STA242 - HW03 - BML Traffic Model attempt in Python

Hugh Crockford

02/16/2013

1 Drawing map

I tried two methods of representing BML model, as discussed in class. Initially I tried representing each cell in the matrix as covered or not with an integer corresponding to color or car, and The issue here was I was unfamiliar with pythons indexing/slicing methods, and was hence unable to select cars to move. (I tried enumerate, logic operations, itertools, but kept running into immutable tuple's that didn't allow me to move cars)

The second attempt was to replicate the method used in H02 using R, where each car's address was contained within a data frame with a code for color. This was then transformed to a matrix with my plot function. Randomly selecting location to place the cars at the beginning of the simulation was also problematic, as I was unable to work out how to select a random subset of positions by index as in R. My solution was to generate a list of all possible locations and then randomise in place (using numpy's shuffle) and select the first row * col * rho number of locations, and divide evenly between car colors.

2 Plotting

Plotting the resulting grid was achieved using scipy's toimage function. Each car cell had to be described in RGB format, which was mapped using integer describing color. One bug in my code I was unable to identify resulted in vertical bars of red cars located along left of plot. This seemed to only occur when generating large maps (larger than 100 squared), and I assume had something to do with my random selection code.

3 Moving cars.

Moving cars was achieved easily by selecting all color based on time, then incrementing their x or y positions respectively. Any cars exceeding the dimensions of map in either dimension was placed back at beginning. I didn't include dimensions with map object as in H02, but perhaps this could be achieved using a similar list function, and remove the need to include map dimensions in call to move function. The move function worked well for smaller maps where I could visually check the results, however because I couldn't get it to function within a loop I was unable to assess it on larger maps.

4 Checking Jams.

Using Python I struggled with the referencing of car locations that would allow me to check jams. As discussed above I tried numerous methods but continued to come up against immutable tuples, a concept I was not familiar with before this project. Another issue that confused me in my initial attempts was the concept of modifying lists/arrays in place versus merely changing a view.

I've since found a few books on basic python which should improve my understanding of the language, and pandas, scikit-learn, and numpy tutorials and books online which will assist my application of machine learning tasks in this language. I hope to complete my project for this course using image recognition techniques (possibly neural nets??) that are present in these packages, assuming I can overcome the steep learning curve.

5 Git

I've been using git since the start of the year to manage projects for classes and research projects I'm involved in. Using ssh keypairs and running git from the shell allows easy version control between multiple computers and is a great tool to collaborate on writing grants/papers etc, especially when using raw text such as latex documents. My github repo can be found here: ¡https://github.com/hughec1329¿.

6 CODE

```
import numpy as np
import scipy.misc.pilutil as smp
data = np.zeros( (1024,1024,3))
data[3,] = [255,0,0]
img = smp.toimage( data )
img.show()
def car(p,n):
        posx = np.repeat(np.arange(n,dtype=int),n)
                # possible x
        posy = np.tile(np.arange(n,dtype=int),n)
        poss = np.vstack((posx,posy))
        poscar = np.arange(n*n)                   # index
            of cats
        np.random.shuffle(poscar)                 #
           randomise (Wasnt working on 2dim)
        ncars = np.round(n*n*p)
        carrs = poss[:,poscar[0:ncars]]           #
           choose random cars
        cols = np.repeat(np.arange(1,3),ncars/2)
        cars = np.vstack((carrs,cols))            #
           colorize
        return cars

def move(cars):
        redcar = cars[:,cars[2,] == 1]
        bluecar = cars[:,cars[2,] == 2]
        carsnew = cars
        #move cars
        carsnew[0,:] = carsnew[0,:] + 1
        # check and thow to start if off edge.
        off = carsnew[0,:] > n
        carsnew[0,off] = 0

        old = ''
        for i in range(len(cars[1,:])):
            old += str(cars[:,i])        # build string
               of old positions.
        block = np.repeat(bool(),len(cars[0,:]))
        for i in range(len(cars[1,:])):
            block[i] =  str(carsnew[:,i]) in y  #
               compare old and new, any conflict marked
                true.

        return block


for i in range(len(t[1,:])):
            y += str(t[:,i])     # build string of old
               positions.

for i in range(len(t[1,:])):
            print str(tnew[:,i]) in y   # compare old
               and new, any conflict marked true.



#       for i range(o,t):
#               if t%2==0:      #move red across

#               else:





def draw(cars,n):
        dat = np.zeros( (n,n,3))
        redcar = cars[:,cars[2,] == 1]
        bluecar = cars[:,cars[2,] == 2]
        dat[redcar[0,:],redcar[1:],] = [255,0,0]
        dat[bluecar[0,:],bluecar[1:],] = [0,0,255]
        img = smp.toimage( dat )
        img.show()
```