CSCI 134: halloween

The purpose of this lab is to help you practice nested lists, tuples, classes, and mutability! All while maintaining your chops with some of the old stuff like lists, loops, and boolean logic. This one's a spoooooky one! 👻🎃

---

0. Housekeeping (try before Monday/Tuesday)

- Create a folder called `hw07` in your `cs134` folder using the terminal... You know the rest :) The link to download the homework is here, `https://rbhatta8.github.io/cs134-nomekop/hw_files/hw07.zip`
- **Open up a terminal and type the command** `pip install Pillow`
- It's been a week since your capture. The CEO of Evil Corp has you locked up in one of their themed mega prisons — this one's an icy prison in the frozen north
- Waiting in your cell, you count the days to your favorite day of the year, Halloween. You imagine the leaves on the trees turning yellow and orange and red, and same with the leaves on the grass nomekop. You miss time spent with the Professor decorating pumpkins in their lab
- "Miss decorating pumpkins, you say? Well, let's do it then!"
- "Professor?! How'd you get in here.. Are you here to get me out?!" you exclaim. "Oh easy, even corporate mega prisons have visitation rights, so I just booked one for Halloween so we can decorate pumpkins together!" they reply.
- You remain puzzled by the Professor's ability to remain frivolous in such a dire situation, but you sense that they might have a plan. Besides, it *is* Halloween… and what's Halloween without a little light-heartedness and fun!

0a. not your average pixel

- Just like old times, the Professor gives you a quick tutorial on painting before you pick up the real pumpkins and decorations
- Check out the `Pixel` class defined in `halloween.py`
- This class represents an individual pixel drawn on screen. It has 4 attributes

    - `red, green, blue`: integers corresponding to intensity of these colors for the pixel. Each of these attributes is an integer ranging from 0 to 255
    - `alpha`: an integer corresponding to how opaque/transparent the pixel is. A value of 0 corresponds to the pixel being fully transparent (or invisible) and a value of 255 corresponds to it being fully opaque
    - `brightness`: a float calculated as a function of red, green, and blue that corresponds to the "brightness" of the pixel as defined by a standard formula that lines up with human perceptions of brightness (source, if you're curious where this comes from!)
- Now implement the function `average_rgb_of_red_pixels`
- The function takes as input a 2d list of Pixel objects representing an image
    - E.g., img = [[Pixel(120, 1, 100, 0), Pixel(240, 100, 100, 255)],
      [Pixel(120, 120, 1, 120), Pixel(181, 120, 120, 255)]]
      is such a list. Here, the top left pixel `p = img[0][0]` is a Pixel object and `p.red` is 120, `p.green` is 1, `p.blue` is 100, `p.alpha` is 0
- The function should return a **tuple** of 3 elements, where the first element is the average red attribute of all pixels that are more red than green and more red than blue, the second is the average green attribute for the same pixels, and the last is the average blue attribute for these pixels. **NOTE:** since r, g, b values must be integers, you should convert the averages to ints before returning them. Also here's sample code for creating a tuple: `x = (1, 2, 10+5-2)`
- Run the command `python halloween.py pixel` to test your function. It should return the values (180, 73, 106) for the average
    - It's worth checking out the actual test in the `elif len(sys.argv) > 1 and sys.argv[1] == "pixel":` block of code in `halloween.py` to see if you agree with the output

0b. turning red (movie)

- Now implement the function `update_to_fall_colors`. Notice this is a function within the Pixel class (aka a method), so it has the argument `self`!

- This function should operate as follows:
    - If the pixel is more green than both red and blue, it should modify the red, green, and blue attributes based on the color orange as defined in the global `COLORS` dictionary
- Run the command `python halloween.py pixel` to test your function. Add another two tests to the `elif len(sys.argv) > 1 and sys.argv[1] == "pixel"` to ensure your function is working as expected. Note the current definition of orange in the dictionary is simply the color red, but we'll fix that using a reference image of fall leaves in part 1a

0c. painting a canvas

- Now implement the function `create_blank_canvas`. Notice this is just an ordinary function (outside of the class)
- This function takes three arguments:
    - rows: an integer corresponding to number of rows in the image
    - cols: an integer corresponding to number of columns in the image
    - color: a tuple corresponding to the r, g, b values for the color
- The function should return a 2d list corresponding to an image, where every element is a pixel object with the given r, g, b values and an alpha of 255
- Run the command `python halloween.py canvas` to test your function. Add another two tests to the `elif len(sys.argv) > 1 and sys.argv[1] == "canvas"` to ensure your function is working as expected
- "Ah the soothing color of Fall at Williams" you say, reminiscing about your time as a student in the little college tucked away in the Purple Valley of Nomekop land.

1a. leaf peeping!

- "Is it peak Fall season for the trees and nomekop yet?" you ask the Professor. "Yes yes yes! Let's paint a picture so you can see what it's like!" they reply.

