HANDICRAFT

A digital art developing program created using Python.

TABLE OF CONTENTS

resources can be found on the back of the binder

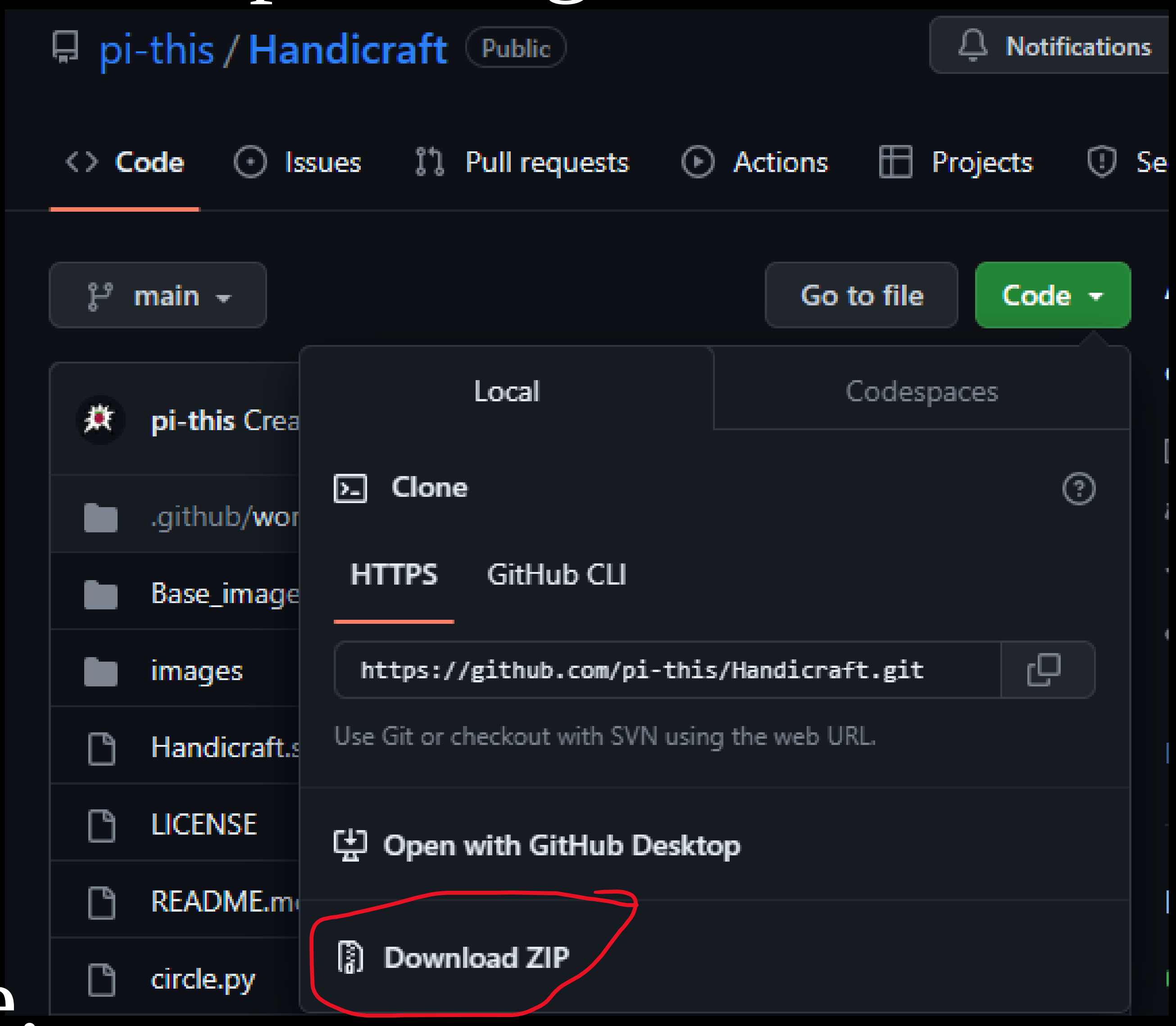
HOW TO USE HANDICRAFT

Handicraft is a versatile tool designed for digital art development and creative exploration. It offers a unique design process and user experience that encourages creativity. Although it may not be as extensive as other digital art creator software, it still serves its purpose effectively.

Installation instructions:

I had a blast experimenting with Handicraft after creating it, and it brought me so much joy that I felt compelled to share it with the world. I uploaded the software for the art development application on GitHub, and you can easily install and enjoy it by following the instructions below:

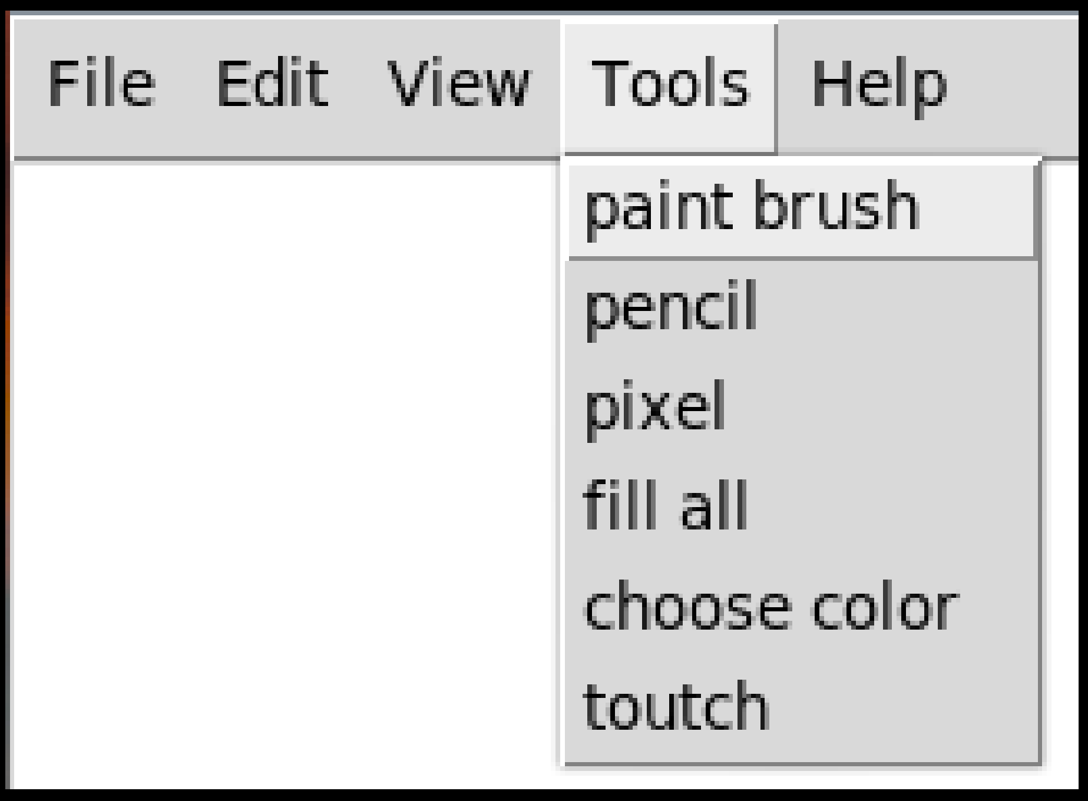
1. To begin, open your web browser and input the given link: https://github.com/pi-this/Handicraft
2. Afterwards, proceed to click on the green button labeled 'code' and select 'download ZIP'.
3. After successfully installing the file, locate and click on the 'main.py' file to launch The Handicraft software.



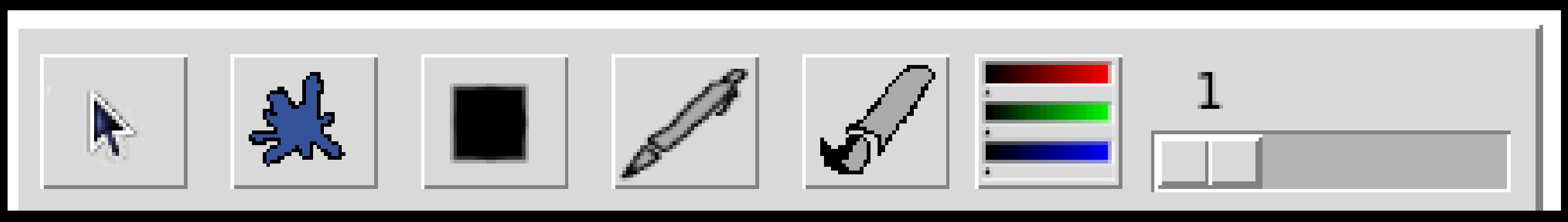
If It Doesn't Work…

If you are experiencing issues with running the code, it's possible that not all necessary components have been installed. Please ensure that Python is properly installed on your device and also check if the following Python modules have been installed: web browser, tkinter, time, random, and PIL.

