Pucong Han
Visual Interfaces to Computer COMS W4735
Professor John Kender
Assignment 3 (Due April 4 2013)

System and Tool Configurations

Operating System: Mac OS 10.7
Programming Language: Python 2.7 (JPEG library + Python Imaging Library + OpenCV)

Python Imaging Library

Installation Process
$ curl -O -L http://effbot.org/downloads/Imaging-1.1.7.tar.gz
$ tar -xzf Imaging-1.1.7.tar.gz cd Imaging-1.1.7
$ python setup.py build
$ sudo python setup.py install

An alternative installing approach using pip:
$ sudo pip install PIL

OpenCV for Python

Installation Process
##Install numpy with Macports
$ sudo port install py27-numpy

##Install OpenCV with Python:
$ sudo port install opencv +python27

##Edit your ~/.bash_profile with:
$ open -t ~/.bash_profile

##Add the line:
export
PYTHONPATH=/opt/local/var/macports/software/opencv/2.2.0_0+python27/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages:$PYTHONPATH

Programming Tools: Sublime Text
Imported Libraries:

```
# This module provides a portable way of using operating system
dependent functionality, such as reading and writing files
import os
# The glob module finds all the pathnames matching a specified
pattern
import glob
#Python OpenCV 2.4.3 library
import cv2
#The Image module provides a number of factory functions,
including functions to load images from files, and to create new
images.
from PIL import Image
#NumPy Array library
```

```
import numpy as np
#Terminal colored text library
from termcolor import colored
#Python math library
import math
```

Step 1: Basic Infrastructure and Building Features and Descriptions

In the first step, I implemented a Python application, back_end_vision_processing.py, to describe each building in terms of a vocabulary of shapes, including geometric figures, sizes, orientations and extremes.

My app paired the building names stored in the ass3-table.txt and the integer values stored in the integer-valued image ass3-labeled.pgm using a hash table. My app first loaded the building name from the ass3-table.txt file and saved into a hash table by its integer value:

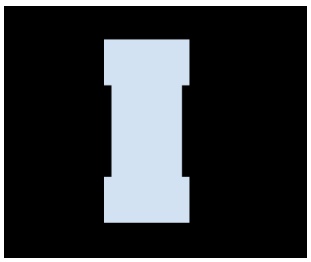
```
#Create a hashtable, in Python it is called 'dictionaries' or
'associative arrays,' associates the building integers with the
building names (building names and integers are from ass3-table.txt
file).
label = {}
label[1]="Pupin"
label[2]="Schapiro CEPSR"
label[3]="Mudd, Engineering Terrace, Fairchild & Computer Science"
label[4]="Physical Fitness Center"
label[5]="Gymnasium & Uris"
label[6]="Schermerhorn"
label[7]="Chandler & Havemeyer"
label[8]="Computer Center"
label[9]="Avery"
label[10]="Fayerweather"
label[11]="Mathematics"
label[12]="Low Library"
label[13]="St. Paul's Chapel"
label[14]="Earl Hall"
label[15]="Lewisohn"
label[16]="Philosophy"
label[17]="Buell & Maison Francaise"
label[18]="Alma Mater"
label[19]="Dodge"
label[20]="Kent"
label[21]="College Walk"
label[22]="Journalism & Furnald"
label[23]="Hamilton, Hartley, Wallach & John Jay"
label[24]="Lion's Court"
label[25]="Lerner Hall"
label[26]="Butler Library"
label[27]="Carman"
```

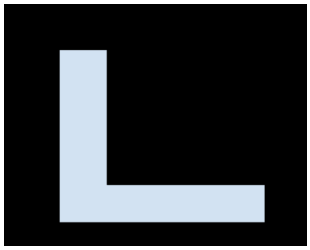
This hash table allows my application to quickly find the name of the building using a filtered integer value from the integer-value map.

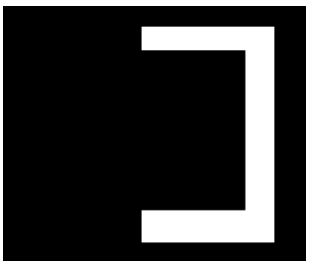
My application loaded six default shapes using the following code:

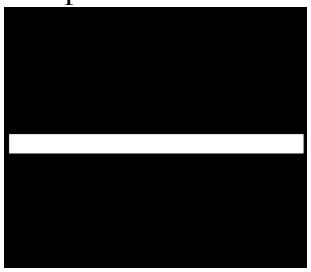
```
#Create a list storing the contours of default shapes.
shapes = []
for shape_image in glob.glob( os.path.join("shapes", '*.png') ):
      shape = cv2.imread(shape_image)
      shape_gray = cv2.cvtColor(shape,cv2.COLOR_BGR2GRAY)
      ret_shape, thresh_shape = cv2.threshold(shape_gray,127,255,0)
      contours_shape, hierarchy_shape =
      cv2.findContours(thresh_shape,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMP
      LE)
      shapes.append([contours_shape[0], shape_image.replace("shapes/",
      "").replace(".png", "").replace("_", " ")])
```

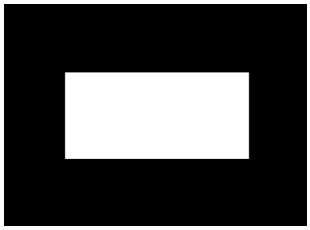
Here is the six default shape images matching and describing the building shapes stored in the map:

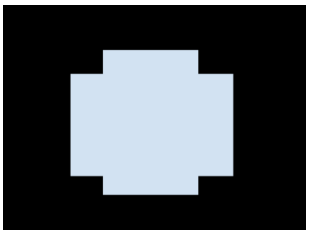

I shape


L shape


Numpy


Oriented from east to west


rectangular


symmetric

This six default shape images allow my application to acknowledge the shape of the buildings in the map. By comparing the contour of the default shape image and the contour of the buildings, my application can find the best-matched shape and add to the description list.

Using binary images improves the accuracy of analyzing the body part in the shape images. In order to get high quality binary images of the hand gestures, I take advantage of one useful function in the OpenCV library called threshold(). This function is available in Python.

The threshold() function provides the most common way to segment a region of an image. According to the OpenCV documentation, the threshold() function applies fixed-level thresholding to a single-channel array. The function is typically used to get a binary image out of a gray scale image or for removing a noise, that is, filtering out pixels with too small or too large values.

My application loaded all image files from the shape folder and open them using OpenCV:

```
for shape_image in glob.glob( os.path.join("shapes", '*.png') ):
      shape = cv2.imread(shape_image)
```

Since the threshold() function requires gray scale image input, we need to transfer the shape image to gray scale:

```
shape_gray = cv2.cvtColor(shape,cv2.COLOR_BGR2GRAY)
```

Then, we call the threshold() function to compute the binary image and pass the gray scale images:

```
ret_shape, thresh_shape = cv2.threshold(shape_gray,127,255,0)
```

Since I have produced binary images for the six shape image, my program directly calls the findContours() function and passes the binary images to the parameter of the function:

```
cv2.findContours(thresh_shape,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

Each contour is stored as a vector of points. The output contour is a list of these boundary points. According to the OpenCV documentation, hierarchy variable is an optional output vector, containing information about the image topology.

My application stored the names and the shape contours in a list:

```
shapes.append([contours_shape[0], shape_image.replace("shapes/",
"").replace(".png", "").replace("_", " ")])
```

My application uses this list when finding the closest description of buildings' shape.

My application then opened the integer value campus image using the Python PIL library:

```
#Open the integer value campus image
integer_value_campus = Image.open("ass3-labeled.pgm")
```

The Python PIL library provides a simple way to extract the pixel value using getpixel() function.

My application then opened the campus image using the Python OpenCV library and computed the contour for each building object in the campus image:

```
#Open the campus image
campus = cv2.imread("ass3-campus.pgm")
#Convert campus image to gray scale.
campus_gray = cv2.cvtColor(campus,cv2.COLOR_BGR2GRAY)
#Create binary images for the campus.
ret, thresh = cv2.threshold(campus_gray,127,255,0)
#Compute contours (boundaries) of buildings on campus.
```

```
contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

To draw boundaries on the images, my application used the drawContours() function. According to the OpenCV documentation, this function draws contour outlines in the image using the returned contours variable. My application using this function to test the accuracy of the contours' computation:

```
#[TESTING] Drawing building contours
cv2.drawContours(campus,contours,-1,(0,255,0),3)
```

Here is the resulted contour image after drawing the boundaries:

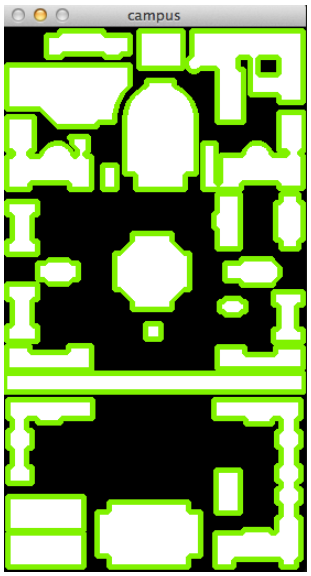


Figure 1: Contours of all buildings.

My application uses a for-loop to go through each contour and process further computation:

```
#For each building contour, this for-loop assigns proper descriptions
(shapes, sizes and location)
for h,cnt in enumerate(contours):
```

Inside the for-loop, my application computes the center of the contour, the upper left pixel of the minimum bounding box and the lower right pixel of the minimum bounding box. My application also extracts the average integer values stored in the integer-valued campus image. Using the pre-computed hash table, my application pairs the name of the building to the building contour:

```
    #Get the integer value from the integer value campus image.
    count = 0
    xValue = 0
    yValue = 0
    integer_value = 0
    for demensions in cnt:
```

```
            for values in demensions:
                  integer_value +=
                  integer_value_campus.getpixel((int(values[0]),
                  int(values[1])))
                  xValue += values[0]
                  yValue += values[1]
                  count += 1
      #Compute the center of the contours.
      xCenter = xValue / count
      yCenter = yValue / count

      #Store the center of the contour into the building_contour_center
      variable.
      building_contour_center = "(" + str(xCenter) + ", " + str(yCenter)
      + ")"

      x,y,w,h = cv2.boundingRect(cnt)

      #Store the contour upper left pixel value into the
      building_contour_upper_left variable.
      building_contour_upper_left = "(" + str(x) + ", " + str(y) + ")"
      #Store the contour lower right pixel value into the
      building_contour_lower_right variable.
      building_contour_lower_right = "(" + str(x+w) + ", " + str(y+h) +
      ")"

      #Get the name of the building by passing the variable to the hash
      table label.
      building_name = label[integer_value / count]
```

To test the computations, I implemented a testing method to print out these properties:

```
dyn-209-2-225-214:assignment3 puconghan$ python back_end_vision_processing.py
Upper left: (4, 459)
Lower right: (74, 491)
Carman
Upper left: (85, 431)
Lower right: (180, 491)
Butler Library
Upper left: (4, 426)
Lower right: (74, 458)
Lerner Hall
Upper left: (193, 402)
Lower right: (216, 442)
Lion's Court
Upper left: (191, 338)
Lower right: (271, 491)
Hamilton, Hartley, Wallach & John Jay
Upper left: (4, 338)
Lower right: (82, 415)
Journalism & Furnald
Upper left: (1, 314)
Lower right: (274, 332)
College Walk
Upper left: (194, 290)
Lower right: (273, 311)
Kent
Upper left: (3, 289)
Lower right: (81, 312)
Dodge
Upper left: (129, 269)
Lower right: (144, 284)
Alma Mater
Upper left: (196, 246)
Lower right: (221, 262)
Buell & Maison Francaise
Upper left: (245, 240)
Lower right: (273, 287)
Philosophy
Upper left: (3, 233)
Lower right: (32, 286)
Lewisohn
Upper left: (31, 211)
Lower right: (69, 234)
Earl Hall
Upper left: (201, 210)
Lower right: (252, 235)
St. Paul's Chapel
Upper left: (101, 187)
```

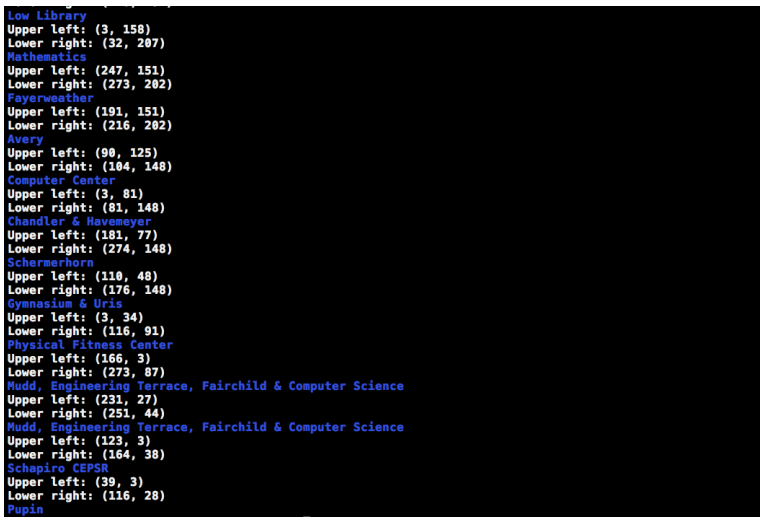


Figure 2: Testing results for contour properties (name of the building and pixel values).

If you closely look at the campus map, the printed building names and pixel values in Figure 2 are fairly accurate.

In the next step, I designed the extremes of the building location (Figure 3).

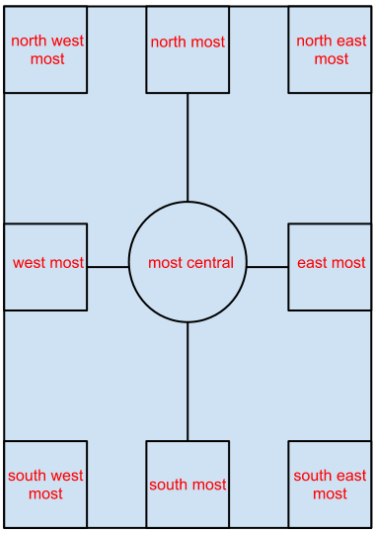


Figure 3: extremes of the building location.

