CS417: Lab 5

Overview

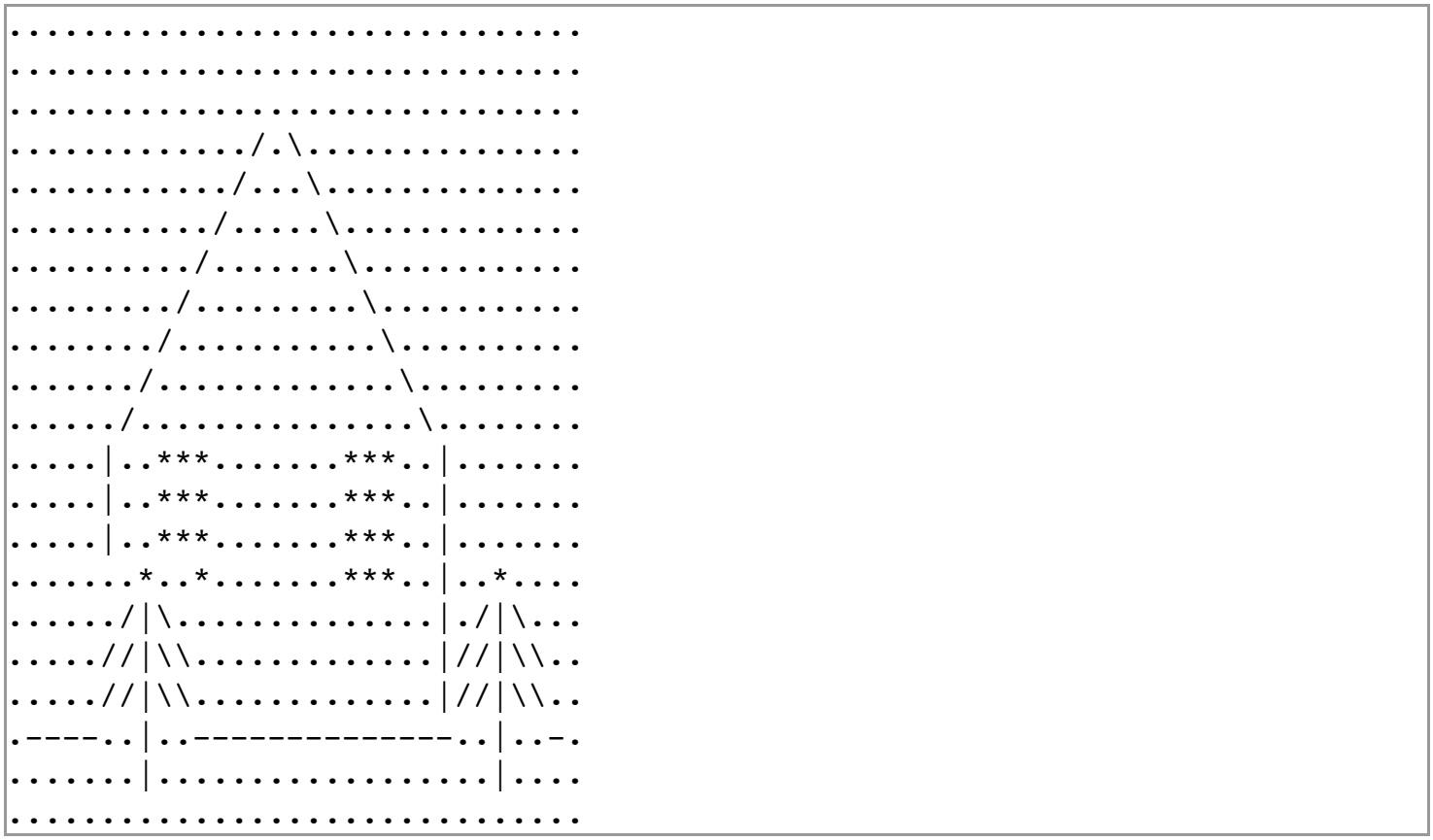
This lab builds your skills in two-dimensional arrays of pixels. You will implement functions for drawing lines, rectangular blocks, and sprites.

Getting Started

Begin by creating a folder for you work. Then, go to `mycourses.unh.edu`, find CS417, click on the `unh.box.com` link, and find Lab06. Then download these files into your folder:

- `graphics.py` This is the program.
- `scene.txt` This is a sequence of drawing instructions, one per line. The `main()` function needs it.

