Hey there! Welcome to the one and only (at this time) tutorial on creating an animation app with python. In this tutorial, we’ll learn how we can create a 2d animation app with python 3 while learning some other stuff too, to follow this tutorial in creating the animation app, you’ll need to have at least a basic knowledge on the python 3 programming language, if you haven’t, you can learn that here>>>*[refer to python beginner blog post(s)]*, here we’ll be using some packages in python 3, and it includes: tkinter(for our GUI(Graphic User Interface)), opencv(for creating our animations), pillow (for interacting with images), pygame(for some demonstration), so what should be done is to install this packages with the pip command, you can learn how to use the pip command here>>> *[refer to pip tutorial]*, by typing *pip install opencv-python* in the command prompt or powershell to install opencv and *pip install pillow* to install pillow, as well as pip install pygame, note that the tkinter library comes preinstalled with python.

So, let’s understand how the 2d animation concept works, it is like a bunch of images playing over and over. For instance, if you take a look at a camera that flashes light while taking a picture, you see it flashes once for one picture, and for a video, you don’t see the light going off, because it’s continuously taking the picture until you stop recording the video, so a video is literally taking a lot of pictures and show them one by one to view motion at a particular frame rate (speed of the animation, i.e how long one picture would be viewed).

Now, we’re going to use this same concept to create an animating app with python, so what we just need to do is have the user pass in a bunch of pictures, set the frame rate and then we let it view the animation and also export it to a video that can be played on TVs, phones, computers etc.

So, first thing we’ll need to do is create the user interface with tkinter and implement the importing of images for the animation.

So we’ll just create a simple UI for this project, to create our window where we’ll be working with tkinter, here’s some code to create a simple blank window, I’ll be explaining code with comments

#first import everything we need from tkinter

from tkinter import \*

#create a class that inherits the Tk window class

class Main(Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

#set the title of our window

self.title('Simple Animator App')

#set the size of the window

#the first one is for the width and the second for height

#feel free to mess around with the screen size

self.geometry('900x600')

#make the window not resizable(optional)

#the first parameter is for the width and second for height

#we use True or False to determine if it should be

#resizable or not

#here we make both not resizable (False)

self.resizable(False, False)

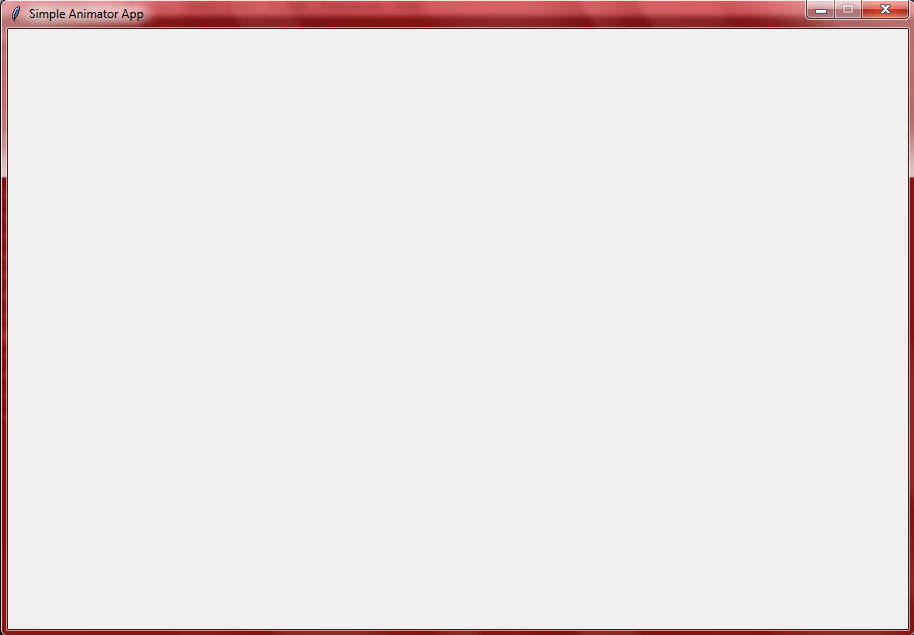
#run the main code

if \_\_name\_\_=='\_\_main\_\_':

#call the main function and access an attribute for our window

Main().mainloop()

Now, we’ve created a blank window, you should get something like this when you run the code:



There’s our window, now let’s put some things on it to allow the user open an image.

