**DRAWING APPLICATION SYSTEM**

**PROJECT REPORT**

OF MAJOR PROJECT

**BACHELOR OF COMPUTER APPLICATION (BCA)**

SUBMITTED BY

**DINESH SINGH**

**BCA 6th Semester**

Batch Year - 2020-2023

Enrollment No. - U2046035

**PROJECT GUIDE: - Mr. ANAND DURGA SINGH**

****

**Centre of Computer Education & Training**

**Institute of Professional Studies**

**University of Allahabad, Prayagraj, Uttar Pradesh**

**ACKNOWLEDGEMENT**

This project report is based on **‘DRAWING APPLICATION SYSTEM’**. I have taken all my efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Mr. ANAND DURGA SINGH** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project. I would also like to express my gratitude towards my parents & members of **Institute of Professional Studies (IPS), University of Allahabad** for their kind cooperation and encouragement which helped me in completion of this project.

My thanks and appreciation also go to my Friends in developing the project and people who have willingly helped me out with their abilities.

DINESH SINGH

BCA, 6th Semester

Enrollment No. - U2046035

**CERTIFICATE**

It is here by certified that the project work entitled DRAWING APPLICATION SYSTEM using PYTHON is a bonafide work carried out by DINESH SINGH Enrollment NO: U2046035, Bachelor of Computer Application in Centre of Computer Education & Training, Institute of Professional Studies, University of Allahabad during the year 2020-2023.

It is certified that all the correction/suggestion indicated for Major Project have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work allotted for the said degree.

**Name of supervisor:** Mr. ANAND DURGA SINGH

**DECLARATION**

I, **DINESH SINGH,** solemnly declare that the project report **DRAWING APPLICATION SYSTEM** is based on my own work carried out during the course of our study under the supervision of **Mr. ANAND DURGA SINGH.**

I assert the statements made and conclusions drawn are an outcome of my research work. I further certify that

1. The work contained in the report is original and has been done by me under the general supervision of my supervisor.
2. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or Abroad.
3. I have followed the guidelines provided by the university in writing the report.
4. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and given their details in the references.

DINESH SINGH

BCA, 6th Semester

Enrollment No. – U2046035

SYNOPSIS

**TABLE OF CONTENT**

1. Introduction… 7
2. Problem Definition… 7
3. Motivation… 8
4. Objective 8
5. Requirement Analysis… 9
6. System Design… 10
7. Project Architecture 11

**INTRODUCTION**

Drawing Application System is the application software used to create simple images like vector graphics on canvas using various tools like Pencil, Eraser, Brushes, colors and various shapes for example rectangle, circle, oval etc.

User can also import files like images from the local computer and use it in their Drawing project. This application also provides features to save, save as, edit, open, close files and so on.

This application provides support of large number of colors and provides features to insert text which helps user to make their drawing project more interactive and eye-catching.

# **PROBLEM DEFINITION**

Drawing Application helps user to draw sketches, images and pictures according to the user necessity. This application will also provide features like edit, where we can resize, change its appearance, compress the selected images or pictures.

User can register or login him/her with User Id and password which create separate workspace for them where they can save their work and made it inaccessible from the other user. User can also access their works remotely by simply login in the application.

This application is useful for a beginner for creating simple drawing as well as to create advanced designs as it has vast range of tools support.

# **MOTIVATION**

The Graphical User Interface of this application is very interactive and user friendly which provides ease of use to the user so they can use this application without any difficulty. In this we have all important graphical icons like pencil, brush, eraser, coloring tools, text boxes and all by which user can easily use these tools to create their illustrations to fulfill their requirements.

User can also create their own personal lobby where they can hide their work from other users by creating User id and Password.

# **OBJECTIVE**

The main objective of this project to build a Drawing Application System using Python which provides the following features.

* Drawing tools to draw objects, for example lines, arcs, polygons etc.
* Vast amount of coloring tools to make the drawing objects more attractive.
* Facility to import other image files which can be used in the current work.
* User can have their separate workspace which made this application multiuser.
* User ID and Password make user’s files secure from unwanted access and modification.

# **REQUIREMENT ANALYSIS**

**SOFTWARE REQUIREMENT**

* **Operating System:**  Window/Linux/Mac
* **Language Used:**  Python
* **Additional Software:**  Photoshop
* **IDE:**  VsCode/PyCharm/IDLE

**HARDWARE REQUIREMENT**

* **CPU:** Intel I5 Gen-10
* **RAM:** 8 GB or Higher

# **SYSTEM DESIGN**

System Design includes the various modules of the system which will build. In other word it describes about the features that the software is going to provide and how it will be helpful to the target user/client.

# Main Window

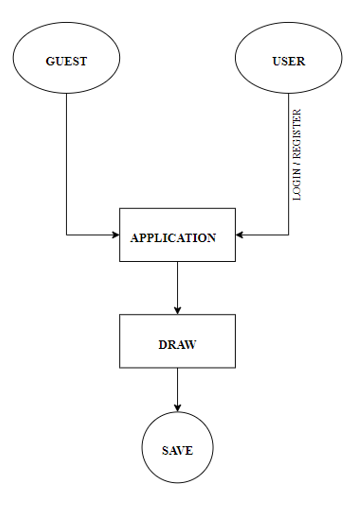
Basic Interface with all the important tools and the canvas area for drawing the sketches.

# Login/Register

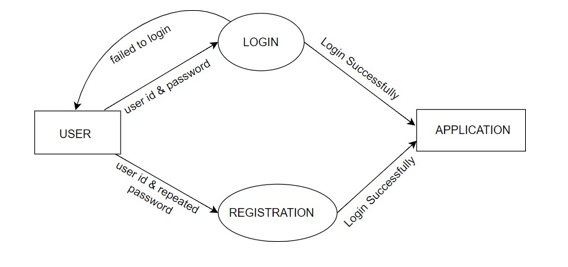
Here user can register/login him/her to create their own workspace to accomplish security and confidentiality.

# **PROJECT ARCHITECTURE**

**BASIC PROJECT STRUCTURE**



LOGIN & REGISTRATION



PROJECT REPORT

**TABLE OF CONTENT**

1. SYNOPSIS.……………………………………………………………………………5-11
2. INTRODUCTION………………………………………………………………….…....14
   1. Objective
   2. Project Description
3. MOTIVATION…………………………………………………………..........................15
4. REQUIREMENT & SOFTWARE ANALYSIS………………………............................16
   1. Software Requirement
   2. Hardware Requirement
5. SOFTWARE ANALYSIS……………………………………………………………….17
6. TECHNOLOGIES USED……………………………………………………………….18
7. SYSTEM DESIGN…………………………...………………………………………….19
8. ACTIVITY DIAGRAM………………………………………………….........................20
9. CODING………………………………………………….……………………...…...21-79
10. SNAPSHOTS.……………………………………………………………………………80
11. APPLICATIONS……………………………………………………………………. ….81
12. FUTURE SCOPE……………………………………………………………………. ….82
13. CONCLUTION……………………………………………………………….………….83
14. MILESTONE…………………………………………………………………………… 84
15. MEETING WITH THE SUPERVISOR……………………………………………….....85
16. REFERENCES……………………………………..........................................................86

**INTRODUCTION**

## 1.1 Objective

The main objective of the project is to build a Drawing Application System using Python as a programming language which helps users to draw sketches and drawing as per their requirement.

## 1.2 Project Description

* + - Drawing Application System is the application software used to create simple images like vector graphics on canvas using various tools like Pencil, Eraser, Brushes, colors and various shapes for example rectangle, circle, oval etc.
    - User can also import files like images from the local computer and use it in their Drawing project. This application also provides features to save, save as, edit, open, close files and so on.
    - This application provides support of large number of colors and provides features to insert text which helps user to make their drawing project more interactive and eye-catching.

**MOTIVATION**

The motivation for creating a drawing application can vary depending on the goals of project and the target audience. Some common motivations for creating a drawing application includes:

1. Creative Expression: People enjoy using drawing applications to create digital art, express themselves, and experiment with different colors, textures, and styles.
2. Productivity: Drawing applications can be useful tools for graphic designers, artists, and other professionals who need to create visual content quickly and efficiently.
3. Education: Drawing applications can be used in educational settings to teach art and design concepts, as well as to facilitate collaborative learning and creativity.
4. Entertainment: Drawing applications can be designed to be fun and engaging, offering users a way to unwind and express their creativity in a casual setting.

Overall, a drawing application can provide a versatile and accessible platform for people to express their creativity, communicate visually, and explore new ideas.

**REQUIREMENT & SOFTWARE ANALYSIS**

The process of deciding on the requirement of a software system, which determines the responsibilities of a system, is called requirement analysis. Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.

Requirement engineering activities result in the specification of software’s operational characteristics, indicate the software’s interface with other system elements and establish constraints that the software must meet. The following section presents the detailed requirement analysis of our project

**3.1 SOFTWARE REQUIREMENT**

* **Operating System:**  Window/Linux/Mac
* **Language Used:**  Python
* **Additional Software:**  Photoshop
* **IDE:**  VsCode/PyCharm/IDLE

**3.2 HARDWARE REQUIREMENT**

* **CPU:** Intel I5 Gen-10
* **RAM:** 8 GB or Higher

**SOFTWARE ANALYSIS**

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human readable requirements to be transformed into actual code.

In order to achieve the objectives of coming up with the DRAWING APPLICATION SYSTEM, the preliminary investigation about the workability of the software is necessary and need to be carried out first. This will equip me with the relevant materials and knowledge on how to carry out the implementation.

