g.glob_label(directory)
            else:
                return 'Error: the file extension must be json'
        else:
            return 'Error: this directory does not exist'
        response = self.save_gif_event_frame(label_files, new_dir_gif)
        return response
```

gifEventFrame.py

```
61      def save_gif_event_frame(self, label_files, new_dir_gif):
62          """
63          For all annotations files found, define the dictionary from JSON object
64          Iterating through the json list
65          The glob will search for files with video extension in the subfolders of the base directory
66          For all video files found, capture the video with OpenCV
67          For all annotations, checks if the first character from gameTime annotation
68          is the same as the first character from video file name
69          Get the minute and the second of gameTime annotation
70          Define the number of the start frame and the end frame, multiplying by fps (frames per second)
71          If the user click on the gif button in the graphical interface, call the save_gif method
72          If the user click on the event frame button in the graphical interface, call the save_event_frame method
73          :param label_files: String (the directory with annotations file)
74          :param new_dir_gif: String
75          :return String (success or error)
76          """
77          response = "Success!"
78          for file_path in label_files:
79              json_file = open(file_path)
80              data = json.load(json_file)
81              dict_annotation = []
82
83              for i in data['annotations']:
84                  dict_annotation.append(i)
85
86              json_file.close()
87              path = os.path.dirname(file_path)
88              video_directory = self.search_in_path + '/**/*' + self.video_extension
89              video_files, new_dir_video = self.g.glob_video(video_directory)
90
91              for video_file in video_files:
92                  video_name = os.path.basename(Path(video_file).stem)
93                  cap = cv2.VideoCapture(video_file)
94                  fps = cap.get(cv2.CAP_PROP_FPS)
95                  parts = []
96                  label = []
```

gifEventFrame.py

```
97
98                  for a in dict_annotation:
99                      if a['gameTime'][:1] == video_name[:1]:
100                         label.append(a['label'])
101                         minutes = a['gameTime'][4:-3]
102                         seconds = a['gameTime'][-2:]
103                         total_seconds = int(60 * int(minutes) + int(seconds))
104                         frame_start = ((total_seconds + int(self.sec_start)) * fps)
105                         frame_end = ((total_seconds + int(self.sec_end)) * fps)
106                         if frame_start != 0:
107                             frame_start = frame_start - 1
108                         parts.append((frame_start, frame_end))
109
110                 if self.action == "EVENT_FRAME":
111                     response = self.save_event_frame(path, parts, label, video_name, cap)
112                 elif self.action == "GIF":
113                     response = self.save_gif(cap, parts, label, video_name, new_dir_gif)
114                 else:
115                     return "Could not identify the action. Try again!"
116         return response
```

gifEventFrame.py

```python
def save_gif(self, cap, parts, label, video_name, new_dir_gif):
    """
    Create and save the parts of the video (gifs)
    Read the video
    Define the fourcc for windows
    Save the parts of video in new directory with the name consisting of:
    label name + video name + start frame number + end frame number
    :param cap: video capture object from OpenCV
    :param parts: list (the frame numbers of the annotations)
    :param label: list (the label names of the annotations)
    :param video_name: String
    :param new_dir_gif: String (path to save gifs)
    :return String (success or error)
    """
    ret, frame = cap.read()
    h, w, _ = frame.shape
    fourcc = cv2.VideoWriter_fourcc(*"XVID")
    writers = []

    for index, frame_number in enumerate(parts):
        start, end = frame_number
        writers.append(cv2.VideoWriter
                       (f"{new_dir_gif}/{label[index]}-{video_name}-{start}-{end}{self.video_extension}",
                        fourcc, 25.0, (w, h)))

    f = 0
    while ret:
        f += 1
        for i, part in enumerate(parts):
            start, end = part
            if start <= f <= end:
                writers[i].write(frame)
        ret, frame = cap.read()

    for writer in writers:
        writer.release()

    cap.release()
    return "Success!"
```

gifEventFrame.py

```python
def save_event_frame(self, path, parts, label, video_name, cap):
    """
    Create and save the event frames
    Define the name of the new directory
    Checks if the new directory exists before creating it
    id_frame has two numbers: the frame to start and to end
    Use id_start as the frame you want
    Read the frame
    Save the frame in new directory with the name consisting of: label name + video name + frame number
    :param path: String (the directory with annotations file)
    :param parts: list (the frame numbers of the annotations)
    :param label: list (the label names of the annotations)
    :param video_name: String
    :param cap: video capture object from OpenCV
    :return String (success or error)
    """
    new_dir_event_frame = path + '/' + self.dir_event_frame
    try:
        if not os.path.exists(new_dir_event_frame):
            os.mkdir(new_dir_event_frame)
    except OSError:
        return 'Error: creating directory of data'