We’ll be building this UI in a very simple manner, so we want to create a simple deck of buttons at the top for clicks, and then we’ll arrange them with the label frame in tkinter, this buttons would allow us to open images and navigate through them, and then eventually we’ll get to the animating, c’mon, let’s code together:

from tkinter import \*

#import the LabelFrame widget from ttk in tkinter

from tkinter.ttk import \*

#that imports everything in ttk including label frame

class Main(Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title('Simple Animator App')

self.geometry('900x600')

self.resizable(False, False)

self.widgets()#call the widgets method

def widgets(self):

#created a method for initiating the UI

#create label frame

self.frame = LabelFrame(self, text="buttons", width = 600, height = 55)

#place the frame at x and y coordinates on the screen

self.frame.place(x = 10, y = 15)

#we're putting all the buttons in self.frame

#so we pass it in as the frame/window param

btn1 = Button(self.frame, text = "open", command = None)

#created a button for the opening images

#not given any commands yet, so we set the commmand = None

btn1.place(x = 2) #place btn1 in coordinates in self.frame

#next is the "save" button to save our animation

btn2 = Button(self.frame, text = "save", command = None)

btn2.place(x = 80)

#then the "new" button to create fresh animation

btn3 = Button(self.frame, text = "new", command = None)

btn3.place(x = 158)

#next is the "prev" button for navigation

btn4 = Button(self.frame, text = "prev", command = None)

btn4.place(x = 236)

#"next" button for navigation

btn5 = Button(self.frame, text = "next", command = None)

btn5.place(x = 314)

#so let's create some more widget for future use

#first would be a widget to set our frames per second

#the frames per second i mentioned stff about it earlier

# is gonna be between a range of numbers for the loop speed

#so we'll be using the spin box widget

#we'll be putting it in a label frame to make our ui more understandable

fpsframe = LabelFrame(self.frame, text = "FPS", width = 150, height = 35)

fpsframe.place(x = 392)

#create the spin box for selecting numbers

fps = Spinbox(fpsframe, from\_ = 0,to = 2000, width = 5)

#the from\_ is where it starts and to for where it stops (counting)

fps.pack()#just another layout manager

#the preview button for previewing the animation

btn6 = Button(self.frame, text = "preview", command = None)

btn6.place(x = 442)

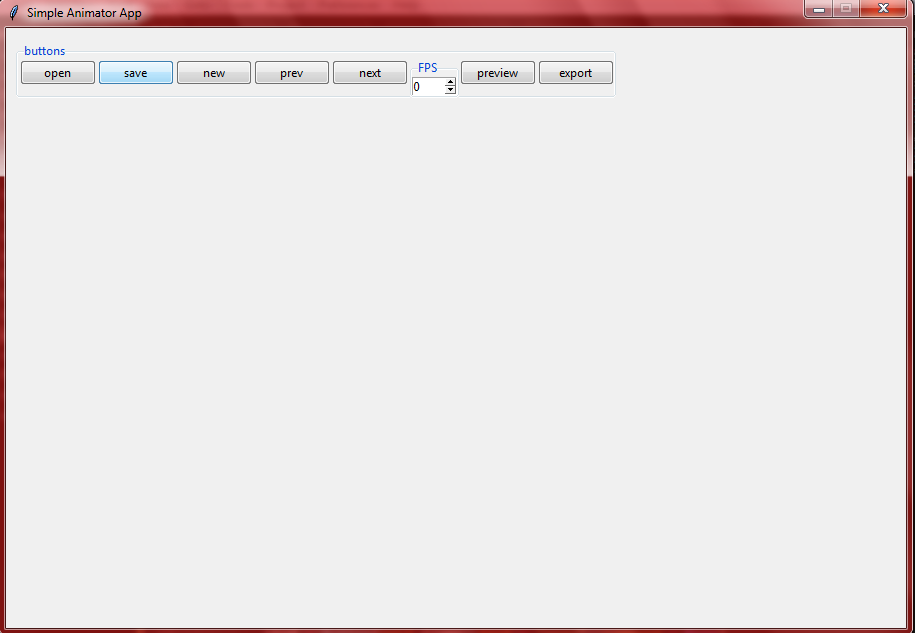
#export button for exporting our animation into a playable video

btn7 = Button(self.frame, text = "export", command = None)

btn7.place(x = 520)

if \_\_name\_\_=='\_\_main\_\_':