The areas of rectangular extreme regions are 40 pixels by 40 pixels. The radius of the circular center is 40 pixels. My application computed the extremes inside the for-loop using the following codes:

```
if 117.5 <= xCenter >= 157.5 and yCenter <= 40:
        building_location_extrema_description = "north most"
if 117.5 <= xCenter >= 157.5 and yCenter >= 455:
        building_location_extrema_description = "south most"
if xCenter <= 40 and 227.5 <= yCenter <= 267.5:
```

```
        building_location_extrema_description = "west most"
    if xCenter >= 235 and 227.5 <= yCenter <= 267.5:
        building_location_extrema_description = "east most"
    if xCenter <= 40 and yCenter <= 40:
        building_location_extrema_description = "north west most"
    if xCenter >= 235 and yCenter <= 40:
        building_location_extrema_description = "north east most"
    if xCenter <= 40 and yCenter >= 455:
        building_location_extrema_description = "south west most"
    if xCenter >= 235 and yCenter >= 455:
        building_location_extrema_description = "south east most"
    if math.sqrt((xCenter - 137.5)*(xCenter - 137.5) + (yCenter - 247.5)*(yCenter -
    247.5)) <= 40:
        building_location_extrema_description = "most central"
```

These if-statements check whether the center of the building contours is fall into the bounded regions in Figure 3. Here are the printed results for the extremes:

```
dyn-209-2-225-214:assignment3 puconghan$ python back_end_vision_processing.py
Carman
south west most
Butler Library
Lerner Hall
Lion's Court
Hamilton, Hartley, Wallach & John Jay
Journalism & Furnald
College Walk
Kent
Dodge
Alma Mater
most central
Buell & Maison Francaise
Philosophy
east most
Lewisohn
west most
Earl Hall
St. Paul's Chapel
Low Library
most central
Mathematics
Fayerweather
Avery
Computer Center
Chandler & Havemeyer
Schermerhorn
Gymnasium & Uris
Physical Fitness Center
Mudd, Engineering Terrace, Fairchild & Computer Science
north most
Mudd, Engineering Terrace, Fairchild & Computer Science
north most
north east most
Schapiro CEPSR
Pupin
```

Figure 4: Testing results for extremes

If you closely look at the campus map, the testing results for the extremes in Figure 4 are fairly accurate. All building locates in extreme locations are captured.

After this section, my application computes the area of the contours using the OpenCV contourArea() function inside the for-loop using the following codes:

```
#Compute the area of the contour
building_contour_area = cv2.contourArea(cnt)
```

According to the computed area, my application describes the size of the building using the following code:

```
#Compute the size of the countour area
if cv2.contourArea(cnt) <= 500:
    building_size_description = "tiny size"
```

```
if 500 < cv2.contourArea(cnt) <= 1200:
      building_size_description = "small size"
if 1200 < cv2.contourArea(cnt) <= 2500:
      building_size_description = "average size"
if cv2.contourArea(cnt) > 2500:
      building_size_description = "large size"
```

My application classifies the size of buildings into four categories including tiny size, small size, average size and large size. My application then compares the contour with the default contour shapes in the list and match the shape descriptions using the matchShapes() function provided by Python OpenCV. Computing the matching-value of the building contour with each shape contour in the list, my application finds the best match (shape contour with the largest value) and stores them in the shape_value variable:

```
#Compare with the coutours in the shape list and match shape
description
shape_name = shapes[0][1]
shape_value = cv2.matchShapes(shapes[0][0], cnt,
cv2.cv.CV_CONTOURS_MATCH_I1, 0)
for item in shapes:
      if shape_value > cv2.matchShapes(item[0], cnt,
      cv2.cv.CV_CONTOURS_MATCH_I1, 0):
            shape_value = cv2.matchShapes(item[0], cnt,
            cv2.cv.CV_CONTOURS_MATCH_I1, 0)
            shape_name = item[1]
building_shape = shape_name
```

My application prints the results of basic infrastructure, building features and descriptions using the following code:

```
print colored(building_name, 'red')
print colored("Building contour center is: " +
building_contour_center, 'green')
print colored("Building contour area is: " +
str(building_contour_area), 'green')
print colored("Building contour upper left is: " +
building_contour_upper_left, 'green')
print colored("Building contour lower right is: " +
building_contour_lower_right, 'green')
print colored("Building shape is: " + building_shape, 'blue')
print colored("Building size description: " +
building_size_description, 'blue')
if building_location_extrema_description is not "":
      print colored("Building shape description (extrema): " +
      building_location_extrema_description, 'blue')
print ""
```

Here are my testing results:

```
dyn-207-10-137-13:assignment3 puconghan$ python describing_compact_spatial_relations.py
Carman
Building contour center is: (38, 474)
Building contour area is: 2139.0
Building contour upper left is: (4, 459)
Building contour lower right is: (74, 491)
Building shape is: rectangular
Building size description: average size
Building shape description (extrema): south west most

Butler Library
Building contour center is: (132, 460)
Building contour area is: 5130.0
Building contour upper left is: (85, 431)
Building contour lower right is: (180, 491)
Building shape is: symmetric
Building size description: large size

Lerner Hall
Building contour center is: (38, 441)
Building contour area is: 2139.0
Building contour upper left is: (4, 426)
Building contour lower right is: (74, 458)
Building shape is: rectangular
Building size description: average size

Lion's Court
Building contour center is: (204, 421)
Building contour area is: 858.0
Building contour upper left is: (193, 402)
Building contour lower right is: (216, 442)
Building shape is: rectangular
Building size description: small size

Hamilton, Hartley, Wallach & John Jay
Building contour center is: (249, 414)
Building contour area is: 5532.0
Building contour upper left is: (191, 338)
Building contour lower right is: (271, 491)
Building shape is: numpy
Building size description: large size

Journalism & Furnald
Building contour center is: (22, 372)
Building contour area is: 2444.0
Building contour upper left is: (4, 338)
Building contour lower right is: (82, 415)
Building shape is: L shape
Building size description: average size

College Walk
Building contour center is: (137, 322)
Building contour area is: 4624.0
Building contour upper left is: (1, 314)
Building contour lower right is: (274, 332)
Building shape is: oriented east to west
Building size description: large size

Kent
Building contour center is: (233, 296)
Building contour area is: 1365.0
Building contour upper left is: (194, 290)
Building contour lower right is: (273, 311)
Building shape is: L shape
Building size description: average size

Dodge
Building contour center is: (41, 295)
Building contour area is: 1485.0
Building contour upper left is: (3, 289)
Building contour lower right is: (81, 312)
Building shape is: I shape
Building size description: average size

Alma Mater
Building contour center is: (136, 276)
Building contour area is: 196.0
Building contour upper left is: (129, 269)
Building contour lower right is: (144, 284)
Building shape is: symmetric
Building size description: tiny size
Building shape description (extrema): most central

Buell & Maison Francaise
Building contour center is: (208, 253)
Building contour area is: 302.0
Building contour upper left is: (196, 246)
Building contour lower right is: (221, 262)
Building shape is: symmetric
Building size description: tiny size

Philosophy
Building contour center is: (258, 263)
Building contour area is: 1002.0
```

```
Building contour upper left is: (245, 240)
Building contour lower right is: (273, 287)
Building shape is: I shape
Building size description: small size
Building shape description (extrema): east most

Lewisohn
Building contour center is: (17, 259)
Building contour area is: 1218.0
Building contour upper left is: (3, 233)
Building contour lower right is: (32, 286)
Building shape is: I shape
Building size description: average size
Building shape description (extrema): west most

Earl Hall
Building contour center is: (49, 221)
Building contour area is: 706.0
Building contour upper left is: (31, 211)
Building contour lower right is: (69, 234)
Building shape is: rectangular
Building size description: small size

St. Paul's Chapel
Building contour center is: (226, 221)
Building contour area is: 1019.5
Building contour upper left is: (201, 210)
Building contour lower right is: (252, 235)
Building shape is: rectangular
Building size description: small size

Low Library
Building contour center is: (135, 221)
Building contour area is: 3782.0
Building contour upper left is: (101, 187)
Building contour lower right is: (170, 257)
Building shape is: symmetric
Building size description: large size
Building shape description (extrema): most central

Mathematics
Building contour center is: (17, 182)
Building contour area is: 1106.0
Building contour upper left is: (3, 158)
Building contour lower right is: (32, 207)
Building shape is: I shape
Building size description: small size
Fayerweather
Building contour center is: (259, 176)
Building contour area is: 1108.0
Building contour upper left is: (247, 151)
Building contour lower right is: (273, 202)
Building shape is: rectangular
Building size description: small size

Avery
Building contour center is: (197, 175)
Building contour area is: 1090.0
Building contour upper left is: (191, 151)
Building contour lower right is: (216, 202)
Building shape is: rectangular
Building size description: small size

Computer Center
Building contour center is: (96, 136)
Building contour area is: 286.0
Building contour upper left is: (90, 125)
Building contour lower right is: (104, 148)
Building shape is: rectangular
Building size description: tiny size

Chandler & Havemeyer
Building contour center is: (45, 114)
Building contour area is: 3429.0
Building contour upper left is: (3, 81)
Building contour lower right is: (81, 148)
Building shape is: rectangular
Building size description: large size

Schermerhorn
Building contour center is: (234, 118)
Building contour area is: 3696.0
Building contour upper left is: (181, 77)
Building contour lower right is: (274, 148)
Building shape is: I shape
Building size description: large size

Gymnasium & Uris
Building contour center is: (143, 77)
Building contour area is: 5606.0
Building contour upper left is: (110, 48)
Building contour lower right is: (176, 148)
Building shape is: symmetric
Building size description: large size
```

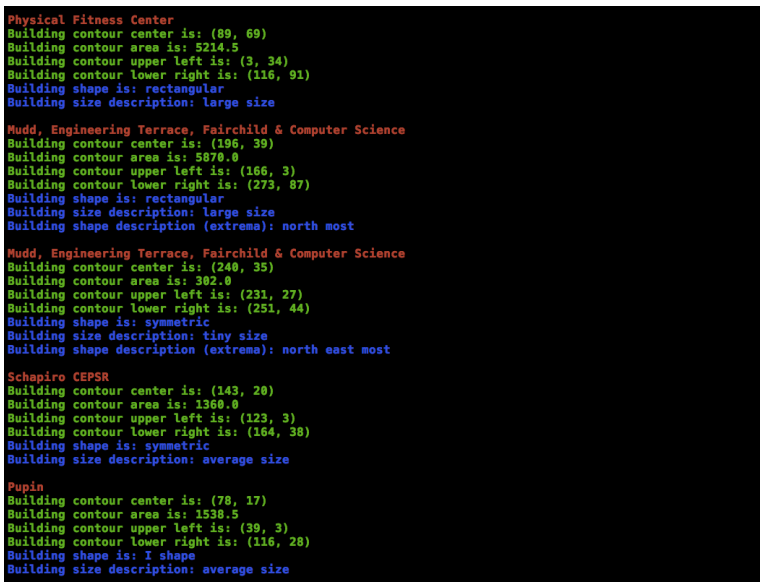


Figure 5: Results of basic infrastructure and building features and descriptions.

As we can see from Figure 5, the results include the building name, the (x, y) of the center of mass, the area, and the upper left and lower right coordinates of its minimum bounding rectangle, the description of its shapes, and size description. If the building locates in the extreme locations of the map, the results will include the extreme descriptions.

Step 2: Describing Compact Spatial Relations

I implemented step 2 in Python application describing_compact_spatial_relations.py. I kept most of my codes from step 1. In my new program, I made the following changes to make my application easier to retrieve coordinates for center, upper left corner and lower right corner.

Instead of storing coordinates for contour center, contour upper left corner and contour lower right corner as strings, my new application describing_compact_spatial_relations.py stores these three pairs of coordinates in a list:

```
building_contour_center = [xCenter, yCenter]
building_contour_upper_left = [x, y]
building_contour_lower_right = [x+w, y+h]
```

My new application stores building names, coordinates of the center of contour, the area of contour, coordinates of the upper left corner of the minimum bounding rectangle of the contour and coordinates of the lower right corner of the minimum bounding rectangle of the contour, the description of shapes, and size description into a list called building_properties using the following codes:

```
building_properties.append([building_name,
building_contour_center, building_contour_area,
```

```
building_contour_upper_left, building_contour_lower_right,
building_shape, building_size_description,
building_location_extrema_description])


for item in building_properties:
      print item
```

The above printing for-loop prints the following results:

```
dyn-207-10-137-13:assignment3 puconghan$ python describing_compact_spatial_relations.py
['Carman', [38, 474], 2139.0, [4, 459], [74, 491], 'rectangular', 'average size', 'south west most']
['Butler Library', [132, 460], 5130.0, [85, 431], [180, 491], 'symmetric', 'large size', '']
['Lerner Hall', [38, 441], 2139.0, [4, 426], [74, 458], 'rectangular', 'average size', '']
["Lion's Court", [204, 421], 858.0, [193, 402], [216, 442], 'rectangular', 'small size', '']
['Hamilton, Hartley, Wallach & John Jay', [249, 414], 5532.0, [191, 338], [271, 491], 'numpy', 'large size', '']
['Journalism & Furnald', [22, 372], 2444.0, [4, 338], [82, 415], 'L shape', 'average size', '']
['College Walk', [137, 322], 4624.0, [1, 314], [274, 332], 'oriented east to west', 'large size', '']
['Kent', [233, 296], 1365.0, [194, 290], [273, 311], 'L shape', 'average size', '']
['Dodge', [41, 295], 1485.0, [3, 289], [81, 312], 'I shape', 'average size', '']
['Alma Mater', [136, 276], 196.0, [129, 269], [144, 284], 'symmetric', 'tiny size', 'most central']
['Buell & Maison Francaise', [208, 253], 302.0, [196, 246], [221, 262], 'symmetric', 'tiny size', '']
['Philosophy', [258, 263], 1002.0, [245, 240], [273, 287], 'I shape', 'small size', 'east most']
['Lewisohn', [17, 259], 1218.0, [3, 233], [32, 286], 'I shape', 'average size', 'west most']
['Earl Hall', [49, 221], 706.0, [31, 211], [69, 234], 'rectangular', 'small size', '']
["St. Paul's Chapel", [226, 221], 1019.5, [201, 210], [252, 235], 'rectangular', 'small size', '']
['Low Library', [135, 221], 3782.0, [101, 187], [170, 257], 'symmetric', 'large size', 'most central']
['Mathematics', [17, 182], 1106.0, [3, 158], [32, 207], 'I shape', 'small size', '']
['Fayerweather', [259, 176], 1108.0, [247, 151], [273, 202], 'rectangular', 'small size', '']
['Avery', [197, 175], 1090.0, [191, 151], [216, 202], 'rectangular', 'small size', '']
['Computer Center', [96, 136], 286.0, [90, 125], [104, 148], 'rectangular', 'tiny size', '']
['Chandler & Havemeyer', [45, 114], 3429.0, [3, 81], [81, 148], 'rectangular', 'large size', '']
['Schermerhorn', [234, 118], 3696.0, [181, 77], [274, 148], 'I shape', 'large size', '']
['Gymnasium & Uris', [143, 77], 5606.0, [110, 48], [176, 148], 'symmetric', 'large size', '']
['Physical Fitness Center', [89, 69], 5214.5, [3, 34], [116, 91], 'rectangular', 'large size', '']
['Mudd, Engineering Terrace, Fairchild & Computer Science', [196, 39], 5870.0, [166, 3], [273, 87], 'rectangular', 'large size', 'north most']
['Mudd, Engineering Terrace, Fairchild & Computer Science', [240, 35], 302.0, [231, 27], [251, 44], 'symmetric', 'tiny size', 'north east most'
]
['Schapiro CEPSR', [143, 20], 1360.0, [123, 3], [164, 38], 'symmetric', 'average size', '']
['Pupin', [78, 17], 1538.5, [39, 3], [116, 28], 'I shape', 'average size', '']
```

Figure 6: Printed results for building properties.

