Computer Science Courses

Project 4 - The Dragon Curve

Due date: November 7th, 23:59PM

This is a team project.

The project is worth 100 points.

All the team members will get an equal grade.

ONLY the team leader must turn-in the project.

VERY IMPORTANT!!!

You **MUST** specify the following information at the beginning of the project4 file:

```
# Team leader: Last name, First name, Purdue account
# Team member 1: Last name, First name, Purdue account
# Team member 2: Last name, First name, Purdue account
```

Before submitting the file project4.py, check carefully that the header above is correctly completed:

- If the file project4.py does not contain the header above, **only the student submitting the file will receive the grade**. This will apply also in the case that the team was registered.
- **Only the team members listed in the header of project4.py file will receive the grade.**

Project description

In this project you will use the PIL functions to render a 2D curve encoded by a text string. The text string consists of a sequence of N, S, W, and E characters. N,S,W,E mean respectively:

- go forward in the north direction,
- go forward in the south direction,
- go forward in the west direction
- go forward in the east direction

You can think of the curve as describing the movement of a turtle (or the movement of a car, or the movement of a person). When dealing with movements in the 2D plane, we need to specify the ***initial position*** and the ***initial orientation*** of the moving subject, as well as the ***length*** of the movement.

To simplify the problem, you can assume that the initial orientation of the "turtle" is toward north and parallel to the Y axis of the 2D plane, and that each movement has the same length (10 units). The initial position of the "turtle" will be at the center of a window. The dimensions (width and height) of this window will be given to you in a file.

The curve described by the character string is known as the ***Dragon curve***.

- The Dragon curve is a member of a family of so-called self-similar fractal curves, which can be approximated by recursive methods (you will learn about recursion later in the course).
- As you can easily guess, there is some math involved to understand how they work. If you are interested, you can read

the wiki page here: http://en.wikipedia.org/wiki/Dragon_curve [http://en.wikipedia.org/wiki/Dragon_curve].

- Self-similar fractal curves produce beautiful images. But the striking facts are that i) a number of natural phenomena exhibit fractal features (see http://en.wikipedia.org/wiki/Fractal#Natural_phenomena_with_fractal_features [http://en.wikipedia.org/wiki/Fractal#Natural_phenomena_with_fractal_features]); ii) these phenomena, such as the growth of a plant, can be described by (relatively) simple mathematical models. The Dragon curve, for example, is the graphical representation of the Lindenmayer system (or L-system). A L-system provides a formal description of the development of simple multi-cellular organisms (see http://en.wikipedia.org/wiki/Lindenmayer_system [http://en.wikipedia.org/wiki/Lindenmayer_system]).

For simplicity we have provided a text file that contains a dragon curve as a string of cardinal directions (NSEW) and the dimensions of the window over which you must draw.

Your job will be to take this string as input and draw the corresponding curve in a 2D window.

To draw the segment corresponding to the movement, you have to set the color of the pixels corresponding to the movement to a given color. Your program will ask to the user the color to be used (see TODO 4).

Libraries:

- This project will use the Python libraries, PIL Image, and random
- You will need to use the random library to generate a random integer for the project. This can be done using the following function:
  - random.randint(a,b). This function will return a random integer N such that

```
a <= N <= b
```

SETUP

Create a folder **project4** under your folder CS177.

Download there the file :dragon_inputs.zip. It contains the files for the Test Cases.

filename project4-sk.py

```
# CS177 Fall 2012 - Project4
#
# IMPORTANT: The following information MUST be filled!!!
#
# Team leader: Last name, First name, Purdue account
#
# Team member 1: Last name, First name, Purdue account
#
# Team member 2: Last name, First name, Purdue account
#



#TODO1 write function colorPixel



#TODO2 write function flipAxis



#TODO3 write function checkBounds
```

```
#TODO4 write drawLine function



#TODO5 write main function


        #ask user for file to open


        #convert the color string to (r,g,b) color
```

Input File:

The input file(s) provided to you (dragon_1.txt, dragon_2.txt, and dragon_3.txt) have the following format:

- one or more character strings. Each string consists of a combination of 'N', 'S', 'E', or 'W' and is terminated by the <newline> character.
- the width of image, followed by <newline>
- the height of image followed by <newline>

NOTES

1. <newline> represents the '\n' character which encodes a new line in a text document.
2. the input file may contain the encoding of more than one curve; in that case you have to draw all curves

Look at the files dragon_1.txt, dragon_2.txt and dragon_3.txt, provided to you in the dragon_inputs.zip file, to see how a curve is encoded. Look at the output of the test cases in the section Test Cases of this document to get an idea of the curve that must be drawn.

Movement directions:

- The directions for curve creation are given in the form of cardinal directions (NSEW)
- To make things easier for a novice user (and harder for you!), we want to use a standard coordinate system with the origin (0,0) in the bottom left corner of the image.
- If the program detects the input 'N' then the image should draw a line North(upwards) from the current position

Project details

TODO 1:

Write the function `colorPixel`.

- This function will take in three input arguments, *position*, *color*, and *myImage*, where:
    - `position` is the 2D coordinate of the current position, that is, a couple of integers x and y, where x is the x-axis position, and y is the y-axis position
    - `color` is a triple (r,g,b) containing the color to be used when drawing the pixel
    - `myImage` is the PIL image we are editing

The function must take the pixel at `position` and set its color to the `color` specified as input argument

TODO 2:

Write the function `flipAxis`.

