Pixel Art Editor with Python and Tkinter
私のPythonとTkinterの学習記録
デスクトップアプリを作りながら

梅屋萬年堂

2022/045 ver 0.5,0.6

何年前のことかも忘れてしまいましたが、Python でデスクトップアプリを作るときに、その GUI の作成に Tkiniter を使ってみたことがありました。その時は日本語入力がうまくできず（？）Tkinter は断念して wxPython を使いました。wxPython は wxFormBuilder もあって簡単にユーザーインターフェイス部分を作成できる優れものです。そうであるのですが、Tkinter は Python の標準モジュールであり、”軽い”のではないかという期待と日本語入力もできるようだったので再挑戦してみることにしました。

ネットで「Tkinter 使い方」で検索すると参考になるものがたくさん出てきます。それらを参考にしながら一つのアプリを作成して Tkinter の勉強をしていきます。題材は Pixel Art Editor です。ドット絵を作るアプリです。高機能のドット絵作成ツールは、EDGE などたくさんあります。ここでつくいるものは、そんな高機能ではなく、自分でツールを作って、自分で簡単なドット絵を描くものを目指しています。16x16 ドット、色は２色のものから始めて、できるだけですが、欲しい機能を追加していきます。それで Python と Tkinter の勉強ができるであろうと思っています。

このノートは TexShop で書き、画面のスナップショット作成加工に、ScreenShot[*1]を使用しました。アプリ作成 IDE には PyCharm を使い、実行形式の作成には pyinstaller を使用しました。

以下に、このノートで記述される内容について列記します。

第１節　まずは 16X16 ドット、赤白２色のドット絵作成

第２節　データをファイルに保存、読み込みの機能を追加

第３節　使用する色を増やす

第４節　作成した「ドット絵」の画像ファイルとして保存と表示

第５節　キャンバスサイズの拡張と表示の拡大縮小

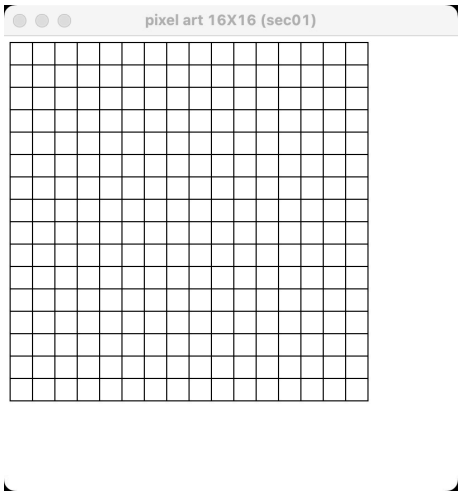
第６節　少しだけ描画ツールを

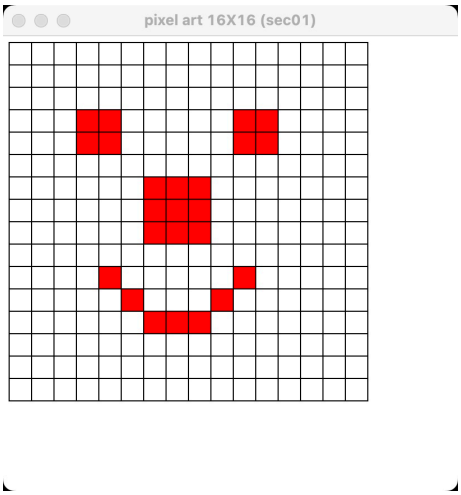
[*1] Python と wxPython で作った自作アプリです。Vector にアップロードしてあります。(https://www.vector.co.jp/soft/mac/util/se524071.html)

目次

1　まずは 16X16 ドット、赤白２色のドット絵作成

下の２つの図が実行結果です。





左が起動直後のもので、このマス目をクリックすることで赤いドットを描きます。右がいくつかのドットを赤にしたものです。ここでは２色だけなので赤い部分をクリックすると白に戻しています。

下のソースコード Px16ArtEditor.py で説明していきます。

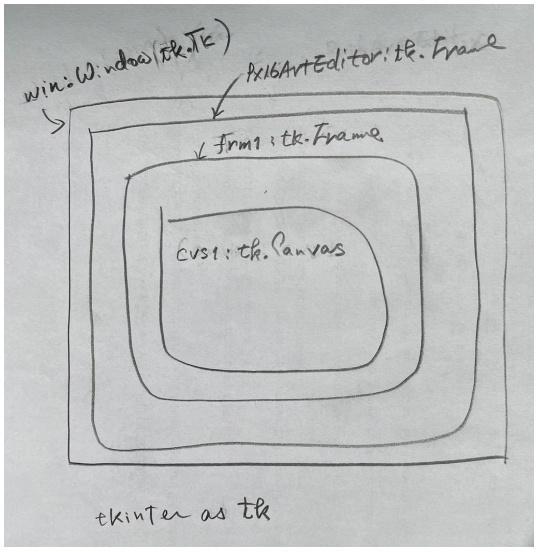
Px16ArtEditor01.py

```
1  import tkinter as tk
2
3  class Px16ArtEditor(tk.Frame):
4    def __init__(self, master:tk.Tk):
5      super().__init__(master)
6      master.title('pixel art 16X16 (sec01)')
7      self.pack()
8      self.frm1 = tk.Frame(self)
9      self.cvs1 = tk.Canvas(self.frm1
10       , width=400, height=400, bg='white',highlightthickness=0)
11     self.cvs1.bind('<Button>', self.on_click)
12     self.cvs1.pack(padx=10, pady=10)
13     self.frm1.pack()
14     self.cvs1_data = [[0]*16 for _ in range(16)]
15     self.draw_squares(self.cvs1)
16
17
18   def draw_squares(self, cvs:tk.Canvas):
19     for r in range(17):
20       cvs.create_line(0, r * 20, 20 * 16, r * 20
21         , fill='black', width=1, tag='squares')
22     for c in range(17):
23       cvs.create_line(c*20, 0, c*20, 20*16
24         , fill='black', width=1, tag='squares')
25     cvs.update()
26
27   def draw_pixels_on_cvs_data(self, cvs:tk.Canvas, data):
28     for r in range(16):
29       for c in range(16):
```

```
30            if data[r][c] == 0:
31              color = 'white'
32            else:
33              color = 'red'
34            cvs.create_rectangle(c*20, r*20, c*20+20, r*20+20
35              , fill=color, outline='black', tag='pix')
36        cvs.update()
37
38    def on_click(self, e):
39        c, r = e.x//20, e.y//20
40        if 0<=c<=15 and 0<=r<=15:
41          self.cvs1_data[r][c] = 1 - self.cvs1_data[r][c]
42          self.draw_pixels_on_cvs_data(self.cvs1, self.cvs1_data)
43
44  def main_loop():
45    win = tk.Tk()
46    app = Px16ArtEditor(master=win)
47    app.mainloop()
48
49  if __name__ == '__main__':
50    main_loop()
```