**TECHNOLOGIES USED**

**PYTHON** is a language used to build proposed project on Drawing Application System. In this Project we have used GUI feature of Python called tkinter and various other modules as follows:-

* **Tkinter:** Tkinter is a de-facto standard GUI package for Pythons (Graphical User Interface). It is on top of Tcl/Tk, a thin object-oriented layer. TKinter is not the only Python toolkit for Gui Programming. However, it is the most commonly used one. Cameron Laird calls the annual decision to keep Tkinter "one of the minor traditions of the Python world." If you run python-m tkinter from the command prompt, it is important to open a window showing a simple Tk interface, letting you know that tkinter is correctly installed on your device, and also showing which version of Tcl/Tk is running.
* **Pillow:** The Standard Python Library (abbreviated as PIL) (known in newer versions as Pillow) is an additional free and open-source Python programming language library that supports the opening, manipulation, and saving of several different image file formats.
* **Time:** As the name suggests Python time module allows to work with time in Python. It allows functionality like getting the current time, pausing the Program from executing, etc. So before starting with this module we need to import it.

**SYSTEM DESIGN**

System Design includes various modules of the system which is build. In other word it describes about the features that software is going to provide and how it will be helpful to the target user/client.

1. **Main Window**

Main window is divided into two parts, 1st upper part for controllers and 2nd lower for drawing sketches

1. **Controllers**

Controllers includes various sub parts providing buttons dedicated for different purposes.

These sub parts are as follows: -

* **Tools: -** Contains Buttons like pencil, eraser, color pen, clear, text box and select region.
* **Width Controller: -** Contains slider to choose the width of outline and eraser.
* **Index Box: -** Contains the recent drawn object as number, user can manipulate these objects by selecting in the index box
* **Shapes & Lines: -** Contains lines and shapes like, rectangle, parallelogram, triangle, pentagon, hexagon, arrow, circle etc.
* **Color Box: -** Contains various colors for drawing purpose.
* **Mode Box: -** Contains coloring modes as fill (fill the shapes and line), outline (change the outline color of the shapes) and permanent fill and outline color for choosing the fill and outline color for upcoming shapes.

1. **Menu Bar**

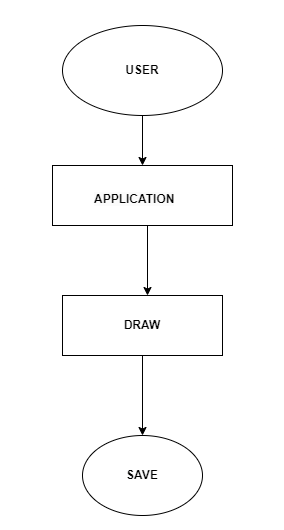
Menu bar contains following options: - new, open, save, undo, cut, copy, paste, screenshot, clear, change background color, zoom, color pen width controller, movement, about, tips, about etc.

1. **Status Bar**

Display Coordinate position of the cursor along with the message according to the selected tool.

**ACTIVITY DIAGRAM**

Activity diagram provides a graphical representation of various activities that are carried out in an application. A clear overview of the DRAWING APPLICATION SYSTEM will be provided from the beginning till the end. The workflow of the activities and their dependencies with each other during the execution of the tasks and processes is shown in the activity diagram. The major functionalities of the site have been depicted using the diagram as shown below.



**CODING**

#Importing All Necessary libraries

from tkinter import \*

from tkinter import messagebox,colorchooser,filedialog

from PIL import Image,ImageTk,ImageGrab

import time

class Draw:

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

self.window = Tk()

self.window.title("DRAWINGO : wings to your imagination")

self.window.geometry("1536x864")

self.window.maxsize(1536,864)

self.window.minsize(1536,864)

self.window.config(bg="linen")

self.window.iconbitmap("Icons/appicon.ico")

#Variable

self.main\_menu = None

self.file\_menu = None

self.edit\_menu = None

self.color\_menu = None

self.option\_menu = None

self.help\_menu = None

self.coord = None

self.status\_msg = None

self.notation\_box = None

self.tools = None

self.width\_controller = None

self.Index\_Box = None

self.shapes\_lines = None

self.delete\_seg = None

self.top = None

self.choosing\_color = None

self.intro\_img = None

#All necessary variables

self.shape\_outline\_width\_label = None

self.eraser\_width\_label = None

self.eraser\_controller = None

#Input variable initialization

self.fill\_information = IntVar()

self.outline\_information = IntVar()

self.input\_take = StringVar()

self.fill\_information.set(0)

self.outline\_information.set(0)

self.input\_take.set(" ")

#All scale initialization under Text button

self.font\_size = Scale()

self.font\_size.set(20)

#Coordinate variable

self.old\_x = None

self.old\_y = None

self.new\_x = None

self.new\_y = None

#Some value initialization

self.color\_circle\_width\_maintainer = 15

self.img\_counter = -1

self.width\_controller\_scale = 0

self.counter = -1

self.width\_maintainer = 2

self.erase\_width\_maintainer = 5

self.active\_coloring = 2

#All initialize lists

self.img\_container = []

self.cut\_copy\_img = []

self.undo\_container = []

self.redo\_container = []

self.temp = []

self.tools\_container = []

self.shapes\_container = []

self.colors\_container = []

self.modes\_container = []

self.menu\_img\_container = []

self.about\_img = []

#By default set color

self.fill\_color = "white"

self.fill\_color\_line = "black"

self.outline\_color\_line = "black"

self.text\_fg = "black"

self.color\_container\_box = "black"

#All necessary Buttons

self.tool\_btn = Button()

self.shape\_btn = Button()

self.color\_btn = Button()

self.mode\_btn = Button()

#All necessary frame

self.main\_frame = LabelFrame()

self.tools = LabelFrame()

self.width\_controller = LabelFrame()

self.Index\_Box = LabelFrame()

self.shapes\_lines = LabelFrame()

self.color\_Box = LabelFrame()

self.color\_mode = LabelFrame()

#All necessary canvas

self.drawing\_canvas = Canvas()

#Creating Frame for placing frame for tool, shapes, colors etc

self.main\_frame = LabelFrame(self.window,text="CONTROLLERS",cursor="arrow",labelanchor=SE,relief=GROOVE,font=("timesnewroman",12,"bold"),bg="snow",height=160,width=1536,highlightbackground="grey",highlightthickness=2)

self.main\_frame.grid(row=0,column=0,sticky=NSEW,pady=5)

#Creating canvas for drawing

self.drawing\_canvas = Canvas(self.window,bg = "white",width=1510,height=570, highlightthickness=2, highlightbackground="grey",relief=GROOVE)

self.drawing\_canvas.grid(row=1,column=0,sticky=NSEW,padx = 10, pady = 1)

#Calling controls method which contains frame and buttons for tools, shapes, color etc

self.controls(0)

#Creating Controllers to various tools,shape, color box etc

self.controllers()

#Creating all necessary menu option

self.make\_menu()

#Creating status bar

self.make\_status()

#setting up zoom feature in canvas

self.drawing\_canvas.bind("<Control-MouseWheel>",self.zoom\_controller)

#seting up color box width controller

self.drawing\_canvas.bind("<Shift-MouseWheel>",self.color\_box\_width\_controller)

#displaying pixel position in status bar

self.drawing\_canvas.bind("<Motion>",self.movement\_cursor)

self.window.mainloop()

def make\_menu(self):

self.main\_menu = Menu(self.window)

self.window.config(menu=self.main\_menu)

menu\_img = ["new.png", "open.png", "save.png", "exit.png", "undo.png", "clear.png", "cut.png", "copy.png", "paste.png","screenshot.png", "bgcolor.png", "fill\_outline.png", "zoom\_in.png", "zoom\_out.png", "color\_pen.png", "movement.png","about.png","tips.png"]

for i in range(18):

self.menu\_img\_container.append(i)

self.menu\_img\_container[i] = ImageTk.PhotoImage(Image.open("Icons/"+menu\_img[i]).resize((30,30)))

#Creating File Menu

self.file\_menu = Menu(self.main\_menu,tearoff=False)

self.main\_menu.add\_cascade(label="File",menu=self.file\_menu)

#Placing options to file menu

self.file\_menu.add\_command(label="New",accelerator="(Ctrl+N)",command=lambda: self.controls(19),image=self.menu\_img\_container[0],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.file\_menu.add\_command(label="Open",accelerator="(Ctrl+O)",command=lambda: self.open(False),image=self.menu\_img\_container[1],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.file\_menu.add\_command(label="Save",accelerator="(Ctrl+S)",command=lambda: self.save(False),state=DISABLED,image=self.menu\_img\_container[2],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.file\_menu.add\_command(label="Exit",command=lambda: self.controls(20) ,image=self.menu\_img\_container[3],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.window.bind('<Control-Key-n>', lambda e: self.controls(19))

self.window.bind('<Control-Key-o>', self.open)

self.window.bind('<Control-Key-s>', self.save)

#Creating edit menu

self.edit\_menu = Menu(self.main\_frame,tearoff=False)

self.main\_menu.add\_cascade(label="Edit",menu=self.edit\_menu)