The advantage of building this building_properties list is to allow my application retrieves all needed information using one for-loop.

After this step, my application computes the compacted spatial relations. Before implementing the computations, I designed a definition model for defining the spatial relations.

Here is my design for defining the spatial relations:

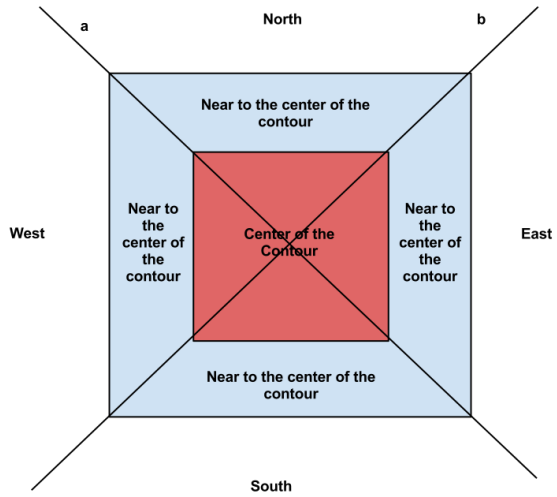


Figure 7: Design for the spatial relations.

In Figure 7, the red square is the contour of the targeted building. In my design, any object center falls into the upper triangular region is considered to the north of the contour. If the distance between the lower y value of the object contour and the upper y value of the target contour is less than 6, the two contours are close to each other (within the blue region).

In my design, any object center falls into the lower triangular region is considered to the south of the contour. If the distance between the lower y value of the target contour and the upper y value of the object contour is less than 6, the two contours are close to each other (within the blue region).

In my design, any object center falls into the left triangular region is considered to the west of the contour. If the distance between the right x value of the object contour and the left x value of the target contour is less than 6, the two contours are close to each other (within the blue region).

In my design, any object center falls into the right triangular region is considered to the east of the contour. If the distance between the right x value of the target contour and the left x value of the object contour is less than 6, the two contours are close to each other (within the blue region).

The threshold value 6 (for determining nearest building) needs to be adjusted according to the area of the targeted contour. The function for adjusting the threshold value is 6 * contour area / 2500. Note that the median size of the contour is 2500. As a result, threshold values for contours with larger area will be multiplied up by the factor (contour area / 2500) and threshold values for contours with smaller area will be multiplied down by the factor (contour area / 2500).

This designed model is one of the best models because it captures spatial relations of all surrounded building without ambiguity. Spatial relations of all buildings surrounded the target building are describable.

Before calculating the relations between contours, I implemented a helper function isLeft() to check whether a point is to the left of a line through two other points:

```
def isLeft(a, b, c):
      if ((b[0] - a[0])*(c[1] - a[1]) - (b[1] - a[1])*(c[0] -
      a[0])) > 0:
            return True
      else:
            return False
```

The isLeft() function is inspired by the article “Find which side of a line a point is on” available on wiki (http://wiki.processing.org/w/Find_which_side_of_a_line_a_point_is_on). According to the article, the functions (b[0] - a[0])*(c[1] - a[1]) - (b[1] - a[1])*(c[0] - a[0]) returns a negative value if the point is "to the left" of the line, zero if the point is on the line and a

positive value if it's on "the right". My isLeft() function will return true if the point (c) is at the left of the line through points (a) and (b) or return false if the point (c) is at the right of the line through points (a) and (b).

This function allows my program to compute the relation (north, south, west and east) of the center of the testing building contour and the targeted contour.

Here are my implementations to compute the relations among building contours:

```
for building in building_properties:
      for building_sub in building_properties:
            if building[0] != building_sub[0]:
                  if (isLeft(building[1], building[3], building_sub[1])
                  == False) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == True):
                        building_relations[building[0],  building_sub[0],
                        "relation"] = "north"
                        if math.sqrt((building_sub[3][1] -
                        building[4][1])*(building_sub[3][1] -
                        building[4][1])) <= 6 * building[2] / 2500:
                              building_relations[building[0],
                              building_sub[0], "near"] = "near"
                        else:
                              building_relations[building[0],
                              building_sub[0], "near"] =
                              math.sqrt((building_sub[3][1] -
                              building[4][1])*(building_sub[3][1] -
                              building[4][1]))
                  if (isLeft(building[1], building[3], building_sub[1])
                  == True) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == False):
                        building_relations[building[0],  building_sub[0],
                        "relation"] = "south"
                        if math.sqrt((building[3][1] -
                        building_sub[4][1])*(building[3][1] -
                        building_sub[4][1])) <= 6 * building[2] / 2500:
                              building_relations[building[0],
                              building_sub[0], "near"] = "near"
                        else:
                              building_relations[building[0],
                              building_sub[0], "near"] =
                              math.sqrt((building_sub[3][1] -
                              building[4][1])*(building_sub[3][1] -
                              building[4][1]))
                  if (isLeft(building[1], building[3], building_sub[1])
                  == True) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == True):
                        building_relations[building[0],  building_sub[0],
                        "relation"] = "west"
                        if math.sqrt((building_sub[3][0] -
                        building[4][0])*(building_sub[3][0] -
                        building[4][0])) <= 6 * building[2] / 2500:
                              building_relations[building[0],
                              building_sub[0], "near"] = "near"
                        else:
```

```
                        building_relations[building[0],
                        building_sub[0], "near"] =
                        math.sqrt((building_sub[3][1] -
                        building[4][1])*(building_sub[3][1] -
                        building[4][1]))
            if (isLeft(building[1], building[3], building_sub[1])
            == False) and (isLeft(building[1], [building[4][0],
            building[3][1]], building_sub[1]) == False):
                  building_relations[building[0],  building_sub[0],
                  "relation"] = "east"
                  if math.sqrt((building[3][0] -
                  building_sub[4][0])*(building[3][0] -
                  building_sub[4][0])) <= 6 * building[2] / 2500:
                        building_relations[building[0],
                        building_sub[0], "near"] = "near"
                  else:
                        building_relations[building[0],
                        building_sub[0], "near"] =
                        math.sqrt((building_sub[3][1] -
                        building[4][1])*(building_sub[3][1] -
                        building[4][1]))
```

The two for-loops will compare the location of every building to each of the rest buildings. The if-statement with expression building[0] != building_sub[0] will make sure that buildings do not compare to themselves.

If a point is to the right of the line (a) as shown in Figure 7 and to the left of the line (b) as shown in Figure 7, the point is to the north of the contour. If the distance between the y value of the bottom right corner of the testing gesture and the y value of the upper left corner of the targeted gesture is less than 6 * contour area / 2500, the testing gesture is near to the targeted gesture.

If a point is to the left of the line (a) as shown in Figure 7 and to the right of the line (b) as shown in Figure 7, the point is to the south of the contour. If the distance between the y value of the upper left corner of the testing gesture and the y value of the bottom right corner of the targeted gesture is less than 6 * contour area / 2500, the testing gesture is near to the targeted gesture.

If a point is to the left of the line (a) as shown in Figure 7 and to the left of the line (b) as shown in Figure 7, the point is to the west of the contour. If the distance between the x value of the bottom right corner of the testing gesture and the x value of the upper left corner of the targeted gesture is less than 6 * contour area / 2500, the testing gesture is near to the targeted gesture.

If a point is to the right of the line (a) and to the right of the line (b), the point is to the east of the contour. If the distance between the x value of the upper left corner of the testing gesture and the x value of the bottom right corner of the targeted gesture is less than 6 * contour area / 2500, the testing gesture is near to the targeted gesture.

The relations are stored in a hash table. The keys for the relations are the name of the first target building, the name of the second building and the string “relation” for the

directional relations (including north, south, west and east) or the string “near” for the distance relation (including near or empty string).

Using the following code, my application prints and tests the hash table that storing the relations:

```
a = 1
while a < 27 :
        b = 1
        while b < 27 :
                if label[a] != label[b]:
                        print label[a] + " is to the " + building_relations[label[a], label[b],
                        "relation"] + " of " + label[b]
                        if building_relations[label[a], label[b], "near"] == "near":
                                print colored(label[a] + " is " + building_relations[label[a],
                                label[b], "near"] + " to the " + label[b], 'red')
                b += 1
        a += 1
```