Tools:

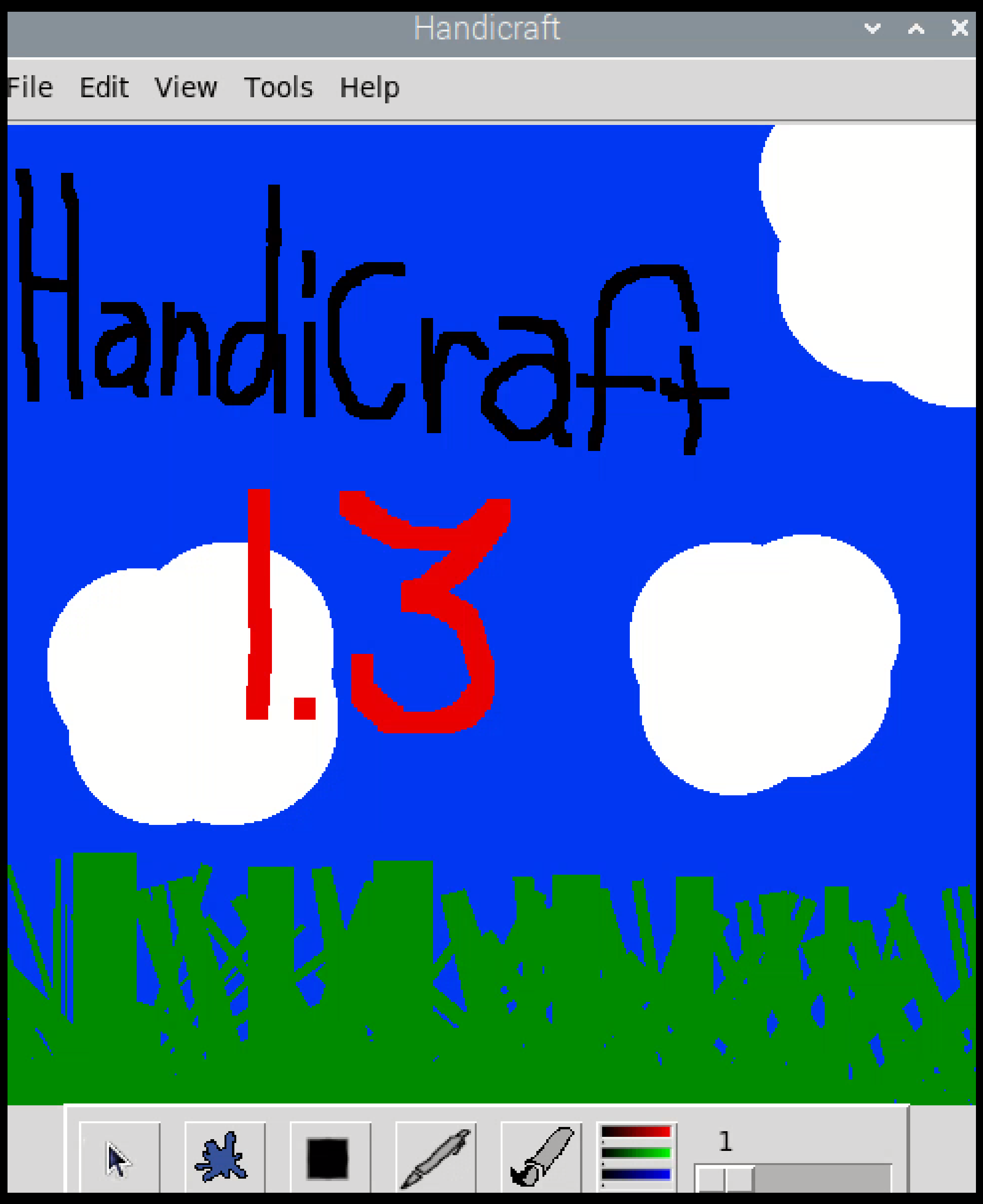


To find the tools, simply look at the bottom toolbar on your screen or the top navigation menu under 'tools.' Now, let's get into the details of each tool. First up, the paintbrush. Keep in mind that each drawing tool has a unique shape, so each click will produce a different result. In the case of the paintbrush, it creates a circular shape.



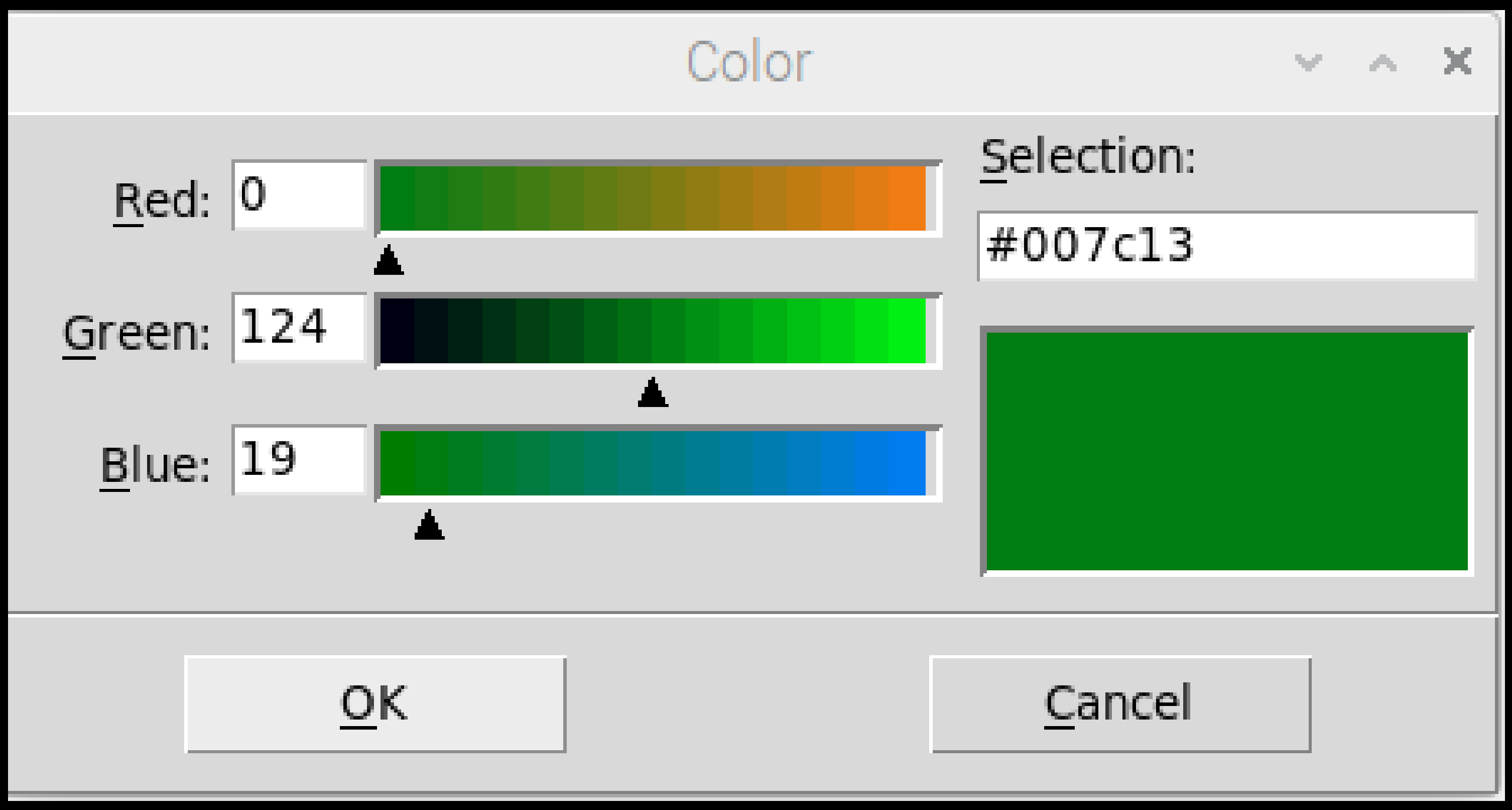
Let's talk about the pencil. Just like any other tool, it has a specific shape - a line. This shape comes in handy when drawing things such as grass, hair, or other spiked objects.

Explore and experiment with each tool to gain knowledge through hands-on experience.



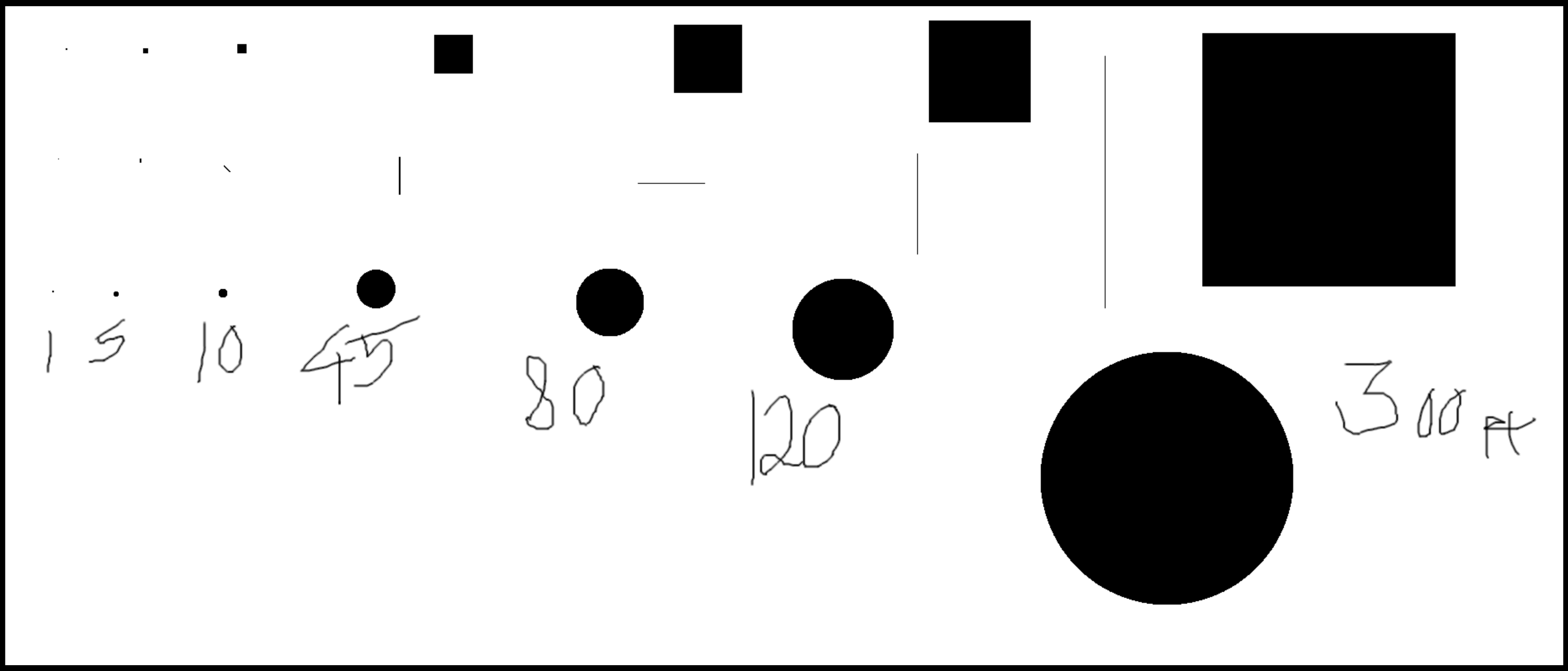
Can you identify the tool used to create the grass in the photo above? The grass serves as a perfect illustration of the pencil tool.