#Placing options in edit menu

self.edit\_menu.add\_command(label="Undo",accelerator="(Ctrl+Z)",command=lambda: self.undo(False),image=self.menu\_img\_container[4],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.edit\_menu.add\_command(label="Clear",command=lambda: self.controls(18),image=self.menu\_img\_container[5],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.edit\_menu.add\_command(label="Cut",accelerator="(Ctrl+X)",command=lambda: self.cut(False),image=self.menu\_img\_container[6],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.edit\_menu.add\_command(label="Copy",accelerator="(Ctrl+C)",command=lambda: self.copy(0),image=self.menu\_img\_container[7],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.edit\_menu.add\_command(label="Paste",accelerator="(Ctrl+V)",command=lambda: self.paste(False),image=self.menu\_img\_container[8],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.edit\_menu.add\_command(label="Screenshot",accelerator="(Ctrl+Alt+C)",command=lambda: self.screenshot(False),image=self.menu\_img\_container[9],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.window.bind("<Control-Key-z>",self.undo)

self.window.bind("<Control-Key-x>", self.cut)

self.window.bind("<Control-Key-c>",self.copy)

self.window.bind("<Control-Key-v>", self.paste)

self.window.bind("<Control-Alt-Key-c>",self.screenshot)

#Creating edit menu

self.color\_menu = Menu(self.main\_frame,tearoff=False)

self.main\_menu.add\_cascade(label="Color",menu=self.color\_menu)

self.color\_menu.add\_command(label="Change Background Color",command=lambda: self.controls(21),image=self.menu\_img\_container[10],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.color\_menu.add\_command(label="Change Permanent Fill and Outline Color",command=self.change\_permanent\_fill\_outline\_color,image=self.menu\_img\_container[11],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

#Creating option menu

self.option\_menu = Menu(self.main\_menu,tearoff=False)

self.main\_menu.add\_cascade(label="Option",menu=self.option\_menu)

#Placing options in option menu

self.option\_menu.add\_command(label="Zoom in",accelerator="(Ctrl+Scroll up)",command=lambda: self.zoom\_controller(1),image=self.menu\_img\_container[12],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.option\_menu.add\_command(label="Zoom out",accelerator="(Ctrl+Scroll down)",command=lambda: self.zoom\_controller(0),image=self.menu\_img\_container[13],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.option\_menu.add\_separator(background="linen")

self.option\_menu.add\_command(label="Color Pen Width Increase",accelerator="(Shift+Scroll up)",command=lambda: self.color\_box\_width\_controller(1),image=self.menu\_img\_container[14],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.option\_menu.add\_command(label="Color Pen Width Decrease",accelerator="(Shift+Scroll down)",command=lambda: self.color\_box\_width\_controller(0),image=self.menu\_img\_container[14],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.option\_menu.add\_separator(background="linen")

self.option\_menu.add\_command(label="Movement",command=lambda: self.controls(22),image=self.menu\_img\_container[15],compound=LEFT,background="linen",foreground="black",state=DISABLED,font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

#Create Help Menu

self.help\_menu = Menu(self.main\_menu,tearoff=False)

self.main\_menu.add\_cascade(label="Help",menu=self.help\_menu)

#Placing Options in Help menu

self.help\_menu.add\_command(label="About",command=self.about,image=self.menu\_img\_container[16],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

self.help\_menu.add\_command(label="Tips",command=self.tips,image=self.menu\_img\_container[17],compound=LEFT,background="linen",foreground="black",font=("timesnewroman",12,"bold"),activebackground="silver",activeforeground="black")

def make\_status(self):

#Creating label for status message

self.status\_msg = Label(self.window,text="Draw With Fun\t",bg = "grey",fg="black",font=("timesnewroman",10,"bold"),anchor=NE)

self.status\_msg.grid(row=2,column=0,sticky=NSEW)

#Creating label for coordinate values

self.coord = Label(self.status\_msg,text="",bg="grey",fg="black",font=("timesnewroman",10,"bold"),anchor=NW)

self.coord.grid(row=2,column=0)

#For cursor position by movement

def movement\_cursor(self,event):

self.coord.config(text=f"{event.x},{event.y}px")

def controls(self,notation):

if self.temp:

self.drawing\_canvas.delete(self.temp.pop())

if self.notation\_box:

if self.notation\_box["state"] == DISABLED:

self.notation\_box["state"] = NORMAL

self.drawing\_canvas.config(cursor="TCROSS")

self.drawing\_canvas.unbind("<B1-Motion>")

self.drawing\_canvas.unbind("<ButtonRelease-1>")

self.drawing\_canvas.unbind("<Button-1>")

if notation == 1:

self.drawing\_canvas.config(cursor="Pencil")

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_with\_pencil)

elif notation == 2:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_circle)

elif notation == 3:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_rectangle)

elif notation == 4:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_bent\_line)

self.drawing\_canvas.bind("<Shift-B1-Motion>",self.draw\_straight\_line)

elif notation == 5:

self.drawing\_canvas.config(cursor=DOTBOX)

self.drawing\_canvas.bind("<B1-Motion>",self.eraser)

elif notation == 6:

self.text\_creation\_input\_take()

elif notation == 7:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_triangle)

elif notation == 8:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_parallelogram)

elif notation == 9:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_pentagon)

elif notation == 10:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_hexagon)

elif notation == 11:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_arrow\_up\_down)

elif notation == 12:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_dashed\_line)

elif notation == 13:

self.drawing\_canvas.bind("<B1-Motion>",self.select\_region)

self.drawing\_canvas.bind("<Delete>",self.delete\_selected\_region)

elif notation == 14:

self.drawing\_canvas.config(cursor="circle")

self.color\_container\_box = colorchooser.askcolor()[1]

self.drawing\_canvas.bind("<B1-Motion>",self.color\_boxer)

elif notation == 15:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_arrow\_left\_right)

elif notation == 16:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_rounded\_rectangle)

elif notation == 17:

self.drawing\_canvas.bind("<B1-Motion>",self.draw\_right\_angled\_triangle)

elif notation == 18 or notation == 19:

if notation == 18:

take = messagebox.askyesno("Clear Conformation","Are you sure to Clear?")

else:

take = messagebox.askyesno("New Window Conformation","Are you want to open new window?")

if take is True:

self.drawing\_canvas.delete("all")

self.clear()

elif notation == 20:

take = messagebox.askyesno("Exit Confermation","Are you sure to Exit?")

if take is True:

self.window.destroy()

elif notation == 21:

take = colorchooser.askcolor()[1]

if take:

self.drawing\_canvas["bg"] = take

self.drawing\_canvas.update()

elif notation == 22:

messagebox.showinfo("Movement Direction","At first click on the shape or line number from indexing box\n\n1. Right Arrow----> Right Movement\n\n2. Left Arrow----> Left Movement\n\n3. Up Arrow---->Up Movement\n\n4. Down Arrow--->Down Movement\n\n5. Space Button--->Stop Movement")

def controllers(self):

#Creating frame for tools

self.tools = LabelFrame(self.main\_frame,text="Tools",labelanchor=S,bg="white",height=130,width=150,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.tools.grid(row=0,column=0,padx=10,pady=5)

tools\_storage = ["pencil.png","eraser.png","color\_box.png","clear.png","text\_box.png","selection\_box.png"]

for i in range(6):

self.tools\_container.append(i)

self.tools\_container[i] = ImageTk.PhotoImage(Image.open("Icons/"+tools\_storage[i]).resize((20,20)))

#Pencil

self.tool\_btn = Button(self.tools,image=self.tools\_container[0],relief=RAISED,command=lambda: self.controls(1))

self.tool\_btn.grid(row=0,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Eraser

self.tool\_btn = Button(self.tools,image=self.tools\_container[1],relief=RAISED,command=lambda: self.controls(5))

self.tool\_btn.grid(row=0,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Color Box

self.tool\_btn = Button(self.tools,image=self.tools\_container[2],relief=RAISED,command=lambda: self.controls(14))

self.tool\_btn.grid(row=1,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Clear

self.tool\_btn = Button(self.tools,image=self.tools\_container[3],relief=RAISED,command=lambda: self.controls(18))

self.tool\_btn.grid(row=1,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Text box

self.tool\_btn = Button(self.tools,image=self.tools\_container[4],relief=RAISED,command=lambda :self.controls(6))

self.tool\_btn.grid(row=2,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Selection

self.tool\_btn = Button(self.tools,image=self.tools\_container[5],relief=RAISED,command=lambda: self.controls(13))

self.tool\_btn.grid(row=2,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Movement Keyboard setup

self.window.bind("<space>",self.movement)

self.window.bind("<Left>",self.movement)

self.window.bind("<Right>",self.movement)

self.window.bind("<Up>",self.movement)

self.window.bind("<Down>",self.movement)

#Creating frame for width controller

self.width\_controller = LabelFrame(self.main\_frame,text="Width Controller",labelanchor=S,bg="white",height=130,width=150,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.width\_controller.grid(row=0,column=1,padx=10,pady=5)

#Shape Border Width Controller

def shape\_outline\_width\_controller(event):

self.width\_maintainer = event

#Eraser Width Controller

def eraser\_width\_controller(event):

self.erase\_width\_maintainer = event

#Shape Border width label

self.shape\_outline\_widht\_label = Label(self.width\_controller,text="Outline Width :- ",font=("timesnewroman",12,"bold"),bg="white",fg="black")

self.shape\_outline\_widht\_label.grid(row=0,column=0,padx=2,pady=2)

#Shape Border width scale

self.width\_controller\_scale = Scale(self.width\_controller,orient=HORIZONTAL,from\_=0,to=100,bg="black",fg="white",font=("timesnewroman",10,"bold"),relief=RAISED,bd=2,command=shape\_outline\_width\_controller,activebackground="silver")

self.width\_controller\_scale.set(self.width\_maintainer)

self.width\_controller\_scale.grid(row=0,column=1,padx=2,pady=2)