Here are my printed results:

```
dyn-207-10-137-13:assignment3 puconghan$ python describing_compact_spatial_relations.py
Pupin is to the west of Schapiro CEPSR
Pupin is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Pupin is to the north of Physical Fitness Center
Pupin is to the north of Gymnasium & Uris
Pupin is to the north of Schermerhorn
Pupin is to the north of Chandler & Havemeyer
Pupin is to the north of Computer Center
Pupin is to the north of Avery
Pupin is to the north of Fayerweather
Pupin is to the north of Mathematics
Pupin is to the north of Low Library
Pupin is to the north of St. Paul's Chapel
Pupin is to the north of Earl Hall
Pupin is to the north of Lewisohn
Pupin is to the north of Philosophy
Pupin is to the north of Buell & Maison Francaise
Pupin is to the north of Alma Mater
Pupin is to the north of Dodge
Pupin is to the north of Kent
Pupin is to the north of College Walk
Pupin is to the north of Journalism & Furnald
Pupin is to the north of Hamilton, Hartley, Wallach & John Jay
Pupin is to the north of Lion's Court
Pupin is to the north of Lerner Hall
Pupin is to the north of Butler Library
Schapiro CEPSR is to the east of Pupin
Schapiro CEPSR is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Schapiro CEPSR is to the north of Physical Fitness Center
Schapiro CEPSR is to the north of Gymnasium & Uris
Schapiro CEPSR is to the north of Schermerhorn
Schapiro CEPSR is to the north of Chandler & Havemeyer
Schapiro CEPSR is to the north of Computer Center
Schapiro CEPSR is to the north of Avery
Schapiro CEPSR is to the north of Fayerweather
Schapiro CEPSR is to the north of Mathematics
Schapiro CEPSR is to the north of Low Library
Schapiro CEPSR is to the north of St. Paul's Chapel
Schapiro CEPSR is to the north of Earl Hall
Schapiro CEPSR is to the north of Lewisohn
Schapiro CEPSR is to the north of Philosophy
Schapiro CEPSR is to the north of Buell & Maison Francaise
Schapiro CEPSR is to the north of Alma Mater
Schapiro CEPSR is to the north of Dodge
Schapiro CEPSR is to the north of Kent
Schapiro CEPSR is to the north of College Walk
Schapiro CEPSR is to the north of Journalism & Furnald
```

```
Schapiro CEPSR is to the north of Hamilton, Hartley, Wallach & John Jay
Schapiro CEPSR is to the north of Lion's Court
Schapiro CEPSR is to the north of Lerner Hall
Schapiro CEPSR is to the north of Butler Library
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Pupin
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Schapiro CEPSR
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Physical Fitness Center
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Gymnasium & Uris
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Schermerhorn
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Chandler & Havemeyer
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Computer Center
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Avery
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Fayerweather
Mudd, Engineering Terrace, Fairchild & Computer Science is to the east of Mathematics
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Low Library
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of St. Paul's Chapel
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Earl Hall
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Lewisohn
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Philosophy
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Buell & Maison Francaise
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Alma Mater
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Dodge
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Kent
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of College Walk
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Journalism & Furnald
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Hamilton, Hartley, Wallach & John Jay
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Lion's Court
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Lerner Hall
Mudd, Engineering Terrace, Fairchild & Computer Science is to the north of Butler Library
Physical Fitness Center is to the south of Pupin
Physical Fitness Center is near to the Pupin
Physical Fitness Center is to the west of Schapiro CEPSR
Physical Fitness Center is near to the Schapiro CEPSR
Physical Fitness Center is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Physical Fitness Center is to the west of Gymnasium & Uris
Physical Fitness Center is near to the Gymnasium & Uris
Physical Fitness Center is to the west of Schermerhorn
Physical Fitness Center is to the east of Chandler & Havemeyer
Physical Fitness Center is to the north of Computer Center
Physical Fitness Center is to the north of Avery
Physical Fitness Center is to the north of Fayerweather
Physical Fitness Center is to the north of Mathematics
Physical Fitness Center is to the north of Low Library
Physical Fitness Center is to the north of St. Paul's Chapel
Physical Fitness Center is to the north of Earl Hall
Physical Fitness Center is to the north of Lewisohn
Physical Fitness Center is to the north of Philosophy
Physical Fitness Center is to the north of Buell & Maison Francaise
Physical Fitness Center is to the north of Alma Mater
Physical Fitness Center is to the north of Dodge
Physical Fitness Center is to the north of Kent
Physical Fitness Center is to the north of College Walk
Physical Fitness Center is to the north of Journalism & Furnald
Physical Fitness Center is to the north of Hamilton, Hartley, Wallach & John Jay
Physical Fitness Center is to the north of Lion's Court
Physical Fitness Center is to the north of Lerner Hall
Physical Fitness Center is to the north of Butler Library
Gymnasium & Uris is to the south of Pupin
Gymnasium & Uris is to the south of Schapiro CEPSR
Gymnasium & Uris is near to the Schapiro CEPSR
Gymnasium & Uris is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Gymnasium & Uris is to the east of Physical Fitness Center
Gymnasium & Uris is near to the Physical Fitness Center
Gymnasium & Uris is to the west of Schermerhorn
Gymnasium & Uris is near to the Schermerhorn
Gymnasium & Uris is to the east of Chandler & Havemeyer
Gymnasium & Uris is to the north of Computer Center
Gymnasium & Uris is to the north of Avery
Gymnasium & Uris is near to the Avery
Gymnasium & Uris is to the west of Fayerweather
Gymnasium & Uris is to the east of Mathematics
Gymnasium & Uris is to the north of Low Library
Gymnasium & Uris is to the north of St. Paul's Chapel
Gymnasium & Uris is to the north of Earl Hall
Gymnasium & Uris is to the north of Lewisohn
Gymnasium & Uris is to the north of Philosophy
Gymnasium & Uris is to the north of Buell & Maison Francaise
Gymnasium & Uris is to the north of Alma Mater
Gymnasium & Uris is to the north of Dodge
Gymnasium & Uris is to the north of Kent
Gymnasium & Uris is to the north of College Walk
Gymnasium & Uris is to the north of Journalism & Furnald
Gymnasium & Uris is to the north of Hamilton, Hartley, Wallach & John Jay
Gymnasium & Uris is to the north of Lion's Court
Gymnasium & Uris is to the north of Lerner Hall
Gymnasium & Uris is to the north of Butler Library
Schermerhorn is to the east of Pupin
Schermerhorn is to the south of Schapiro CEPSR
Schermerhorn is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Schermerhorn is to the east of Physical Fitness Center
Schermerhorn is to the east of Gymnasium & Uris
Schermerhorn is near to the Gymnasium & Uris
Schermerhorn is to the east of Chandler & Havemeyer
Schermerhorn is to the east of Computer Center
Schermerhorn is to the north of Avery
Schermerhorn is near to the Avery
Schermerhorn is to the north of Fayerweather
Schermerhorn is near to the Fayerweather
Schermerhorn is to the east of Mathematics
Schermerhorn is to the north of Low Library
Schermerhorn is to the north of St. Paul's Chapel
Schermerhorn is to the east of Earl Hall
Schermerhorn is to the east of Lewisohn
Schermerhorn is to the north of Philosophy
Schermerhorn is to the north of Buell & Maison Francaise
Schermerhorn is to the north of Alma Mater
Schermerhorn is to the east of Dodge
Schermerhorn is to the north of Kent
Schermerhorn is to the north of College Walk
Schermerhorn is to the north of Journalism & Furnald
Schermerhorn is to the north of Hamilton, Hartley, Wallach & John Jay
Schermerhorn is to the north of Lion's Court
Schermerhorn is to the north of Lerner Hall
Schermerhorn is to the north of Butler Library
Chandler & Havemeyer is to the south of Pupin
Chandler & Havemeyer is to the south of Schapiro CEPSR
Chandler & Havemeyer is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Chandler & Havemeyer is to the south of Physical Fitness Center
Chandler & Havemeyer is to the west of Gymnasium & Uris
Chandler & Havemeyer is to the west of Schermerhorn
Chandler & Havemeyer is to the west of Computer Center
Chandler & Havemeyer is to the west of Avery
Chandler & Havemeyer is to the west of Fayerweather
Chandler & Havemeyer is to the north of Mathematics
Chandler & Havemeyer is to the north of Low Library
Chandler & Havemeyer is to the west of St. Paul's Chapel
Chandler & Havemeyer is to the north of Earl Hall
Chandler & Havemeyer is to the north of Lewisohn
Chandler & Havemeyer is to the west of Philosophy
Chandler & Havemeyer is to the north of Buell & Maison Francaise
Chandler & Havemeyer is to the north of Alma Mater
Chandler & Havemeyer is to the north of Dodge
Chandler & Havemeyer is to the north of Kent
Chandler & Havemeyer is to the north of College Walk
Chandler & Havemeyer is to the north of Journalism & Furnald
Chandler & Havemeyer is to the north of Hamilton, Hartley, Wallach & John Jay
Chandler & Havemeyer is to the north of Lion's Court
Chandler & Havemeyer is to the north of Lerner Hall
Chandler & Havemeyer is to the north of Butler Library
Computer Center is to the south of Pupin
Computer Center is to the south of Schapiro CEPSR
```

```
Computer Center is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Computer Center is to the south of Physical Fitness Center
Computer Center is to the west of Gymnasium & Uris
Computer Center is to the west of Schermerhorn
Computer Center is to the east of Chandler & Havemeyer
Computer Center is to the west of Avery
Computer Center is to the west of Fayerweather
Computer Center is to the east of Mathematics
Computer Center is to the north of Low Library
Computer Center is to the west of St. Paul's Chapel
Computer Center is to the north of Earl Hall
Computer Center is to the north of Lewisohn
Computer Center is to the west of Philosophy
Computer Center is to the west of Buell & Maison Francaise
Computer Center is to the north of Alma Mater
Computer Center is to the north of Dodge
Computer Center is to the west of Kent
Computer Center is to the north of College Walk
Computer Center is to the north of Journalism & Furnald
Computer Center is to the west of Hamilton, Hartley, Wallach & John Jay
Computer Center is to the north of Lion's Court
Computer Center is to the north of Lerner Hall
Computer Center is to the north of Butler Library
Avery is to the east of Pupin
Avery is to the east of Schapiro CEPSR
Avery is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Avery is to the east of Physical Fitness Center
Avery is to the east of Gymnasium & Uris
Avery is to the south of Schermerhorn
Avery is to the east of Chandler & Havemeyer
Avery is to the east of Computer Center
Avery is to the west of Fayerweather
Avery is to the east of Mathematics
Avery is to the east of Low Library
Avery is to the west of St. Paul's Chapel
Avery is to the east of Earl Hall
Avery is to the east of Lewisohn
Avery is to the west of Philosophy
Avery is to the north of Buell & Maison Francaise
Avery is to the north of Alma Mater
Avery is to the east of Dodge
Avery is to the west of Kent
Avery is to the north of College Walk
Avery is to the east of Journalism & Furnald
Avery is to the north of Hamilton, Hartley, Wallach & John Jay
Avery is to the north of Lion's Court
Avery is to the north of Lerner Hall
Avery is to the north of Butler Library
Fayerweather is to the east of Pupin
Fayerweather is to the east of Schapiro CEPSR
Fayerweather is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Fayerweather is to the east of Physical Fitness Center
Fayerweather is to the east of Gymnasium & Uris
Fayerweather is to the south of Schermerhorn
Fayerweather is to the east of Chandler & Havemeyer
Fayerweather is to the east of Computer Center
Fayerweather is to the east of Avery
Fayerweather is to the east of Mathematics
Fayerweather is to the east of Low Library
Fayerweather is to the east of St. Paul's Chapel
Fayerweather is to the east of Earl Hall
Fayerweather is to the east of Lewisohn
Fayerweather is to the north of Philosophy
Fayerweather is to the east of Buell & Maison Francaise
Fayerweather is to the east of Alma Mater
Fayerweather is to the east of Dodge
Fayerweather is to the north of Kent
Fayerweather is to the east of College Walk
Fayerweather is to the east of Journalism & Furnald
Fayerweather is to the north of Hamilton, Hartley, Wallach & John Jay
Fayerweather is to the north of Lion's Court
Fayerweather is to the east of Lerner Hall
Fayerweather is to the north of Butler Library
Mathematics is to the south of Pupin
Mathematics is to the west of Schapiro CEPSR
Mathematics is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Mathematics is to the west of Physical Fitness Center
Mathematics is to the west of Gymnasium & Uris
Mathematics is to the west of Schermerhorn
Mathematics is to the south of Chandler & Havemeyer
Mathematics is to the west of Computer Center
Mathematics is to the west of Avery
Mathematics is to the west of Fayerweather
Mathematics is to the west of Low Library
Mathematics is to the west of St. Paul's Chapel
Mathematics is to the west of Earl Hall
Mathematics is near to the Earl Hall
Mathematics is to the north of Lewisohn
Mathematics is to the west of Philosophy
Mathematics is to the west of Buell & Maison Francaise
Mathematics is to the west of Alma Mater
Mathematics is to the north of Dodge
Mathematics is to the west of Kent
Mathematics is to the west of College Walk
Mathematics is to the north of Journalism & Furnald
Mathematics is to the west of Hamilton, Hartley, Wallach & John Jay
Mathematics is to the west of Lion's Court
Mathematics is to the north of Lerner Hall
Mathematics is to the north of Butler Library
Low Library is to the south of Pupin
Low Library is to the south of Schapiro CEPSR
Low Library is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Low Library is to the south of Physical Fitness Center
Low Library is to the south of Gymnasium & Uris
Low Library is to the south of Schermerhorn
Low Library is to the south of Chandler & Havemeyer
Low Library is to the south of Computer Center
Low Library is to the west of Avery
Low Library is to the west of Fayerweather
Low Library is to the east of Mathematics
Low Library is to the west of St. Paul's Chapel
Low Library is to the east of Earl Hall
Low Library is to the east of Lewisohn
Low Library is to the west of Philosophy
Low Library is to the west of Buell & Maison Francaise
Low Library is to the north of Alma Mater
Low Library is to the east of Dodge
Low Library is to the west of Kent
Low Library is to the north of College Walk
Low Library is to the north of Journalism & Furnald
Low Library is to the north of Hamilton, Hartley, Wallach & John Jay
Low Library is to the north of Lion's Court
Low Library is to the north of Lerner Hall
Low Library is to the north of Butler Library
St. Paul's Chapel is to the south of Pupin
St. Paul's Chapel is to the south of Schapiro CEPSR
St. Paul's Chapel is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
St. Paul's Chapel is to the south of Physical Fitness Center
St. Paul's Chapel is to the south of Gymnasium & Uris
St. Paul's Chapel is to the south of Schermerhorn
St. Paul's Chapel is to the south of Chandler & Havemeyer
St. Paul's Chapel is to the south of Computer Center
St. Paul's Chapel is to the south of Avery
St. Paul's Chapel is to the south of Fayerweather
St. Paul's Chapel is to the east of Mathematics
St. Paul's Chapel is to the east of Low Library
St. Paul's Chapel is to the east of Earl Hall
St. Paul's Chapel is to the east of Lewisohn
St. Paul's Chapel is to the north of Philosophy
St. Paul's Chapel is to the north of Buell & Maison Francaise
St. Paul's Chapel is to the north of Alma Mater
```

```
St. Paul's Chapel is to the east of Dodge
St. Paul's Chapel is to the north of Kent
St. Paul's Chapel is to the north of College Walk
St. Paul's Chapel is to the north of Journalism & Furnald
St. Paul's Chapel is to the north of Hamilton, Hartley, Wallach & John Jay
St. Paul's Chapel is to the north of Lion's Court
St. Paul's Chapel is to the north of Lerner Hall
St. Paul's Chapel is to the north of Butler Library
Earl Hall is to the south of Pupin
Earl Hall is to the south of Schapiro CEPSR
Earl Hall is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Earl Hall is to the south of Physical Fitness Center
Earl Hall is to the south of Gymnasium & Uris
Earl Hall is to the south of Schermerhorn
Earl Hall is to the south of Chandler & Havemeyer
Earl Hall is to the south of Computer Center
Earl Hall is to the west of Avery
Earl Hall is to the west of Fayerweather
Earl Hall is to the south of Mathematics
Earl Hall is to the west of Low Library
Earl Hall is to the west of St. Paul's Chapel
Earl Hall is to the north of Lewisohn
Earl Hall is near to the Lewisohn
Earl Hall is to the west of Philosophy
Earl Hall is to the west of Buell & Maison Francaise
Earl Hall is to the north of Alma Mater
Earl Hall is to the north of Dodge
Earl Hall is to the west of Kent
Earl Hall is to the north of College Walk
Earl Hall is to the north of Journalism & Furnald
Earl Hall is to the north of Hamilton, Hartley, Wallach & John Jay
Earl Hall is to the north of Lion's Court
Earl Hall is to the north of Lerner Hall
Earl Hall is to the north of Butler Library
Lewisohn is to the south of Pupin
Lewisohn is to the south of Schapiro CEPSR
Lewisohn is to the west of Mudd, Engineering Terrace, Fairchild & Computer Science
Lewisohn is to the south of Physical Fitness Center
Lewisohn is to the west of Gymnasium & Uris
Lewisohn is to the west of Schermerhorn
Lewisohn is to the south of Chandler & Havemeyer
Lewisohn is to the west of Computer Center
Lewisohn is to the west of Avery
Lewisohn is to the west of Fayerweather
Lewisohn is to the south of Mathematics
Lewisohn is to the west of Low Library
Lewisohn is to the west of St. Paul's Chapel
Lewisohn is to the west of Earl Hall
Lewisohn is near to the Earl Hall
Lewisohn is to the west of Philosophy
Lewisohn is to the west of Buell & Maison Francaise
Lewisohn is to the west of Alma Mater
Lewisohn is to the west of Dodge
Lewisohn is to the west of Kent
Lewisohn is to the west of College Walk
Lewisohn is to the north of Journalism & Furnald
Lewisohn is to the west of Hamilton, Hartley, Wallach & John Jay
Lewisohn is to the west of Lion's Court
Lewisohn is to the north of Lerner Hall
Lewisohn is to the west of Butler Library
Philosophy is to the east of Pupin
Philosophy is to the south of Schapiro CEPSR
Philosophy is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Philosophy is to the east of Physical Fitness Center
Philosophy is to the east of Gymnasium & Uris
Philosophy is to the south of Schermerhorn
Philosophy is to the east of Chandler & Havemeyer
Philosophy is to the east of Computer Center
Philosophy is to the east of Avery
Philosophy is to the south of Fayerweather
Philosophy is to the east of Mathematics
Philosophy is to the east of Low Library
Philosophy is to the east of St. Paul's Chapel
Philosophy is to the east of Earl Hall
Philosophy is to the east of Lewisohn
Philosophy is to the east of Buell & Maison Francaise
Philosophy is to the east of Alma Mater
Philosophy is to the east of Dodge
Philosophy is to the east of Kent
Philosophy is to the east of College Walk
Philosophy is to the east of Journalism & Furnald
Philosophy is to the north of Hamilton, Hartley, Wallach & John Jay
Philosophy is to the north of Lion's Court
Philosophy is to the east of Lerner Hall
Philosophy is to the north of Butler Library
Buell & Maison Francaise is to the south of Pupin
Buell & Maison Francaise is to the south of Schapiro CEPSR
Buell & Maison Francaise is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Buell & Maison Francaise is to the south of Physical Fitness Center
Buell & Maison Francaise is to the south of Gymnasium & Uris
Buell & Maison Francaise is to the south of Schermerhorn
Buell & Maison Francaise is to the south of Chandler & Havemeyer
Buell & Maison Francaise is to the south of Computer Center
Buell & Maison Francaise is to the south of Avery
Buell & Maison Francaise is to the south of Fayerweather
Buell & Maison Francaise is to the east of Mathematics
Buell & Maison Francaise is to the east of Low Library
Buell & Maison Francaise is to the south of St. Paul's Chapel
Buell & Maison Francaise is to the east of Earl Hall
Buell & Maison Francaise is to the east of Lewisohn
Buell & Maison Francaise is to the west of Philosophy
Buell & Maison Francaise is to the east of Alma Mater
Buell & Maison Francaise is to the east of Dodge
Buell & Maison Francaise is to the north of Kent
Buell & Maison Francaise is to the north of College Walk
Buell & Maison Francaise is to the north of Journalism & Furnald
Buell & Maison Francaise is to the north of Hamilton, Hartley, Wallach & John Jay
Buell & Maison Francaise is to the north of Lion's Court
Buell & Maison Francaise is to the north of Lerner Hall
Buell & Maison Francaise is to the north of Butler Library
Alma Mater is to the south of Pupin
Alma Mater is to the south of Schapiro CEPSR
Alma Mater is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Alma Mater is to the south of Physical Fitness Center
Alma Mater is to the south of Gymnasium & Uris
Alma Mater is to the south of Schermerhorn
Alma Mater is to the south of Chandler & Havemeyer
Alma Mater is to the south of Computer Center
Alma Mater is to the south of Avery
Alma Mater is to the west of Fayerweather
Alma Mater is to the east of Mathematics
Alma Mater is to the south of Low Library
Alma Mater is to the west of St. Paul's Chapel
Alma Mater is to the east of Earl Hall
Alma Mater is to the east of Lewisohn
Alma Mater is to the west of Philosophy
Alma Mater is to the west of Buell & Maison Francaise
Alma Mater is to the east of Dodge
Alma Mater is to the west of Kent
Alma Mater is to the north of College Walk
Alma Mater is to the east of Journalism & Furnald
Alma Mater is to the north of Hamilton, Hartley, Wallach & John Jay
Alma Mater is to the north of Lion's Court
Alma Mater is to the north of Lerner Hall
Alma Mater is to the north of Butler Library
Dodge is to the south of Pupin
Dodge is to the south of Schapiro CEPSR
Dodge is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Dodge is to the south of Physical Fitness Center
Dodge is to the south of Gymnasium & Uris
Dodge is to the south of Schermerhorn
```

```
Dodge is to the south of Computer Center
Dodge is to the south of Avery
Dodge is to the south of Fayerweather
Dodge is to the south of Mathematics
Dodge is to the south of Low Library
Dodge is to the south of St. Paul's Chapel
Dodge is to the south of Earl Hall
Dodge is to the south of Lewisohn
Dodge is near to the Lewisohn
Dodge is to the west of Philosophy
Dodge is to the south of Buell & Maison Francaise
Dodge is to the south of Alma Mater
Dodge is to the west of Kent
Dodge is to the north of College Walk
Dodge is near to the College Walk
Dodge is to the north of Journalism & Furnald
Dodge is to the north of Hamilton, Hartley, Wallach & John Jay
Dodge is to the north of Lion's Court
Dodge is to the north of Lerner Hall
Dodge is to the north of Butler Library
Kent is to the south of Pupin
Kent is to the south of Schapiro CEPSR
Kent is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Kent is to the south of Physical Fitness Center
Kent is to the south of Gymnasium & Uris
Kent is to the south of Schermerhorn
Kent is to the south of Chandler & Havemeyer
Kent is to the south of Computer Center
Kent is to the south of Avery
Kent is to the south of Fayerweather
Kent is to the south of Mathematics
Kent is to the south of Low Library
Kent is to the south of St. Paul's Chapel
Kent is to the south of Earl Hall
Kent is to the south of Lewisohn
Kent is to the south of Philosophy
Kent is near to the Philosophy
Kent is to the south of Buell & Maison Francaise
Kent is to the south of Alma Mater
Kent is to the east of Dodge
Kent is to the north of College Walk
Kent is near to the College Walk
Kent is to the north of Journalism & Furnald
Kent is to the north of Hamilton, Hartley, Wallach & John Jay
Kent is to the north of Lion's Court
Kent is to the north of Lerner Hall
Kent is to the north of Butler Library
College Walk is to the south of Pupin
College Walk is to the south of Schapiro CEPSR
College Walk is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
College Walk is to the south of Physical Fitness Center
College Walk is to the south of Gymnasium & Uris
College Walk is to the south of Schermerhorn
College Walk is to the south of Chandler & Havemeyer
College Walk is to the south of Computer Center
College Walk is to the south of Avery
College Walk is to the south of Fayerweather
College Walk is to the south of Mathematics
College Walk is to the south of Low Library
College Walk is to the south of St. Paul's Chapel
College Walk is to the south of Earl Hall
College Walk is to the south of Lewisohn
College Walk is to the south of Philosophy
College Walk is to the south of Buell & Maison Francaise
College Walk is to the south of Alma Mater
College Walk is to the south of Dodge
College Walk is near to the Dodge
College Walk is to the south of Kent
College Walk is near to the Kent
College Walk is to the north of Journalism & Furnald
College Walk is near to the Journalism & Furnald
College Walk is to the north of Hamilton, Hartley, Wallach & John Jay
College Walk is near to the Hamilton, Hartley, Wallach & John Jay
College Walk is to the north of Lion's Court
College Walk is to the north of Lerner Hall
College Walk is to the north of Butler Library
Journalism & Furnald is to the south of Pupin
Journalism & Furnald is to the south of Schapiro CEPSR
Journalism & Furnald is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Journalism & Furnald is to the south of Physical Fitness Center
Journalism & Furnald is to the south of Gymnasium & Uris
Journalism & Furnald is to the south of Schermerhorn
Journalism & Furnald is to the south of Chandler & Havemeyer
Journalism & Furnald is to the south of Computer Center
Journalism & Furnald is to the south of Avery
Journalism & Furnald is to the south of Fayerweather
Journalism & Furnald is to the south of Mathematics
Journalism & Furnald is to the south of Low Library
Journalism & Furnald is to the south of St. Paul's Chapel
Journalism & Furnald is to the south of Earl Hall
Journalism & Furnald is to the south of Lewisohn
Journalism & Furnald is to the west of Philosophy
Journalism & Furnald is to the south of Buell & Maison Francaise
Journalism & Furnald is to the south of Alma Mater
Journalism & Furnald is to the south of Dodge
Journalism & Furnald is to the west of Kent
Journalism & Furnald is to the west of College Walk
Journalism & Furnald is to the west of Hamilton, Hartley, Wallach & John Jay
Journalism & Furnald is to the west of Lion's Court
Journalism & Furnald is to the north of Lerner Hall
Journalism & Furnald is to the west of Butler Library
Journalism & Furnald is near to the Butler Library
Hamilton, Hartley, Wallach & John Jay is to the south of Pupin
Hamilton, Hartley, Wallach & John Jay is to the south of Schapiro CEPSR
Hamilton, Hartley, Wallach & John Jay is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Hamilton, Hartley, Wallach & John Jay is to the south of Physical Fitness Center
Hamilton, Hartley, Wallach & John Jay is to the south of Gymnasium & Uris
Hamilton, Hartley, Wallach & John Jay is to the south of Schermerhorn
Hamilton, Hartley, Wallach & John Jay is to the south of Chandler & Havemeyer
Hamilton, Hartley, Wallach & John Jay is to the south of Computer Center
Hamilton, Hartley, Wallach & John Jay is to the south of Avery
Hamilton, Hartley, Wallach & John Jay is to the south of Fayerweather
Hamilton, Hartley, Wallach & John Jay is to the east of Mathematics
Hamilton, Hartley, Wallach & John Jay is to the south of Low Library
Hamilton, Hartley, Wallach & John Jay is to the south of St. Paul's Chapel
Hamilton, Hartley, Wallach & John Jay is to the east of Earl Hall
Hamilton, Hartley, Wallach & John Jay is to the east of Lewisohn
Hamilton, Hartley, Wallach & John Jay is to the south of Philosophy
Hamilton, Hartley, Wallach & John Jay is to the south of Buell & Maison Francaise
Hamilton, Hartley, Wallach & John Jay is to the east of Alma Mater
Hamilton, Hartley, Wallach & John Jay is to the east of Dodge
Hamilton, Hartley, Wallach & John Jay is to the south of Kent
Hamilton, Hartley, Wallach & John Jay is to the east of College Walk
Hamilton, Hartley, Wallach & John Jay is to the east of Journalism & Furnald
Hamilton, Hartley, Wallach & John Jay is to the east of Lion's Court
Hamilton, Hartley, Wallach & John Jay is to the east of Lerner Hall
Hamilton, Hartley, Wallach & John Jay is to the east of Butler Library
Hamilton, Hartley, Wallach & John Jay is near to the Butler Library
Lion's Court is to the south of Pupin
Lion's Court is to the south of Schapiro CEPSR
Lion's Court is to the south of Mudd, Engineering Terrace, Fairchild & Computer Science
Lion's Court is to the south of Physical Fitness Center
Lion's Court is to the south of Gymnasium & Uris
Lion's Court is to the south of Schermerhorn
Lion's Court is to the south of Chandler & Havemeyer
Lion's Court is to the south of Computer Center
Lion's Court is to the south of Avery
Lion's Court is to the south of Fayerweather
Lion's Court is to the east of Mathematics
Lion's Court is to the south of Low Library
Lion's Court is to the south of St. Paul's Chapel
```