行番号１で Tkinter をインポートし、Tk をその省略名としています。Tkinter でのプログラムは行番号４５から４８のようにして無限ループを作ります。４６行目でウィンドウを作り、４７行目の app は、ウィンドウ (win) の中に作るフレームが書かれたクラス Px16ArtEditor から生成されます。４８行目でループを作っています。この部分はクラス名やウィンドウの名前が違う程度で定番のコードになります。本体は３から４３行までのクラス Px16ArtEditor になります。このウィンドウ上にさまざまな GUI 構成部品（widget）を配置していきます。

今回は、下のスケッチのように、ウィンドウ上にフレーム、そのフレーム上にキャンバスを配置することにします。ここで使うものはキャンバスだけですが、機能拡張したときに部品の配置がしやすくなるかもしれないので直接ウィンドウ上にキャンバスを置かないでフレームをはさんで置きます。



このクラス Px16ArtEditor は Tk.Frame を継承して、一つのフレームを作っています。Tkinter ではボタンとかキャンバスとかさまざまな部品 (widget) を配置していきます。配置する方法は pack、grid、place とありますが、ここでは pack で作成していきます。一つのウィンドウ上に Frame を置いて、その中にさらに

Frame を置いたり、部品（widget）を置きます。Frame はいくつかの部品をひとまとめにする囲いのようなものだと思っています。pack での配置はコードを書いた順に並べるというもので簡単です。これに Frame を使えば、ある程度のものはできる（？）と思います。

では、クラス Px16ArtEditor のコードを順にみていきます。

4 から 15 行　`__init__`メソッドについて。

6-8 行　47 行の引数 master に win を渡されて、5 行目のスーパークラス（親クラス）をその master（win）にして初期化しています。6 行で master のサイズ変更をできないようにして、7 行でウィンドウのタイトルを設定し、８行でこのフレームを master の上に配置しています。

9,13 行　　新たにフレーム frm1 を作り配置します

9 から 12 行　　幅 400(ピクセル)、高さ 400（ピクセル)、背景色を白、囲み線なしにしてキャンバス cvs1 を作り、このキャンバス上でマウスクリックしたときのイベント処理を `on_click` メソッドにします。そしてフレーム frm1 上に置きます。

14 行　　２次元のリストの配列 `cvs1_data` を初期化します。各要素の値が 0 で白、1 で赤を表します。要素数は 16X16 ドットに必要な個数を用意しています

16 行　　キャンバス上にマス目を描く `draw_squares` メソッドを呼び出します

18 から 25 行　関数 `draw_squsres` はキャンバス cvs1 上に直線をひきマス目を作ります。

27 から 36 行　関数 `draw_pixels_on_cvs_data` は 配列 `cvs1_data` の各要素の値が 0 で白、1 で赤に塗りつぶ長方形（縦横同じだから正方形）を描きます。

38 から 42 行　関数 `on_click`] は キャンバス上でマウスをクリックした時の処理です。e.x と e.y からクリックした時の座標が得られ、それをマス目の番号にします。これが配列 `cvs1_data` のインデックスになり、1 のとき 0,0 のとき 1 に要素の値を設定します。変更後、再描画しています。

Tkinter の使い方などは、ネットにたくさん載っています。それも参考にコードを読んでください。

2　データをファイルに保存、読み込みの機能を追加

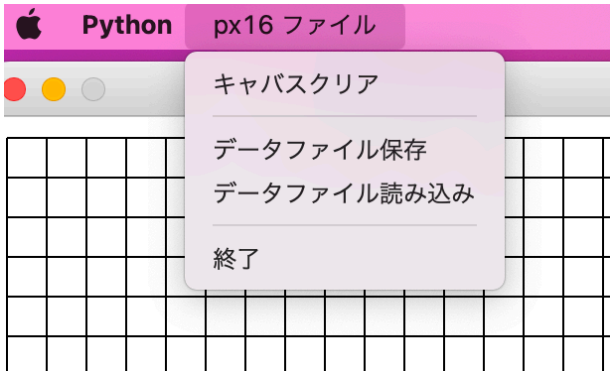
作成したデータは cvs1_data の２次元リストにあります。これをファイルに保存したり、データをファイルから読み込みます。json 形式でファイルで取り扱います。

Px16ArtEditor02.py へ Px16ArtEditor01.py から追加・変更されたコードを見ていきます。

Px16ArtEditor02.py(No1)

```
1  # coding: utf-8
2  import tkinter as tk
3  from tkinter import filedialog
4  import json
5
6  class Px16ArtEditor(tk.Frame):
7    def __init__(self, master:tk.Tk):
8      super().__init__(master)
9      master.title('pixel art 16X16')
10     self.pack()
11     self.frm1 = tk.Frame(self)
12     # ———— menu
13     menubar = tk.Menu(master=self.master)
14     filemenu = tk.Menu(menubar, tearoff=0)
15     filemenu.add_command(label=キャバスクリア='', command=lambda :self.art_clear(cvs=self.cvs1))
16     filemenu.add_separator()
17     filemenu.add_command(label=データファイル保存='', command=self.art_save)
18     filemenu.add_command(label=データファイル読み込み='', command=self.art_load)
19     filemenu.add_separator()
20     filemenu.add_command(label=終了='', command=self.master.quit)
21     menubar.add_cascade(label='px16 ファイル', menu=filemenu)
22     self.master.config(menu=menubar)
```