#Eraser Width label

self.eraser\_width\_label = Label(self.width\_controller,text="Eraser Width :- ",font=("timesnewroman",12,"bold"),bg="white",fg="black")

self.eraser\_width\_label.grid(row=1,column=0,padx=2,pady=2)

#Eraser Width Scale

self.eraser\_controller = Scale(self.width\_controller,orient=HORIZONTAL,from\_=0,to=100,bg="black",fg="white",activebackground="silver",font=("timesnewroman",10,"bold"),relief=RAISED,bd=2,command=eraser\_width\_controller)

self.eraser\_controller.set(self.erase\_width\_maintainer)

self.eraser\_controller.grid(row=1,column=1,padx=2,pady=2)

#Creating frame for Index box

self.Index\_Box = LabelFrame(self.main\_frame,text="Index Box",labelanchor=S,bg="white",height=130,width=100,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.Index\_Box.grid(row=0,column=2,padx=10,pady=5)

#Creating index box

self.notation\_box = Listbox(self.Index\_Box,width=5,height=4,font=("timesnewroman",12,"bold"),fg="black",bg="linen",relief=GROOVE,bd=5)

self.notation\_box.grid(row=0,column=0,padx=10,pady=5)

#Creating frame for shapes and lines

self.shapes\_lines = LabelFrame(self.main\_frame,text="Shape & Lines",labelanchor=S,bg="white",height=130,width=300,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.shapes\_lines.grid(row=0,column=3,padx=10,pady=5)

shape\_storage = ["line.png","dashed\_line.png","rectangle.png","parallelogram.png","triangle.png","pentagon.png","hexagon.png","arrow.png","circle.png","right\_angled\_triangle.png","rounded\_rectangle.png","left\_arrow.png"]

for i in range(12):

self.shapes\_container.append(i)

self.shapes\_container[i] = ImageTk.PhotoImage(Image.open("Icons/"+shape\_storage[i]).resize((20,20)))

#Creating line

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[0],relief=RAISED,command=lambda: self.controls(4))

self.shape\_btn.grid(row=0,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Creating dashed line

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[1],relief=RAISED,command=lambda: self.controls(12))

self.shape\_btn.grid(row=0,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Creating rectangle

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[2],relief=RAISED,command=lambda: self.controls(3))

self.shape\_btn.grid(row=0,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Creating parallelogram

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[3],relief=RAISED,command=lambda: self.controls(8))

self.shape\_btn.grid(row=0,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Creating triangle

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[4],relief=RAISED,command=lambda: self.controls(7))

self.shape\_btn.grid(row=1,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Creating pentagon

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[5],relief=RAISED,command=lambda: self.controls(9))

self.shape\_btn.grid(row=1,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Creating hexagon

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[6],relief=RAISED,command=lambda: self.controls(10))

self.shape\_btn.grid(row=1,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Creating Arrow

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[7],relief=RAISED,command=lambda: self.controls(11))

self.shape\_btn.grid(row=1,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Creating Circle

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[8],relief=RAISED,command=lambda: self.controls(2))

self.shape\_btn.grid(row=2,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Creating Right Angled Triangle

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[9],relief=RAISED,command=lambda: self.controls(17))

self.shape\_btn.grid(row=2,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Creating Rounded Rectangle

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[10],relief=RAISED,command=lambda: self.controls(16))

self.shape\_btn.grid(row=2,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Creating Left arrow

self.shape\_btn = Button(self.shapes\_lines,image=self.shapes\_container[11],relief=RAISED,command=lambda: self.controls(15))

self.shape\_btn.grid(row=2,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Creating frame for color box

self.color\_Box = LabelFrame(self.main\_frame,text="Color Box",labelanchor=S,bg="white",height=130,width=300,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.color\_Box.grid(row=0,column=4,padx=10,pady=5)

color\_storage = ["red.png","brown.png","blue.png","grey.png","yellow.png","green.png","orange.png","black.png","white.png","pink.png","indigo.png","voilet.png","light\_green.png","olive.png","maroon.png","blush.png","silver.png","saffron.png","cream.png","peach.png","choose\_color.png"]

for i in range(21):

self.colors\_container.append(i)

self.colors\_container[i] = ImageTk.PhotoImage(Image.open("Icons/"+color\_storage[i]).resize((20,20)))

#Button for red color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[0],relief=RAISED,command=lambda: self.check(0,self.active\_coloring))

self.color\_btn.grid(row=0,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Button for brown color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[1],relief=RAISED,command=lambda: self.check(1,self.active\_coloring))

self.color\_btn.grid(row=0,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Button for blue color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[2],relief=RAISED,command=lambda: self.check(2,self.active\_coloring))

self.color\_btn.grid(row=0,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Button for grey color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[3],relief=RAISED,command=lambda: self.check(3,self.active\_coloring))

self.color\_btn.grid(row=0,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Button for yellow color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[4],relief=RAISED,command=lambda: self.check(4,self.active\_coloring))

self.color\_btn.grid(row=0,column=4,padx=2,pady=2,ipadx=2,ipady=2)

#Button for green color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[5],relief=RAISED,command=lambda: self.check(5,self.active\_coloring))

self.color\_btn.grid(row=0,column=5,padx=2,pady=2,ipadx=2,ipady=2)

#Button for orange color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[6],relief=RAISED,command=lambda: self.check(6,self.active\_coloring))

self.color\_btn.grid(row=0,column=6,padx=2,pady=2,ipadx=2,ipady=2)

#Button for black color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[7],relief=RAISED,command=lambda: self.check(7,self.active\_coloring))

self.color\_btn.grid(row=1,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Button for white color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[8],relief=RAISED,command=lambda: self.check(8,self.active\_coloring))

self.color\_btn.grid(row=1,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Button for pink color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[9],relief=RAISED,command=lambda: self.check(9,self.active\_coloring))

self.color\_btn.grid(row=1,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Button for indigo color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[10],relief=RAISED,command=lambda: self.check(10,self.active\_coloring))

self.color\_btn.grid(row=1,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Button for voilet color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[11],relief=RAISED,command=lambda: self.check(11,self.active\_coloring))

self.color\_btn.grid(row=1,column=4,padx=2,pady=2,ipadx=2,ipady=2)

#Button for light green color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[12],relief=RAISED,command=lambda: self.check(12,self.active\_coloring))

self.color\_btn.grid(row=1,column=5,padx=2,pady=2,ipadx=2,ipady=2)

#Button for olive color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[13],relief=RAISED,command=lambda: self.check(13,self.active\_coloring))

self.color\_btn.grid(row=1,column=6,padx=2,pady=2,ipadx=2,ipady=2)

#Button for maroon color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[14],relief=RAISED,command=lambda: self.check(14,self.active\_coloring))

self.color\_btn.grid(row=2,column=0,padx=2,pady=2,ipadx=2,ipady=2)

#Button for blush color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[15],relief=RAISED,command=lambda: self.check(15,self.active\_coloring))

self.color\_btn.grid(row=2,column=1,padx=2,pady=2,ipadx=2,ipady=2)

#Button for silver color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[16],relief=RAISED,command=lambda: self.check(16,self.active\_coloring))

self.color\_btn.grid(row=2,column=2,padx=2,pady=2,ipadx=2,ipady=2)

#Button for night color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[17],relief=RAISED,command=lambda: self.check(17,self.active\_coloring))

self.color\_btn.grid(row=2,column=3,padx=2,pady=2,ipadx=2,ipady=2)

#Button for cream color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[18],relief=RAISED,command=lambda: self.check(18,self.active\_coloring))

self.color\_btn.grid(row=2,column=4,padx=2,pady=2,ipadx=2,ipady=2)

#Button for peach color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[19],relief=RAISED,command=lambda: self.check(19,self.active\_coloring))

self.color\_btn.grid(row=2,column=5,padx=2,pady=2,ipadx=2,ipady=2)

#Button for choose color

self.color\_btn = Button(self.color\_Box,image=self.colors\_container[20],relief=RAISED,command=lambda: self.check(20,self.active\_coloring))

self.color\_btn.grid(row=2,column=6,padx=2,pady=2,ipadx=2,ipady=2)

#Frame for color mode

self.color\_mode = LabelFrame(self.main\_frame,text="Mode",labelanchor=S,bg="white",height=130,width=300,highlightbackground="grey",highlightthickness=2,relief=GROOVE,font=("timesnewroman",10,"bold"))

self.color\_mode.grid(row=0,column=5,padx=10,pady=5)

modes\_storage = ["fill.png","outline.png","permanent.png"]

for i in range(3):

self.modes\_container.append(i)

self.modes\_container[i] = ImageTk.PhotoImage(Image.open("Icons/"+modes\_storage[i]).resize((20,20)))

#Button for Fill

self.mode\_btn = Button(self.color\_mode,image=self.modes\_container[0],relief=RAISED,command=lambda: self.activate\_coloring(1))

self.mode\_btn.grid(row=0,column=0,padx=10,pady=2,ipadx=2,ipady=2)

#Button for Outline

self.mode\_btn = Button(self.color\_mode,image=self.modes\_container[1],relief=RAISED,command=lambda: self.activate\_coloring(2))

self.mode\_btn.grid(row=1,column=0,padx=10,pady=2,ipadx=2,ipady=2)

#Button for permanent color

self.mode\_btn = Button(self.color\_mode,image=self.modes\_container[2],relief=RAISED,command=self.change\_permanent\_fill\_outline\_color)

self.mode\_btn.grid(row=2,column=0,padx=10,pady=2,ipadx=2,ipady=2)