- Implement the function `transform_img_to_fall_colors` fall that takes a target image and a reference image. The function should operate as follows
    - It should first compute the average values of the red pixels in the reference image and replace the value of the “orange” key in the COLORS dictionary with this color
    - It should then iterate through all the pixels of the target image, and use the method from 0b to update the pixels to their fall colors
    - The function does not need to return anything—write a one sentence comment at the end of this function explaining why the changes made within this function will still be reflected outside of it
- Run the command `python halloween.py fall` to test your function. Yikes! That’s a pretty sad Fall transformation for Cleaf. That’s not at all how you remember them you tell the Professor!
- Transformation from one color to another color is not that simple it turns out… we also need to maintain the same level of perceived brightness. To achieve this, we need to modify `update_to_fall_colors`!
- In addition to the logic you already have, when producing an update to the r, g, b values to the desired color, you also want to rescale the values based on the original brightness of the image.
    - The scaling factor is just `old_brightness/new_brightness` (where new_brightness is based on the new r, g, b values of the pixel and old_brightness is from the old).
    - **NOTE:** When implementing this new logic, you should ensure that the new scaled values are still integers and lie between 0 to 255. Also remember to watch out for division by zero errors — you can use the global variable SMALL_NUMBER in the denominator to help you overcome that!
    - Last, you should make a final update to the brightness attribute based on the brightness formula to reflect the scaling you just did
- To test your function, re-run the command the command `python halloween.py fall`

- You can also try this command after changing the nomekop in the `elif len(sys.argv) > 1 and sys.argv[1] == "fall":` to “Pluma.png”
- For reference, this is the reference image we use for this function

2a. pumpkin patch

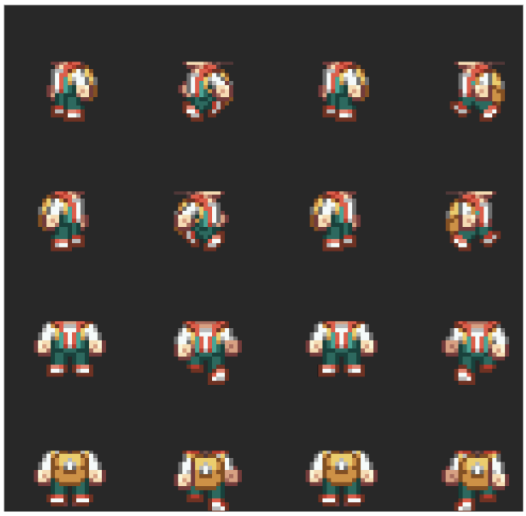
- “Now it’s time for our favorite activity, pumpkin carving and painting!” the Professor says excitedly. Then in a whisper.. “Psst, once you put on the pumpkin head and the prison guards are distracted with all the mayhem of you being the pumpkin king, I’m going to slip something into your pocket”. You nod, understanding the plan.
- Take a look at the `pumpkin_heads.png` file in the characters subfolder of the graphics folder. Each of these pumpkin heads is a 65x65 image, so the total size of the image is 65x260
- Implement the `extract_pumpkin_head` function
    - The first argument of this function is a 2d list consisting of all the pumpkins as you saw in `pumpkin_heads.png`

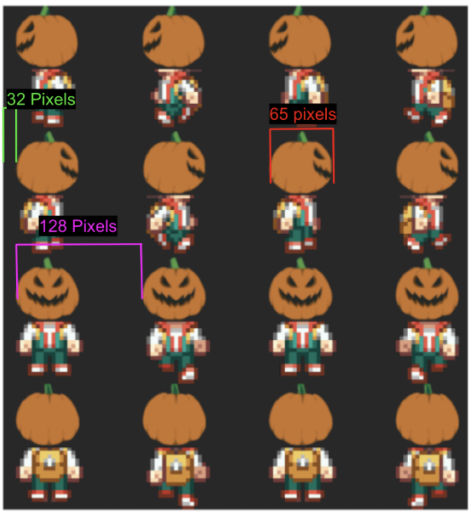
    - The second argument is an integer – 0, 1, 2, or 3
- When the integer is 0, the function should return a 2d list corresponding to the first pumpkin head, if it is 1 it should return the second pumpkin, and so on
- To test your function run `python halloween.py pumpkin`

2b. eraser head (movie) & pumpkin(g) king (movie)

- Now it's time to wear our pumpkins! Implement the function `place_pumpkin_head` to replace all the player heads with pumpkins!
- Here's a very minimal two step pseudocode for this function (text after pics)

In the above: all pixels in rows 0-65 has their alpha value set to 0 (transparent). Then we jump to row 128 and set the next 65 rows to have alpha value 0, then we jump to row 256 and do the same, and the same starting at row 384





- For the top left character pumpkin head, we are replacing all pixels in rows 0 to 65 and columns 32 to 32 + 65 with pixels from the pumpkin image
- For the next left walking image, we are replacing all pixels in rows 0-65 and columns (128+32, 128+32+65), and so on

- STEP 1: First produce an intermediate result as shown by the image above where we set the heads to be transparent. You can run the command `python halloween.py heads` to test this
- STEP2: Now we add in the pumpkins heads as shown above!
- You can do a final test with `python halloween.py heads`
- As a final final test for all your functions, update the PLAYER_FILE and PROFESSOR_FILE in `settings.py`. For the PLAYER_FILE, I recommend one of the characters c1, c5, c6, c7, or c8 — it turns out the pumpkin transformation doesn't work uniformly well across all characters depending on how large their hat/hair is, but it does work well for those ones. It's a really fun computer vision challenge trying to get it to work for all the characters we have, that's our bonus for the week :) ! Once you've updated the file names, run `main.py`
- "Happy Halloween! Look at all the spookies!" the Professor says with a wry smile

---

3. Bonus (optional):

- Experiment with other reference images and see what cool transformations you can get when using the `transform_img_to_fall_colors` function!
- Try to fix the `place_pumpkin_head` function to work correctly for all characters!

---

4. Final checklist

Upload the `halloween.py` file to Gradescope with the following:

- ☐ The functions in 0a, b, c
- ☐ The function in 1a as well as the written answer/comment
- ☐ The functions in 2a, b