Let's talk about the third tool in our toolbox - the pixel tool. With this tool, you can create squares exactly where you click your mouse. Its strength is in producing bold, prominent lines. Each tool we've discussed so far has been designed for a specific purpose - the paintbrush for fluffy clouds and circular foliage, the pencil tool for spiky objects like grass and hair, and the pixel tool for creating thick, unbroken lines for words and other objects. But what if I want to fill the entire background with a specific color, like blue for the sky? No problem. The fill-all tool can fill the background with any color you select.



Red: 0
Green: 124
Blue: 19
Selection: #007c13

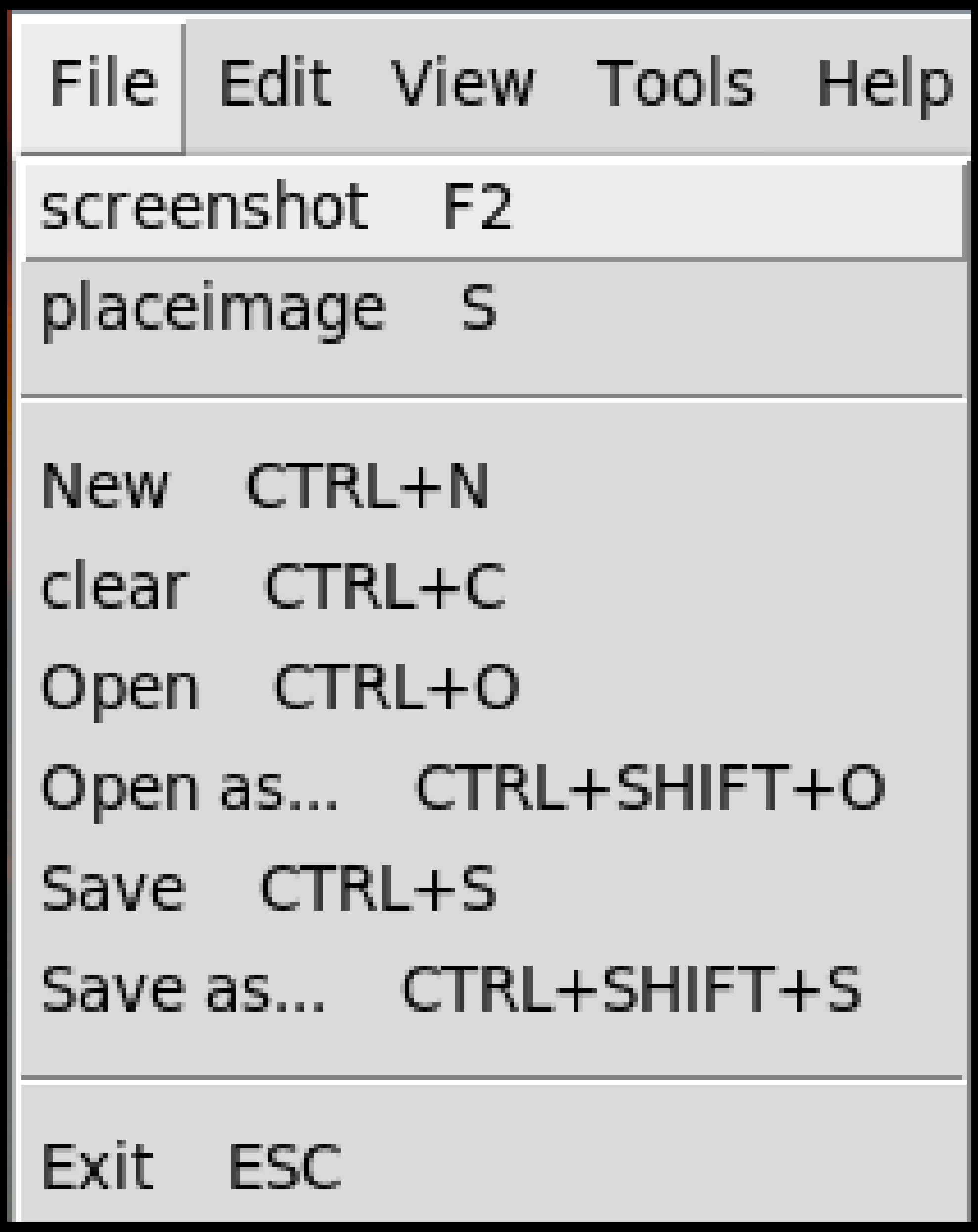
Creating computer art using only black and white with Handicraft can be uninteresting and lackluster. That's why having the ability to select your own colors can enhance the experience. If you're particular about the color of your artwork, simply adjust the triangles to set the red, green, and blue values or input your desired color in the selection box above the color display. Adding color to your art breathes life into it. However, it would be limited if only one size option for drawing existed. Fortunately, Handicraft gives you the freedom to choose from sizes ranging from 1-300px. Check out the image below to see the sizes of 1, 5, 10, 45, 80, 120, and 300px.



Out of all the tools mentioned, the touch tool may not be the most essential in this software, but it still serves a purpose. If you simply want to showcase your artwork to friends without having to draw anything, the touch tool is the one to use. It allows you to easily click on the canvas and display your work.

The File Tab:

In order to avoid losing your hard work, it's important to know how to save and open files. The next topic we'll cover is the File tab, which includes these features.



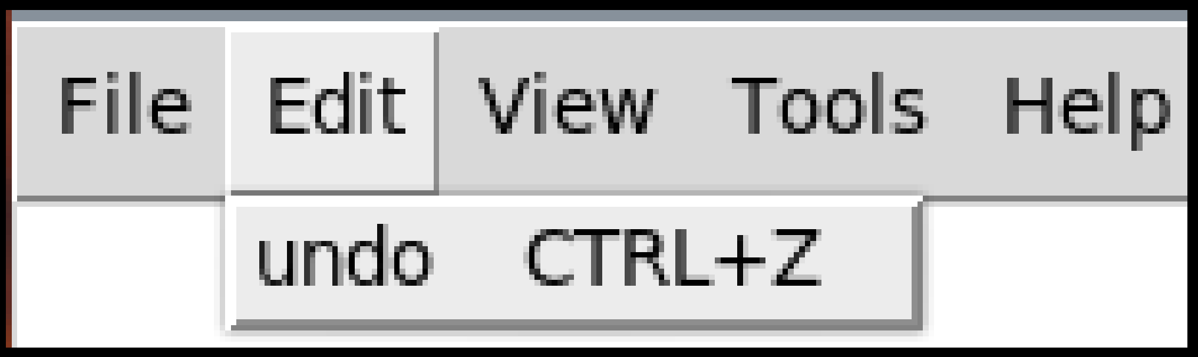
To access the options under the "File" category, simply look towards the right side of your screen where a photo is displayed. You'll find a variety of choices such as taking screenshots, creating new files, clearing artwork, opening files, renaming files, saving files, saving files under a new name, and exiting the program. Click on the corresponding button or use the keyboard shortcut to select the option of your choice.

Let's start by discussing Handicraft's screenshot feature located on the top button of the "File" drop-down menu. This option allows you to capture a screenshot of your entire computer screen, which can be useful in case any errors occur while saving your artwork. However, this issue will be resolved in future versions of Handicraft. Additionally, the "placeimage" option shown in the image can be accessed by using the shortcut 'S.' This feature allows you to place an image on your canvas, just like adding a sticker. However, it needs improvement since it only places the image in the center of the canvas and cannot be moved. Nonetheless, this is something that can be addressed in future versions. The other options in the drop-down menu are probably familiar to you, as they are self-explanatory. The "new" option will open a new file and clear all data. The "clear" option only clears all canvas data, while the "open" option allows you to open an image file of any kind. If the same image is opened again, it will erase your art data and open the same image. To open an entirely different image, use the "open as" option. "Save" and "save as" give you the same options, allowing you to save the file with a different name and in a different place. Lastly, at the bottom of the drop-down menu is the exit button. This button closes the art program and Python as well, stopping the entire software and not saving any of your work unless saved beforehand.

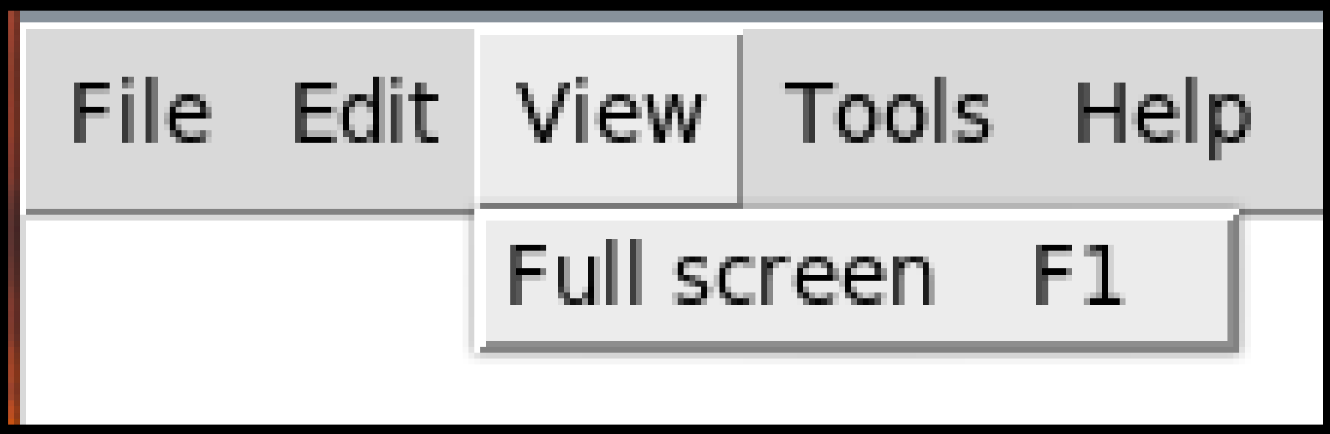
Warning!

