Homework 5: Programming II - Robot Simon

Due: October 10, 2014

You must hand in your code on Owlspace. We'll print it for you this time because it is due on Friday. All of your code should be contained in a single python file called PS05_*netid*.py, where *netid* is your netid. Be sure to comment your code so that the graders will know what you are trying to do.

Simon

For this assignment, you will implement the game Simon on your robot. Traditionally, Simon is a device with four colored buttons, each with a particular sound that will play when pressed. It is a memory game where the player must reproduce the flashed lights in order by pushing the colored buttons. In the first round, a single light will flash. The player must then press the button corresponding to the same color. In the second round, the same first light will flash and then a second light will flash. This will continue with the list of lights flashing growing longer and longer until the player presses a wrong button. In the game of Simon that you will create, there are three colors: red, green, and blue. Each color is represented by a ring of LEDs. The button corresponding to each color is in the center of the ring of LEDs. The primary difference between your game and the traditional game of Simon is that your robot has three colors instead of four.

To play Simon and get a feeling for the game, you can go to this link (Just be careful not to play too long. Your game will be much more fun to play once you finish.):

http://www.freegames.ws/games/kidsgames/simon/mysimon.htm

Hints

At the top of your file you'll need to have three import statements:

```
import sys, rone, math
```

The math and list imports will be used in a function provided for you. You will need functions from rone and sys in your code. For example, you will find the `sys.sleep(ms)` function useful. Read the online documentation to determine what it does.

We have provided you with a few functions to simplify the assignment. These functions are shown in this handout for clarity. Download the PS05.py file to get started.

Please remember to save your code now and frequently. Save your file PS05_*netid*.py. It's never fun to get a piece of your code working, and lose it because you forgot to save frequently, and something crashes.

As always, we strongly suggest you do two things:

1. Start the assignment early!
2. Get help from the course staff!

You will be doing this assignment individually. You can still talk to your collaborators and friends, but your work must be your own.

1 Blink a light

One important part to the game of Simon is blinking lights. For Simon, you need to blink a ring of LEDs to represent a color in the game. Think about how you would change how fast the LEDs

blink. What would you need to do? Write a function called `blink` that blinks a light at a certain speed for a given number of times.

```
def blink(color, duration, numblinks):
```

There are three arguments passed into `blink`. The first, `color`, is the color of the LED ring to be blinked. Since you have imported `rone`, you can call functions from it. You can use the following function to turn on a ring of LEDs (or turn them off by setting the intensity to zero):

```
rone.led_set_group(color,intensity)
```

The color can be 'r', 'g', or 'b' and the intensity is a number between 0 and 127 which changes the brightness of the LEDs. We find 40 to be a bright, but not blinding, intensity. We've defined an intensity value LED_BRIGHTNESS at the top of the handout file. The second argument to `blink`, `duration`, is the duration that the LED ring is on and then the duration it is off. The duration will be in milliseconds. So, a duration of 500 milliseconds means to turn the LED ring on for one half second and then turn it off for a half second and repeat the indicated number of times (specified by the third argument, `numblinks`). As this function does not compute anything that has a result, it does not need to return anything.

Hand-in: Write `blink`. (3 pts)

2 Blink a list of lights

Once you have one light blinking, how about blinking a list of lights? Write a function called `blink_list` that uses `blink`. It should take a list of colors and a single duration. It blinks the lights in the list of colors, in order, each with the given duration.

```
def blink_list(colorlist, duration):
```

The function `blink_list` has two arguments, `colorlist` and `duration`. The `colorlist` is a list of colors to be blinked. An example list is `['r','r','b','g']`. In this example, the red LEDs should blink, then the red LEDs should blink again, then blue and finally the green LEDs should blink. The `duration` is the same as it was in the function `blink`: the duration for which the LED ring is on and then off. Again, this function does not compute a result, so it does not need to return anything.