When your program runs correctly, it should produce a picture like this one:



Your Tasks

You will be working with `picture`, which is a two-dimensional array of single letters. Each letter will be a "pixel". Pixels can be accessed by row (i.e. y coordinate) and column (i.e. x coordinate):

```
picture[y][x]
```

or picture[row][column]

Initially, every letter in the array is a dot: `"."` , to help you see where the pixels are.

As `x` increases, we go to the right.

As `y` increases, we go down the page.

1. (5 points) Implement `put_pixel(x, y, symbol, picture)`. This is just a one-liner:

   ```
   picture[y][x] = symbol
   ```

---

2. (5 points) Implement `print_picture(picture)`. You will need a `for` loop that visits each row in the picture (a row is a list of pixels):

   ```
   for row in picture:
   ```

   and you should join the pixels in the row together, and print them:

   ```
       print("".join(row))
   ```

---

3. (15 points) Implement `h_line(x1, x2, y, symbol, picture)`. This draws a horizontal line of pixels. Assume that `x1` will always be ≤ `x2`. You need a `for` loop that goes from `x1` to `x2`:

   ```
   for x in range(x1, x2+1)
   ```

   Call `put_pixel(x, y, symbol, picture)` inside the loop.

---

4. (15 points) Implement `v_line(x, y1, y2, symbol, picture)`. It draws a vertical line. This is similar to `h_line`, but now you are keeping the `x` unchanged, and visiting various values of `y` in a `for` loop. Assume that `y1` will always be ≤ `y2`.

---

5. (15 points) Implement `block(x1, y1, w, h, symbol, picture)`. This fills in a rectangular block of pixels, with top-left corner at `x1 y1`. You need one `for x` loop inside a `for y` loop. `x` ranges from `x1` to `x1+w`, and `y` ranges from `y1` to `y1+h`.

---

6. (15 points) Implement `draw_sprite(x1, y1, sprite, picture)`. This draws a small rectangular image into the `picture`. The `sprite` is a two-dimensional array. It's just like a `block()`, but each pixel may be a different `symbol`.

   First, get the width (`len(sprite[0])`) and height (`len(sprite)`) of the image.

   Then, write a double `for` loop, just like in `block()`.

   `x` and `y` may be large numbers. To access a pixel from the sprite, you need small numbers. The pixel's coordinates are *offsets* from the top-left. So, you should call `put_pixel()`, and pass it this `symbol`:

```
symbol = sprite[y-y1][x-x1]
```

   IMPORTANT: You will need something like this in the `banner.py` assignment!

---

7. (30 points) Implement `draw_line(x1, y1, x2, y2, symbol, picture)`. This draws a line with endpoints `x1 y1` and `x2 y2`. It is the most challenging task. You can certainly try to do this yourself, but it's hard. I suggest you follow the method below:

   - Compute the run: *dx = x2 - x1*
   - Compute the rise: *dy = y2 - y1*
   - Compute the sign of *dx*: if *dx* is positive, *sign_x* is +1, otherwise, it's -1.
   - Compute *sign_y*, the sign of *dy*, the same way.

   To draw the line, you will initialize `x` and `y` to be `x1` and `y1`, and then enter a loop that updates `x` and `y`. For each update, you will add *step* amounts to both `x` and `y`, to advance to the next pixel on the line.

   What are the step amounts? They depend on whether `abs(dx)` or `abs(dy)` is bigger:

   - if `abs(dx)` is bigger, then
     a. *step_x* should be *sign_x*,
     b. *step_y* should be *sign_x* × *dy/dx*, and
     c. num_steps should be `abs(dx)`.
   - if `abs(dy)` is bigger, then
     a. *step_y* should be *sign_y*,
     b. *step_x* should be *sign_y* × *dx/dy*, and
     c. num_steps should `abs(dy)`.

   After setting up `step_x`, `step_y`, and `num_steps`, write a loop:

   for step in range(num_steps):

In the loop, call `put_pixel()`, and then add a step amount to both `x` and to `y`.

Your program will crash, because `x` and `y` are `floats`, and you need `ints`. So call `put_pixel` using `round(x)` and `round(y)`.

Turn Your Work In

When you are finished, go to `mycourses.unh.edu`, find `CS417`, and find `Lab06`. Then click `Submit`, and upload `graphics.py`.