Remember to save your work as Handicraft does not have an automatic save feature.

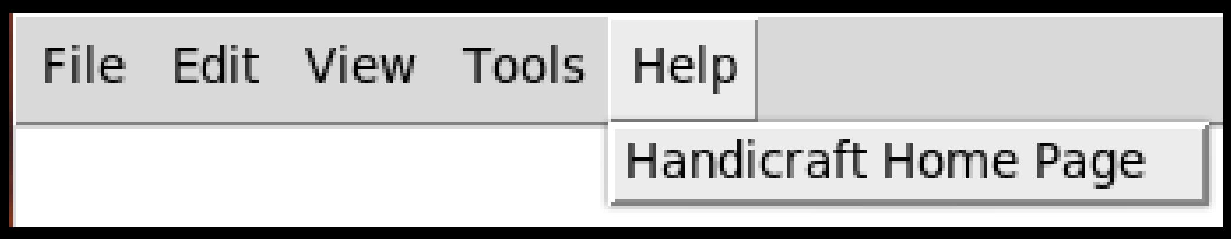
Other Tabs:

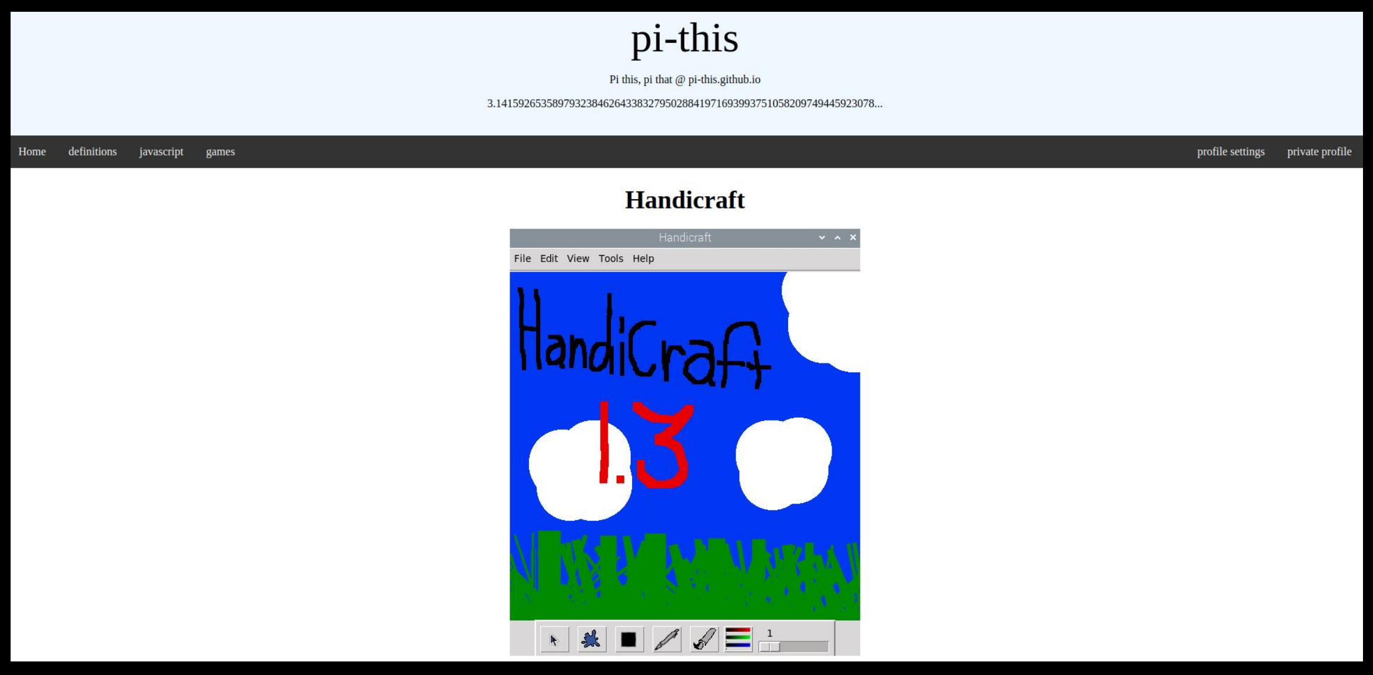


The option to undo is shown in the picture above.



The picture above shows the View tab's full-screen button.



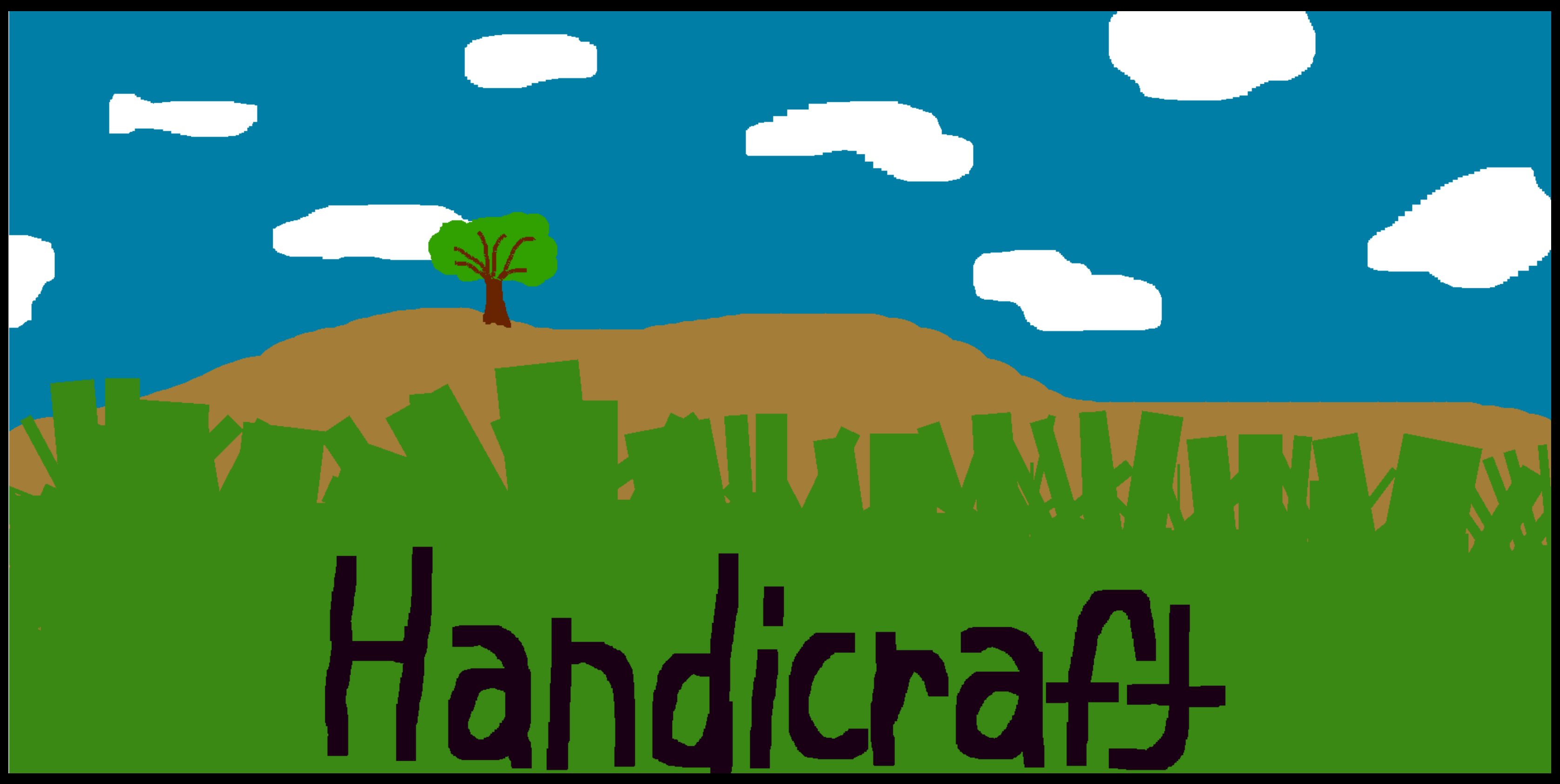


We discussed the "File" and "Tools" tabs, but there are three other tabs that are not as busy yet still offer useful features. The "Edit" tab has an undo feature, which can also be accessed using the CTRL+Z shortcut. Unfortunately, there is no redo option in this version, but it will be included in a future release. The View tab is located to the right of the Edit tab and contains all the options that let you modify how the software appears. Currently, there is only one option in this version - the full-screen option. This option provides more room to create bigger and better projects. Handicraft has several useful features to explore, with many more on the way. However, if you are ever unsure about a button, shortcut, or error, this art editor software has a dedicated website on GitHub, called pi-this. Click the "Handicraft Home Page" button under the Help tab to access information about Handicraft.

Shortcuts:

I briefly discussed the availability of shortcuts, which are visible in the images. But what advantages do shortcuts offer? They provide users with more flexibility in utilizing the software, and they make it more efficient to create and discover. Every new option added comes with a corresponding shortcut, so it's recommended to commit the shortcuts to memory for easy access and use.

The Overlapping Concept:



While this software provides many features, there are some options that would be useful but are currently unavailable. Due to the absence of certain tools, Handicraft users must find ways to work around these challenges.