Main().mainloop()

Yeah, we’ve setup our simple front end design, you should have something like this when you run the code we just wrote together: 

Next thing we’ll want to do is to start making our app functional, so we’ll be writing some code for the functions of those buttons and connect them to the buttons and then our app is ready.

First should be the open button for opening the images, so let’s code!

To create our open method, we'll need to import the open method from tkinter filedialog, so that we can open images and then we'll display them on the screen with pillow, and add them to a list we'll need the list because it's a good way to keep track of all the images for navigation and we’ll use the list of images to create the animation, we’ll separate the functions here a bit, so we won’t type the whole code but we’ll show the full code later

#import what we need from the tkinter filedialog

#which is the open method and saveas method

self.images = []#define the image list

from tkinter.filedialog import \*

#import PIL to handle image

from PIL import Image, ImageTk

#create method for opening images

def open(self):

global img#for purpose of image view

#run the askopenfilename method that and store the image in

#a variable named file, so we can use it later in the code

#we define an optional parameter title and title the open file pop up

file = askopenfilename(title = "Open image - animator app")

self.images.append(file)

#so, now we want to view the file in a label, there's another complex

#way of viewing the image in a scrollable view, but we want to make things

#simple in this tutorial, so the images would have a limited size in this tutorial

#we want to display the images in a label below the widgets

#here, we want to open the file and prepare it for view on the label

img = ImageTk.PhotoImage(Image.open(file))

#create a label and make the open image the image

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

#now let's get to the save method

def save(self):

#first run the asksaveasfilename method from the filedialog

file = asksaveasfilename(title = "Save Project - Simple animation app", initialfile = 'Untitled.txt', defaultextension = ".txt", filetypes = [("Text Files", "\*.txt")])

if file == '':#check if the file is none

file == "Untitled"

else:

#attach the filename for the project to the windows title

self.title(os.path.basename(self.File).replace('.txt', '') + "- Simple Animation app")

#open the file for writing

with open(self.File, 'w+') as f:

#turn the list of images into a string for writing

data = str(self.images)

#write the list of images into the file

f.write(data)

#close the file

f.close()

#the "new" function is quite very simple

def new(self):

#simply make the image list empty again

self.images = []

#then change/keep the window title

self.title("Simple Animation app")

#then we move to the navigation tools, first the "prev"

#we initially define a variable to allocate the location of the navigator

#so that we can increment and decrement this value to navigate through

self.n = 0

def prev\_frame(self):

global img

self.n-=1 #decrement the value to move backwards

#open the image with its in images with the use of n variable

#so the self.n is literally our image list index

img = ImageTk.PhotoImage(Image.open(self.images[self.n]))

#create label and put image in

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

#for the "next" function, we'll do the same thing

# but increase the value of n this time

def next\_frame(self):

global img

self.n += 1 #increment the value of n

img = ImageTk.PhotoImage(Image.open(self.images[self.n]))

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

#so let's get to the main animation function

#first we'll create a new file where we'll implement the algorithm

#with python, and then we'll import and use it for the preview

#in a new file, we'll save it as preview.py:

import pygame #import pygame as it would be used for preview

from time import sleep #import sleep, it would be used to control the fps

from PIL import Image, ImageTk#for image processing

from tkinter.messagebox import showerror#for displaying errors

class Animate:#create a class for our algorithm

pygame.init()#initialize the pygame module features

#the main method, passing image list and fps as parameters

def animate(images, fps):

try:

#in a try except block

def get\_pic\_size(filename):

