Creating an LED Canvas App

Creating the app

Each app for the LED Canvas is a python class which is imported into the main python program of the LED Canvas, named “AppHome.py”. This section will assist in the creation of the python class as well as define the required variables.

1. To start, create a new python file to store your app. The standard naming convention is in Pascal case and ending in “App.py”. Example: “MyExampleApp.py”
2. In your new file, import any libraries you wish to use at the top and create your class with the name of your app. Continuing the previous example this should look like

```
class myExampleApp:
```

3. On the next line, create an initializing function of the app. This will run when the board is turned on, and be used to set up any variable your app will use. In this function, the following variables are required:
    a. touch_grid - a 192 array of tuples which corresponds to the LED grid pixels
    b. IS_TIMER_BASED - A boolean which if true will have the main program run a move function each cycle
    c. SPEED - An integer which when the app is timer based determines the number of milliseconds between each run of the move function

In addition to these functions, it is also generally recommended you have a separate setup function, to handle most of the setting of pixels or initial values. This will make resetting the app, in the event of a game over or a user initiated reset, easier to program. An example of this function is below. Note that any variable or function defined in a class should have “self.” when referencing it in python

```
def __init__(self):
    self.touch_grid = [(0, 0, 0)] * 192 # Create a blank array of tuples for the pixels
    self.IS_TIMER_BASED = False # A non-timer based app
    self.SPEED = 0 # While it is not timer based it is still recommended to set SPEED to a value
    self.setup_my_example()
```

4. In addition to the __init__ function, you will also need a paint, web_paint, and move function. The paint function will handle a user input from the physical board and requires an x and y input. The web_paint function will handle a user input from the remote app and requires an index and web_color input. The move function will handle any animation that is to take place if the app is timer based, and does not require any additional inputs.

Examples of these can be seen below. Additionally, the convert function is shown, which helps to convert x and y to the appropriate pixel array index

```
def convert(self, x, y):
        if (x % 2 != 0): # if an in odd numbered column, reverse the order
            y = 15 - y
        return (x * 16) + y

def move(self):
        pass # Since our example is not timer based, this function will not do anything

def web_paint(self, n, web_color):
        x = int(n / 16) # Converts the websites input to an x coordinate on the grid
        y = int(n - x * 16) # Converts the website input to a y coordinate on the grid
        self.touch_grid[self.convert(x, y)] = web_color # Changes the pixel array

def paint(self, x, y):
        self.touch_grid[self.convert(x, y)] = (255, 0 , 0) # Changes the press pixel to red
```

5. With this, your app is now ready to be run on the program. The example code all together is below

```
class myExampleApp:
    def __init__(self):
        self.touch_grid = [(0, 0, 0)] * 192 # Create a blank array of tuples for the pixels
        self.IS_TIMER_BASED = False # A non-timer based app
        self.SPEED = 0 # While it is not timer based it is still recommended to set SPEED to a value
        self.setup_my_example()

    def setup_my_example(self):
        self.touch_grid = [(0, 0, 0)] * 192 # Resets the pixel grid to be blank

    def convert(self, x, y):
        if (x % 2 != 0): # if an in odd numbered column, reverse the order
            y = 15 - y
        return (x * 16) + y

    def move(self):
        pass # Since our example is not timer based, this function will not do anything

    def web_paint(self, n, web_color):
        x = int(n / 16) # Converts the websites input to an x coordinate on the grid
        y = int(n - x * 16) # Converts the website input to a y coordinate on the grid
        self.touch_grid[self.convert(x, y)] = web_color # Changes the pixel array
```

```
def paint(self, x, y):
    self.touch_grid[self.convert(x, y)] = (255, 0 , 0) # Changes the press pixel to red
```

Testing and Implementing Your App

In order to properly test and implement your app, you will first need to save it in the “apps” directory. After doing this you will need to import into either the simulation “simple_sim.py”, if you wish to run it without the physical board or in the RaspiApp “AppHome.py” if you wish to test it with the physical board. This section will walk you through how to do this for each.
For both example follow these instructions first, and then branch off based on your case later:

1. At the top of your intended file (AppHome.py or simple_sim.py), add to the App imports section an import for your app. It should look like the following

```
from apps.MyExampleApp import MyExampleApp
```

2. At the bottom of the file, in the “if __name__ == ‘__main__” section, find the apps map definition and add your app to the end. A very small apps with only the Menu, Painting, and your new app would look like the following

```
apps = {
    'Menu': MenuApp(0),
    'Painting': PaintingApp(),
    ‘MyExampleApp’: MyExampleApp()}
```

\Now, if you are modifying the simulation, there is a MenuApp.py file in the simulation directory. If you are modifying the Raspberry Pi app, you will edit the MenuApp.py file in the “apps” directory.

3. In this class, there is an array called self.app_array. To the end of this, add your apps name as you defined in the map. For the above example, it should look like the following:

```
self.app_array = ['Painting', 'MyExampleApp’]
```

4. In the setup_menu function, it is recommended you set the next pixel to have a color to make it more obvious where the pixel grid needs to be touched to go into that app. By default, valid inputs are from x values 2-9 and y values of 10.
    a. If you wish to add more apps beyond this. The easiest way is to go into the next y value. This will require some more modification of the paint function of

"MenuApp.py". An example of a menu paint function that uses both the 10th and 11th y value is below.

```
def paint(self, x, y):
    if y == 10:
        if (x >= 2 and x <= 9):
            self.next_app = self.app_array[x - 2]
            self.new_app_selected = 1
    if y == 11:
        if (x >= 2 and x <= 3):
            self.next_app = self.app_array[x + 6]
            self.new_app_selected = 1
```

With this your app should be fully implemented and you can test, use, and modify further.