ファイルの保存や読み込みに filedialog を、json 形式の取り扱いに json モジュールを行 3 と 4 でインポートします。行 13 から 22 で下の図のようなメニューバーをセットします。13 行でメニューバーを作ります。14 行で filemenu という名前のメニューを作り、15 から 20 行でその filemenu のメニュウアイテムを実装します。command オプションでメニュアイテムがクリックされたときに呼び出す関数を指定しておきます。それぞれのメニューがクリックされたときに呼び出される関数を command オプションで設定しています。他に表示されたドット絵をクリアするメニューとこのアプリの終了メニューも作っておきます。20 行の quit はアプリを終了させるコマンドです。21 行でメニューバーに filemenu を繋ぎます（cascade します）



「キャンパスクリア」メニューで呼び出されるのが次の art_clear です。

Px16ArtEditor02.py(No2)

```
56    def art_clear(self, cvs:tk.Canvas):
57      cvs.delete('all')
58      self.draw_squares(cvs)
```

そしてファイルの保存と読み込みが art_save と art_load になります。

Px16ArtEditor02.py(No3)

```
62    def art_save(self):
63      filename = filedialog.asksaveasfilename(
64
65        title='Pixel ART Editor: データの保存',
66        filetypes=(('json file', '*.json'),)
67      )
68      if filename:
69
70        with open(filename, 'w') as f:
71          json.dump(self.cvs1_data, f)
72
73    def art_load(self):
74      filename = filedialog.askopenfilename(
75
76        title='Pixel Art Editor: データの読み込み',
77        filetypes=(('json file', '*.json'),)
78      )
79      if filename:
80        with open(filename) as f:
81          self.cvs1_data = json.load(f)
82
83        self.draw_pixels_on_cvs_data(self.cvs1, self.cvs1_data)
```

キャンバスのクリア　art_clear 関数

行 56 でキャンバスをクリアします。キャンバスに描かれた図形にはタグ (tag) を付けることができ、'all' は描かれた全てを表しています。

ファイルへの書き込み　art_save 関数

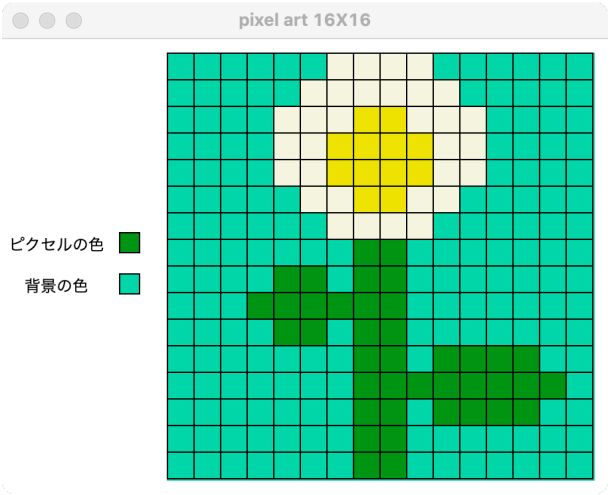
63 から 67 行でファイルを保存するためのダイアログを開き、ファイル名を取得します。セルボタン (Cancal)」や「閉じるボタン (X)」をクリックした場合、 判定で偽 (False) となる値が返ってきます。あとは、70 と 71 行で json 形式のファイルを書き込みます。

ファイルから読み込み　art_load 関数

書き込みとほぼ同じで、行 74 から 78 で読み込みするファイル名を取得し、80、81 行で読み込み、そのデータを表示しています。

3　使用する色を増やす

ここまでは白と赤の２色のみでしたが、色を増やし、下の図のようなものもできるようにします。



ウィンドウ左側に現在のピクセルの色を正方形の枠内に表示します。これをクリックすることで色指定ができ、その指定された色が表示されるようにします。白赤の２色のときは０と１で色を区別していましたが、これからは１６進カラーコードを使うことにして、データは、たとえば白は'#ffffff'といった文字列として扱います。指定がなかったピクセルは、０で表します。さらに背景色も指定できるようにします。

Px16ArtEditor03.py

```
4  from tkinter import colorchooser
```

この colorchooser を使ってカラーを選択します。
次にフレーム frm0 を frm1 の左に配置し、ここにラベルと小さななキャンバスを２組作ます。配置は grid を使います。このキャンバスをクリックしてカラーを選択ダイアログを表示する `on_click_pixel_color` と `on_click_bg_color` をそれぞれバインドしておきます。

```
25      # ----------frm0
26      self.frm0 = tk.Frame(self)
27      self.frm0.pack(side=tk.LEFT, anchor=tk.W)
28      self.label_color_title = tk.Label(self.frm0, text色='')
29      self.label_color_title.grid(column=0, row=0)
30      self.label_pixel_color = tk.Label(self.frm0,
31                          text=uピクセル'')
32      self.label_pixel_color.grid(column=0, row=1)
33      self.cvs_pixel_color = tk.Canvas(self.frm0,
34                          width=20, height=20, bg='white')
35      self.cvs_pixel_color.bind('<Button>', self.on_click_pixel_color)
36      self.cvs_pixel_color.grid(column=1, row=1)
37      self.pixel_color = '#ffffff'
38      self.label_bg_color = tk.Label(self.frm0,
39                          text=u背景'')
40      self.label_bg_color.grid(column=0, row=2)
41      self.cvs_bg_color = tk.Canvas(self.frm0,
```

```
42                      width=25, height=25, bg='white')
43      self.cvs_bg_color.bind('<Button>', self.on_click_bg_color)
44      self.cvs_bg_color.grid(column=1, row=2)
45      self.bg_color = '#ffffff'

57      self.draw_squsre_with_selected_color(self.cvs_pixel_color, '#ffffff')
58      self.draw_squsre_with_selected_color(self.cvs_bg_color, '#ffffff')
```

色選択用のキャンバスの初期値を白にして表示しています。
キャンバスクリアの関数 art_clear では、データ配列 cvs1_data の 0 クリアを追加します。

```
62    def art_clear(self, cvs:tk.Canvas):
63      cvs.delete('all')
64      for r in range(16):
65        for c in range(16):
66          self.cvs1_data[r][c] = 0
67      self.draw_squares(cvs)
```