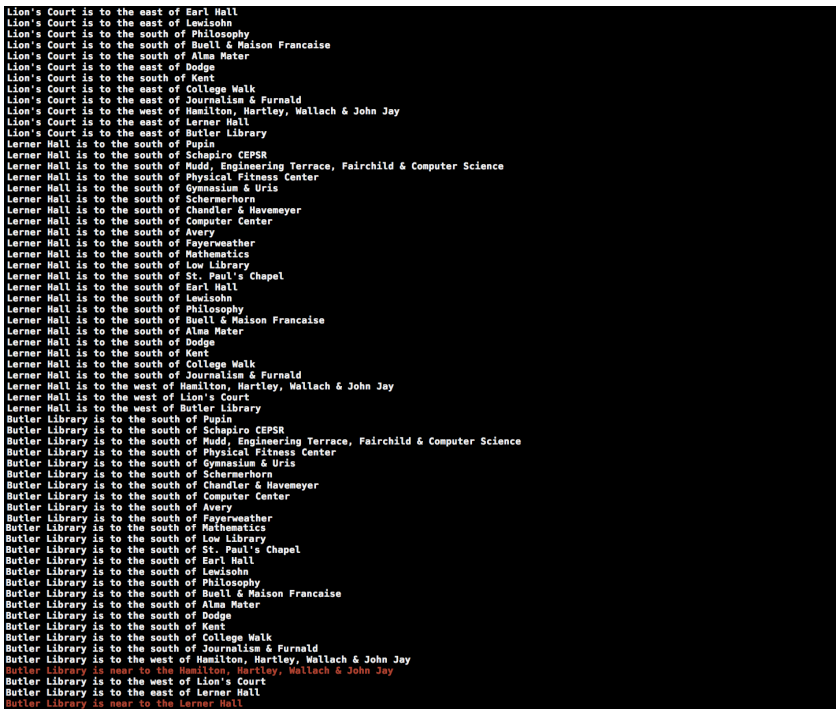


Figure 8: Printed testing results.

As we can see from Figure 8, the computation generates on the order of O(B*B*R) binary relationships, where B = 27 buildings, and R = 4 (north, south, west and east) + 1 (near) relationships.