#method to get the size of a pic

image = Image.open(filename)

#open the pic

width, height = image.size

# extract the size

return width, height

#then return the size

imagelist = [i for i in images]#get relocate the list items

try:

#in a nested try except block

size = get\_pic\_size(imagelist[2])

#used the function for getting size of pic to get size of

#the second picture, that'll be the size of our window

except IndexError:#check if it encounters and index error

#then display an error message

showerror('An Error Occured', 'there are not enough frames to animate')

# print(size)

#now, set screen size with size of the image

screen = pygame.display.set\_mode((size))

#set the caption of the window we created

pygame.display.set\_caption("Preview - Simple Animator app")

imagelist = [i for i in images]

#for loop, to loop throught the images

for i in imagelist:

try:

#in a try except block

#load each image in imagelist

img = pygame.image.load(i)

#set it on the screen from x=0, y=0 coords respectively

screen.blit(img, [0, 0])

except:

pass

#to let the window pause for the fps time before the next image

sleep(1/fps)

#update the pygame window

pygame.display.update()

while True:

#event loop

for event in pygame.event.get():

#check for quit events(to close window)

if event.type == pygame.QUIT:

pygame.quit()#exit when the window is closed

if event.type == pygame.KEYDOWN:#check for keydown events

#exit when keyboard Ctrl+A is pressed

if event.key == pygame.K\_a & pygame.KMOD\_CTRL:

pygame.quit()

#update pygame screen

pygame.display.update()

except Exception as e:

pass

#next is the export function, to export the animation

# to a playable video

#first we import the necessary things

from tkinter.messagebox import \*#import messagebox for displaying info

import cv2, os#import opencv-python and os

def createVid(self, images, fps, title):

imgs = []

for i in images:#loop through images

img = cv2.imread(i)#read each image

#get the shape properties of the image

height, width, layer = img.shape

#get the size of the image

size = (width, height)

#append each image into a new list for animation

imgs.append(img)

#replace the .txt in the title

Title = title.replace('.txt', '')

#create the animation with opencv, by passing in parameters

#of the fps, size, and title

output = cv2.VideoWriter(Title, cv2.VideoWriter\_fourcc(\*'DIVX'), fps, size)

for i in range(len(imgs)):

#loop through the images and append each image into the animation

output.write(imgs[i])

#release the animation

output.release()

#this is optional and is for adding sound to the animation, where title

#is the animation and audio is the sound, you'll need to have ffmpeg installed

#to be able to do this:

# os.popen(f'CMD /K ffmpeg -i ' + Title + ' -i ' + audio +' -map 0:0 -map 1:0 -c:v copy -c:a copy ' + title)

#now we need to integrate the createVid method to the export method

def export(self):

#get the images from the image list

images = self.images

#notice that we'll define a new variable there for the textvariable

#--snip--self.FPS = IntVar()--snip--

# and the spinbox will be edited and would have

#--snip-- to = 2000, width = 5, text\_variable = self.FPS)

#now, here we get the fps from the spinbox variable

fps = self.FPS.get()

#set the title

title = 'animation' + '.avi'

# we call the createVid function and pass in the parameters

#that we obtained from the user's input

self.createVid(images, fps, title)

#inform the user that the animation was exported successfully

showinfo("Success", "Your animation was exported succesfully exported to "+title)

There, we’ve implemented all the features of our buttons, now we’ll connect them to the buttons and show the full code, explaining the features and testing them out.