Hand-in: Write `blink_list`. (3 pts)

3 Adding a random color to the colorlist

Next, you need to add a random color to the list of colors. Write a function called `append_random_color`, which takes in a list of colors and returns the same list with one more color appended to it. To use this function, you would write:

```
colorlist = append_random_color(colorlist)
```

Where `colorlist` is a list of colors to be blinked (such as `['g', 'b']`). The function `append_random_color` returns the list of colors passed in with a random color added to the end. This means if you pass in an empty list, `[]`, the function will return either `['r']`, `['g']`, or `['b']`. You set `colorlist` equal to the function call so that when the function returns the new list, it is saved in the variable called `colorlist`. Remember, the syntax to append a variable `x` to a list `l` is: `l.append(x)`. See Lab03c handout for more list examples.

Printing variables is critical to debug your code. In `append_random_color`, we print the `colorlist` each time the function is called.

Hand-in: Write `append_random_color`. (5 pts)

4 Checking the buttons

Next, you need a function that will detect which button is being pushed. This function is called `button_check`. It has no arguments, and returns the color of the button that is pushed. To use this function, write:

```
button = button_check()
```

This function will wait until a button is pushed, and then return 'r','g', or 'b' depending on which button is pushed. The code for this function is included in the PS05.py file, and copied here:

```
def button_check():
    while rone.button_get_value('r') == 0 and \
          rone.button_get_value('g') == 0 and \
          rone.button_get_value('b') == 0:
        pass ## while no button is being pushed, do nothing
    for button in ['r', 'g', 'b']:
        if rone.button_get_value(button) == 1:
            rone.led_set_group(button,10) ## turn on LEDs
            while rone.button_get_value(button) == 1:
                pass ## while button is pushed, do nothing
            rone.led_set_group(button,0) ## turn off LEDs
            ## return string representing the color pushed ('r', 'g', or 'b')
            return button
```

Note that the "\" characters indicate that the Python statement continues on the following line. You could put all three elements of the condition for the while loop onto one line, but it would likely wrap around your screen and be difficult to read.

Given the button that was pushed, you now need to check that it corresponds to the next color in the list. If not, the player has pushed the incorrect button and should lose the game.

5 Checking the list

Write a function called `list_check` which takes one argument, `colorlist`:

```
def list_check(colorlist):
```

This function should run through all of the colors in `colorlist` and make sure they are pushed in the correct order (be sure to use `button_check`). This function should retrun `True` if the player correctly pressed all of the buttons in `colorlist` or `False` if they did not.

Hand-in: Write `list_check`. (6 pts)

6 Putting it all Together: Robot Simon

You now have all of the components to make your robot run Simon! Write a game function that calls the functions you've written and runs until the player loses.

```
def game():
```

You need to set all the variables passed into the functions to a starting value. To continue until the game is over, make a while loop with the condition that `gameinprogress` be equal to `True`.

```
colorlist = []
gameinprogress = True
while gameinprogress:
```

When the player loses (*i.e.*, `list_check` returns `False`), `gameinprogress` should become `False` causing the game loop to end. After the loop exits, you should then clearly indicate that the player has lost. You can turn on all the lights, have them blink in a certain pattern, or whatever you want. Just be sure that it is clear that the player has lost.

Hand-in: Write `game`. (8 pts)

7 Have some fun!

If you are enjoying working with your Simon game, here are a couple of additional suggestions:

You can make your game a little more user friendly by putting a delay between rounds. By waiting a fraction of a second each time through the game loop, the player will not miss the first blink because they just finished pushing the last button.

One way to make Simon more challenging than just having to remember more lights each round, is to increase the rate at which the lights blink. How would you increase how fast the lights blink each time through? Or should it only get faster after each 5 rounds or 10 rounds?

How can you allow the player to start a new game after they have lost without having to reset the robot?

Feel free to add other features you think make the game more fun!

Don't forget to stop playing robot Simon in time to come to class!