One challenge that users of this software may encounter is the inability of the fill tool to fill specific parts of the screen. Although it can fill the entire screen in this version, it cannot fill specific objects. However, there is a workaround for this issue. Users of the art software can employ a technique called the overlapping concept, which is also used in traditional painting and is not too difficult to learn. In the image shown above, I utilized the overlapping concept by first filling the sky, followed by the clouds, sand, tree, grass, and finally, the word Handicraft. When applying this technique, it is recommended to begin with the bottom layers and work your way up to the top layers.

Software Versions:

Greetings! Earlier, I discussed how different versions of Handicraft may vary in features. However, I have yet to provide information on the past and present versions of this software. Currently, Handicraft is running on version 1.3, while the previous version was 1.0. To compare the two versions, images can be found on the right corners for your convenience. The latest update of this digital art software offers numerous enhancements, such as expanded color options, customizable size, perspective tools, and various options under the 'File' tab.

The version above is 1.3 while the one below is 1.0.

Did You Know?

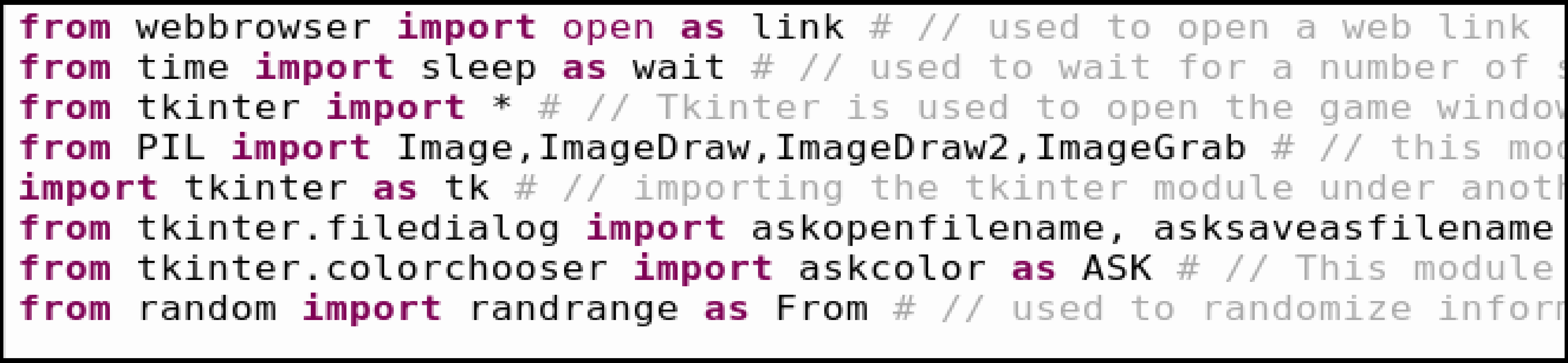
To download Handicraft 1.0, simply visit the same location where version 1.3 was installed - on GitHub.

HOW HANDICRAFT WORKS

Let's take a look at some key sections of the code and their descriptions. It's important to note that we won't be covering the entire software's code, but rather focusing on Handicraft's inner workings. The commented code can be found in the last section of the binder. Our focus will be on understanding the crucial aspects of how the code was created.

Modules:

The digital art development software was crafted using Python 3. Although, if a user intends to develop a more intricate Python program beyond a basic greeting modules must be integrated into their code. So, what exactly is a module? Python modules are simply files that contain commands used within the software. They are an effective way of simplifying code and making it more understandable and user-friendly.



```
from webbrowser import open as link # // used to open a web link
from time import sleep as wait # // used to wait for a number of
from tkinter import * # // Tkinter is used to open the game windo
from PIL import Image,ImageDraw,ImageDraw2,ImageGrab # // this mo
import tkinter as tk # // importing the tkinter module under anot
from tkinter.filedialog import askopenfilename, asksaveasfilename
from tkinter.colorchooser import askcolor as ASK # // This module
from random import randrange as From # // used to randomize infor
```

In the picture displayed above, you can see the various Python modules utilized in Handicraft. The webbrowser module enables the opening of web links. The time module facilitates a waiting period for a designated number of seconds. The tkinter assists in managing the software window. PIL (Python Imaging Library) is responsible for opening and closing images. Lastly, the random module plays a significant role in randomly selecting any list of data. Every module has a crucial role to play in the code's functioning. Without these modules, the code would not work correctly.

The Mouse:

One crucial component of the software's code is the section that interprets mouse movements. The mouse serves as the primary tool for performing various tasks such as drawing, selecting, and closing windows.

```
 def motion(self,event): # // function for mouse motion
        self.line_width = self.choose_size_button.get()
        if self.b1 == "down":
            if self.tool_option == 'paintbrush':

                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_oval(self.xold,self.yold,event.x,event.y,width=self.line_width,outline=self.col
or,fill=self.color)

self.draw.ellipse((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width), 
fill=self.color, outline=self.color)
                    self.stack.append(self.x)
            elif self.tool_option == 'pencil':
                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_line(self.xold,self.yold,event.x,event.y,width=self.line_width,fill=self.color)

self.draw.line(((self.xold,self.yold),(event.x,event.y)),(self.color),width=self.line_width)
                    self.stack.append(self.x)
            elif self.tool_option == 'pixel':

                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_rectangle(self.xold,self.yold,event.x,event.y,outline=self.color,fill=self.colo
r,width=self.line_width)

self.draw.rectangle((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width), 
fill=self.color, outline=self.color)
                    self.stack.append(self.x)
        self.xold = event.x
        self.yold = event.y
```

The code above is included in a class called "GameClass" and is contained within a function named "motion." The "motion" function is responsible for detecting mouse movements and clicks. If a click is detected, the code checks which tool option is selected. The shape that is drawn will vary depending on the tool option chosen.

Let's break down the "motion" function by examining each section step by step. We'll begin with "self.line_width = self.choose_size_button.get()" to gain a better understanding.

In case You Were Wondering...

We won't be discussing the pixel or pencil drawing section in the "motion" function since there isn't a significant difference between that and the code displayed below.

```
self.line_width = self.choose_size_button.get()
```

The code displayed above sets the size of the art in the software. The size is determined by the toolbar at the bottom of the canvas and will be drawn accordingly.

```
if self.b1 == "down":
            if self.tool_option == 'paintbrush':

                if self.xold is not None and self.yold is not None:

                    self.x =
event.widget.create_oval(self.xold,self.yold,event.x,event.y,width=self.line_width,outlin
e=self.color,fill=self.color)

self.draw.ellipse((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width),
fill=self.color, outline=self.color)
                    self.stack.append(self.x)
```

Within the motion function, the code above follows the previous lines of code. Initially, the code checks if the left button on the mouse is pressed down and if so, it executes the code that follows. This code specifically checks if the paintbrush is selected. If it is, a circle is drawn at the exact position of the mouse cursor. Moreover, if the left mouse button is held down while moving the mouse, a series of circles are created, forming a line. Lastly, the line of code "self.stack.append(self.x)" adds each circle drawn to a list, which can be gradually removed when undoing a portion of the artwork.

```
self.w.bind("<Motion>", self.motion) # // defines what motion is with the mouse
```

The creation of circles, squares, and lines through the "motion" function is not a standalone process. Several lines of code are required to execute this function. For instance, the utilization of a window for a canvas is only possible with the incorporation of modules like Tkinter. Additionally, the code line, "self.w.bind("<Motion>", self.motion)", is responsible for detecting motion from the computer mouse and storing the data in the variable, "self.motion." These examples emphasize the importance of code working together to achieve desired outcomes.

```
# // The code below outlines the action to be taken
when the left mouse button is clicked
        self.w.bind("<ButtonPress-1>", self.b1down)
        self.w.bind("<ButtonRelease-1>", self.b1up)
```

In the previous two pages, we discussed the significance of mouse motion in this software. However, it is crucial to note that none of it would be possible without the user clicking the left mouse button. At the beginning of the page, the code presented is designed to detect whether the left button on the mouse is up or down. In case the left button is pushed down, the value of self.b1down changes to "True." Conversely, if the button is released, the value of self.b1up becomes "True." Utilizing variables is highly effective and efficient in programming, and I utilized these two variables to convey information to the "motion" function regarding the status of the mouse button.

```
def b1down(self,event): # // what happens when the mouse is clicked

        self.line_width = self.choose_size_button.get()
        self.b1 = "down"
        if self.tool_option == "paintbrush": # // if the tool option is set to paintbrush, use the
paintbrush
            self.x =
event.widget.create_oval(self.xold,self.yold,event.x,event.y,width=self.line_width,outline=self.color,fi
ll=self.color)
            self.draw.ellipse((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width),
fill=self.color, outline=self.color)
            self.stack.append(self.x)
        elif self.tool_option == "pencil": # // if the tool option is set to pencil, use the pencil
            self.x =
event.widget.create_line(self.xold,self.yold,event.x,event.y,width=self.line_width,fill=self.color)
            self.draw.line(((self.xold,self.yold),(event.x,event.y)),(self.color),width=self.line_width)
            self.stack.append(self.x)
        elif self.tool_option == "pixel": # // if the tool option is set to pixel, use the pixel
            self.x =
event.widget.create_rectangle(self.xold,self.yold,event.x,event.y,outline=self.color,fill=self.color,wid
th=self.line_width)

self.draw.rectangle(((self.xold,self.yold),(event.x+self.line_width,event.y+self.line_width)),(self.colo
r))
            self.stack.append(self.x)
```

During the initial development of Handicraft version 1.0, I encountered a problem with the "motion" feature. While it allowed for drawing when the mouse was dragged and clicked, it failed to produce a dot on the canvas if the mouse was clicked but not dragged. Although several solutions were available, I chose to create a new function that replicated the capabilities of the "motion" feature except it activates when the left mouse button is clicked, rather than detecting motion. I have also developed a function named "b1up." This function informs additional variables that the mouse button is no longer clicked, providing extra benefits. The code for this function is shown at the bottom right of the page.

```
def b1up(self,event): # // when the mouse is released
        self.b1 = "up"
        self.xold = None
        self.yold = None
```

New Or Clear:

It can be difficult to distinguish between the "new" and "clear" commands, as they may seem alike at first glance. Nonetheless, there are distinct differences between them. I will provide an explanation of the distinctions between the "new" and "clear" commands and suggest the appropriate usage for each one.