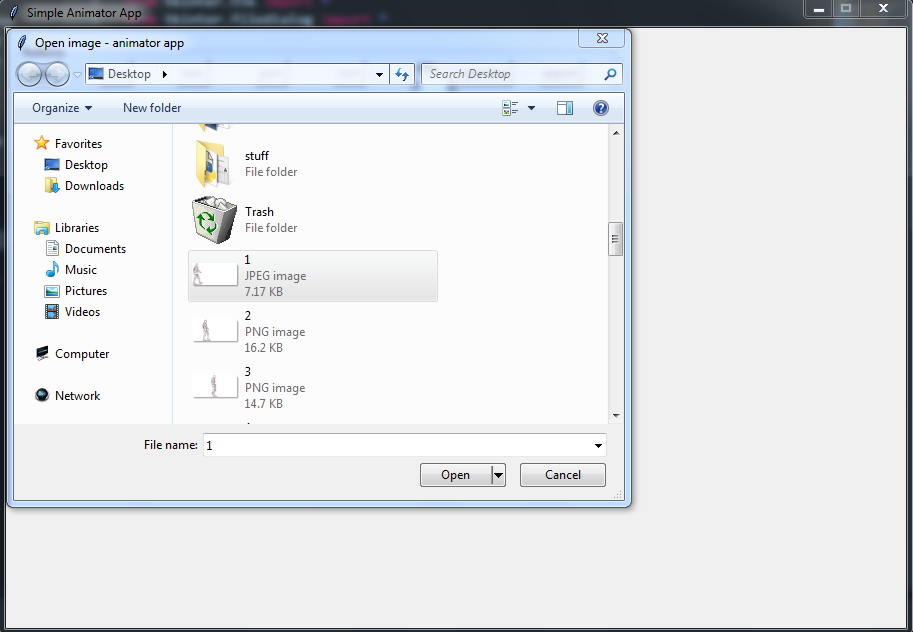
So, first would be the open button, we’ll put the open function in the main class, and connect it to the button, where we have:

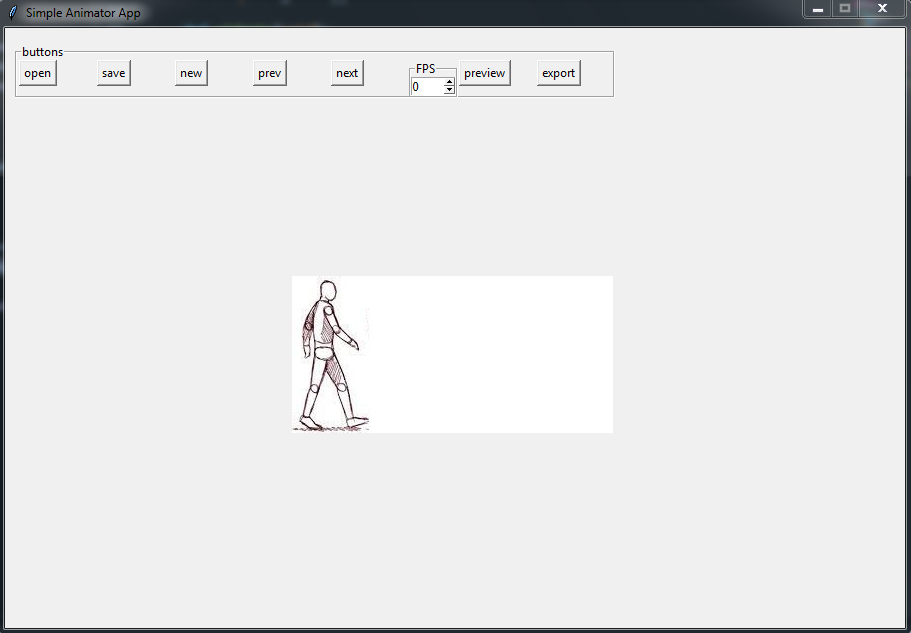
--snip—

btn1 = Button(self.frame, text = "open", command = None)