Before filtering the results, I added one hash table building_distance and two lists reduced_relations and near_relations to my program:

```
building_distance = {}
reduced_relations = []
near_relations = []
```

I changed the implementations to compute the relations:

```
for building in building_properties:
      for building_sub in building_properties:
            if building[0] != building_sub[0]:
                  if (isLeft(building[1], building[3], building_sub[1])
                  == False) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == True):
                        building_relations[building[0], building_sub[0],
                        "relation"] = "north"
                         building_distance[building[0], building_sub[0],
                        "relation"] = math.sqrt((building[1][0] -
                        building_sub[1][0])*(building[1][0] -
                        building_sub[1][0]) + (building[1][1] -
                        building_sub[1][1])*(building[1][1] -
                        building_sub[1][1]))
```

```
     if math.sqrt((building_sub[3][1] -
     building[4][1])*(building_sub[3][1] -
     building[4][1])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
if (isLeft(building[1], building[3], building_sub[1])
== True) and (isLeft(building[1], [building[4][0],
building[3][1]], building_sub[1]) == False):
     building_relations[building[0], building_sub[0],
     "relation"] = "south"
      building_distance[building[0], building_sub[0],
     "relation"] = math.sqrt((building[1][0] -
     building_sub[1][0])*(building[1][0] -
     building_sub[1][0]) + (building[1][1] -
     building_sub[1][1])*(building[1][1] -
     building_sub[1][1]))
     if math.sqrt((building[3][1] -
     building_sub[4][1])*(building[3][1] -
     building_sub[4][1])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
if (isLeft(building[1], building[3], building_sub[1])
== True) and (isLeft(building[1], [building[4][0],
building[3][1]], building_sub[1]) == True):
     building_relations[building[0], building_sub[0],
     "relation"] = "west"
      building_distance[building[0], building_sub[0],
     "relation"] = math.sqrt((building[1][0] -
     building_sub[1][0])*(building[1][0] -
     building_sub[1][0]) + (building[1][1] -
     building_sub[1][1])*(building[1][1] -
     building_sub[1][1]))
     if math.sqrt((building_sub[3][0] -
     building[4][0])*(building_sub[3][0] -
     building[4][0])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
if (isLeft(building[1], building[3], building_sub[1])
== False) and (isLeft(building[1], [building[4][0],
building[3][1]], building_sub[1]) == False):
```

```
                        building_relations[building[0], building_sub[0],
                        "relation"] = "east"
                         building_distance[building[0], building_sub[0],
                        "relation"] = math.sqrt((building[1][0] -
                        building_sub[1][0])*(building[1][0] -
                        building_sub[1][0]) + (building[1][1] -
                        building_sub[1][1])*(building[1][1] -
                        building_sub[1][1]))
                         if math.sqrt((building[3][0] -
                        building_sub[4][0])*(building[3][0] -
                        building_sub[4][0])) <= 6 * building[2] / 2500:
                              building_relations[building[0],
                              building_sub[0], "near"] = "near"
                         else:
                              building_relations[building[0],
                              building_sub[0], "near"] =
                              math.sqrt((building_sub[3][1] -
                              building[4][1])*(building_sub[3][1] -
                              building[4][1]))
```

This new computation procedure stores the distance between the testing building contour and the targeted building contour into the hash table building_distance.

Using the following 2D while-loop, my application filters these relationships and leaves only the ones that cannot be inferred by the usual transitivity rules:

```
a = 1
while a < 27 :
      b = 1
      target_building_N = ""
      distance_count_N = math.sqrt(495*495+275*275)
      testing_building_N = ""
      target_building_S = ""
      distance_count_S = math.sqrt(495*495+275*275)
      testing_building_S = ""
      target_building_W = ""
      distance_count_W = math.sqrt(495*495+275*275)
      testing_building_W = ""
      target_building_E = ""
      distance_count_E = math.sqrt(495*495+275*275)
      testing_building_E = ""
      while b < 27 :
            if label[a] != label[b]:
                  if building_relations[label[a], label[b], "relation"]
                  == "north":
                        if distance_count_N >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_N =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_N = label[b]
                              testing_building_N = label[a]
                  if building_relations[label[a], label[b], "relation"]
                  == "south":
```

```
                        if distance_count_S >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_S =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_S = label[b]
                              testing_building_S = label[a]
                  if building_relations[label[a], label[b], "relation"]
                  == "west":
                        if distance_count_W >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_W =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_W = label[b]
                              testing_building_W = label[a]
                  if building_relations[label[a], label[b], "relation"]
                  == "east":
                        if distance_count_E >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_E =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_E = label[b]
                              testing_building_E = label[a]
                  if building_relations[label[a], label[b], "near"] ==
                  "near":
                        near_relations.append([label[a], label[b]])
            b += 1
      a += 1
      if distance_count_N != math.sqrt(495*495+275*275):
            reduced_relations.append([testing_building_N, "north" ,
            target_building_N])
      if distance_count_S != math.sqrt(495*495+275*275):
            reduced_relations.append([testing_building_S, "south" ,
            target_building_S])
      if distance_count_W != math.sqrt(495*495+275*275):
            reduced_relations.append([testing_building_W, "west" ,
            target_building_W])
      if distance_count_E != math.sqrt(495*495+275*275):
            reduced_relations.append([testing_building_E, "east" ,
            target_building_E])
```

For each of the four relations (north, south, west and east), my program keeps the one that have the shortest distance between the testing building contour and the targeted building contour. The filtered results of the four relations are stored in the reduced_relations list. The results of the near relations are stored in the near_relations list.

```
for item in reduced_relations:
      print item
for item in near_relations:
      print item
```

Using the above printing loops, my application prints the following filtered results:

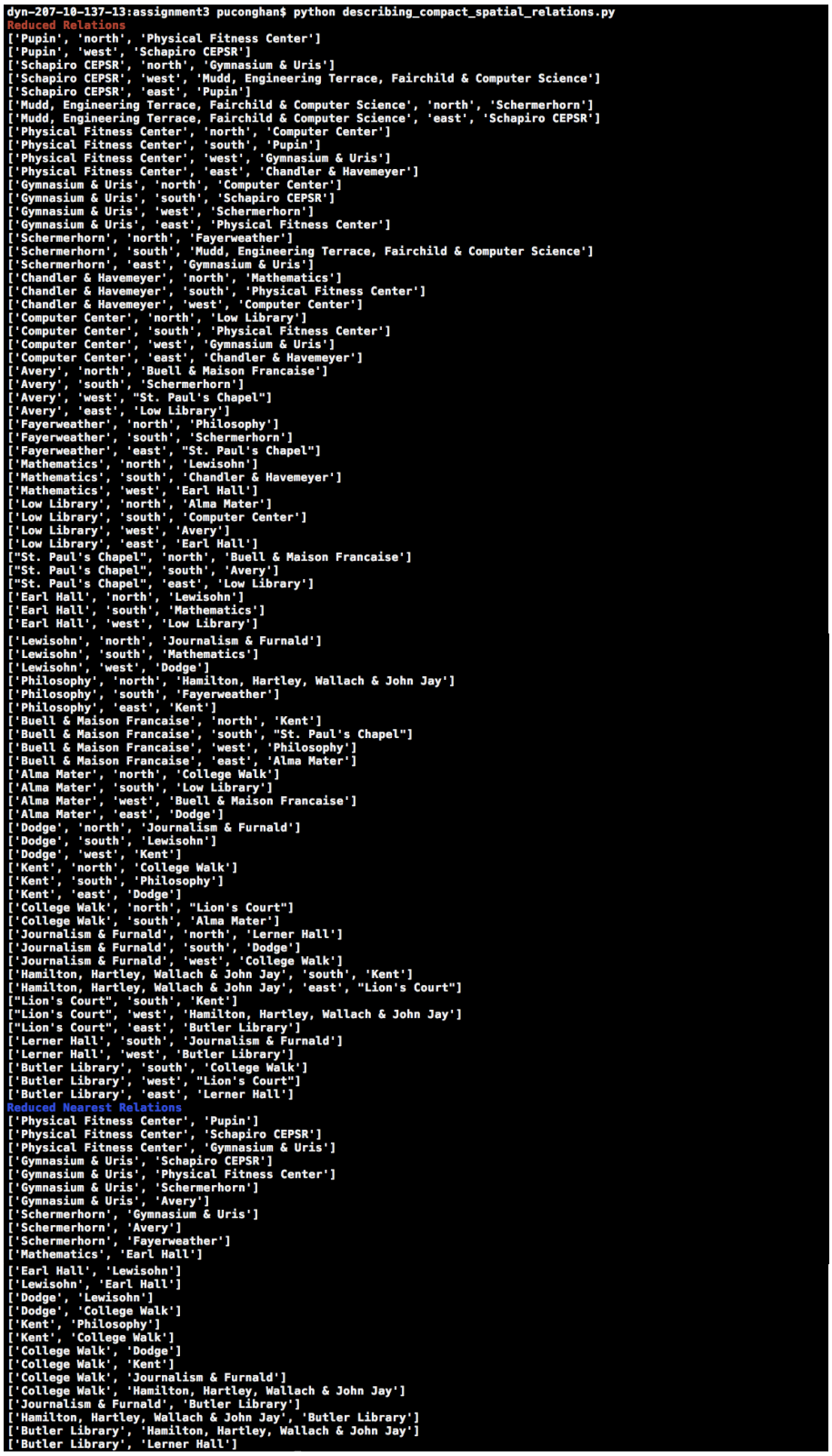


Figure 9: Printed filtered results.

As we can see the reduced relations from Figure 9, the first testing building (first element of the list) is to the relation (second element of the list) of the targeted building (third element of the list).
The number of relationships survived from my filtering steps are accurate (leaving only the ones that cannot be inferred by the usual transitivity rules) and much fewer than the 27*5 possible relationships each building can enter into.

Step 3: Source and Goal Description and User Interface

In step 3, I designed a user interface using the Python Tkinter library. According to the Python wiki documentation, Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is not the only GuiProgramming toolkit for Python. It is however the most commonly used one. The installation of Tkinter on Python 2.7 is a little bit complicated. Following the instruction (http://www.python.org/getit/mac/tcltk/), I installed ActiveTcl on my Mac machine. I also run the following command to make my Python 2.7 connect to the ActiveTcl library:

```
sudo port install py27-tkinter
```

I setup the basic user interface in Figure 10 using the following codes:

```
# Importing Tkinter GUI library
from Tkinter import *

master = Tk()

# Callback function for mouse clicks
def callback(event):
	***
photo = PhotoImage(file="ass3-campus.pgm")
w = Label(None, image=photo)
w.photo = photo
w.bind("<Button-1>", callback)
w.pack()

mainloop()
```

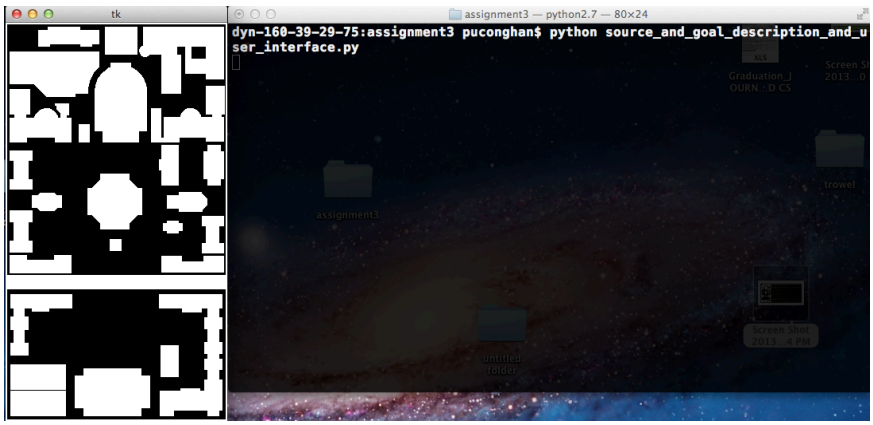


Figure 10: Basic user interface.

This user interface reads and displays the campus image to the user interface. It has a callback function to capture mouse left clicks. In other words, the callback function is triggered by left clicks. Code for description and location analyzing should go into the mouse callback function.

The first portion of the codes goes into this callback function checks whether the captured x and y values fall into any building gestures. If the x and y values fall into a particular building contour, the code will print the name of the building. Here is my implementation for the first portion inside the callback function:

```
#[Testing] Print the mouse click x and y values
print colored("clicked at: " + "(" + str(event.x) + ", " +
str(event.y) + ")", "red")

# The following code computes the descriptions for the locations
contour_fall = ""
contour_name = ""
descritions = []
for h,cnt in enumerate(contours):
      #If the point is in any building contours, codes within
      this if-statement will print the building name.
      if cv2.pointPolygonTest(cnt,(event.x, event.y),True) >= 0:
            contour_fall = cnt
            count = 0
            integer_value = 0
            for demensions in cnt:
                  for values in demensions:
                        integer_value +=
                        integer_value_campus.getpixel((int(values
                        [0]), int(values[1])))
                        count += 1
                        #Get the name of the building by passing
                        the variable to the hashtable label
                        contour_name = label[integer_value /
                        count]
            print colored(contour_name, 'blue')
```

This portion of the code takes advantage of the pointPolygonTest() function provided by Python OpenCV. According to the Python OpenCV documentation, this function determines whether the point is inside a contour, outside, or lies on an edge (or coincides with a vertex). The pointPolygonTest() function returns positive (inside), negative (outside), or zero (on an edge) value, correspondingly. The if-statement checks whether an x and y values (captured from the mouse) fall into a building gesture. Codes inside the if-statement find and print the name of the building gesture.

The second portion of the codes checks the immediate buildings to the (north/ south/ west/ east) of the mouse locations. I take advantages of code from step 2. I first compute the relation description of the point to all building contours, and then reduced the size of the relations. Here are my implementations for the second portion:

```
#If the point is not in any building contours, codes within this
if-statement compute the location description of the point.
```

```
if contour_fall == "":
      #Storing spatial relations of the point to all buildings.
      This is a temporary variable.
      spatial_relations = []
      #This for-loop goes over all buildings in the
      building_properties variable (including a list of
      building_name, building_contour_center,
      building_contour_area, building_contour_upper_left,
      building_contour_lower_right, building_shape,
      building_size_description,
      building_location_extrema_description).
      for building in building_properties:
            #These if-and-else-statement checks and store
            building names and relations to the points to the
            spatial_relations list variable.
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == True):
                  if math.sqrt((event.y -
                  building[4][1])*(event.y - building[4][1])) <=
                  10 * building[2] / 2500:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == False):
                  if math.sqrt((building[3][1] -
                  event.y)*(building[3][1] - event.y)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == True):
                  if math.sqrt((event.x -
                  building[4][0])*(event.x - building[4][0])) <=
                  10 * building[2] / 2500:
```

```
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == False):
                  if math.sqrt((building[3][0] -
                  event.x)*(building[3][0] - event.x)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""
#This for-loop goes over the spatial_relations and stores
the most immediate relation (nearest/ north/ south/ west/
east) of a point (with shortest distance to a building
gesture) to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
```

```
                near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check all other points
with same relation descriptions.
point_stack = ""

#These if-statements update and save the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture) from
temporary variables to descriptions list and point_stack
string variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            descritions.append("Point is near and is to the
            north of " + building_N)
            point_stack = point_stack + "near_north" +
            building_N

      else:
            descritions.append("Point is to the north of "
            + building_N)
            point_stack = point_stack + "north" +
            building_N
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            descritions.append("Point is near and is to the
            south of " + building_S)
            point_stack = point_stack + "near_south" +
            building_S
      else:
            descritions.append("Point is to the south of "
            + building_S)
            point_stack = point_stack + "south" +
            building_S
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
            descritions.append("Point is near and is to the
            west of " + building_W)
            point_stack = point_stack + "near_west" +
            building_W
      else:
            descritions.append("Point is to the west of " +
            building_W)
            point_stack = point_stack + "west" + building_W
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
            descritions.append("Point is near and is to the
            east of " + building_E)
```

```
                point_stack = point_stack + "near_east" +
                building_E
          else:
                descritions.append("Point is to the east of " +
                building_E)
                point_stack = point_stack + "east" + building_E
#This for-loop prints all immediate relation (nearest/
north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from the descriptions list.
for item in descritions:
      print colored(item, 'green')
```