If I needed to clarify the distinction to someone without displaying the code, I would describe how the "new" command restarts the canvas, whereas the "clear" command simply removes the content on it.

In the code below, you can see the "new" function on the left and the "clear" function on the right. You may notice that the "new" function shares many lines of code with the "clear" function. This is because the "new" function essentially starts from scratch, erasing your history, current color, tool selection, tool size, and all other variables. The "new" command resets all data variables to their original values. On the other hand, the "clear" command simply clears the canvas and resets the necessary variables in order to do so.

```
def new(self): # // I need to use this
function to create a fresh canvas for a
brand-new project.        self.fullcolor =
'white'

self.image=Image.new("RGB",(self.sizex,self.
sizey),(self.fullcolor))
        self.draw=ImageDraw.Draw(self.image)
        self.w.delete("all")
        root.config(cursor="left_ptr")
        self.w.configure(bg='white')
        self.color = 'black'
        self.filepathopen = False
        self.tool_option = 'toutch'
        self.choose_size_button =
Scale(self.fr_buttons, from_=1, to=300,
orient=HORIZONTAL)
        self.choose_size_button.grid(row=0,
column=6, sticky="ew", padx=5, pady=5)
```

```
def clear(self): # // To start a
new one, this function clears
the entire canvas.
self.w.delete("all")

self.w.configure(bg='white')
        self.fullcolor = 'white'

self.image=Image.new("RGB",(self
.sizex,self.sizey),(self.fullcol
or))

self.draw=ImageDraw.Draw(self.im
age)
```

Setting a Tool Type:

After selecting your preferred user tool type by clicking on a button, you may wonder how the code switches it to the chosen one. The solution can be found in the straightforward code snippet provided below.

```
def pixel(self): # // The tool type will be changed to "pixel".
        self.tool_option = 'pixel'
        root.config(cursor="dot")
```

In the code above, a function adapts its name according to the tool being used. Since the tool in use is called "pixel," the function is named the same. Only this function is displayed because the others are identical except for using different names and cursors. Despite its size, this function is essential. The first line sets the tool option to "pixel," allowing it to be stored in a variable. All values in the code must be stored in a variable so that they can communicate with the rest of the program. The second line of code changes the cursor to an image that looks like a dot, accomplished through the Tkinter module.

Saving a File:

```
def save(self): # // function for saving an image
        try:
            self.image.save(self.filename)
            Rnumber = From(1000,5000)
            root.config(cursor="watch")
            self.tool_option = 'toutch'
            root.after(Rnumber,self.change)
        except:
            self.saveAs_file()
```

Handicraft is no fun if you can’t save your creations, but how does saving work? After clicking save, the “save” function plays its part.

It appears that the code is attempting to run, but it appears to not be functioning properly, which could indicate that a file has not been selected. In order to proceed, the "saveAs_file" function must be executed to select a file. Assuming a file has been selected, the code will then proceed to save the file name, display a loading cursor for a random period, and then switch to a touch cursor. This brief pause is intended to give Handicraft a more polished and professional appearance. It's worth noting that the saving process is kept short and simple due to the use of a popular image software known as PIL.

Opening a File:

```
def open_file(self): # // Function to Open a File
        try:

self.IMAGEopen=tk.PhotoImage(file=self.filepathopen)
            self.MYimage = self.w.create_image(0, 0,
anchor=tk.NW, image=self.IMAGEopen)
        except:
            self.openAs_file()
```

In case of a code error, the "openAs_file" function is executed for file selection. If there are no errors, the code assumes that a file has already been chosen and proceeds accordingly. The code segment above employs the Tkinter and PIL modules to open the image file and set it on the Tkinter canvas.

Similar to the save function, the open_file function checks if a file has been selected by attempting the code.

Taking a Screenshot:

If you encounter errors while trying to save an image, there is a temporary solution that may help. Take a screenshot of your computer screen if you experience unexpected issues while saving your artwork. Although you may need additional software to crop it later, this method is likely to provide you with the desired outcome. In a previous section, I covered how to take a screenshot. Now, let's examine how the code works. Below is the code that I will go through step by step.

```
 def capture(self): # // captures a screen shot of the entire window
        x0 = self.w.winfo_rootx()
        y0 = self.w.winfo_rooty()
        x1 = x0 + self.w.winfo_width()
        y1 = y0 + self.w.winfo_height()

        saveCapture = asksaveasfilename(title="Save File", filetypes=[("png
files", "*.png")]
        )
        if not saveCapture:
            return

        wait(1)
        im = ImageGrab.grab((0, 0, 1915, 1000))
        im.save(saveCapture)
 def toggleFullScreen(self): # toggles to full screen
        self.fullScreenState = not
self.fullScreenState
        root.attributes("-fullscreen",
self.fullScreenState)
```

I have provided you with the necessary code to switch to full-screen mode, which can prove to be quite advantageous in case you wish to avoid any future cropping of the image. Now let's go through the capture function step by step, starting with the first lines within the function.

```
x0 = self.w.winfo_rootx()
y0 = self.w.winfo_rooty()
x1 = x0 + self.w.winfo_width()
y1 = y0 + self.w.winfo_height()
```

The code above is setting the height and width of the user's computer screen, into variables that can later be used.

```
saveCapture = asksaveasfilename(title="Save File",
filetypes=[("png files", "*.png")]
        )
        if not saveCapture:
            return
```

The code shown above opens a dialog box and lets the user choose where he or she wishes the save the captured screenshot and what it shall be named.

```
wait(1)
im = ImageGrab.grab((0, 0, 1915, 1000))
im.save(saveCapture)
```

The last three lines in the function are above. These lines make Handicraft wait for one second, then grab the entire computer screen and save the captured image using the PIL module.

Guess What!

Since taking a screenshot in Handicraft doesn't only take a shot of the software's screen but the entire computer screen, then you can take a screenshot of anything as long as the handicraft window is open.

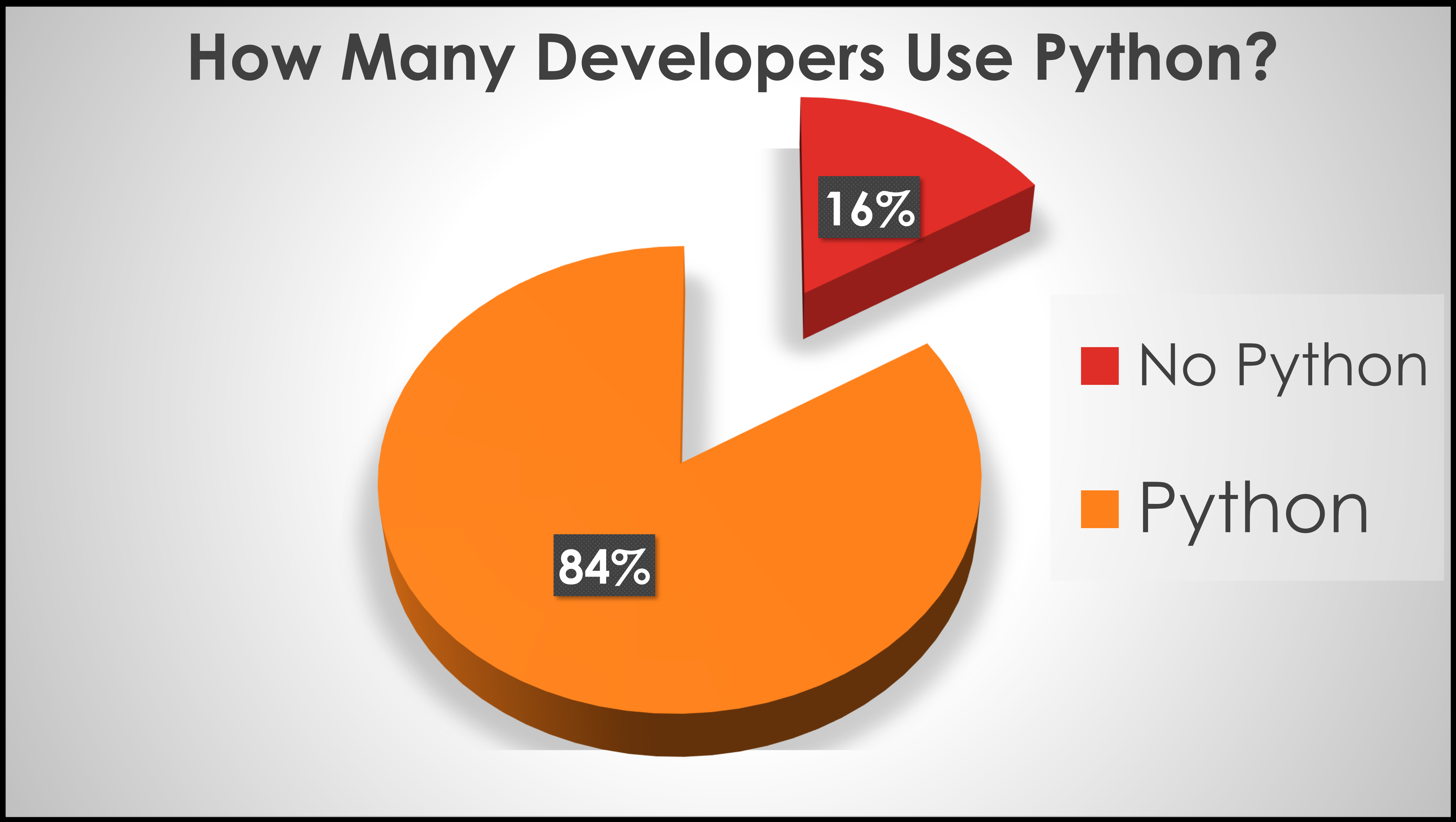
Problem Solving:

Errors are things that come up in big programs a lot, which is why problem-solving is so valuable in programming.

```python
try:
    this
except:
    print("now try this")
```

Python has a type of command that's just for error handling, shown to the left is one example of this command.