- This function will have two arguments, *position* and *bounds*:
  - `position` is the 2D coordinate of the current position, that is, a couple of integers x and y, where x is the x-axis position, and y is the y-axis position
  - `bounds` is a couple of (maxX,maxY), which is the width and height of the current image.

This function will translate the `position` from that of a standard coordinate system - that is, one where the origin (0,0) is in the bottom left corner -, to one which PIL understands. Since we want the origin (0,0) to be in the bottom left corner of the image and PIL images have the origin in the top left corner, we need to convert between the two.

TODO 3:

When drawing the segment corresponding the encoded movement, it may be possible that the segment exceeds the width/height of the image.

In this case you **do not** generate an error and terminate the drawing.

Rather, you must write a function `checkBounds` that :

1. checks the current position, that is the position of the last pixel of the segment corresponding to the last movement generated.
2. If the position is outside the bounds of the image, it must be reset to the position of the nearest edge.
3. If the position is within the bounds, it will simply return the current position.

This function will take two input arguments, position and bounds:

- `position` is a tuple of (x,y) where x is the x position, and y is the y position of the last pixel of the last movement
- `bounds` is a tuple of (maxX,maxY), which is the width and height of the current image.
- it will return the updated tuple for `position`

TODO 4:

Write the function `drawLine`.

This function will draw a line of length 10 pixels in the direction specified by the `direction` input argument.

- This function will take in four input arguments, *position*, *direction*, *color*, and *myImage*:
  - `position` is the 2D coordinate of the current position, that is, a couple of integers x and y, where x is the x-axis position, and y is the y-axis position
  - `direction` is a single character ('N', 'S', 'E', or 'W') that determines the direction of the line
  - `color` is a triple (r,g,b) which will be the desired color of the line
  - `myImage` is the PIL image we are editing
- This function will return the x,y coordinate of the pixel corresponding to the ending point of the drawn line segment

HINTS

1. Start the line by setting the pixels' color at the current position to the specified color.
2. Move one pixel at a time in the appropriate direction and set the pixels' color to the specified color.
3. Be sure you check the pixel position by calling the `checkBounds` function after each move of the current position.

4. If the required color is "random", you need to create a random color. Remember that a color has three color channels and that each of them is an integer >=0 and < 255 (See TODO 5)

TODO 5:

Write the function `main`. This function has no arguments.

- The function must do the following things:
    1. ask the user for the name of the input file, and store it in a variable.
    2. ask the user for the color, and store it in a variable.
    3. convert the color string to (r,g,b) representation.
    4. open the input file.
    5. read the input data from the file. Its contents were described above under section InputFile.
    6. use the acquired data to generate the dragon curve(s) as necessary.
        a. as you are drawing the curve, after each move save the ending line segment position into a list.
    7. save the image when you are finished. The saved file must have the name "dragon_image_#.jpeg" where # can be incremented if the input file has multiple dragon curves. For example, if the input file contains two curves, you must save the two images you generated in the files named "dragon_image_1.jpeg" and "dragon_image_2.jpeg"
    8. save the movement list to a text file. The file should have the name "dragon_moves_#.txt" where # can be incremented if the input file has multiple dragon curves.

HINTS

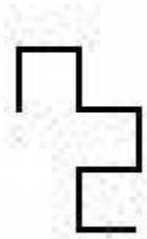
1. Begin by converting the input color to an (r,g,b) representation of the input `color`.
    a. 'red', 'green', 'blue', or 'random' are the only options to the used for `color`
    b. if the color is 'random', use the function random.randint described above to generate a random value for r, g, and b
        I. when the color is 'random' you must generate a new color for each move! This means the output curve will be multi-colored.
    c. if the user does not input a color, or inputs an invalid color, set the color to black
2. When drawing a dragon curve always set the initial position to the center of the blank image.
3. To draw the dragon curve you will need to execute each movement (NSEW). After each movement has been drawn save the updated position tuple in a list. After the curve is completely drawn output that list to the save file.

The prompts should look like the following ones:

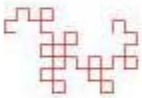
```
Please input file to open:
Please input color (red, green, blue, or random):
Thank you come again
```

Test Cases

1. **TestCase1**
    a. file:dragon_1.txt
    b. color: orange
    c. output: dragon_moves_1.txt

d. 

2. **TestCase2**
    a. file:dragon_2.txt
    b. color: red
    c. output: dragon_moves_2.txt

d. 

3. **TestCase3**
    a. file:dragon_3.txt
    b. color: random
    c. output: dragon_moves_3.txt
        I. Same as TestCase1 but in random color
        II. Same as TestCase2 but in random color

III. 

4. **TestFiles**:dragon_inputs.zip

GRADING RUBRIC (to be revised!)

TODO	Points
TODO1	10
TODO2	10
TODO3	20
TODO4	30
TODO5	30
TOTAL	100

TURNIN INSTRUCTIONS

Login into your UNIX CS account. Then go to your project4 folder using the following commands:

```
$cd CS177
$cd project4
```

Then launch the following command AS IS:

```
$turnin -v -c cs177=COMMON -p project4 *.py
```