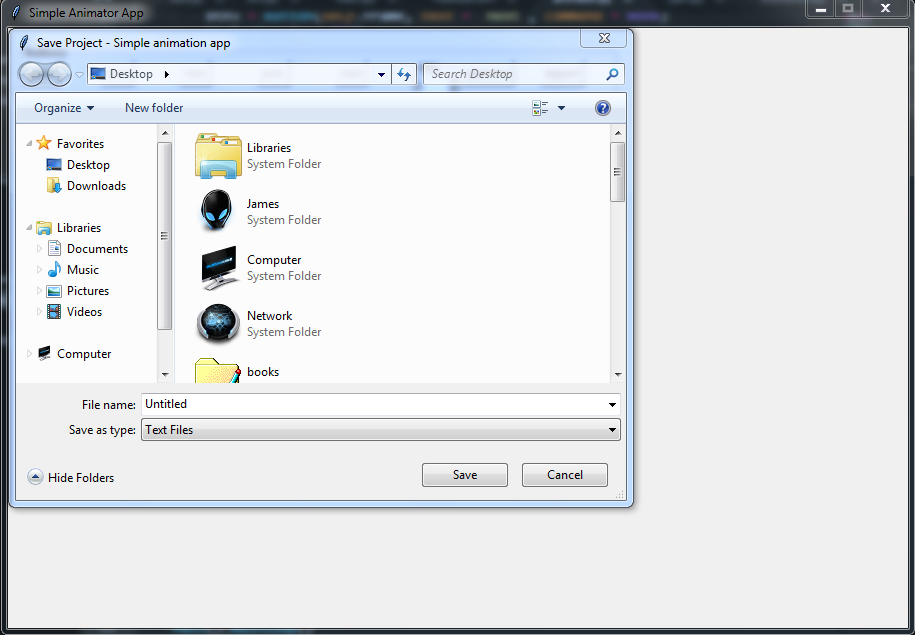
--snip—

Then replace “None” with “self.open”, thus calling the function, when we run the code, and click the open button now, we should get something like this:

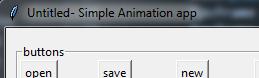




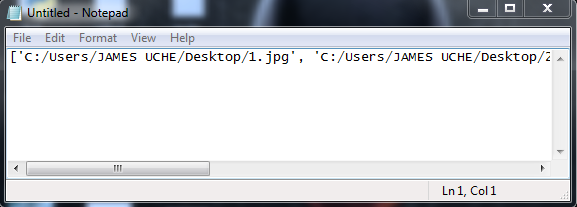
So, now we can open images with the open button, sorry, something went wrong with my pc UI, thus affecting the buttons, you can adjust your buttons coordinates if your UI seems to have a different feel. With that said, you can use the same system to connect the save function, to test the save function, first we open a list of images, and then click save, we should obtain something like this:



So, here we save the file as untitled, notice that the title is changed to work with the name of the file



and when we open the text file, you should see the list of paths of the images, like:

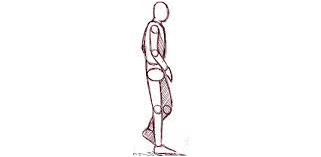


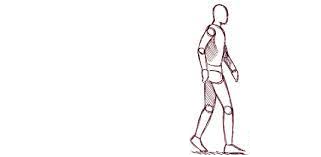
We, can use the file to open it again to continue a project, with an open function, to open saved work, the method would simply open the text file and then use the image list for the animation, that’s how it can be used, but I forgot to cover that, so I’m skipping it, feel free to try it yourself, you can contact me if you need any help.

The “new” button would simply make the imagelist empty and change the title back to ‘Simple Animator app’ while the “prev” and “next” button, can move through the imagelist and display the image.

Final thing is to test the animation. First one would need to create the images before hand, but I have an already made picture we’ll be using for our own, to get stuff like these(for testing the software only), you can just search for “animation frames” on google and you would see some like this you could use.

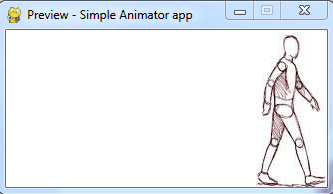








Those are the images we’re using, so we just open them one by one, in our project, so what we do to test the preview, is to first import the preview file where we wrote our preview code, then connect the Animate.animate(images, fps) function to the preview button, and then, what we just have to do is import all the images, set the FPS and click on the preview button. With an fps of 8, you can would have a pygame display, that would flip through the frames and play a short video, that would look like this:



The full animation would be displayed when we export it at the end. So, let’s get to the export method, when it is typed and connected to the export button, the fps should just be set with the project, when clicked it would export our video with the name of the project and we would get something like:

[video of animation]

The full code:

