Final Project Report: Pixel Art Maker by Sandrian

Introduction:

In planning the final project, I initially wanted to create a mini game; I had the idea to do a simple platformer game. The reason for that was because I thought it would be simple and sufficient for my final project, but then I changed my mind and thought I could put a little more thought into it.

Besides being a university student, I am a freelance animator and an artist, and they are my hobbies as well. With that, I thought maybe I could integrate my hobby into a Python Script, thus leading to the idea of making a mini art program. After researching on the internet, I came upon a few more ideas but eventually I settled on specifically a Pixel Art Maker. It was more common to see a pixel art program based on Pygame and looking at some examples, their coding was relatively intermediate-ish.

I do not consider myself a good programmer, so it was a tough challenge to do on my own. I then found a video which helped me finish the script.

- Here is the Youtube video:
  https://youtu.be/N20eXcfyQ_4

I want to say that this video was a big help and most of the credit goes to the creator of the video for helping me finish this.

Project Specifications:

- Goal:

  Make a mini art program that's designed for pixel art creation.

- Description:

  It should have the basics of a drawing program; be able to draw, erase, and clear, and be able to use colors from the basic color wheel. The program is divided into 2

sections, the canvas and the toolbar. The canvas is the area where the user can draw on and the toolbar is the area with all the tools that allow the user to change color, erase, and etc. This program uses Python and Pygame.

Solution Design/The Inner Workings:

- Directory Structure

  There are a total of 4 python files, 1 main driver file and 3 utility files. The 3 utility files are stored in a “utils” folder.

  List of Files:

  - *fpScriptmain.py* - The main driver file which runs the program and the functions created.

  Inside the ‘utils’ folder:

  - *__init__.py* - The file that makes python interpret the ‘utils’ folder as a python package.
  - *settings.py* - The file that stores all the constant variables and values
  - *button.py* - The file that stores the Button class which creates the toolbar buttons in the main program

1) settings.py

```python
import pygame

#Initialize pygame sources
pygame.init()
pygame.font.init()

#the variables are constants, hence why they are all in capital
#defining the colors that will be used in the game
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
#PRIMARY COLORS
RED = (255, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
#SECONDARY COLORS
YELLOW = (255, 255, 0)
ORANGE = (255, 128, 0)
VIOLET = (128, 0, 255)
#OTHER/MIXED COLORS
ROSE = (199, 21, 133)
MAGENTA = (255, 0, 255)
CHARTREUSE = (128, 255, 0)
SPRINGGREEN = (0, 255, 128)
CYAN = (0, 255, 255)
AZURE = (0, 128, 255)



#defining the framerate for game
FPS = 120

#defining the size of the pygame window
WIDTH, HEIGHT = 600, 750

#create the size of the pixels
ROWS = COLS = 70

#defining the size of the toolbar
TOOLBAR_HIEGHT = HEIGHT - WIDTH

#specifying/clarifying the width and height of each pixel
PIXEL_SIZE = WIDTH // COLS

#define the canvas color
BG_COLOR = WHITE

#make the grid to see where pixels are placed on the screen
DRAW_GRID_LINES = False

#function that returns a font object with a specific size
def get_font(size):
    return pygame.font.SysFont("consolas", size)
```

Modules:

- *pygame*
    - *pygame.init()*
    - *pygame.font.init()*

Variables:

- *List of Colors with RGB values set*
    - White, Black, Red, Blue, Green, Yellow, Orange, Violet, Rose, Magenta, Chartreuse, Spring-Green, Cyan, Azure
- *FPS*
    - Sets the frames that the program refreshes per second
- *WIDTH, HEIGHT*
    - Sets the values of the size of the program
- *ROWS, COLS*
    - Sets the value for the size of each pixel that appears on screen
- *TOOLBAR_HEIGHT*
    - The remainder of '*HEIGHT - WIDTH*' is the value set for the size of the toolbar
- *PIXEL_SIZE*
    - This variable is different from the *ROWS* and *COLS* variable, in which this one stores the width and height information of each pixel based on the value that is set for the *ROWS/COLS* variable.
- *BG_COLOR*
    - This stores the background color of the program. Uses the WHITE variable from the list of colors.
- *DRAW_GRID_LINES*
    - This is a boolean variable; whenever it is set to True, the grids in the program will show, otherwise it will not.
- *def get_font(size):*
    - This function's purpose is to be able to return a font object using *pygame.font.SysFont* with the *size* as well.