関数 draw_pixels_on_cvs_data、on_left_click_cvs1、on_right_click_cvs1、art_save、art_load、on_click_pixel_color、 on_click_bg_color での変更追加は以下の通り。

```
78
79    def draw_pixels_on_cvs_data(self, cvs:tk.Canvas, data):
80      for r in range(16):
81        for c in range(16):
82          if data[r][c] !=0:
83            color = data[r][c]
84          else:
85            color = self.bg_color
86          cvs.create_rectangle(c*20,r*20,c*20+20,r*20+20
87           , fill=color, outline='black', tag='pix')
88
89      cvs.update()
90
91    def on_left_click_cvs1(self, e):
92      c, r = e.x//20, e.y//20
93      if 0<=c<=15 and 0<=r<=15:
94        self.cvs1_data[r][c] = self.pixel_color
95        self.draw_pixels_on_cvs_data(self.cvs1, self.cvs1_data)
96
97    def on_right_click_cvs1(self, e):
98      c, r = e.x//20, e.y//20
99      if 0<=c<=15 and 0<=r<=15:
100       self.cvs1_data[r][c] = 0
101       self.draw_pixels_on_cvs_data(self.cvs1, self.cvs1_data)
102
103   def art_save(self):
104     filename = filedialog.asksaveasfilename(
105       title='Pixel ART Editor: データの保存',
106       filetypes=(('json file', '*.json'),)
107     )
108     if filename:
109       with open(filename, 'w') as f:
110       json.dump((self.pixel_color, self.bg_color,self.cvs1_data), f)
111
```

```
112     def art_load(self):
113       filename = filedialog.askopenfilename(
114         title='Pixel Art Editor: データの読み込み',
115         filetypes=(('json file', '*.json'),)
116       )
117       if filename:
118         with open(filename) as f:
119           self.pixel_color, self.bg_color,self.cvs1_data = json.load(f)
120
121         self.draw_squsre_with_selected_color(self.cvs_pixel_color, self.pixel_color)
122         self.draw_squsre_with_selected_color(self.cvs_bg_color, self.bg_color)
123         self.draw_pixels_on_cvs_data(self.cvs1, self.cvs1_data)
124
125     def on_click_pixel_color(self, e):
126       _, color = colorchooser.askcolor()
127       if color is not None:
128         self.pixel_color = color
129         self.draw_squsre_with_selected_color(self.cvs_pixel_color, color)
130
131     def on_click_bg_color(self, e):
132       _, color = colorchooser.askcolor()
133       if color is not None:
134         self.bg_color = color
135         self.draw_squsre_with_selected_color(self.cvs_bg_color, color)
136         self.cvs1.config(bg=color)
137
138     def draw_squsre_with_selected_color(self, cvs:tk.Canvas, color):
139       cvs.create_rectangle(5,5, 20,20, fill=color,outline='black')
```

関数 `draw_pixel_on_cvs_data` では、色の指定があるときはそのカラーコードを使い、色指定がないときは背景色のコードを使って描くようにしました。

関数 `on_left_click_cvs1` はキャンバスを左クリックしたときに選択指定されている色をピクセルに塗る処理です。

関数 `on_right_click_cvs1` ではデータ配列 `cvs1_data[r][c]` に 0 を入れます。これで、そのピクセルに背景色が塗られ、クリック前に塗られていた色を消すことになります。

関数 `art_save` データ配列に、ピクセルの色と背景の色も保存読み込みをします。

関数 `on_click_pixel_color`、`on_click_bg_color` ピクセルに塗る色と背景の色の選択処理部分です。それぞれの色を表示しているキャンバスをクリックして、色選択ダイアログ `colorchooser.askcolor` により色を選択しています。

`draw_squqre_with_selected_color` は、選択された色を表示しています。

4　作成した「ドット絵」の画像ファイルとして保存と表示

ライブラリー `Pillow` を使って画像の保存と表示をします。まず必要な PIL(Pillow) をインポートします。

Px16ArtEditor04.py

```
6  from PIL import Image, ImageDraw
```

「ファイル」メニューにこの三つのメニューアイテムを追加しておきます。

```
22      exportmenu = tk.Menu(filemenu, tearoff=False)
23      exportmenu.add_command(label='BMP 形式',
24          command=lambda : self.export_image_file('bmp', self.cvs1_data, self.bg_color, 16, 16))
25      exportmenu.add_command(label='PNG 形式',
26          command=lambda : self.export_image_file('png', self.cvs1_data, self.bg_color, 16, 16))
27      exportmenu.add_command(label='JPG 形式',
28          command=lambda : self.export_image_file('jpg', self.cvs1_data, self.bg_color, 16, 16))
29      filemenu.add_cascade(labelドット絵のエクスポート（='BMP,PNG,) JPG', menu=exportmenu)
30      filemenu.add_separator()
31      filemenu.add_command(label=uドット絵の表示'(標準表示ソフトで) OS', command=self.show_image)
```

関数 `export_image_file` では、保存ファイル名の選択と画像イメージの作成保存をおこなっています。行 134 から 137 が保存ファイルの選択部分です。行 140 から 148 までで Pillow ライブラリを使ってイメージを作成しています。データ配列の値が 0 でないときはその値（カラーコード`#nnnnnn`）の色で点を打ちます。0 のときは背景色で点を打ちます。 行 148 で作成したイメージを書き出しています。ファイル名の拡張子に応じたフォーマットのファイルになります。