I've faced problems throughout both versions of Handicraft, but I've been able to overcome them. This software version currently has several errors, but that means I'll be able to learn more, create more, and solve more problems in future versions. Working through problems is how Handicraft works behind the scenes, It's one of the reasons that I enjoy programming. I enjoy it because I can fit the puzzle pieces together and make something new and enjoyable while solving problems I might face. In closing, Handicraft is a software that's made for creators to explore art either by using the software or by learning about the code that makes the software and getting inspired to look towards the future of crafting their own project.



The Python Developers 2021 survey by JetBrains found that 84 % of developers use Python as their primary language. For the past four years, this proportion has been constant.

See the pie chart on the left, lots of developers use Python as their primary language, and Handicraft is a great way for new developers to learn about the language and how to interact with programming in general.

THE CODE

The Handicraft commented code is fully provided on the following pages.

```
from webbrowser import open as link # // used to open a web link
from time import sleep as wait # // used to wait for a number of seconds
from tkinter import * # // Tkinter is used to open the game window
from PIL import Image,ImageDraw,ImageDraw2,ImageGrab # // this module is used to open
images
import tkinter as tk # // importing the tkinter module under another name
from tkinter.filedialog import askopenfilename, asksaveasfilename # // This module allows
the user to open and save image files to and from their computer
from tkinter.colorchooser import askcolor as ASK # // This module allows the user to
select any preferred color from a single window
from random import randrange as From # // used to randomize information

class GameClass:

    def __init__(self,parent,posx,posy,*kwargs): # // in python __init__ functions run
when the class is ran

        root.attributes('-fullscreen', False)
        self.fullScreenState = False

        root.bind("<F1>", lambda x: self.toggleFullScreen()) # // if key 'F1' is pressed,
then toggle full screen
        root.bind("<F2>", lambda x: self.capture()) # // if key 'F2' is pressed, then
capture a screen shot

        self.parent = parent

        """
        Global variables are being set
        both above and below, and then
        stored in a variable that can be
        shared within a single class.
        """
        self.posx = posx
        self.posy = posy

        # // bellow the size of the canvas is set
        self.sizex = 2000
        self.sizey = 1000

        self.b1 = "up"

        self.w = Canvas(self.parent,width=self.sizex,height=self.sizey) # // sets canvases
size
        self.w.pack(expand = True, fill = BOTH) # // allows the canvas to be exspanded

        """
        inside the program there are is a window layer called root
        and a canvas layer called w
        ""«
        self.w.place(x=self.posx,y=self.posy) # // an introduction image is placed at the
start-up of the program
        root.rowconfigure(0, minsize=80, weight=1)
        root.columnconfigure(0, minsize=80, weight=1)
        # // above, rows and columns are being configured for buttons to lay onto

        self.openButtonImages()
        self.fr_buttons = tk.Frame(root, relief=tk.RAISED, bd=2)
        fillall_button = tk.Button(self.fr_buttons, image=self.Y, command=self.fill_all)
        fillall_button.grid(row=0, column=1, sticky="ew", padx=5, pady=5)
        btn_pixel = tk.Button(self.fr_buttons, image=self.W, command=self.pixel)
        btn_pencil = tk.Button(self.fr_buttons, image=self.I, command=self.pencil)
        btn_toutch = tk.Button(self.fr_buttons, image=self.Mouse, command=self.toutch)
        btn_toutch.grid(row=0, column=0, sticky="ew", padx=5)
        btn_paintbrush = tk.Button(self.fr_buttons, image=self.T, command=self.paintbrush)
        btn_pixel.grid(row=0, column=2, sticky="ew", padx=5, pady=5)
        btn_pencil.grid(row=0, column=3, sticky="ew", padx=5, pady=5)
        btn_paintbrush.grid(row=0, column=4, sticky="ew", padx=5)

        # // above, the buttons on the bottom row are being configured and aligned

        self.choose_size_button = Scale(self.fr_buttons, from_=1, to=300,
orient=HORIZONTAL)
        self.choose_size_button.grid(row=0, column=6, sticky="ew", padx=5, pady=5)

        # // sets the button size above

        self.color_button = Button(self.fr_buttons, image=self.C,
command=self.choose_color)
        self.color_button.grid(row=0, column=5)
        self.color = 'black'

        self.fr_buttons.grid(row=1, column=0, sticky="ns")
        self.w.bind("<Motion>", self.motion) # // defines what motion is with the mouse

        # // The code below outlines the action to be taken when the left mouse button is
clicked
        self.w.bind("<ButtonPress-1>", self.b1down)
        self.w.bind("<ButtonRelease-1>", self.b1up)

        self.w.bind("<Enter>", lambda x: self.introExit()) # // Once the canvas is click
with the mouse then the introdunction image disapears compleatly
        root.bind("<Control-Shift-S>", lambda x: self.saveAs_file()) # // click control
and shift, and 'S' to save the file as a specific name
        root.bind("<Control-c>", lambda x: self.clear()) # // clear all work by clicking
control c
        root.bind("<Control-s>", lambda x: self.save()) # // if button control and s is
clicked, then save
        root.bind("<Control-n>", lambda x: self.new()) # // if clicked control and n is
clcked, then open a new canvas
        root.bind("<Escape>", lambda x: self.Quit()) # // if the Escape button on the
keyboard is clicked, then close the program
        root.bind("<Control-o>", lambda x: self.open_file()) # // When the user clicks on
the button labeled "control + o," the computer files will open, giving the user a
selection to choose from.
        root.bind("<Control-Shift-O>", lambda x: self.openAs_file()) # //I have created a
unique invention called "Open as File" which is not currently offered by any art editor.
This command functions similarly to the "Save As" command
```

```
 root.bind("<Control-z>", lambda x: self.undo()) # // after control and z are clicked,
then undo
        root.bind("<s>", lambda x: self.opensticker()) # // To add an image to the canvas,
simply press the 'S' key on your keyboard and the computer photos will open up.
        self.menubar = Menu(root)
        self.filemenu = Menu(self.menubar, tearoff=0)
        self.filemenu.add_command(label="screenshot    F2", command=self.capture)
        self.filemenu.add_command(label="placeimage    S", command=self.opensticker)
        self.filemenu.add_separator()
        self.filemenu.add_command(label="New    CTRL+N", command=self.new)
        self.filemenu.add_command(label="clear    CTRL+C", command=self.new)
        self.filemenu.add_command(label="Open    CTRL+O", command=self.open_file)
        self.filemenu.add_command(label="Open as...    CTRL+SHIFT+O",
command=self.openAs_file)
        self.filemenu.add_command(label="Save    CTRL+S", command=self.save)
        self.filemenu.add_command(label="Save as...    CTRL+SHIFT+S",
command=self.saveAs_file)
        self.destroy = False
        self.filemenu.add_separator()

        self.filemenu.add_command(label="Exit    ESC", command=self.Quit)
        self.menubar.add_cascade(label="File", menu=self.filemenu)

        self.editmenu = Menu(self.menubar, tearoff=0)
        self.editmenu.add_command(label="undo    CTRL+Z", command=self.undo)
        self.menubar.add_cascade(label="Edit", menu=self.editmenu)
        root.config(menu=self.menubar)

        self.fullcolor = 'white'
        self.helpmenu = Menu(self.menubar, tearoff=0)
        self.helpmenu.add_command(label="Handicraft Home Page", command=self.homepage)
        root.config(menu=self.menubar)

        self.viewmenu = Menu(self.menubar, tearoff=0)
        self.viewmenu.add_command(label="Full screen    F1",
command=self.toggleFullScreen)
        self.menubar.add_cascade(label="View", menu=self.viewmenu)
        root.config(menu=self.menubar)

        self.toolsmenu = Menu(self.menubar, tearoff=0)
        self.toolsmenu.add_command(label="paint brush", command=self.paintbrush)
        self.toolsmenu.add_command(label="pencil", command=self.pencil)
        self.toolsmenu.add_command(label="pixel", command=self.pixel)

        self.toolsmenu.add_command(label="fill all", command=self.fill_all)
        self.toolsmenu.add_command(label="choose color", command=self.choose_color)
        self.toolsmenu.add_command(label="toutch", command=self.toutch)

        self.menubar.add_cascade(label="Tools", menu=self.toolsmenu)
        root.config(menu=self.menubar)

        self.menubar.add_cascade(label="Help", menu=self.helpmenu)
        root.config(menu=self.menubar)
        """
        The top navigation menu now features labels and buttons that allow users to easily
access digital art tools. These tools can be accessed from both the top and bottom
toolbars, with images positioned at the bottom and words at the top. For quicker
navigation, shortcuts are also available.
        """

        self.line_width = self.choose_size_button.get()
        self.toolsmenu.bind("<Enter>", self.arrow)

        self.image=Image.new("RGB",(self.sizex,self.sizey),(self.fullcolor))
        self.draw=ImageDraw.Draw(self.image)

        self.openINTROimage() # // opens the intro image
        self.stack = []
        self.tool_option = 'toutch'

        """
        When working on handicraft projects, there are numerous tools to choose from such
as brushes, paints, fill, and many others. The "variable" mentioned here signifies that
the currently selected tool is "touch."
        """

    def openImagesticker(self): # // function for opening a sticker image
        root.bind("<Button-1>", self.get_mouseposition)
        root.config(cursor="crosshair")

    def pencil(self): # // function for setting cursor looks to the pencil image
        root.config(cursor="pencil")

    def arrow(self): # // This function sets the cursor to appear as an arrow image.