#Movement of any widget by selecting indexing number

def movement(self,event):

try:

self.status\_msg['text'] = "Movement"

take = self.notation\_box.get(ACTIVE)

self.notation\_box.config(state=DISABLED)

take = self.undo\_container[take]

if event.keycode == 32:

self.notation\_box.config(state=NORMAL)

if event.keycode == 37:

if type(take) == list:

for x in take:

self.drawing\_canvas.move(x, -8, 0)

else:

self.drawing\_canvas.move(take, -8, 0)

if event.keycode == 38:

if type(take) == list:

for x in take:

self.drawing\_canvas.move(x, 0, -8)

else:

self.drawing\_canvas.move(take, 0, -8)

if event.keycode == 39:

if type(take) == list:

for x in take:

self.drawing\_canvas.move(x, 8, 0)

else:

self.drawing\_canvas.move(take, 8, 0)

if event.keycode == 40:

if type(take) == list:

for x in take:

self.drawing\_canvas.move(x, 0, 8)

else:

self.drawing\_canvas.move(take, 0, 8)

except:

messagebox.showerror("Error","Nothing selected from indexing box")

#Method for opening file

def open(self,event):

self.status\_msg["text"] = "Open a File"

if self.notation\_box["state"] == DISABLED:

self.notation\_box["state"] = NORMAL

self.drawing\_canvas.unbind("<B1-Motion>")

self.drawing\_canvas.unbind("<ButtonRelease-1>")

self.drawing\_canvas.unbind("<Button-1>")

img = filedialog.askopenfilename(initialdir="Saved\_file",title="Select an Image",filetypes=(("PNG Images","\*.png"),("All Images","\*,\*")))

if img:

self.img\_container.append(ImageTk.PhotoImage(Image.open(img)))

self.img\_counter+=1

take = self.drawing\_canvas.create\_image(100,200,image=self.img\_container[self.img\_counter])

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

self.controls(1)

#Method For Save a file

def save(self,event):

self.status\_msg['text'] = "Save current file"

file = filedialog.asksaveasfilename(initialdir="Saved\_file",filetypes=[("PNG File","\*.png")])

if file:

x = self.window.winfo\_rootx() + self.drawing\_canvas.winfo\_x()+10

y = self.window.winfo\_rooty() + self.drawing\_canvas.winfo\_y()+10

x1 = x + self.drawing\_canvas.winfo\_width()-20

y1 = y + self.drawing\_canvas.winfo\_height()-20

ImageGrab.grab().crop((x,y,x1,y1)).save(file+'.png')

self.window.title("DRAWINGO : wings to your imagination -----" + file + ".png")

#Method for undo operation

def undo(self,event):

self.status\_msg["text"] = "Undo"

if self.notation\_box:

if self.notation\_box["state"] == DISABLED:

self.notation\_box["state"] = NORMAL

self.notation\_box.delete(END)

if self.undo\_container:

take = self.undo\_container.pop()

#self.redo\_container.append(take)

if type(take) == list:

for x in take:

self.drawing\_canvas.delete(x)

else:

self.drawing\_canvas.delete(take)

if len(self.undo\_container) == 0:

self.clear()

#For clear the drawing canvas

def clear(self):

self.undo\_container.clear()

self.notation\_box.delete(0,END)

self.file\_menu.entryconfig("Save",state=DISABLED)

self.edit\_menu.entryconfig("Undo",state=DISABLED)

self.edit\_menu.entryconfig("Clear",state=DISABLED)

self.edit\_menu.entryconfig("Cut",state=DISABLED)

self.edit\_menu.entryconfig("Copy",state=DISABLED)

self.edit\_menu.entryconfig("Paste",state=DISABLED)

self.edit\_menu.entryconfig("Screenshot",state=DISABLED)

self.option\_menu.entryconfig("Movement",state=DISABLED)

self.temp.clear()

self.img\_container.clear()

self.cut\_copy\_img.clear()

self.img\_counter = -1

self.counter = -1

#Cut the selected region

def cut(self,event):

self.copy(1)

self.delete\_selected\_region(False)

self.status\_msg["text"] = "Selected Region Cut Successfully"

#Copy the selected region

def copy(self,event):

try:

if event!=1:

self.drawing\_canvas.delete(self.temp.pop())

self.status\_msg["text"] = "Selected Region Copied"

else:

self.drawing\_canvas.itemconfig(self.temp[len(self.temp)-1],outline="white")

time.sleep(0.0001)

self.drawing\_canvas.update()

x1 = self.window.winfo\_rootx() + self.drawing\_canvas.winfo\_x()

y1 = self.window.winfo\_rooty() + self.drawing\_canvas.winfo\_y()

ImageGrab.grab().crop((x1 + self.old\_x,y1 + self.old\_y,x1 + self.new\_x,y1 + self.new\_y)).save("cutting.png")

self.counter += 1

self.reset()

except:

if event == 1:

messagebox.showerror("Cut Error","Select a region by selector tool under the 'Tools', then cut the selected region")

else:

messagebox.showerror("Copy Error","Select a region by selector tool under 'Tools', then copy the selected region")

#Paste in the selected region

def paste(self,event):

try:

if self.notation\_box["state"] == DISABLED:

self.notation\_box["state"] = NORMAL

self.cut\_copy\_img.append(ImageTk.PhotoImage(Image.open("cuttion.png")))

take = self.drawing\_canvas.create\_image(100,200,image=self.cut\_copy\_img[self.counter])

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.status\_msg["text"] = "Paste on the screen"

except:

messagebox.showerror("Paste Error","Paste Error")

#Method for selection region

def select\_region(self,event):

try:

self.status\_msg["text"] = "Select a particular region"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_rectangle(self.old\_x,self.old\_y,event.x,event.y)

self.temp.append(take)

def select\_region\_final(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

self.new\_x = event.x

self.new\_y = event.y

self.delete\_seg = self.drawing\_canvas.create\_rectangle(self.old\_x,self.old\_y,self.new\_x,self.new\_y)

self.temp.append(self.delete\_seg)

self.drawing\_canvas.bind("<ButtonRelease-1>",select\_region\_final)

else:

self.old\_x = event.x

self.old\_y = event.y

except:

messagebox.showerror("Error","Select region error")

#Method for deleting selected region

def delete\_selected\_region(self):

self.drawing\_canvas.itemconfig(self.delete\_seg,fill="white",width=0.00001,outline="white")

self.reset()

#Method for Screenshot

def screenshot(self,event):

try:

self.drawing\_canvas.delete(self.temp.pop())

time.sleep(0.0000001)

self.window.update()

x1 = self.window.winfo\_rootx() + self.drawing\_canvas.winfo\_x()

y1 = self.window.winfo\_rooty() + self.drawing\_canvas.winfo\_y()

file = filedialog.asksaveasfilename(initialdir="\Desktop",title="Screenshot save",filetypes=[("PNG File","\*.png")])

if file:

ImageGrab.grab().crop((x1 + self.old\_x,y1 + self.old\_y,x1 + self.new\_x,y1 + self.new\_y)).save(file + ".png")

self.reset()

self.status\_msg["text"] = "Screenshot taken and saved"

except:

print("Screenshot Error")

messagebox.showerror("Selection Error", "At first select a region by selector under 'Tools' then take screen shot")

#Set permanent color: border and background

def change\_permanent\_fill\_outline\_color(self):

self.status\_msg["text"] = "Set Permanent fill and outline color"

top = Toplevel()

top.title("Set Permanent Fill and Outline Color")

top.config(bg="linen")

top.geometry("400x200")

top.resizable(False,False)

top.wm\_iconbitmap("Icons/permanent.ico")

#take color and set

def color\_set(choice):

take\_color = colorchooser.askcolor()[1]

if choice == 1:

self.fill\_color = take\_color

self.fill\_color\_line = take\_color

self.fill\_information.set(1)

else:

self.outline\_color\_line = take\_color

self.outline\_information.set(1)

self.window.update()

fill\_check = Checkbutton(top,variable=self.fill\_information,text="Permanent fill color",font=("timesnewroman",12,"bold"),bg="snow",fg="black")

fill\_check.place(x=50,y=20)

self.choosing\_color = ImageTk.PhotoImage(Image.open("Icons/choose\_color.png").resize((10,10)))

fill\_colorchooser = Button(top,image=self.choosing\_color,bg="black",command=lambda: color\_set(1))

fill\_colorchooser.place(x=280,y=28)

outline\_check = Checkbutton(top,variable=self.outline\_information,text="Permanent outline color",font=("timesnewroman",12,"bold"),bg="snow",fg="black")

outline\_check.place(x=50,y=80)

outline\_colorchooser = Button(top,image=self.choosing\_color,bg="black",command=lambda: color\_set(2))

outline\_colorchooser.place(x=320,y=90)

def message():

messagebox.showinfo("Information","For any kind of line online Fill is available")

information = Button(top,text="Information",font=("timesnewroman",14,"bold"),bg="black",fg="white",command=message,relief=RAISED,bd=8)

information.place(x=50,y=140)

ok = Button(top,text="OK",font=("timesnewroman",14,"bold"),bg="black",fg="white",command=top.destroy,relief=RAISED,bd=8)

ok.place(x=280,y=140)

#Method for zoom controller

def zoom\_controller(self,event):

self.status\_msg["text"] = "Zoom Controller"

try:

if event.delta > 0:

self.drawing\_canvas.scale("all",event.x,event.y,1.1,1.1)

elif event.delta < 0:

self.drawing\_canvas.scale("all",event.x,event.y,0.9,0.9)

except:

if event == 1:

self.drawing\_canvas.scale("all",550,350,1.1,1.1)

else:

self.drawing\_canvas.scale("all",550,350,0.9,0.9)