*animator.py*

from tkinter import \*

from tkinter.ttk import \*

from tkinter.filedialog import \*

from PIL import Image, ImageTk

import preview

from tkinter.messagebox import \*

import cv2, os

class Main(Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title('Simple Animation App')

self.geometry('900x600')

self.resizable(False, False)

self.n = 0

self.images = []

self.widgets()

def widgets(self):

self.frame = LabelFrame(self, text="buttons", width = 600, height = 55)

self.frame.place(x = 10, y = 15)

btn1 = Button(self.frame, text = "open", command = self.open)

btn1.place(x = 2)

btn2 = Button(self.frame, text = "save", command = self.save)

btn2.place(x = 80)

btn3 = Button(self.frame, text = "new", command = self.new)

btn3.place(x = 158)

btn4 = Button(self.frame, text = "prev", command = self.prev\_frame)

btn4.place(x = 236)

btn5 = Button(self.frame, text = "next", command = self.next\_frame)

btn5.place(x = 314)

fpsframe = LabelFrame(self.frame, text = "FPS", width = 150, height = 35)

fpsframe.place(x = 392)

self.FPS = IntVar()

fps = Spinbox(fpsframe, from\_ = 0,to = 2000, width = 5, textvariable = self.FPS)

fps.pack()

btn6 = Button(self.frame, text = "preview",

command = lambda:[preview.Animate.animate(self.images, self.FPS.get())])

btn6.place(x = 442)

btn7 = Button(self.frame, text = "export", command = self.export)

btn7.place(x = 520)

def open(self):

global img

file = askopenfilename(title = "Open image - animator app")

self.images.append(file)

img = ImageTk.PhotoImage(Image.open(file))

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

def save(self):

file = asksaveasfilename(title = "Save Project - Simple animation app", initialfile = 'Untitled.txt', defaultextension = ".txt", filetypes = [("Text Files", "\*.txt")])

if file == '':

file == "Untitled"

else:

self.title(os.path.basename(file).replace('.txt', '') + "- Simple Animation app")

with open(file, 'w+') as f:

data = str(self.images)

f.write(data)

f.close()

def new(self):

self.images = []

self.title('Simple Animation app')

def prev\_frame(self):

global img

self.n-=1

img = ImageTk.PhotoImage(Image.open(self.images[self.n]))

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

def next\_frame(self):

global img

self.n += 1

img = ImageTk.PhotoImage(Image.open(self.images[self.n]))

self.view = Label(self, image = img, width = 852, height = 450)

self.view.place(x = 20, y = 100)

def createVid(self, images, fps, title):

imgs = []

for i in images:

img = cv2.imread(i)

height, width, layer = img.shape

size = (width, height)

imgs.append(img)

Title = title.replace('.txt', '')

output = cv2.VideoWriter(Title, cv2.VideoWriter\_fourcc(\*'DIVX'), fps, size)

for i in range(len(imgs)):

output.write(imgs[i])

output.release()

def export(self):

images = self.images

fps = self.FPS.get()

title = 'animation' + '.avi'

self.createVid(images, fps, title)

showinfo("Success", "Your animation was exported succesfully exported to "+title)

if \_\_name\_\_=='\_\_main\_\_':

Main().mainloop()

*preview.py*

import pygame

from time import sleep

from PIL import Image, ImageTk

from tkinter.messagebox import showerror

class Animate:

pygame.init()

def animate(images, fps):

try:

def get\_pic\_size(filename):

image = Image.open(filename)

width, height = image.size

return width, height

imagelist = [i for i in images]

try:

size = get\_pic\_size(imagelist[2])

except IndexError:

showerror('An Error Occured', 'there are not enough frames to animate')

screen = pygame.display.set\_mode((size))

pygame.display.set\_caption("Preview - Simple Animator app")

imagelist = [i for i in images]

for i in imagelist:

try:

img = pygame.image.load(i)

screen.blit(img, [0, 0])

except:

pass

sleep(1/fps)

pygame.display.update()

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_a & pygame.KMOD\_CTRL:

pygame.quit()

pygame.display.update()

except Exception as e:

pass

Thank you for reading.