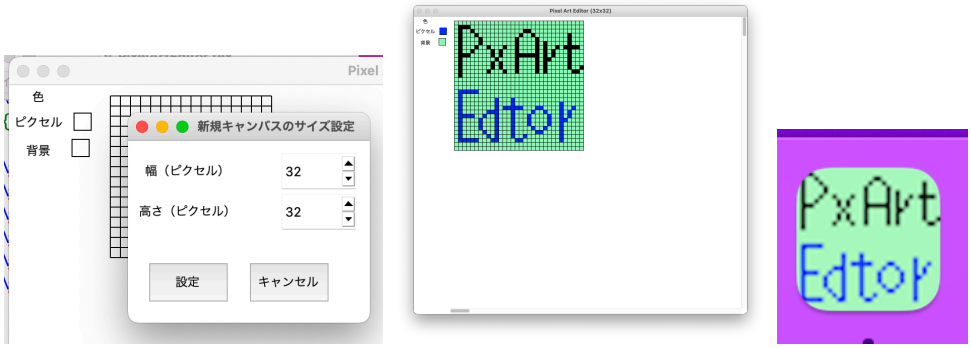
行 151 から 161 の イメージの作成は `export_image_file` と同じで、161 行の部分が `im.save()` ではなく、`im.show()` となっているだけです。Mac だと「プレビュー」、windows だと「フォト」でイメージが表示されます。

```
133   def export_image_file(self, fmt:str, data:list, bg, width, height):
134     filename = filedialog.asksaveasfilename(
135       title='Pixel ART Editor: ' + fmt形式でのイメージファイルの保存+''+'',
136       filetypes=((fmt+' file', '*.'+fmt),)
137     )
138     if not filename:
139       return
140     im = Image.new('RGB',(width, height), '#ffffff')
141     draw = ImageDraw.Draw(im)
142     for r in range(height):
143       for c in range(width):
144         if data[r][c] != 0:
145           draw.point(((c, r)), fill=data[r][c])
146         else:
147           draw.point(((c, r)), fill=bg)
148     im.save(filename)
149
150
151   def show_image(self):
152     data, bg, width, height = self.cvs1_data, self.bg_color, 16, 16
153     im = Image.new('RGB',(width, height), '#ffffff')
154     draw = ImageDraw.Draw(im)
155     for r in range(height):
156       for c in range(width):
```

```
157             if data[r][c] != 0:
158               draw.point(((c, r)), fill=data[r][c])
159             else:
160               draw.point(((c, r)), fill=self.bg_color)
161         im.show()
```

5　キャンバスサイズの拡張と表示の拡大縮小

サイズが 16x16 では小さすぎるので、16、32、64、128、256、512、1024 から選ぶことができるようにします。これまでのマス目を 20 から 10 にしてサイズが大きくなったときにも表示エリアが対応できるようにします。サイズが 128 以上になると表示しきれなくなるのでキャンバスにスクロールバーをつけます。さらに表示の拡大縮小もできるようにしておきます。下の図は 32x32 で新規のキャンパスを作り（左端の図）、そのキャンパスで `PxArt Editr` のドット絵を書いたものです（中央の図）。pyinstaller を使ってこのスクリプトを Mac 実行形式にするときのアイコンをここで作成したもの使用しました。右端の図はそのアプリを実行したときの Dock に表示されたアイコンを撮影したものです。



ここで追加した機能のコードはリスト `PixelArtEditorV05.py` をみてもらえば良いかと思っています。

行 14 から 19　いくつかの定数をここで定義しています。キャンバスのサイズが変更されたときに変数に値を入れることにします。キャンバスのサイズが `art_width` 、 `art_height` で表示は、`square_length` の値でコントロールしています。

行 28　新規キャンパスの設定用のメニューです。このメニューにバイドした `new_canvas_menu` から `new_canvas` を呼び出しています。行 117 から 145 コードはサブウィンドウを表示して、キャンバスのサイズを選択し、それを取得しているところです。

行 47 から 51　表示の拡大縮小のメニューです。行 254 から 270 の `zoom_display` で処理しています。引数 ratio の値をこれまでの拡大率（縮小率）にかけています。倍率は 0.5 から 2.0 までとしています。この倍率を `square_length10(=10 ピクセル)` にかけ `square_length` を計算して、それを使って網目とドット絵を再表示しています。