    for index, id_frame in enumerate(parts):
        id_start, id_end = id_frame
        cap.set(1, id_start)
        ret, frame = cap.read()
        cv2.imwrite(f'{new_dir_event_frame}{label[index]}-{video_name}-{id_start}.jpg', frame)

    return "Success!"
```

Classe GlobLabelVideo:

globLabelVideo.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

import glob
import os


class GlobLabelVideo:

    def __init__(self):
        self.dir_frame = 'frames/'
        self.dir_gif = 'gifs/'

    def glob_label(self, directory):
        """
        Search annotations files in base directory
        Uses glob which returns the files that match the directory specified in the argument and the extension file
        Checks if the annotations file directory exists before continuing
        Define the name of the new directory
        Checks if the new directory exists before creating it
        Save the directory of all found files
        :param directory: String
        :return list (label_files) and String (new_dir_gif)
        """
        new_dir_gif = ""
        label_files = []
        for path in glob.glob(directory):
            if os.path.exists(path):
                new_dir_gif = os.path.dirname(path) + "/" + self.dir_gif
                try:
                    if not os.path.exists(new_dir_gif):
                        os.mkdir(new_dir_gif)
                    label_files.append(path)
                except OSError:
                    return 'Error: creating directory of data'
            else:
                return 'Error: this directory does not exist'
        return label_files, new_dir_gif
```

globLabelVideo.py

```python

    def glob_video(self, directory):
        """
        Search video files in base directory
        Uses glob which returns the files that match the directory specified in the argument and the extension file
        Define the name of the new directory
        Checks if the new directory exists before creating it
        Save the directory of all found files
        :param directory: String
        :return list video_files and String new_dir_video
        """
        video_files = []
        new_dir_video = ""
        for path in glob.glob(directory):
            if os.path.exists(path):
                new_dir_video = os.path.dirname(path) + '/' + self.dir_frame
                try:
                    if not os.path.exists(new_dir_video):
                        os.mkdir(new_dir_video)
                    video_files.append(path)
                except OSError:
                    return 'Error: creating directory of data'
            else:
                return 'Error: this directory does not exist'
        return video_files, new_dir_video

```

Classe Tests:

tests.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

import pathlib
import unittest
import os
import sys
import glob
import logging
from src.frame import Frame
from src.gifEventFrame import GifEventFrame


class Tests(unittest.TestCase):

    def test_error_directory(self):
        """
        Checks if directory exists
        :return: Error
        """
        vf = Frame('.mkv', 'test')
        result = vf.create_frame()
        expected = 'Error: this directory does not exist'
        log = logging.getLogger("Testing directory")
        log.debug("sent path: test")
        log.debug("return message: Error: this directory does not exist")
        self.assertEqual(result, expected)
```

tests.py

```
29  def test_create_frame(self):
30      """
31      Checks if create frames in path
32      :return: True
33      """
34      directory = self.get_directory("frames")
35      number_files = self.get_number_files(directory[0])
36      vf = Frame('.mkv', '')
37      vf.create_frame()
38      number_frames = self.get_number_files(directory[0])
39      log = logging.getLogger("Testing frames")
40      log.debug("files before create frames= %r", number_files)
41      log.debug("files after create frames= %r", number_frames)
42      self.assertTrue(number_files < number_frames)
43
44  def test_create_gif(self):
45      """
46      Checks if create gifs in path
47      :return: True
48      """
49      directory = self.get_directory("gifs")
50      number_files = self.get_number_files(directory[0])
51      gef = GifEventFrame('.mkv', '', '', '', "GIF")
52      gef.create_gif_event_frame()
53      number_gifs = self.get_number_files(directory[0])
54      log = logging.getLogger("Testing gifs")
55      log.debug("files before create gifs= %r", number_files)
56      log.debug("files after create gifs= %r", number_gifs)
57      self.assertTrue(number_files < number_gifs)
```

tests.py