The above code prints the most immediate relations (nearest/ north/ south/ west/ east) of a point in the console window. According to the assignment, descriptions I give for the point will be shared by other pixels as well. I need to implement my application to display the uncertainty that this click causes, as a sort of pixel cloud. In the following implementations, my application not only gives the description, but also displays on the image all the pixels in this equivalence class (I color them all red). Here are my implementations:

```
# The following code finds and draws all points sharing
similar location descriptions.

#Open the integer value campus image
campus_region = cv2.imread("ass3-campus.pgm")

#Improve running efficiency using the following loops. They
go over all point within a region of interest 100 pixel *
100 pixel (mouse pixel in the middle)
# x = event.x - 50
# while x <= event.x + 50:
#     y = event.y - 50
#     while y <= event.y + 50:

#The following while-loops go over all pixels in the image.
x = 0
while x <= integer_value_campus.size[0]:
      y = 0
      while y <= integer_value_campus.size[1]:
            spatial_relations = []
            #Similar to the previous portion, these loops
            and if-statement compute all relational
            descriptions of a point to all building
            contours.
            for building in building_properties:
                  if (isLeft(building[1], building[3], [x,
                  y]) == False) and (isLeft(building[1],
                  [building[4][0], building[3][1]], [x, y])
                  == True):
                        if math.sqrt((y -
                        building[4][1])*(y -
                        building[4][1])) <= 10 * building[2]
                        / 2500:
                              spatial_relations.append(["so
                              uth", building[0],
                              math.sqrt((building[1][0] -
```

```
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "near"])
          else:
                spatial_relations.append(["so
                uth", building[0],
                math.sqrt((building[1][0] -
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "not near"])
    if (isLeft(building[1], building[3], [x,
    y]) == True) and (isLeft(building[1],
    [building[4][0], building[3][1]], [x, y])
    == False):
          if math.sqrt((building[3][1] -
          y)*(building[3][1] - y)) <= 10 *
          building[2] / 2500:
                spatial_relations.append(["no
                rth", building[0],
                math.sqrt((building[1][0] -
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "near"])
          else:
                spatial_relations.append(["no
                rth", building[0],
                math.sqrt((building[1][0] -
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "not near"])
    if (isLeft(building[1], building[3], [x,
    y]) == True) and (isLeft(building[1],
    [building[4][0], building[3][1]], [x, y])
    == True):
          if math.sqrt((x -
          building[4][0])*(x -
          building[4][0])) <= 10 * building[2]
          / 2500:
                spatial_relations.append(["ea
                st", building[0],
                math.sqrt((building[1][0] -
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "near"])
          else:
                spatial_relations.append(["ea
                st", building[0],
                math.sqrt((building[1][0] -
                x)*(building[1][0] - x) +
                (building[1][1] -
                y)*(building[1][1] - y)),
                "not near"])
```

```
            if (isLeft(building[1], building[3], [x,
            y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [x, y])
            == False):
                  if math.sqrt((building[3][0] -
                  x)*(building[3][0] - x)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])
#These temporary variables help to find the
most immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations
and stores the most immediate relation
(nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture)
to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
```

```
                    distance_W = item[2]
                    near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                    building_E = item[1]
                    distance_E = item[2]
                    near_E = item[3]

#This temporary variable used to check points
with same relation descriptions.
new_point_stack = ""

#These if-statements update and save the most
immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture) from temporary variables
to descriptions list and point_stack string
variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            new_point_stack = new_point_stack +
            "near_north" + building_N
      else:
            new_point_stack = new_point_stack +
            "north" + building_N
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            new_point_stack = new_point_stack +
            "near_south" + building_S
      else:
            new_point_stack = new_point_stack +
            "south" + building_S
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
            new_point_stack = new_point_stack +
            "near_west" + building_W
      else:
            new_point_stack = new_point_stack +
            "west" + building_W
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
            new_point_stack = new_point_stack +
            "near_east" + building_E
      else:
            new_point_stack = new_point_stack +
            "east" + building_E

#This if-statement will check points with same
relation descriptions to the mouse click point
and draw red circles on a temporary campus
image.
if new_point_stack == point_stack:
      overlap = False
      for h,cnt in enumerate(contours):
            if cv2.pointPolygonTest(cnt,(x,
            y),True) >= 0:
                  overlap = True
      if overlap == False:
```

```
                                cv2.circle(campus_region,(x,
                                y),1,[0,0,255],-1)
                        y += 1
                x += 1

        #Save the modified image to image_with_points.png file.
        cv2.imwrite('image_with_points.png', campus_region)
#Read the modified image and replace the label image of the GUI.
As a result, the red circles will be displayed on the map.
photo = PhotoImage(file='image_with_points.png')
w.config(image=photo)
w.photo = photo
```

Here are the testing results:

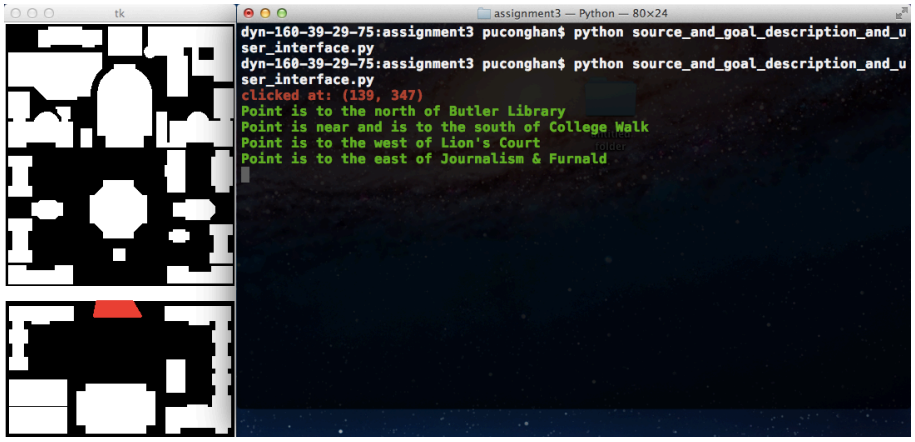


Figure 11: Testing results for point (139, 347). This point is near and is to the south of College Walk, to the north of Butler library, to the west of Lion's Court and to the east of Journalism & Furnald.

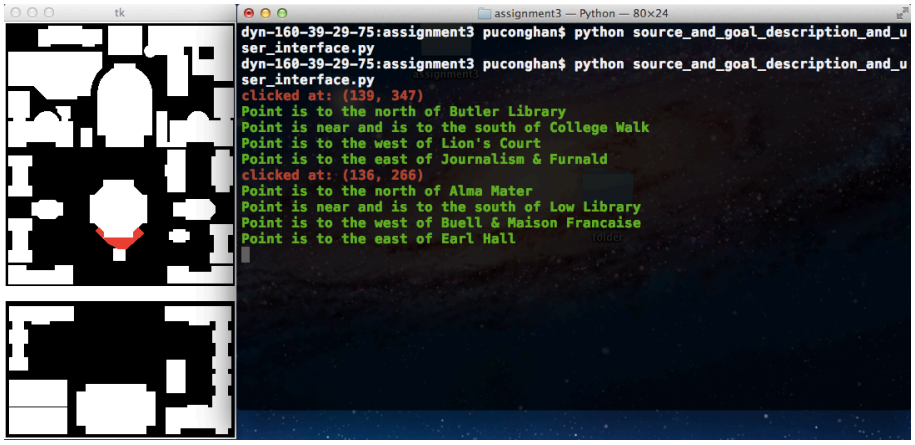


Figure 12: Testing results for point (136, 266). This point is near and is to the south of Low Library, to the north of Alma Mater, to the west of Buell & Maison Francaise and to the east of Earl Hall.

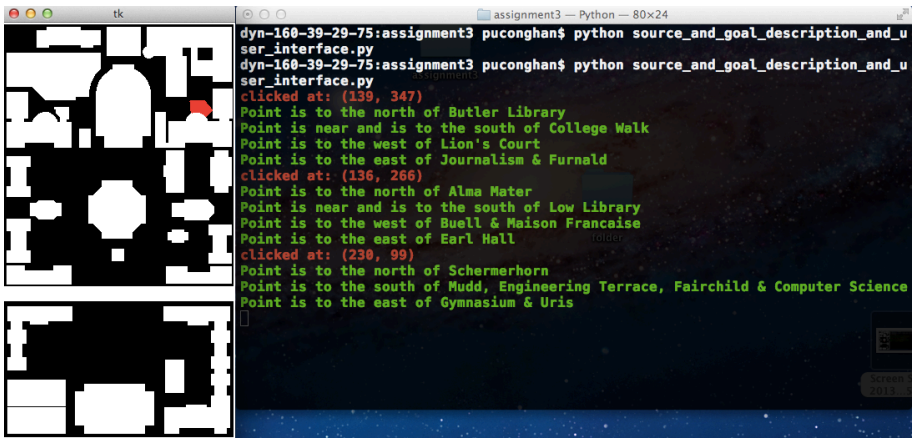


Figure 13: Testing results for point (230, 99). This point is to the south of Mudd, to the north of Schermerhorn, and to the east of Gymnasium & Uris.

As we can see from these testing results, the most immediate relational descriptions of the location of the points are accurately displayed. All pixels in this equivalence class are colored in the map in red.

Points have the **smallest cloud** are between Carman and Lerner Hall, as shown in Figure 14. This region is hard to be selected.

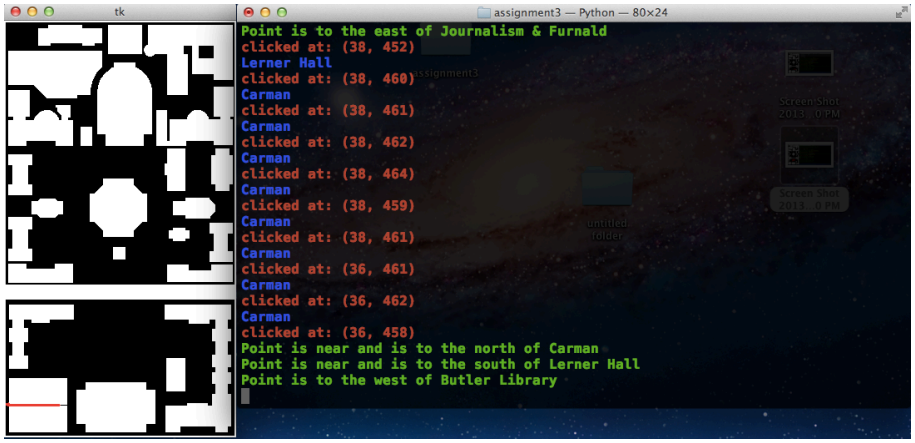


Figure 14: Points have the smallest cloud.

Points that have the **biggest cloud** are to the north of Butler Library, to the south of College Walk, to the west of Lion's Court and to the east of Journalism & Furnald (The grass area in front of Butler library), as shown in Figure 15.

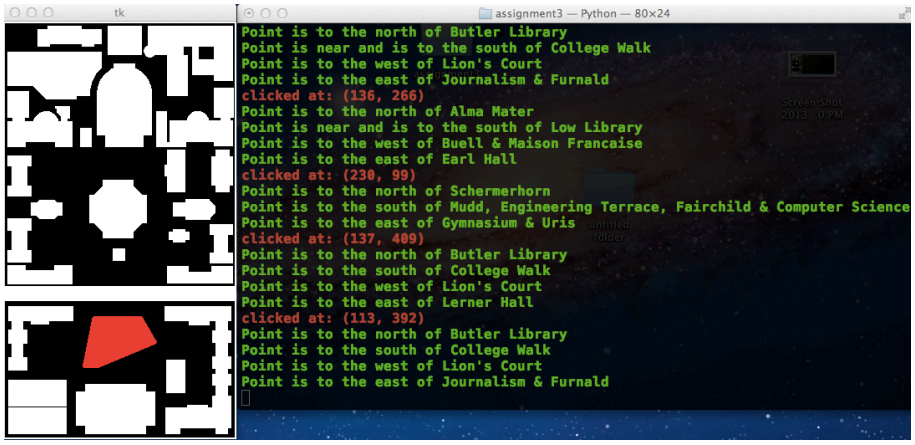


Figure 15: Points have the biggest cloud.