2) __init__.py

```python
import pygame
from .button import Button
from .settings import * #import everything from the settings.py file for this init file (use the '*')

#Initialize pygame sources
pygame.init()
pygame.font.init()

```

Modules:

- *pygame*
    - *pygame.init()*
    - *pygame.font.init()*

Purpose

As stated before, this file is mainly used as the initializer file so that Python would interpret the '*utils*' folder as a Python package, which would allow everything in the folder to be imported into the main driver file.

The other utility files are imported here as well just so everything can transfer smoothly, via:

- *from button import Button* (importing Button class)
- *from .settings import ** (import stored variables in settings.py)

3) button.py

```python
from .settings import *

class Button:
    ##-------------- The initializer function

    def __init__(self, x, y, width, height, color, text = None, text_color = BLACK):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.color = color
        self.text = text
        self.text_color = text_color

    ##-------------- The function to draw the buttons and it's characteristics

    def draw(self, win):
        pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.height)) #Draw the rectangle itself
        pygame.draw.rect(win, BLACK, (self.x, self.y, self.width, self.height), 2) #Draw the outline of the rectangle

        #If there is text, draw the text
        if self.text:
            button_font = get_font(17) #set size of font
            text_surface = button_font.render(self.text, 1, self.text_color) #render the text
            #place the text on the window/screen
            win.blit(text_surface, (self.x + self.width/2 - text_surface.get_width()/2, self.y + self.height/2 - text_surface.get_height()/2))

    ##-------------- The function to check if a certain position of the mouse is clicked by the user

    def clicked(self, pos):
        x, y = pos

        if not (x >= self.x and x <= self.x + self.width): #check if the mouse position is not in the x bound of the rect
            return False
        if not (y >= self.y and y <= self.y + self.height): #check if the mouse position is not in the y bound of the rect
            return False

        return True #if the mouse position is in the x and y bound of the rect, click.
```

Purpose

This file holds the Button class. In this Button class, it includes the initializer function, a function to draw the rectange and text inside the rectangle, and a function that checks the position of the mouse in the program if it's on a rectangle or not.

Functions

- *def __init__(self, x, y, width, height, color, text = None, text_color = BLACK)*
    - The initializer function. It can take in a x, y, width, height, color, text, and text color input. These things define the characteristics of the rectangle which helps in creating one.
- *def draw(self, win)*
    - This function draws the rectangle. To go into detail, it uses *pygame.draw.rect* to draw the rectangle, and it also draws the outline of

the rectangle. There is also a component to add text inside the rectangle. This function was used to create the Erase and Clear buttons.

- *def clicked(self, pos)*
    - This function checks for the mouse position and if it is on or off the rectangle or not. Basically, it checks if the mouse is not in the x and y of the rectangle area, which would return False, and if it was inside, then it would return True.

4) fpScriptmain.py

Purpose

This is the main driver file where the program is run. It draws in all of the constant variables from the settings.py file and the functions in the button.py file and uses them to bring the program to life.

Script Sections

- **Setup**

```
1   from utils import * #import everything from the utils folder for this main script
2
3   #set/define the window of the pygame
4   WIN = pygame.display.set_mode((WIDTH, HEIGHT))
5
6   #change the title/caption of the window
7   pygame.display.set_caption("Sans' Pixel Art Maker")
8
9
```

    - This section focuses on setting up some basic things.
        - Importing the utils folder into the main .py file
        - Creating WIN variable to set/define the size of the window
        - With the *set.caption* line, it changes the name on the top part of the program.