これ以外のコードは、定数を値にもつ変数に書き換えただけです。

PixelArtEditorV05.py

```
1  # coding: utf-8
2  """
3  PixelArtEditor ver0.5
4  """
5  import tkinter as tk
6  from tkinter import filedialog
7  from tkinter import colorchooser
8  import json
9  from PIL import Image, ImageDraw
```

```
10
11  class PxArtEditor(tk.Frame):
12    def __init__(self, master:tk.Tk):
13      super().__init__(master)
14      # 定数 ----------------
15      self.zoom_ratio = 1.0 # 表示倍率
16      self.square_length10 = 10 # 網目の正方形の初期値
17      self.square_length = self.square_length10* self.zoom_ratio # 網目の正方形の一辺の長さ（ピクセル）
18      self.cvs_art_length = 700 ピクセル絵を描くキャンバスのサイズ#
19      self.art_lengt_values = [16, 32, 64, 128, 256, 512, 1024] # 設定セル数
20      self.art_width = self.art_height = self.art_lengt_values[0] # ドット絵の幅と高さ
21      self.art_data = [[0] * self.art_width for _ in range(self.art_height)]
22      # ---------------------
23      master.title(f'Pixel Art Editor ({self.art_width}x{self.art_height})')
24      self.pack()
25      # ---------- menu
26      menubar = tk.Menu(master=self.master)
27      filemenu = tk.Menu(menubar, tearoff=0)
28      filemenu.add_command(label新規キャンバス='', command=self.new_canvas_menu)
29      filemenu.add_command(labelキャンバスクリア='', command=lambda: self.art_clear(cvs=self.cvs_art))
30      filemenu.add_separator()
31      filemenu.add_command(labelデータファイル保存='', command=self.art_save)
32      filemenu.add_command(labelデータファイル読み込み='', command=self.art_load)
33      filemenu.add_separator()
34      exportmenu = tk.Menu(filemenu, tearoff=False)
35      exportmenu.add_command(label='BMP 形式',
36                             command=lambda: self.export_image_file('bmp', self.cvs1_data, self.bg_color, 16, 16))
37      exportmenu.add_command(label='PNG 形式',
38                             command=lambda: self.export_image_file('png', self.cvs1_data, self.bg_color, 16, 16))
39      exportmenu.add_command(label='JPG 形式',
40                             command=lambda: self.export_image_file('jpg', self.cvs1_data, self.bg_color, 16, 16))
41      filemenu.add_cascade(labelドット絵のエクスポート（='BMP,PNG,) JPG', menu=exportmenu)
42      filemenu.add_separator()
43      filemenu.add_command(label=uドット絵の表示'(標準表示ソフトで) OS', command=self.show_image)
44      filemenu.add_separator()
45      filemenu.add_command(label終了='', command=self.master.quit)
46      menubar.add_cascade(labelファイル='', menu=filemenu)
47      zoommenu = tk.Menu(menubar, tearoff=0)
48      zoommenu.add_command(label拡大='(x 1.1)', command=lambda :self.zoom_display(1.1))
49      zoommenu.add_command(labelリセット（等倍）='', command=lambda :self.zoom_display(1))
50      zoommenu.add_command(label縮小='(x 0.9)', command=lambda :self.zoom_display(0.9))
51      menubar.add_cascade(label表示='', menu=zoommenu)
52      self.master.config(menu=menubar)
53      # ----------frm0
54      self.frm0 = tk.Frame(self)
55      self.frm0.pack(side=tk.LEFT, anchor=tk.NW)
56      self.label_color_title = tk.Label(self.frm0, text色='')
57      self.label_color_title.grid(column=0, row=0)
58      self.label_pixel_color = tk.Label(self.frm0,
59                            text=uピクセル'')
60      self.label_pixel_color.grid(column=0,row=1)
61      self.cvs_pixel_color = tk.Canvas(self.frm0,
62                           width=20, height=20, bg='white')
63      self.cvs_pixel_color.bind('<Button>', self.on_click_pixel_color)
64      self.cvs_pixel_color.grid(column=1,row=1)
65      self.pixel_color = '#ffffff'
66      self.label_bg_color = tk.Label(self.frm0,text=u背景'')
```

```
67        self.label_bg_color.grid(column=0,row=2)
68        self.cvs_bg_color = tk.Canvas(self.frm0,width=25, height=25, bg='white')
69        self.cvs_bg_color.bind('<Button>', self.on_click_bg_color)
70        self.cvs_bg_color.grid(column=1,row=2)
71        self.bg_color = '#ffffff'
72        # ———— frm1
73        self.frm1 = tk.Frame(self,
74                             width=400, height=400)
75        self.cvs_art = tk.Canvas(self.frm1,
76          width=self.cvs_art_length, height=self.cvs_art_length,
77          bg='white',highlightthickness=0,
78          scrollregion=(0, 0,
79                   self.art_lengt_values[-1]*self.square_length,
80                   self.art_lengt_values[-1]*self.square_length))
81        xbar = tk.Scrollbar(self.frm1, orient=tk.HORIZONTAL)
82        xbar.config(command=self.cvs_art.xview)
83        ybar = tk.Scrollbar(self.frm1, orient=tk.VERTICAL)
84        ybar.config(command=self.cvs_art.yview)
85        self.cvs_art.bind('<Button-1>', self.on_left_click_cvs_art)
86        self.cvs_art.bind('<Button-2>', self.on_right_click_cvs_art)
87        self.cvs_art.grid(column=0, row=0,padx=10, pady=10)
88        xbar.grid(column=0, row=1, sticky=tk.W+tk.E)
89        ybar.grid(column=1, row=0, sticky=tk.N+tk.S)
90        self.cvs_art.config(xscrollcommand=xbar.set)
91        self.cvs_art.config(yscrollcommand=ybar.set)
92        self.frm1.pack(side=tk.LEFT)
93
94        self.draw_squares(self.cvs_art)
95        self.draw_square_with_selected_color(self.cvs_pixel_color, '#ffffff')
96        self.draw_square_with_selected_color(self.cvs_bg_color, '#ffffff')
97
98
99    def art_clear(self, cvs:tk.Canvas):
100     cvs.delete('all')
101     for r in range(self.art_height):
102       for c in range(self.art_width):
103         self.art_data[r][c] = 0
104     self.draw_squares(cvs)
105
106   def new_canvas_menu(self):
107     self.new_canvas_info = None
108     self.new_canvas()
109     if self.new_canvas_info is not None:
110       self.art_width, self.art_height = self.new_canvas_info
111       self.master.title(f'Pixel Art Editor ({self.art_width}x{self.art_height})')
112       self.art_data = [[0] * self.art_width for _ in range(self.art_height)]
113       self.zoom_ratio = 1.0
114       self.square_length = self.square_length10 * self.zoom_ratio
115       self.draw_squares(self.cvs_art)
116
117   def new_canvas(self):
118     def sub_win_btn_ok():
119       self.new_canvas_info = (int(spinbox_x.get()), int(spinbox_y.get()))
120       sub_win.destroy()
121     def sub_win_btn_cancel():
122       self.new_canvas_info = None
123       sub_win.destroy()
```