#Color box with maintain by keyboard or mouse event

def color\_box\_width\_controller(self,event):

try:

print(event)

if event.delta > 0:

self.color\_circle\_width\_maintainer += 3

else:

self.color\_circle\_width\_maintainer -= 3

except:

if event == 1:

self.color\_circle\_width\_maintainer += 3

else:

self.color\_circle\_width\_maintainer -= 3

#Method for Reset

def reset(self):

self.status\_msg['text'] = "Draw with Fun"

if self.notation\_box:

self.file\_menu.entryconfig("Save", state=NORMAL)

self.edit\_menu.entryconfig("Undo",state=NORMAL)

self.edit\_menu.entryconfig("Clear", state=NORMAL)

self.edit\_menu.entryconfig("Cut", state=NORMAL)

self.edit\_menu.entryconfig("Copy", state=NORMAL)

self.edit\_menu.entryconfig("Paste", state=NORMAL)

self.edit\_menu.entryconfig("Screenshot", state=NORMAL)

self.option\_menu.entryconfig("Movement", state=NORMAL)

if self.notation\_box['state'] == DISABLED:

self.notation\_box['state'] = NORMAL

self.new\_x = None

self.new\_y = None

self.old\_x = None

self.old\_y = None

self.temp= []

def about(self):

self.top = Toplevel()

self.top.geometry("1000x500")

self.top.wm\_iconbitmap("Icons/about.ico")

self.top.title("About")

self.top.resizable(False,False)

self.top.config(bg="linen")

img = PhotoImage(file="Icons\DraWingo.png")

Label(self.top,image=img).grid(row=0,column=0)

about\_frame = Frame(self.top,height=450,width=470,relief=RAISED,highlightbackground="silver",highlightthickness=5,bg="snow")

about\_frame.grid(row=0,column=1,padx=10,pady=5)

Label(about\_frame,text="DRAWINGO",bg="white",fg="black",font=("timesnewroman",24,"bold")).place(x=150,y=10)

Text = "This is the Major Project on 'Drawing Application' created by\nDinesh Singh(me) of BCA-6 from University of Allahabad."

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=80)

Text = "This project is created using Python as a programming\nlanguage and provides the following features:- \t"

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=160)

Text = "1. Drawing tools like pencil, eraser, color box, text box etc."

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=240)

Text = "2. Width Controller for adjust the size of pencil,shapes\n& eraser\t\t\t\t\t"

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=270)

Text = "3. Index box for object manipulation."

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=320)

Text = "4. Provides various shapes and color box."

Label(about\_frame,text=Text,font=("timesnewroman",12,"italic"),bg="snow",fg="black").place(x=10,y=350)

self.top.mainloop()

def tips(self):

self.top = Toplevel()

self.top.geometry("1200x700")

self.top.resizable(False,False)

self.top.title("Tips")

self.top.iconbitmap(r"Icons\tips.ico")

self.top.config(bg="linen")

tips\_frame = Frame(self.top,height=675,width=1175,relief=RAISED,highlightbackground="silver",highlightthickness=5,bg="snow")

tips\_frame.grid(row=0,column=0,padx=10,pady=10)

Label(tips\_frame,text="IMPORTANT TIPS",font=("timesnewroman",24,"bold","underline"),bg="snow",fg="black").place(x=450,y=20)

#Tips for straight line

Label(tips\_frame,text="Shift +",font=("timesnewroman",12,),bg="snow",fg="black").place(x=100,y=100)

line\_img = ImageTk.PhotoImage(Image.open("Icons/line.png").resize((20,20)))

Label(tips\_frame,image=line\_img).place(x=200,y=100)

Label(tips\_frame,text="=> Used to make Straight Line: Horizontal and Vertical",font=("timesnewroman",12,),bg="snow",fg="black").place(x=250,y=100)

#Tips for modes

#tips for fill

fill\_img = ImageTk.PhotoImage(Image.open("Icons/fill.png").resize((20,20)))

Label(tips\_frame,image=fill\_img).place(x=200,y=150)

Label(tips\_frame,text="=> Used to change background color of the Selected Shape or Line",font=("timesnewroman",12,),bg="snow",fg="black").place(x=250,y=150)

#tips for outline

outline\_img = ImageTk.PhotoImage(Image.open("Icons/outline.png").resize((20,20)))

Label(tips\_frame,image=outline\_img).place(x=200,y=200)

Label(tips\_frame,text="=> Used to change outline color of the Selected Shape",font=("timesnewroman",12,),bg="snow",fg="black").place(x=250,y=200)

#tips for permanent color

permanent\_img = ImageTk.PhotoImage(Image.open("Icons/permanent.png").resize((20,20)))

Label(tips\_frame,image=permanent\_img).place(x=200,y=250)

Label(tips\_frame,text="=> Used to change background and outline color permanently of the upcoming shapes and lines",font=("timesnewroman",12,),bg="snow",fg="black").place(x=250,y=250)

#tips for movement of drawn element

Label(tips\_frame,text="Shift + Arrow Keys => Used for move selected drawing element Left, Right, Up and Down",font=("timesnewroman",12,),bg="snow",fg="black").place(x=100,y=350)

#tips for zoom in/ zoom out

Label(tips\_frame,text="Ctrl + Scroll => Used for zoom in and zoom out",font=("timesnewroman",12,),bg="snow",fg="black").place(x=100,y=400)

#tips for inc/dec the size of color pen

Label(tips\_frame,text="Shift + Scroll => Used for increasing and decreasing the size of color pen",font=("timesnewroman",12,),bg="snow",fg="black").place(x=100,y=450)

#More tips video link

# Label(tips\_frame,text="For Complete project demonstration, click on the below link:- ",font=("timesnewroman",12,),bg="snow",fg="black").place(x=100,y=550)

self.top.mainloop()

#Colorbox under "Tolls" for pen color

def color\_boxer(self,event):

self.status\_msg["text"] = "Draw with the color pen"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,fill=self.color\_container\_box,width=self.color\_circle\_width\_maintainer,smooth=True,capstyle=ROUND)

self.temp.append(take)

self.old\_x = event.x

self.old\_y = event.y

def color\_input(event):

self.undo\_container.append(self.temp)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

self.drawing\_canvas.bind("<ButtonRelease-1>",color\_input)

#Method to draw using pencil

def draw\_with\_pencil(self,event):

self.status\_msg["text"] = "Draw With Pencil"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,fill=self.fill\_color\_line,width=self.width\_maintainer,smooth=True,capstyle=ROUND)

self.temp.append(take)

self.old\_x = event.x

self.old\_y = event.y

def push\_value(event):

self.undo\_container.append(self.temp)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

self.drawing\_canvas.bind("<ButtonRelease-1>",push\_value)

#Method eraser

def eraser(self,event):

self.status\_msg["text"] = "Erasing"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_rectangle(self.old\_x,self.old\_y,event.x,event.y,width=self.erase\_width\_maintainer,fill="white",outline="white")

self.temp.append(take)

self.old\_x = event.x

self.old\_y = event.y

def real\_erasing(event):

self.undo\_container.append(self.temp)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

self.drawing\_canvas.bind("<ButtonRelease-1>",real\_erasing)

def text\_creation\_input\_take(self):

def message\_show():

messagebox.showinfo("Done","Click on targeting position on the main window to input text")

self.status\_msg["text"] = "Make your own Text"

self.top = Toplevel()

self.top.title("Text Here")

self.top.geometry("400x500")

self.top.wm\_iconbitmap("Icons/text\_box.ico")

self.top.config(bg="linen")

self.top.resizable(False,False)

label\_1 = Label(self.top,text="Enter the text",font=("timesnewroman",25,"bold"),fg="Black",bg="snow")

label\_1.pack(pady=20)

entry\_take = Entry(self.top,width=20,font=("timesnewroman",20,"bold","italic"),bg="snow",fg="black",textvariable=self.input\_take,relief = SUNKEN,bd = 10)

entry\_take.pack(pady=10)

entry\_take.focus()

ok\_btn = Button(self.top,text = "OK", fg = "snow", bg = "black",width=10,font=("timesnewroan",15,"bold"),relief=RAISED,bd=5,command=message\_show)

ok\_btn.pack(pady=20)

self.text\_collection = Listbox(self.top,width=17,height=9,font=("timesnewroman",13,"bold"),bg="snow",fg="black",relief=SUNKEN,bd=8)

self.text\_collection.place(x=10,y=280)

text\_list = ["Times New Roman","Arial","Courier New","Comic Sans MS","Fixedsys","MS Sans Serif","System","Verdana","Symbol"]

for x in text\_list:

self.text\_collection.insert(END,x)

self.text\_collection.activate(0)

self.text\_collection.selection\_set(0)

#For text color set

def color\_choose():

self.text\_fg = colorchooser.askcolor()[1]

color\_chooser = Button(self.top,text = "Text Color",fg="white",bg="black",font=("timesnewroman",15,"bold"),relief=RAISED,bd=5,command=color\_choose)

color\_chooser.place(x=200,y=280)

self.font\_size = Scale(self.top,from\_=1,to=100,orient=HORIZONTAL,bg="black",fg="white",font=("timesnewroman",10,"bold"),activebackground="grey")

self.font\_size.place(x=200,y=433)

#For make text on the screen by click

def text\_creation(event):

take = self.drawing\_canvas.create\_text(event.x, event.y, text=self.input\_take.get(), font=(self.text\_collection.get(ACTIVE), self.font\_size.get(), "bold", "italic"), fill=self.text\_fg)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.input\_take.set(" ")

self.top.destroy()

self.drawing\_canvas.bind("<Button-1>", text\_creation)