- Grid Setup

```
10
11  ##----------- Setting up the GRID on screen
12
13  #setting up the individual pixel to create the grid
14  def init_grid(rows, cols, color):
15      grid = []
16
17      for i in range(rows): #loop these row of arrays (pixels) to create the grid pattern
18          grid.append([])
19          for _ in range(cols): #add the color for all these individual pixels
20              grid[i].append(color)
21
22      return grid
23
```

  - This is the section that sets up the grid. The reason for the grid is so that we can get color data for every pixel in the drawable area or in other words it creates the individual pixels to create that data. This part only creates the arrays.
    - Uses an array (grid = [])
    - A for loop is used to create the array. This array holds in color RGB data and it loops depending on the amount of rows and columns there are in the canvas. (Line 17-18)
    - In the for loop above, there is another for loop which is intended to add the color for the individual pixels. (Line 19-20)

- Drawing the Functions on the window/program

```python
##---------- Draw functions on the window/program

#function to draw the grid
def draw_grid(win, grid):
    for i, row in enumerate(grid): # for i and for j lines determine the position of each of the individual pixel of the grid
        for j, pixel in enumerate(row):
            pygame.draw.rect(win, pixel, (j * PIXEL_SIZE, i * PIXEL_SIZE, PIXEL_SIZE, PIXEL_SIZE))

    #the + 1 is to add the last missing line on the sides of the window on x and y coordinate
    if DRAW_GRID_LINES:
        for i in range(ROWS + 1): #draw grid lines on the X coordinate/ for the horizontal lines
            pygame.draw.line(win, BLACK, (0, i * PIXEL_SIZE), (WIDTH, i * PIXEL_SIZE))

        for i in range(COLS + 1): #draw grid lines on the y coordinate / for the verticle lines
            pygame.draw.line(win, BLACK, (i * PIXEL_SIZE, 0), (i * PIXEL_SIZE, HEIGHT - TOOLBAR_HIEGHT))


#function to call of the window's characteristics
def draw(win, grid, buttons):
    win.fill(BG_COLOR) #fill with bg color (from settings.py)
    draw_grid(win, grid) #draw the grid

    for button in buttons: #draw all the buttons created
        button.draw(win)

    pygame.display.update() #update the display of the program

```

  - This section focuses on drawing all the needed visuals in the program.
    - The *def draw_grid(win, grid)* function takes the grid array that has been created and enumerates it (which basically creates a value for each list in the array) and then uses the *pygame.draw.rect* line to draw the rectangle based on the enumerated array list. The draw.rect line takes a surface, color, and rect (the specifications of it) to create the rectangle/square.
    - In the if statement below, it calls the *DRAW_GRID_LINES* variable which is a boolean variable. This part basically functions to draw the actual grid lines based on the rows and columns that are available. The if statement is checking if the *DRAW_GRID_LINES* is True or False and depending on that, it would either draw or not draw the grid lines.
    - The *def draw(win, grid, buttons)* function draws ALL of the elements in the program. Firstly, it fills the background color, secondly, it calls the function above it to draw the grid of pixels, and then third and last, it draws the buttons that are created (shown in the next lines of this code).

- Event Loop/Running the Program

```python
#find the position of the pixels on the canvas in accordance to the mouse position
def get_row_col_from_pos(pos):
    x, y = pos
    row = y // PIXEL_SIZE
    col = x // PIXEL_SIZE

    if row >= ROWS: #detect the non-drawable area
        raise IndexError

    return row, col


##----------- EVENT LOOP/RUNNING THE PROGRAM: Loop to let everything run until the user exits the program

run = True
clock = pygame.time.Clock() #setting up a clock
grid = init_grid(ROWS, COLS, BG_COLOR) #call function to create the grid
drawing_color = BLACK

## Creating the buttons

button_preY1 = HEIGHT - TOOLBAR_HIEGHT/2 - 60 #preset 1 of Y variable to replace in creating Buttons
button_preY2 = HEIGHT - TOOLBAR_HIEGHT/2 - 1 #preset 1 of Y variable to replace in creating Buttons

#Buttons list
buttons = [
    #TOP ROW
    Button(10, button_preY1, 50, 50, RED), #create the red color button
    Button(70, button_preY1, 50, 50, ROSE), #create the rose color button
    Button(130, button_preY1, 50, 50, MAGENTA), #create the magenta color button
    Button(190, button_preY1, 50, 50, VIOLET), #create the violet color button
    Button(250, button_preY1, 50, 50, BLUE), #create the blue color button
    Button(310, button_preY1, 50, 50, AZURE), #create the purple color button
    Button(370, button_preY1, 50, 50, BLACK), #create the black color button
    Button(430, button_preY1, 50, 50, WHITE, "Erase", BLACK), #create the erase button
    Button(490, button_preY1, 50, 50, WHITE, "Clear", BLACK), #create the clear button
    #BOTTOM ROW
    Button(10, button_preY2, 50, 50, ORANGE), #create the orange color button
    Button(70, button_preY2, 50, 50, YELLOW), #create the yellow color button
    Button(130, button_preY2, 50, 50, CHARTREUSE), #create the chartreuse color button
    Button(190, button_preY2, 50, 50, GREEN), #create the green color button
    Button(250, button_preY2, 50, 50, SPRINGGREEN), #create the spring green color button
    Button(310, button_preY2, 50, 50, CYAN), #create the cyan color button
]