```
59      def test_create_event_frame(self):
60          """
61          Checks if create event frames in path
62          :return: True
63          """
64          directory = self.get_directory("eventFrames")
65          number_files = self.get_number_files(directory[0])
66          gef = GifEventFrame('.mkv', '', '', '', "EVENT_FRAME")
67          gef.create_gif_event_frame()
68          number_event_frames = self.get_number_files(directory[0])
69          log = logging.getLogger("Testing event_frames")
70          log.debug("files before create event_frames= %r", number_files)
71          log.debug("files after create event_frames= %r", number_event_frames)
72          self.assertTrue(number_files < number_event_frames)
73
74      def get_directory(self, path):
75          """
76          Get the directory with wanted files
77          :return: String
78          """
79          directory = glob.glob(str(pathlib.Path(__file__).parents[2]) + '/**/' + path)
80          return directory
81
82      def get_number_files(self, directory):
83          """
84          Get the number of files found
85          :return: int
86          """
87          list_files = os.listdir(directory)
88          number_files = len(list_files)
89          return number_files
90
91      logging.basicConfig(stream=sys.stderr)
92      logging.getLogger("Testing directory").setLevel(logging.DEBUG)
93      logging.getLogger("Testing frames").setLevel(logging.DEBUG)
94      logging.getLogger("Testing gifs").setLevel(logging.DEBUG)
95      logging.getLogger("Testing event_frames").setLevel(logging.DEBUG)
96      unittest.main(argv=[''], verbosity=2, exit=False)
97
```

Classe GraphicalInterface:

graphicalInterface.py

```python
# @Author: Debora Stuck
# @Time: 06.2022

from tkinter import *
from src.frame import Frame
from src.gifEventFrame import GifEventFrame


class GraphicalInterface:

    def __init__(self):
        window = Tk()
        window.title("Video Frame")
        window.geometry("500x320")

        text_orientation_title = Label(window, justify="left", font=("Arial", 12), fg='#00f',
                                       text="Fill in the inputs and click on one option to generate video frames")
        text_orientation_title.place(x=10, y=10)
        text_orientation = Label(window, justify="left",
                                 text="*No field is required. \n"
                                      "*If don't pass the path, the directory where this .exe was saved will be used."
                                      "\n*If don't pass the other fields, the default value will be used")
        text_orientation.place(x=10, y=40)

        self.response_text = Label(window, text="")
        self.response_text.place(x=50, y=280)

        input_path_label = Label(window, justify="left", text="Path to search videos/annotations")
        input_path_label.place(x=10, y=100)
        self.input_path_text = Entry(window)
        self.input_path_text.place(x=250, y=100)

        input_seconds_before_label = Label(window, justify="left", text="Seconds Before")
        input_seconds_before_label.place(x=10, y=130)
        self.input_seconds_before_text = Entry(window)
        self.input_seconds_before_text.insert(0, "0")
        self.input_seconds_before_text.place(x=250, y=130)
```

graphicalInterface.py

```python

        input_seconds_after_label = Label(window, justify="left", text="Seconds After")
        input_seconds_after_label.place(x=10, y=160)
        self.input_seconds_after_text = Entry(window)
        self.input_seconds_after_text.insert(0, "5")
        self.input_seconds_after_text.place(x=250, y=160)

        input_video_extension_label = Label(window, justify="left", text="Video Extension")
        input_video_extension_label.place(x=10, y=190)
        self.input_video_extension_text = Entry(window)
        self.input_video_extension_text.insert(0, ".mkv")
        self.input_video_extension_text.place(x=250, y=190)

        frame_button = Button(window, text="Create frames", command=self.call_create_frame)
        frame_button.place(x=10, y=230)

        event_frame_button = Button(window, text="Create event frames", command=self.call_create_event_frame)
        event_frame_button.place(x=145, y=230)

        gif_button = Button(window, text="Create gifs", command=self.call_create_gif)
        gif_button.place(x=305, y=230)

        cancel_button = Button(window, text="Cancel", command=window.destroy)
        cancel_button.place(x=400, y=280)

        window.mainloop()
```

graphicalInterface.py

```
64
65      def print_response(self, response):
66          if response == "Success!":
67              self.response_text.config(fg='#008000')
68          else:
69              self.response_text.config(fg='#f00')
70          self.response_text["text"] = response
71
72      def call_create_frame(self):
73          path_text = self.input_path_text.get()
74          video_ext = self.input_video_extension_text.get()
75          vf = Frame(video_ext, path_text)
76          response = vf.create_frame()
77          self.print_response(response)
78
79      def call_create_gif(self):
80          path_text = self.input_path_text.get()
81          video_ext = self.input_video_extension_text.get()
82          seconds_before = self.input_seconds_before_text.get()
83          seconds_after = self.input_seconds_after_text.get()
84          vf = GifEventFrame(video_ext, path_text, seconds_before, seconds_after, "GIF")
85          response = vf.create_gif_event_frame()
86          self.print_response(response)
87
88      def call_create_event_frame(self):
89          path_text = self.input_path_text.get()
90          video_ext = self.input_video_extension_text.get()
91          vf = GifEventFrame(video_ext, path_text, "", "", "EVENT_FRAME")
92          response = vf.create_gif_event_frame()
93          self.print_response(response)
94
```