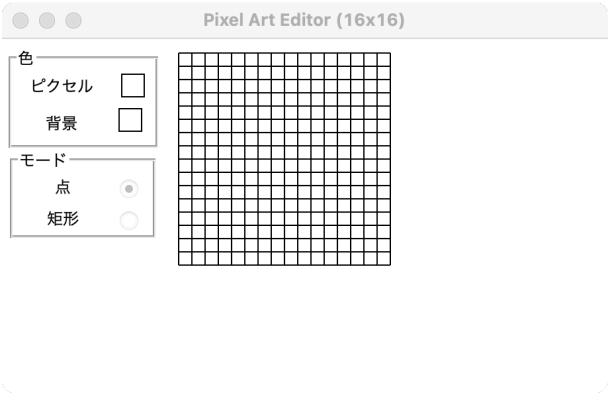
```
124       sub_win = tk.Toplevel()
125       sub_win.title新規キャンバスのサイズ設定('')
126       sub_win.geometry("240x180")
127       label_sub1 = tk.Label(sub_win, text=u幅（ピクセル）'')
128       label_sub1.place(width=100, height=40, x=10, y=10)
129       spinbox_x = tk.Spinbox(sub_win, state='readonly',
130                   values=self.art_lengt_values)
131       spinbox_x.place(width=80, height=40, x=150, y=10)
132       label_sub2 = tk.Label(sub_win, text=u高さ（ピクセル）'')
133       label_sub2.place(width=100, height=40, x=10, y=50)
134       spinbox_y = tk.Spinbox(sub_win, state='readonly',
135                   values=self.art_lengt_values)
136       spinbox_y.place(width=80, height=40, x=150, y=50)
137       button_ok = tk.Button(sub_win, text=u設定'',command=sub_win_btn_ok)
138       button_ok.place(width=80, height=40, x=20, y=120)
139       button_cancel = tk.Button(sub_win, text=uキャンセル'',command=sub_win_btn_cancel)
140       button_cancel.place(width=80, height=40, x=120, y=120)
141       self.new_canvas_info = None
142       sub_win.grab_set()
143       sub_win.focus_set()
144       sub_win.transient(self.master)
145       self.master.wait_window(sub_win)
146
147
148     def draw_squares(self, cvs:tk.Canvas):
149       for r in range(self.art_height+1):
150         cvs.create_line(0, r * self.square_length,
151                 self.square_length * self.art_width, r * self.square_length
152                 ,fill='black', width=1, tag='squares')
153       for c in range(self.art_width+1):
154         cvs.create_line(c*self.square_length, 0,
155                 c*self.square_length, self.square_length*self.art_height
156                 , fill='black', width=1, tag='squares')
157       cvs.update()
158
159     def draw_pixels_on_cvs_data(self, cvs:tk.Canvas, data):
160       for r in range(self.art_height):
161         for c in range(self.art_width):
162           if data[r][c] !=0:
163             color = data[r][c]
164           else:
165             color = self.bg_color
166           cvs.create_rectangle(c * self.square_length, r * self.square_length,
167                     c * self.square_length + self.square_length, (r+1) * self.square_length
168                     , fill=color, outline='black', tag='pix')
169       cvs.update()
170
171     def on_left_click_cvs_art(self, e):
172       c, r = int(self.cvs_art.canvasx(e.x)/self.square_length), \
173            int(self.cvs_art.canvasy(e.y)/self.square_length)
174       if 0<=c<self.art_width and 0<=r<self.art_height:
175         self.art_data[r][c] = self.pixel_color
176         self.cvs_art.create_rectangle(c * self.square_length, r * self.square_length,
177                       (c+1) * self.square_length, (r+1) * self.square_length
178                       , fill=self.pixel_color, outline='black', tag='pix')
179
180     def on_right_click_cvs_art(self, e):
```

```
181       c, r = int(self.cvs_art.canvasx(e.x) / self.square_length), \
182             int(self.cvs_art.canvasy(e.y) / self.square_length)
183       if 0<=c<self.art_width and 0<=r<self.art_height:
184         self.art_data[r][c] = 0
185         self.cvs_art.create_rectangle(c * self.square_length, r * self.square_length,
186                         (c+1) * self.square_length, (r+1) * self.square_length
187                        , fill=self.bg_color, outline='black', tag='pix')
188
189     def on_click_pixel_color(self, e):
190       _, color = colorchooser.askcolor(parent=self, title=u'ピクセルの色')
191       if color is not None:
192         self.pixel_color = color
193         self.draw_square_with_selected_color(self.cvs_pixel_color, color)
194
195     def on_click_bg_color(self, e):
196       _, color = colorchooser.askcolor(parent=self, title=u'背景色')
197       if color is not None:
198         self.bg_color = color
199         self.draw_square_with_selected_color(self.cvs_bg_color, color)
200         self.draw_pixels_on_cvs_data(self.cvs_art, self.art_data)
201
202     def draw_square_with_selected_color(self, cvs:tk.Canvas, color):
203       cvs.create_rectangle(3,3, 20,20, fill=color,outline='black')
204
205     def art_save(self):
206       filename = filedialog.asksaveasfilename(
207         title='Pixel ART Editor: データの保存',
208         filetypes=(('json file', '*.json'),)
209       )
210       if filename:
211         with open(filename, 'w') as f:
212           json.dump((self.art_width, self.art_height,self.art_data, self.bg_color), f)
213
214     def art_load(self):
215       filename = filedialog.askopenfilename(
216         title='Pixel Art Editor: データの読み込み',
217         filetypes=(('json file', '*.json'),)
218       )
219       if filename:
220         with open(filename) as f:
221           self.art_width, self.art_height,self.art_data, self.bg_color = json.load(f)
222         self.draw_pixels_on_cvs_data(self.cvs_art, self.art_data)
223         self.draw_square_with_selected_color(self.cvs_bg_color, self.bg_color)
224
225     def export_image_file(self, fmt:str, data:list, bg, width, height):
226       filename = filedialog.asksaveasfilename(
227         title='Pixel ART Editor: ' + fmt形式でのイメージファイルの保存+''+'',
228         filetypes=((fmt+' file', '*.'+fmt),)
229       )
230       if not filename:
231         return
232       im = Image.new('RGB',(width, height), '#ffffff')
233       draw = ImageDraw.Draw(im)
234       for r in range(height):
235         for c in range(width):
236           if data[r][c] != 0:
237             draw.point(((c, r)), fill=data[r][c])
```

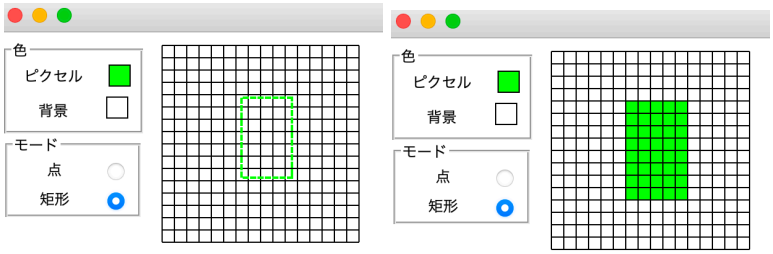
```
238           else:
239             draw.point(((c, r)), fill=bg)
240       im.save(filename)
241
242     def show_image(self):
243       data, bg, width, height = self.art_data, self.bg_color, self.art_width, self.art_height
244       im = Image.new('RGB',(width, height), '#ffffff')
245       draw = ImageDraw.Draw(im)
246       for r in range(height):
247         for c in range(width):
248           if data[r][c] != 0:
249             draw.point(((c, r)), fill=data[r][c])
250           else:
251             draw.point(((c, r)), fill=self.bg_color)
252       im.show()
253
254     def zoom_display(self, ratio):
255       if ratio == 1:
256         self.zoom_ratio = 1.0
257         self.square_length = self.square_length10
258       elif ratio > 1:
259         if self.zoom_ratio < 2.0:
260           self.zoom_ratio *= ratio
261           self.square_length = self.square_length10 * self.zoom_ratio
262       elif ratio < 1:
263         if self.zoom_ratio > 0.5:
264           self.zoom_ratio *= ratio
265           self.square_length = self.square_length10 * self.zoom_ratio
266       else:
267         return
268       self.cvs_art.delete('all')
269       self.draw_squares(self.cvs_art)
270       self.draw_pixels_on_cvs_data(self.cvs_art, self.art_data)
271
272
273
274
275   def main_loop():
276     win = tk.Tk()
277     app = PxArtEditor(master=win)
278     app.mainloop()
279
280   if __name__ == '__main__':
281     main_loop()
```