        root.config(cursor="left_ptr")

    def get_mouseposition(self,event): # // This feature detects the location of the
cursor.
        self.cy = event.y
        self.cx = event.x
        root.unbind("<Button-1>")
        root.config(cursor="arrow")
        self.stickerOpenC()

    def undo(self): # // This function reverses any changes made to your artwork,
effectively undoing your previous actions.
        try:

            self.x = self.stack.pop()
            self.w.delete(self.x)
        except:
            pass
```

```
 def capture(self): # // captures a screenshot of the entire computer screen
        x0 = self.w.winfo_rootx()
        y0 = self.w.winfo_rooty()
        x1 = x0 + self.w.winfo_width()
        y1 = y0 + self.w.winfo_height()

        saveCapture = asksaveasfilename(title="Save File", filetypes=[("png files",
"*.png")]
        )
        if not saveCapture:
            return

        wait(1)
        im = ImageGrab.grab((0, 0, 1915, 1000))
        im.save(saveCapture)

    def toggleFullScreen(self): # toggles to full screen
        self.fullScreenState = not self.fullScreenState
        root.attributes("-fullscreen", self.fullScreenState)

    def fill_all(self): # function for using the fill tool and filling the entire canvas
to one single color
         self.w.configure(bg=self.color)
         self.fullcolor = self.color
         self.image=Image.new("RGB",(self.sizex,self.sizey),(self.fullcolor))
         self.draw=ImageDraw.Draw(self.image)

    def toutch(self): # // setting tool type to toutch
        self.tool_option = 'toutch'
        root.config(cursor="left_ptr")

    def choose_color(self): # // function for choosing a color
        self.color = ASK(color=self.color)[1]

    def paintbrush(self): # // setting tool type to paint brush
        self.tool_option = 'paintbrush'
        root.config(cursor="spraycan")

    def pencil(self): # // setting tool type to pencil
        self.tool_option = 'pencil'
        root.config(cursor="pencil")

    def pixel(self): # // setting tool type to pixel
        self.tool_option = 'pixel'
        root.config(cursor="dot")

    def new(self): # // This function creates a new canvas with a new project
        self.fullcolor = 'white'
        self.image=Image.new("RGB",(self.sizex,self.sizey),(self.fullcolor))
        self.draw=ImageDraw.Draw(self.image)
        self.w.delete("all")
        root.config(cursor="left_ptr")
        self.w.configure(bg='white')
        self.color = 'black'
        self.filepathopen = False
        self.tool_option = 'toutch'
        self.choose_size_button = Scale(self.fr_buttons, from_=1, to=300,
orient=HORIZONTAL)
        self.choose_size_button.grid(row=0, column=6, sticky="ew", padx=5, pady=5)
 def clear(self): # // this function clears the entire canvas to start all over
        self.w.delete("all")
        self.w.configure(bg='white')
        self.fullcolor = 'white'
        self.image=Image.new("RGB",(self.sizex,self.sizey),(self.fullcolor))
        self.draw=ImageDraw.Draw(self.image)

    def b1down(self,event): # // what happens when the mouse is clicked

        self.line_width = self.choose_size_button.get()
        self.b1 = "down"
        if self.tool_option == "paintbrush": # // If the tool option is set to a
paintbrush, use the paintbrush
            self.x =
event.widget.create_oval(self.xold,self.yold,event.x,event.y,width=self.line_width,outline
=self.color,fill=self.color)

self.draw.ellipse((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width),
fill=self.color, outline=self.color)
            self.stack.append(self.x)
        elif self.tool_option == "pencil": # //If the tool option is set to pencil, use
the pencil
            self.x =
event.widget.create_line(self.xold,self.yold,event.x,event.y,width=self.line_width,fill=se
lf.color)

self.draw.line(((self.xold,self.yold),(event.x,event.y)),(self.color),width=self.line_widt
h)
            self.stack.append(self.x)
        elif self.tool_option == "pixel": # // if the tool option is set to pixel, use the
pixel
            self.x =
event.widget.create_rectangle(self.xold,self.yold,event.x,event.y,outline=self.color,fill=
self.color,width=self.line_width)

self.draw.rectangle(((self.xold,self.yold),(event.x+self.line_width,event.y+self.line_widt
h)),(self.color))
            self.stack.append(self.x)

    def introExit(self): # // function for exiting the introduction
        if self.destroy == False:
            self.clear()
            self.destroy = True

    def b1up(self,event): # // when the mouse is released
        self.b1 = "up"
        self.xold = None
        self.yold = None
 def motion(self,event): # // function for mouse motion
        self.line_width = self.choose_size_button.get()
        if self.b1 == "down":
            if self.tool_option == 'paintbrush':

                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_oval(self.xold,self.yold,event.x,event.y,width=self.line_width,outline
=self.color,fill=self.color)

self.draw.ellipse((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width),
fill=self.color, outline=self.color)
                    self.stack.append(self.x)
            elif self.tool_option == 'pencil':
                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_line(self.xold,self.yold,event.x,event.y,width=self.line_width,fill=se
lf.color)

self.draw.line(((self.xold,self.yold),(event.x,event.y)),(self.color),width=self.line_widt
h)
                    self.stack.append(self.x)
            elif self.tool_option == 'pixel':

                if self.xold is not None and self.yold is not None:

                    self.x = 
event.widget.create_rectangle(self.xold,self.yold,event.x,event.y,outline=self.color,fill=
self.color,width=self.line_width)

self.draw.rectangle((self.xold,self.yold,event.x+self.line_width,event.y+self.line_width),
fill=self.color, outline=self.color)
                    self.stack.append(self.x)
        self.xold = event.x
        self.yold = event.y

    def homepage(self):
        link("https://pi-this.github.io/handicraft.html") # // sends the user to a website 
for more information on how to use Handicraft

    def Quit(self):
        root.destroy() # // quits Handicraft

    def opensticker(self): # // function for choosing what sticker to open

        self.filepathopensticker = askopenfilename(title="Open File", filetypes=[("png 
files", "*.png")]
        )
        if not self.filepathopensticker:
            return


        self.openImagesticker()
 def stickerOpenC(self): # // function for opening the chosen sticker
        self.IMAGEopen=tk.PhotoImage(file=self.filepathopensticker)
        self.Sticker = self.w.create_image(self.cx, self.cy, anchor=tk.NW,
image=self.IMAGEopen)

    def openAs_file(self): # open as funtion

        filepathopen = askopenfilename(title="Open File", filetypes=[("png files",
"*.png")]
        )
        if not filepathopen:
            return
        self.IMAGEopen=tk.PhotoImage(file=filepathopen)
        self.MYimage = self.w.create_image(0, 0, anchor=tk.NW, image=self.IMAGEopen)
        self.filepathopen = filepathopen

    def open_file(self): # // open file function
        try:
            self.IMAGEopen=tk.PhotoImage(file=self.filepathopen)
            self.MYimage = self.w.create_image(0, 0, anchor=tk.NW, image=self.IMAGEopen)
        except:
            self.openAs_file()

    def saveAs_file(self): # // save as function
        filepathsave = asksaveasfilename(title="Save File", filetypes=[("png files",
"*.png")]
        )
        if not filepathsave:
            return
        self.image.save(filepathsave)
        self.filename = filepathsave
        Rnumber = From(1000,5000)
        root.config(cursor="watch")
        self.tool_option = 'toutch'
        root.after(Rnumber,self.change)

    def change(self): # // changing the tool option
        if self.tool_option == 'pixel':
            self.pixel()
        if self.tool_option == 'paintbrush':
            self.paintbrush()
        if self.tool_option == 'pencil':
            self.pencil()
        if self.tool_option == 'toutch':
            self.toutch()
 def save(self): # // function for saving an image
        try:
            self.image.save(self.filename)
            Rnumber = From(1000,5000)
            root.config(cursor="watch")
            self.tool_option = 'toutch'
            root.after(Rnumber,self.change)
        except:
            self.saveAs_file()

    def openINTROimage(self): # // function for opening the into image
        self.IMAGEopenINTRO=tk.PhotoImage(file='Base_images/intro.png')
        self.MYimageINTRO = self.w.create_image(0, 0, anchor=tk.NW,
image=self.IMAGEopenINTRO)

    def openButtonImages(self): # // function for opening images for each button

        self.I=tk.PhotoImage(file='Base_images/draw.png')
        self.M = self.w.create_image(0, 0, anchor=tk.NW, image=self.I)

        self.W=tk.PhotoImage(file='Base_images/pixel.png')
        self.K = self.w.create_image(0, 0, anchor=tk.NW, image=self.W)

        self.T=tk.PhotoImage(file='Base_images/paint.png')
        self.B = self.w.create_image(0, 0, anchor=tk.NW, image=self.T)

        self.C=tk.PhotoImage(file='Base_images/color.png')
        self.A = self.w.create_image(0, 0, anchor=tk.NW, image=self.C)

        self.Y=tk.PhotoImage(file='Base_images/fill_all.png')
        self.Z = self.w.create_image(0, 0, anchor=tk.NW, image=self.Y)

        self.Mouse=tk.PhotoImage(file='Base_images/mouse.png')
        self.esuoM = self.w.create_image(0, 0, anchor=tk.NW, image=self.Y)


root=Tk() # // creates the Tkinter window called root
root.wm_geometry("%dx%d+%d+%d" % (500, 550, 500, 100)) # // set's the windows size
root.config(bg='white') # // sets the window's background color to white
root.title( "Handicraft" ) # // names the window's title screen to 'Handicraft'
GameClass(root,0,0) # // runs the game class named 'GameClass'
root.mainloop() # // The window is placed in a main loop, allowing for inputs to be
received at any time.
```

RESOURCES

W3schools.com – website
Github.com – website
GeeksForGeeks.com – website
StackOverflow.com – website
RealPython.com – website
learnPython.org – website
Python.org – website
Invent Your Own Computer Game With Python – Book
Think Python – Book