```

```python
while run:
    clock.tick(FPS) #make sure this loop does not run faster than the set variable of FPS (To balance performance)

    for event in pygame.event.get(): #check all events of the pygame that is happening
        if event.type == pygame.QUIT: #if the user has quit the window
            run = False

        if pygame.mouse.get_pressed()[0]: #check if lmb is pressed
            posOfMouse = pygame.mouse.get_pos() #find the x and y position of the mouse

            #Check if the position of the mouse click is inside the drawable area
            try:
                row, col = get_row_col_from_pos(posOfMouse)
                grid[row][col] = drawing_color #if inside the drawable area, change the pixel color

            except IndexError: #if not inside drawable area
                for button in buttons:
                    if not button.clicked(posOfMouse): #check if the position of mouse is not with the button click function
                        continue

                    drawing_color = button.color #change the color based on where the user clicks the buttons

                    #creating the clear function for clear button
                    if button.text == "Clear":
                        grid = init_grid(ROWS, COLS, BG_COLOR) #to clear the canvas, just reset the canvas (copy the previous grid initializer script)
                        drawing_color = BLACK #after clearing, change the "brush" to black


    draw(WIN, grid, buttons) #call draw function and use the display size to draw the white BG

pygame.quit()
```

- This final section is where the program is run at. Additionally, it has some other functions, arrays, and variables that add towards running the program.
    - The *def get_row_col_from_pos(pos)* function serves to identify the location of the row/column or x/y or a specific pixel based on where the mouse cursor is located. To find the row and column aka position, the x and y are floor divided by the *PIXEL_SIZE;* the remainder of that division is used to find the enumerated *grid* array in previous lines, this picks out the pixel location.
    - For the if statement in the *def get_row_col_from_pos(pos)* function, it checks if whether the position of the *row* is greater than or equal to the *ROWS* constant variable, if it is, that means the cursor is in the toolbar in which it will return an *IndexError* and have the program continue without doing anything.
    - In the next few lines, there are the variables *run, clock, grid,* and *drawing color*. The *run* variable is a boolean variable and is currently at True. The *clock* variable creates a timer with the *pygame.time.Clock()*. The grid variable calls the grid initializer function to create the screen/canvas. The *drawing color* is self explanatory. These will be used in the next few lines.
    - In creating the buttons, it calls the Button class from *button.py.* You enter the parameters created for the Button class and it will be created. The *button_preY1* and *button_preY2* variable acts as the y parameter, since the y position is affected by the toolbar's positioning, the height of the canvas needs to be subtracted by the toolbar's height to get its position.
    - The while loop is where the program gets executed from; it uses the *run* boolean variable to determine whether the while loop continues or not. In the first for loop, it checks on all of the events that are happening in pygame; if the event type happens to be exiting the program, it will then change the *run* boolean to False which then exits the while loop, leading to *pygame.quit()*.
    - In the next lines, the main if statement checks if the user has pressed their left mouse button with *pygame.mouse.get_pressed()[0]* ['0' being the value stored for left mouse button]. If the user has pressed their LMB, it will then try checking if the position of the mouse is in the canvas or

not; if yes, it will change the color of the pixel that is under the cursor with *grid[row][col] = drawing_color*. If it is not however, return an *IndexError* (or do nothing) but with the for loop, it will additionally check if that mouse cursor is not on the position of one of the rectangles or tools; if it is not, then it will let the program continue, but if yes, it will change the *drawing_color* based on the button color (*drawing_color = button.color*).

- The next if statement in the for loop checks whether the rectangle the cursor is the Clear tool, if yes, it will basically re-call the grid initializing function to reset the canvas and also change the drawing color to black. As a note, the reason why the Erase button is not coded specifically is because it is not actually erasing but instead just drawing with a white drawing color, so it technically acts the same as a drawing color tool.