#Method for color activation

def activate\_coloring(self,notation):

if notation == 1:

self.active\_coloring = 1

else:

self.active\_coloring = 2

#Color set finally in the widget

def check(self,number,index):

try:

color\_list = ["red","#8B4513","blue","grey","yellow","green","orange","black","white","pink","sky blue","violet","#90EE90","#808000","#800000","#FFE6E8","#C0C0C0","#FBB917","#FFFFCC","#FFE5B4"]

take = self.undo\_container[self.notation\_box.get(ACTIVE)]

if number == 20:

take\_color = colorchooser.askcolor()[1]

else:

take\_color = color\_list[number]

if index == 1:

if type(take) == list:

for x in take:

self.drawing\_canvas.itemconfig(x,fill=take\_color)

else:

self.drawing\_canvas.itemconfig(self.undo\_container[self.notation\_box.get(ACTIVE)],fill=take\_color)

else:

self.drawing\_canvas.itemconfig(self.undo\_container[self.notation\_box.get(ACTIVE)],outline=take\_color)

self.status\_msg["text"] = "Colored"

except:

if len(self.undo\_container) == 0:

messagebox.showwarning("Nothing Present","Sorry, Nothing present to change color")

else:

messagebox.showwarning("Problem Raise",f"For any kind of line only Fill is allowed")

#Method for draw bent line

def draw\_bent\_line(self,event):

self.status\_msg["text"] = "Draw bent line"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color\_line)

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def bent\_line\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color\_line,capstyle=ROUND)

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","Click only not motion")

self.drawing\_canvas.bind("<ButtonRelease-1>",bent\_line\_draw)

#Method for drawing straight line

def draw\_straight\_line(self,event):

self.status\_msg["text"] = "Draw Straight line"

if self.old\_x and self.old\_y:

if event.x - self.old\_x and event.y - self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x, self.old\_y, event.x, self.old\_y, width=self.width\_maintainer,fill=self.fill\_color\_line)

self.temp.append(take)

else:

take = self.drawing\_canvas.create\_line(self.old\_x, self.old\_y, self.old\_x, event.y, width=self.width\_maintainer,fill=self.fill\_color\_line)

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def straight\_line\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

if event.x - self.old\_x > event.y - self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x, self.old\_y, event.x, self.old\_y,width=self.width\_maintainer, fill=self.fill\_color\_line)

else:

take = self.drawing\_canvas.create\_line(self.old\_x, self.old\_y, self.old\_x, event.y,width=self.width\_maintainer, fill=self.fill\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","Click only not motion")

self.drawing\_canvas.bind("<Shift-ButtonRelease-1>",straight\_line\_draw)

#Method for drawing dashed line

def draw\_dashed\_line(self,event):

self.status\_msg["text"] = "Draw Dash line"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color\_line,dash=(10,1))

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def dashed\_line\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

take = self.drawing\_canvas.create\_line(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color\_line,capstyle=ROUND,dash=(10,1))

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","Click only not motion")

self.drawing\_canvas.bind("<ButtonRelease-1>",dashed\_line\_draw)

#Method for drawing triangle

def draw\_triangle(self,event):

self.status\_msg["text"] = "Draw Triangle"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_polygon(self.old\_x, self.old\_y, self.old\_x-(event.x-self.old\_x), event.y, event.x, event.y,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def triangle\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

take = self.drawing\_canvas.create\_polygon(self.old\_x, self.old\_y, self.old\_x - (event.x - self.old\_x), event.y, event.x,event.y, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","click only not motion")

self.drawing\_canvas.bind("<ButtonRelease-1>",triangle\_draw)

#Method for drawing parallelogram

def draw\_parallelogram(self,event):

self.status\_msg["text"] = "Draw a Parallelogram"

if self.old\_x and self.old\_y:

points = [self.old\_x,self.old\_y,int(self.old\_x)+30,event.y,event.x,event.y,int(event.x)-30,self.old\_y]

take = self.drawing\_canvas.create\_polygon(points,width=1, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def parallelogram\_draw(e):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x, self.old\_y, int(self.old\_x) + 30, event.y, event.x, event.y, int(event.x) - 30, self.old\_y]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error", "click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>', parallelogram\_draw)

#Method for drawing circle

def draw\_circle(self,event):

self.status\_msg["text"] = "Draw a Circle"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_oval(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,outline=self.outline\_color\_line,fill=self.fill\_color)

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def circle\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

take = self.drawing\_canvas.create\_oval(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","click only not motion")

self.drawing\_canvas.bind("<ButtonRelease-1>",circle\_draw)

#Method to draw rectangle

def draw\_rectangle(self,event):

self.status\_msg["text"] = "Draw a Rectangle"

if self.old\_x and self.old\_y:

take = self.drawing\_canvas.create\_rectangle(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x = event.x

self.old\_y = event.y

def rectangle\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

take = self.drawing\_canvas.create\_rectangle(self.old\_x,self.old\_y,event.x,event.y,width=self.width\_maintainer,fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END,len(self.undo\_container)-1)

self.reset()

except:

messagebox.showerror("Error","Click only not motion")

self.drawing\_canvas.bind("<ButtonRelease-1>",rectangle\_draw)

#Method for drawing pentagon

def draw\_pentagon(self,event):

self.status\_msg["text"] = "Draw a Pentagon"

if self.old\_x and self.old\_y:

points = [self.old\_x, self.old\_y, int(self.old\_x), event.y, event.x, event.y, int(event.x), self.old\_y, (self.old\_x+event.x)/2,self.old\_y-20]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def pentagon\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x, self.old\_y, int(self.old\_x), event.y, event.x, event.y, int(event.x), self.old\_y,

(self.old\_x + event.x) / 2, self.old\_y - 20]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error","Click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>', pentagon\_draw)

#Method for drawing hexagon

def draw\_hexagon(self, event):

self.status\_msg['text'] = "Draw a Hexagon"

if self.old\_x and self.old\_y:

points = [self.old\_x, self.old\_y, int(self.old\_x), event.y, (int(self.old\_x)+int(event.x))/2, int(event.y)+50, event.x, event.y, int(event.x), self.old\_y, (self.old\_x+event.x)/2,self.old\_y-50]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def hexagon\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x, self.old\_y, int(self.old\_x), event.y, (int(self.old\_x) + int(event.x)) / 2, int(event.y) + 50,event.x, event.y, int(event.x), self.old\_y, (self.old\_x + event.x) / 2, self.old\_y - 50]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error", "Click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>', hexagon\_draw)

#Method for drawing arrow up down

def draw\_arrow\_up\_down(self,event):

self.status\_msg['text'] = "Draw Arrow"

if self.old\_x and self.old\_y:

points = [self.old\_x, self.old\_y, (int(self.old\_x)+int(self.old\_x+event.x)/2)/2, self.old\_y, (int(self.old\_x)+int(self.old\_x+event.x)/2)/2, int(event.y), ((int(self.old\_x+event.x)/2)+int(event.x))/2, event.y, ((int(self.old\_x+event.x)/2)+int(event.x))/2,self.old\_y, int(event.x),self.old\_y, int(self.old\_x+event.x)/2, self.old\_y+(int((self.old\_y-event.y))/2)]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def arrow\_up\_down\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x, self.old\_y, (int(self.old\_x) + int(self.old\_x + event.x) / 2) / 2, self.old\_y,(int(self.old\_x) + int(self.old\_x + event.x) / 2) / 2, int(event.y),((int(self.old\_x + event.x) / 2) + int(event.x)) / 2, event.y, ((int(self.old\_x + event.x) / 2) + int(event.x)) / 2,self.old\_y, int(event.x), self.old\_y, int(self.old\_x + event.x) / 2,self.old\_y + (int((self.old\_y - event.y)) / 2)]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error","click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>',arrow\_up\_down\_draw)

#Method for drawing left right arrow

def draw\_arrow\_left\_right(self, event):

self.status\_msg['text'] = "Draw Arrow"

if self.old\_x and self.old\_y:

m = (self.old\_x + event.x)/2

points = [self.old\_x, self.old\_y, int(m), self.old\_y+20, int(m), self.old\_y+10, event.x, int(self.old\_y)+10, event.x, int(self.old\_y)-10, int(m), int(self.old\_y)-10, int(m), int(self.old\_y)-20]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def arrow\_left\_right\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x, self.old\_y, int(m), self.old\_y+20, int(m), self.old\_y+10, event.x, int(self.old\_y)+10, event.x, int(self.old\_y)-10, int(m), int(self.old\_y)-10, int(m), int(self.old\_y)-20]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error","click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>',arrow\_left\_right\_draw)