6　少しだけ描画ツールを

点を打つだけでは、サイズが大きくなると大変なので線をひく程度の描画ツールを用意したいと思います。バージョンを v0.6 にして、スクリプトファイル名を PixelArtEditorV06.py にします。下の図のようにウィンドウ左側に「点」か「矩形」かを選択するラジオボタンを作ります。これを変数 pen_mode にそれぞれ'dot'、'line' にして選択制御できるようにします。



下の左の図は、モードを矩形にして、キャンバス上で１回マウスをクリックした後、ドラックすることで矩形の範囲を指定します。ESC キーを押すことで指定をキャンセルできます。指定の範囲が定まって、もう一回クリックすると右の図のようにその部分が現在のピクセルの色で塗られます。



該当するコードを順にみていきます。

PixelArtEditorV06.py

```
24      self.pen_mode = tk.StringVar(value='dot')
25      self.line_point0 = None
```

ここでは、いくつかの変数を用意しています。pen_mode はウィジット変数で、これを二つあるラジオボタンの選択状態の取得に使います。line_point0 は、矩形モードでの１回目のクリック位置を保持します。

```
77      self.frm0b = tk.LabelFrame(self.frm0, textモード='')
78      self.frm0b.pack()
79      lbl0b1 = tk.Label(self.frm0b, text点='', width=8).grid(column=0, row=0)
80      self.radio_pen_dot = tk.Radiobutton(self.frm0b,
81                          value='dot', variable=self.pen_mode,
82                          command=self.radio_pen_click)
83      self.radio_pen_dot.grid(column=1, row=0)
84      lbl0b2 = tk.Label(self.frm0b, text矩形='').grid(column=0, row=1)
85      self.radio_pen_line = tk.Radiobutton(self.frm0b,
```

```
86                          value='line', variable=self.pen_mode,
87                          command=self.radio_pen_click)
88      self.radio_pen_line.grid(column=1, row=1)
```

この 77 から 88 行で frm0b ラベル付きフレーム上にラジオボタンを配置しています。通常ラジオボタンは、そのラベルを text で指定するとボタンが左で右にラベルがくるのですが、ここでは text を指定せずラベルは別に用意してボタンが右で左にラベルがくるようにしています。

関数 `on_left_click_cvs_art` では、「点」か「矩形」のいずれが選択されているかに応じて `on_click_cvs_art_in_dot_mode` か `on_click_cvs_art_in_dot_mode` かを呼び出しそれぞれの処理をしています。関数 `on_click_cvs_art_in_dot_mode` は名前を変えただけです。

```
190   def on_left_click_cvs_art(self, e):
191     if self.pen_mode.get() == 'line':
192       self.on_click_cvs_art_in_line_mode(e)
193     else:
194       self.on_click_cvs_art_in_dot_mode(e)
195
196   def on_click_cvs_art_in_dot_mode(self, e):
197     c, r = int(self.cvs_art.canvasx(e.x)/self.square_length), \
198         int(self.cvs_art.canvasy(e.y)/self.square_length)
199     if 0<=c<self.art_width and 0<=r<self.art_height:
200       self.art_data[r][c] = self.pixel_color
201       self.cvs_art.create_rectangle(c * self.square_length, r * self.square_length,
202                     (c+1) * self.square_length, (r+1) * self.square_length
203                     , fill=self.pixel_color, outline='black', tag='pix')
```

以下の関数 `on_click_cvs_art_in_line_mode` が「矩形」描画の処理のためのものです。対角線の両端の２点の指定で矩形を描画するのですが、まずは１点の目の指定を確認します。299 行で１点目が指定済みかどうか調べています。`line_point0` が１点目の座標を保持しますが、この値が None のときは未指定になります。その場合はクリックされた座標を代入します。指定済みの場合はクリックされた点が対角線のもう一方の端の点になるのでここで矩形を描画します。303 から 317 行までで処理しています。

```
298   def on_click_cvs_art_in_line_mode(self, e):
299     if self.line_point0 is None:
300       self.line_point0 = (self.cvs_art.canvasx(e.x), self.cvs_art.canvasx(e.y))
301
302     else:
303       c0, r0 = int(self.line_point0[0]/self.square_length),\
304           int(self.line_point0[1]/self.square_length)
305       c1, r1 = int(self.cvs_art.canvasx(e.x) / self.square_length), \
306           int(self.cvs_art.canvasy(e.y) / self.square_length)
307       if c0 > c1:
308         c0, c1 = c1, c0
309       if r0 > r1:
310         r0, r1 = r1, r0
311       if 0 <= c0 and c1 <self.art_width and 0<=r0 and r1<self.art_height:
312         for r in range(r0, r1+1):
313           for c in range(c0, c1+1):
314             self.art_data[r][c] = self.pixel_color
315             self.cvs_art.create_rectangle(c * self.square_length, r * self.square_length,
316                         (c + 1) * self.square_length, (r + 1) * self.square_length
317                         , fill=self.pixel_color, outline='black', tag='pix')
318       self.line_point0 = None
```

関数 `move_in_cvs_art` では、`'line'` モードで矩形の対角線の一点が指定されているとき、ドラッグしている点をもう一点とした矩形の辺を点線で描き塗りつぶす句権を確認しています。

```
320     def move_in_cvs_art(self, e):
321       if (self.pen_mode.get() != 'line') or (self.line_point0 is None):
322         return
323       if self.pen_mode.get() == 'line' and self.line_point0 is not None:
324         self.cvs_art.delete('dottedLine')
325         self.cvs_art.create_rectangle(self.line_point0[0], self.line_point0[1],
326                         self.cvs_art.canvasx(e.x), self.cvs_art.canvasx(e.y),
327                         width=2, outline=self.pixel_color, dash=(3,3), tag='dottedLine')
```

次の３つの関数は、矩形描画の第１点をリセットしています。

```
329
330     def key_down(self, e):
331       key_smb = e.keysym
332       if key_smb == 'Escape':
333         self.reset_pen_mode()
334
335     def radio_pen_click(self):
336       self.reset_pen_mode()
337
338     def reset_pen_mode(self):
339       self.line_point0 = None
340       self.cvs_art.delete('dottedLine')
```