#Method for drawing rounded rectangle

def draw\_rounded\_rectangle(self,event):

self.status\_msg['text'] = "Draw Rounded Rectangle"

if self.old\_x and self.old\_y:

points = [self.old\_x,self.old\_y, int(self.old\_x)+3,int(self.old\_y)-5, int(self.old\_x)+7,int(self.old\_y)-7, int(self.old\_x)+11,int(self.old\_y)-9, int(self.old\_x)+13,int(self.old\_y)-9, event.x,int(self.old\_y)-9, int(event.x)+5,int(self.old\_y)-7, int(event.x)+8,int(self.old\_y)-5, int(event.x)+11,self.old\_y, int(event.x)+11,event.y, int(event.x)+8,int(event.y)+5, int(event.x)+5,int(event.y)+7, event.x,int(event.y)+8, int(self.old\_x)+13,int(event.y)+8, int(self.old\_x)+11,int(event.y)+7, int(self.old\_x)+7,int(event.y)+5, int(self.old\_x)+3,int(event.y)+3, int(self.old\_x),int(event.y)-2]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def rounded\_rectangle\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x,self.old\_y, int(self.old\_x)+3,int(self.old\_y)-5, int(self.old\_x)+7,int(self.old\_y)-7, int(self.old\_x)+11,int(self.old\_y)-9, int(self.old\_x)+13,int(self.old\_y)-9, event.x,int(self.old\_y)-9, int(event.x)+5,int(self.old\_y)-7, int(event.x)+8,int(self.old\_y)-5, int(event.x)+11,self.old\_y, int(event.x)+11,event.y, int(event.x)+8,int(event.y)+5, int(event.x)+5,int(event.y)+7, event.x,int(event.y)+8, int(self.old\_x)+13,int(event.y)+8, int(self.old\_x)+11,int(event.y)+7, int(self.old\_x)+7,int(event.y)+5, int(self.old\_x)+3,int(event.y)+3, int(self.old\_x),int(event.y)-2]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

messagebox.showerror("Error: click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>',rounded\_rectangle\_draw)

#Method for drawing right angled triangle

def draw\_right\_angled\_triangle(self,event):

self.status\_msg['text'] = "Draw Right Angled Traingle"

if self.old\_x and self.old\_y:

points = [self.old\_x,self.old\_y, self.old\_x,event.y, event.x,event.y]

take = self.drawing\_canvas.create\_polygon(points,width=self.width\_maintainer, fill=self.fill\_color,outline=self.outline\_color\_line)

self.temp.append(take)

else:

self.old\_x=event.x

self.old\_y=event.y

def right\_angled\_traingle\_draw(event):

for x in self.temp:

self.drawing\_canvas.delete(x)

try:

points = [self.old\_x,self.old\_y, self.old\_x,event.y, event.x,event.y]

take = self.drawing\_canvas.create\_polygon(points, width=self.width\_maintainer, fill=self.fill\_color,

outline=self.outline\_color\_line)

self.undo\_container.append(take)

self.notation\_box.insert(END, len(self.undo\_container) - 1)

self.reset()

except:

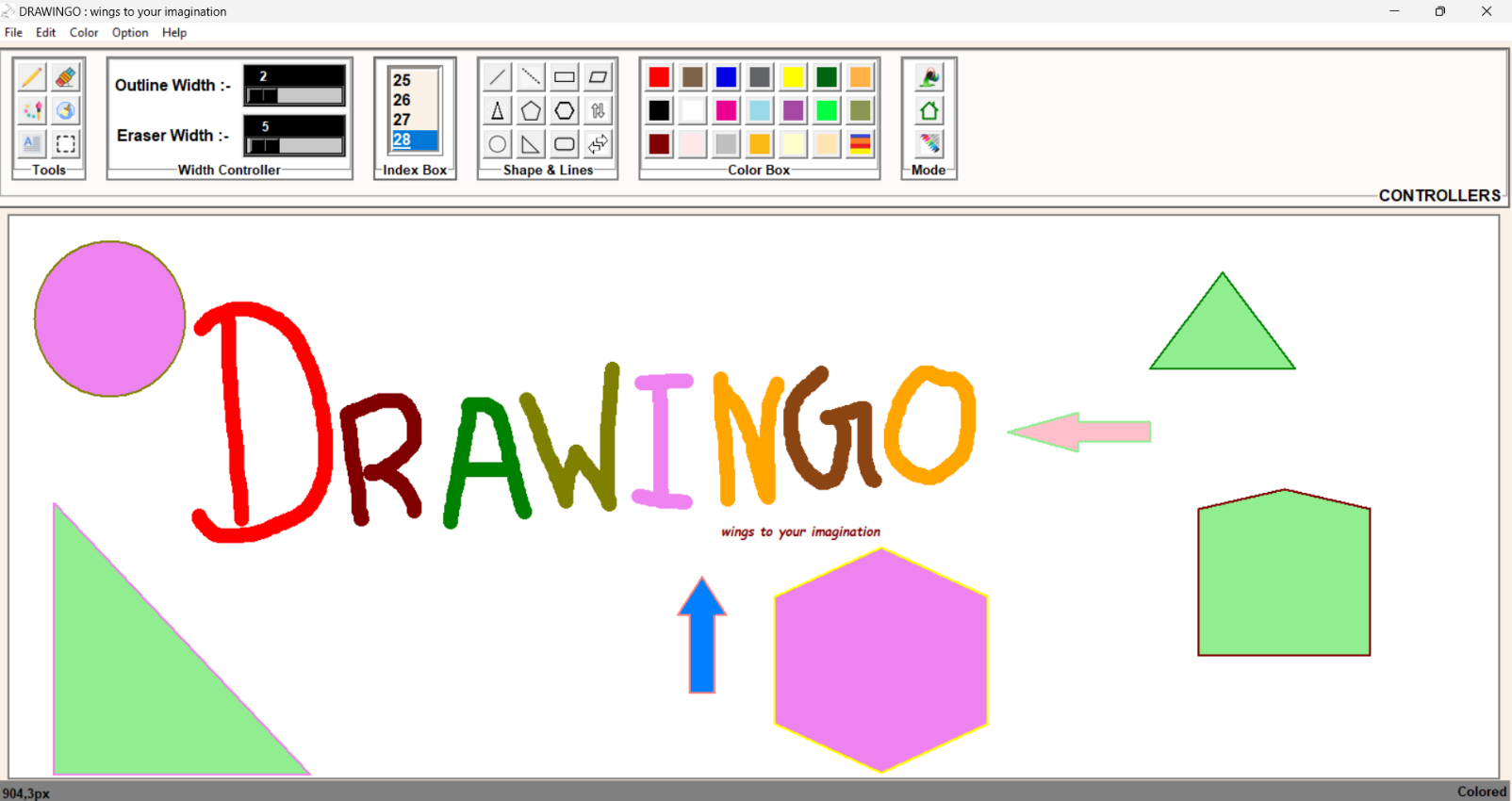
messagebox.showerror("Error","click only not motion")

self.drawing\_canvas.bind('<ButtonRelease-1>', right\_angled\_traingle\_draw)

#calling class constructor

Draw()

**SNAPSHOTS**

****

**APPLICATIONS**

**Diversity:**

This project not only intended to simple drawings, but also allows more complex drawings as it has variety of drawing tools.

**Small Scale Use:**

This application can be used in small institutes like schools for students to explore their creativity.

**Large Scale Use:**

This application can be used at high level for drawing complex design.

**Commercial Use:**

The software can be used as a source of income, because it can be useful for both learners as well as expert. As it is open source, hence can be enhanced with new features so drawing becomes easier and more exciting.

**Future Scope**

The future scope of this drawing application is quite broad and can involve many different areas of development and innovation. Here are some potential areas of future growth and expansion for a drawing application.

1. Integration with emerging technologies: As new technologies continue to emerge, there is a great potential for drawing application to integrate with them. For example, virtual reality (VR) and augmented reality (AR) technologies could be used to create immersive drawing experiences. Additionally, voice and gesture recognition technology could be used to control the application hands-free.
2. Collaborative features: Collaboration is becoming an increasingly important aspect of many creative industries, and drawing applications could include features that allow user to collaborate with others in real-time. This could include features like live editing, chat functionality, and shared project files.
3. Artificial Intelligence (AI): AI could be used to enhance the drawinging experience in a variety of ways. For example, AI could be used to suggest color palettes, provide intelligent brush settings, or event automatically generate background images.
4. Improved user experience: As user experience (UX) continues to be a key focus for digital products, this drawing application could focus on improving the usability and intuitiveness of their interfaces. This could involve redesigning the UI, implementing more intuitive controls, or using advanced UX techniques like gamification.
5. Mobile optimization: As more users move towards mobile devices for their creative work, drawing applications will need to adapt and optimize for mobile platforms. This could involve developing a separate mobile application or optimizing the existing application for use on mobile devices.

**CONCLUSION**

This Drawing Application project aims to provide a user-friendly and efficient drawing application that can be used for a range of purposes. The application will be designed using modern tools and technologies and will incorporate a range of features that enable users to create and edit digital images. The project will be developed using tkinter as a primary module of Python to ensure that the final product meets the user’s requirements.

**MILESTONE**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Project Activity** | **Estimated Start Date** | **Estimated End Date** |
| **1.** | Synopsis Completion | 17/02/2023 | 24/02/2023 |
| **2.** | Synopsis Submission | 27/02/2023 | 27/02/2023 |
| **3.** | Project Completion | 18/02/2023 | 02/04/2023 |
| **4.** | Report Completion | 10/04/2023 | 18/04/2023 |
| **5.** | Pre-Presentation | 20/04/2023 | 20/04/2023 |
| **6.** | Final Presentation |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**MEETING WITH THE SUPERVISOR**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date of**  **the meet** | **Mode** | **Comments by the Supervisor** | **Signature of the**  **Supervisor** |
| 10/01/2023 | offline | Briefly Describe about  the topic |  |
| 25/01/2023 | offline | Project topic  confirmation |  |
| 24/02/2023 | offline | Project Synopsis Demonstration and Sign |  |
| 04/04/2023 | offline | First Demonstration of working project |  |
| 19/04/2023 | offline | Project Report Demonstration and sign |  |
|  | offline | Final Project Report Demonstration and sign |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**REFERENCES**

* https://en.wikipedia.org/wiki/Microsoft\_Drawing
* https://chat.openai.com/
* https://images.google.com/
* <https://docs.python.org/3/>