After this step, we can expand step three to capture an S (source) and a G (destination). I basically repeat the previous step in a new callback function following the first callback function triggered by right clicks:

```
def callback2(event):
      ***

w.bind("<Button-2>", callback2)
```

I changed the circle drawing color in the first callback function to green:

```
      cv2.circle(campus_region,(x, y),1,[0,255,0],-1)
```

Here are my implementations for the second callback function:

```
def callback2(event):
      #[Testing] Print the mouse click x and y values.
      print colored("clicked at: " + "(" + str(event.x) + ", " +
      str(event.y) + ")", "red")

      # The following code computes the descriptions for the locations.
      contour_fall = ""
      contour_name = ""
      descritions = []
      for h,cnt in enumerate(contours):
            #If the point is in any building contours, codes within
            this if-statement will print the building name.
            if cv2.pointPolygonTest(cnt,(event.x, event.y),True) >= 0:
                  contour_fall = cnt
                  count = 0
                  integer_value = 0
                  for demensions in cnt:
                        for values in demensions:
                              integer_value +=
                              integer_value_campus.getpixel((int(values
                              [0]), int(values[1])))
                              count += 1
                              #Get the name of the building by passing
                              the variable to the hashtable label.
```

```
                        contour_name = label[integer_value /
                        count]
            print colored(contour_name, 'blue')
#If the point is not in any building contours, codes within this
if-statement compute the location description of the point.
if contour_fall == "":
      #Storing spatial relations of the point to all buildings.
      This is a temporary variable.
      spatial_relations = []
      #This for-loop goes over all buildings in the
      building_properties variable (including a list of
      building_name, building_contour_center,
      building_contour_area, building_contour_upper_left,
      building_contour_lower_right, building_shape,
      building_size_description,
      building_location_extrema_description).
      for building in building_properties:
            #These if-and-else-statement checks and store
            building names and relations to the points to the
            spatial_relations list variable.
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == True):
                  if math.sqrt((event.y -
                  building[4][1])*(event.y - building[4][1])) <=
                  10 * building[2] / 2500:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == False):
                  if math.sqrt((building[3][1] -
                  event.y)*(building[3][1] - event.y)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
```

```
      [building[4][0], building[3][1]], [event.x, event.y])
      == True):
            if math.sqrt((event.x -
            building[4][0])*(event.x - building[4][0])) <=
            10 * building[2] / 2500:
                  spatial_relations.append(["east",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["east",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
      if (isLeft(building[1], building[3], [event.x,
      event.y]) == False) and (isLeft(building[1],
      [building[4][0], building[3][1]], [event.x, event.y])
      == False):
            if math.sqrt((building[3][0] -
            event.x)*(building[3][0] - event.x)) <= 10 *
            building[2] / 2500:
                  spatial_relations.append(["west",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["west",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""
#This for-loop goes over the spatial_relations and stores
the most immediate relation (nearest/ north/ south/ west/
east) of a point (with shortest distance to a building
gesture) to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
```

```
                near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check all other points
with same relation descriptions.
point_stack = ""

#These if-statements update and save the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture) from
temporary variables to descriptions list and point_stack
string variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            descritions.append("Point is near and is to the
            north of " + building_N)
            point_stack = point_stack + "near_north" +
            building_N
      else:
            descritions.append("Point is to the north of "
            + building_N)
            point_stack = point_stack + "north" +
            building_N
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            descritions.append("Point is near and is to the
            south of " + building_S)
            point_stack = point_stack + "near_south" +
            building_S
      else:
            descritions.append("Point is to the south of "
            + building_S)
            point_stack = point_stack + "south" +
            building_S
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
            descritions.append("Point is near and is to the
            west of " + building_W)
            point_stack = point_stack + "near_west" +
            building_W
      else:
            descritions.append("Point is to the west of " +
            building_W)
            point_stack = point_stack + "west" + building_W
```

```
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
            descritions.append("Point is near and is to the
            east of " + building_E)
            point_stack = point_stack + "near_east" +
            building_E
      else:
            descritions.append("Point is to the east of " +
            building_E)
            point_stack = point_stack + "east" + building_E
#This for-loop prints all immediate relation (nearest/
north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from the descriptions list.
for item in descritions:
      print colored(item, 'green')

# The following code finds and draws all points sharing
similar location descriptions.

#Open the integer value campus image
campus_region = cv2.imread("image_with_points.png")

#The following while-loops go over all pixels in the image.
x = 0
while x <= integer_value_campus.size[0]:
      y = 0
      while y <= integer_value_campus.size[1]:
            #Similar to the previous portion, these loops
            and if-statement compute all relational
            descriptions of a point to all building
            contours.
            spatial_relations = []
            for building in building_properties:
                  if (isLeft(building[1], building[3], [x,
                  y]) == False) and (isLeft(building[1],
                  [building[4][0], building[3][1]], [x, y])
                  == True):
                        if math.sqrt((y -
                        building[4][1])*(y -
                        building[4][1])) <= 10 * building[2]
                        / 2500:
                              spatial_relations.append(["so
                              uth", building[0],
                              math.sqrt((building[1][0] -
                              x)*(building[1][0] - x) +
                              (building[1][1] -
                              y)*(building[1][1] - y)),
                              "near"])
                        else:
                              spatial_relations.append(["so
                              uth", building[0],
                              math.sqrt((building[1][0] -
                              x)*(building[1][0] - x) +
                              (building[1][1] -
                              y)*(building[1][1] - y)),
                              "not near"])
                  if (isLeft(building[1], building[3], [x,
                  y]) == True) and (isLeft(building[1],
```

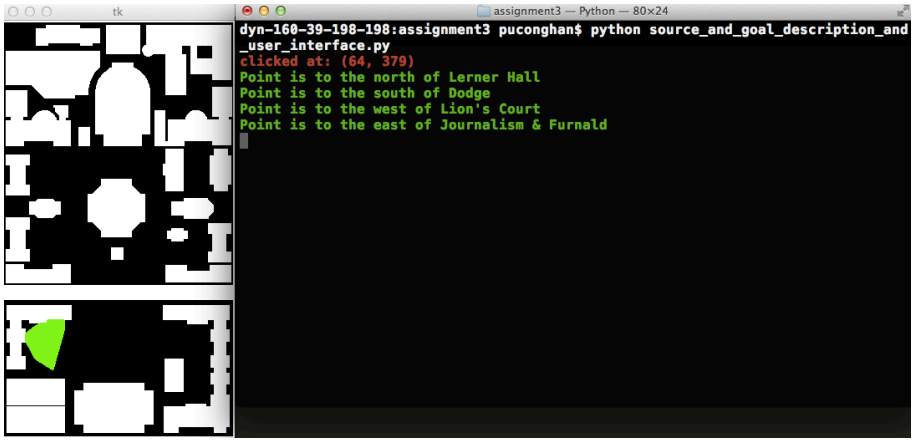
```
[building[4][0], building[3][1]], [x, y])
== False):
      if math.sqrt((building[3][1] -
      y)*(building[3][1] - y)) <= 10 *
      building[2] / 2500:
            spatial_relations.append(["no
            rth", building[0],
            math.sqrt((building[1][0] -
            x)*(building[1][0] - x) +
            (building[1][1] -
            y)*(building[1][1] - y)),
            "near"])
      else:
            spatial_relations.append(["no
            rth", building[0],
            math.sqrt((building[1][0] -
            x)*(building[1][0] - x) +
            (building[1][1] -
            y)*(building[1][1] - y)),
            "not near"])
if (isLeft(building[1], building[3], [x,
y]) == True) and (isLeft(building[1],
[building[4][0], building[3][1]], [x, y])
== True):
      if math.sqrt((x -
      building[4][0])*(x -
      building[4][0])) <= 10 * building[2]
      / 2500:
            spatial_relations.append(["ea
            st", building[0],
            math.sqrt((building[1][0] -
            x)*(building[1][0] - x) +
            (building[1][1] -
            y)*(building[1][1] - y)),
            "near"])
      else:
            spatial_relations.append(["ea
            st", building[0],
            math.sqrt((building[1][0] -
            x)*(building[1][0] - x) +
            (building[1][1] -
            y)*(building[1][1] - y)),
            "not near"])
if (isLeft(building[1], building[3], [x,
y]) == False) and (isLeft(building[1],
[building[4][0], building[3][1]], [x, y])
== False):
      if math.sqrt((building[3][0] -
      x)*(building[3][0] - x)) <= 10 *
      building[2] / 2500:
            spatial_relations.append(["we
            st", building[0],
            math.sqrt((building[1][0] -
            x)*(building[1][0] - x) +
            (building[1][1] -
            y)*(building[1][1] - y)),
            "near"])
      else:
```

```
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])

#These temporary variables help to find the
most immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations
and stores the most immediate relation
(nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture)
to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check points
with same relation descriptions.
new_point_stack = ""

#These if-statements update and save the most
immediate relation (nearest/ north/ south/
```

```
                        west/ east) of a point (with shortest distance
                        to a building gesture) from temporary variables
                        to descriptions list and point_stack string
                        variable.
                        if distance_N != math.sqrt(495*495+275*275):
                              if near_N == "near":
                                    new_point_stack = new_point_stack +
                                    "near_north" + building_N
                              else:
                                    new_point_stack = new_point_stack +
                                    "north" + building_N
                        if distance_S != math.sqrt(495*495+275*275):
                              if near_S == "near":
                                    new_point_stack = new_point_stack +
                                    "near_south" + building_S
                              else:
                                    new_point_stack = new_point_stack +
                                    "south" + building_S
                        if distance_W != math.sqrt(495*495+275*275):
                              if near_W == "near":
                                    new_point_stack = new_point_stack +
                                    "near_west" + building_W
                              else:
                                    new_point_stack = new_point_stack +
                                    "west" + building_W
                        if distance_E != math.sqrt(495*495+275*275):
                              if near_E == "near":
                                    new_point_stack = new_point_stack +
                                    "near_east" + building_E
                              else:
                                    new_point_stack = new_point_stack +
                                    "east" + building_E

                        #This if-statement will check points with same
                        relation descriptions to the mouse click point
                        and draw red circles on a temporary campus
                        image.
                        if new_point_stack == point_stack:
                              overlap = False
                              for h,cnt in enumerate(contours):
                                    if cv2.pointPolygonTest(cnt,(x,
                                    y),True) >= 0:
                                          overlap = True
                              if overlap == False:
                                    cv2.circle(campus_region,(x,
                                    y),1,[0,0,255],-1)
                        y += 1
                  x += 1
            #Save the modified image to image_with_points.png file.
            cv2.imwrite('image_with_points.png', campus_region)

      #Read the modified image and replace the label image of the GUI.
      As a result, the red circles will be displayed on the map.
      photo = PhotoImage(file='image_with_points.png')
      w.config(image=photo)
      w.photo = photo
```

The only difference of this callback function is that it reads in the campus image saved by the first callback function. As a result, this second callback function will draw a second class of points in red color on the same campus image. Here are my three testing results (Figure 16, Figure 17 and Figure 18):

Figure 16: Testing result for S and G (Green cluster is S and red cluster is G).



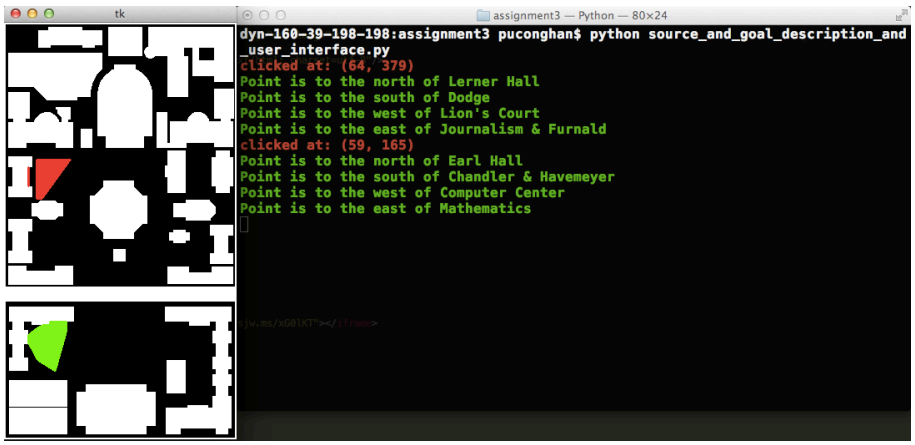


Figure 17: Testing result for S and G (Green cluster is S and red cluster is G).

Figure 18: Testing result for S and G (Green cluster is S and red cluster is G).

As we can see from these testing results, the most immediate relational descriptions of the location of the S and G points are accurately displayed. All pixels in this equivalence class of S are colored in the map in green. All pixels in this equivalence class of G are colored in the map in red.

Step 4: Creativity - Path Generation

I implemented the creativity portion of this assignment using the results of previous three steps, particularly the spatial relation results from step 2.

Using the implementations from step 2, my creativity.py program builds a reduced spatial relation list:

```
#Function to determine whether point a is to the left of line through
points b and c.
def isLeft(a, b, c):
      if ((b[0] - a[0])*(c[1] - a[1]) - (b[1] - a[1])*(c[0] - a[0])) >
      0:
            return True
      else:
            return False

#Create a hashtable, in Python it is called 'dictionaries' or
'associative arrays,' associates the building numbers with the building
names.
label = {}
label[1]="Pupin"
label[2]="Schapiro CEPSR"
label[3]="Mudd, Engineering Terrace, Fairchild & Computer Science"
label[4]="Physical Fitness Center"
label[5]="Gymnasium & Uris"
label[6]="Schermerhorn"
label[7]="Chandler & Havemeyer"
label[8]="Computer Center"
label[9]="Avery"
label[10]="Fayerweather"
label[11]="Mathematics"
label[12]="Low Library"
label[13]="St. Paul's Chapel"
label[14]="Earl Hall"
label[15]="Lewisohn"
label[16]="Philosophy"
label[17]="Buell & Maison Francaise"
label[18]="Alma Mater"
label[19]="Dodge"
label[20]="Kent"
label[21]="College Walk"
label[22]="Journalism & Furnald"
label[23]="Hamilton, Hartley, Wallach & John Jay"
label[24]="Lion's Court"
label[25]="Lerner Hall"
label[26]="Butler Library"
label[27]="Carman"

#Create a list storing the contours of default shapes.
shapes = []
for shape_image in glob.glob( os.path.join("shapes", '*.png') ):
      shape = cv2.imread(shape_image)
      shape_gray = cv2.cvtColor(shape,cv2.COLOR_BGR2GRAY)
      ret_shape, thresh_shape = cv2.threshold(shape_gray,127,255,0)
      contours_shape, hierarchy_shape =
cv2.findContours(thresh_shape,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
      shapes.append([contours_shape[0], shape_image.replace("shapes/",
"").replace(".png", "").replace("_", " ")])
```

```
#Variables for storing temporary building properties.
building_properties = []
building_relations = {}
building_distance = {}
#Variables for storing reduced spatial relations.
reduced_relations = []
#Variables for storing routes.
route = []
#Variables for storing start and destination variables.
start_and_destination = [[], []]

#Open the integer value campus image.
integer_value_campus = Image.open("ass3-labeled.pgm")
#Open the campus image.
campus = cv2.imread("ass3-campus.pgm")
#Convert campus image to gray scale.
campus_gray = cv2.cvtColor(campus,cv2.COLOR_BGR2GRAY)
#Create binary images for the campus.
ret, thresh = cv2.threshold(campus_gray,127,255,0)
#Compute contours (boundaries) of buildings on campus.
contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

#For each building countour, this for-loop assigns proper descriptions
(shapes, sizes and location).
For h,cnt in enumerate(contours):
      building_name = “”
      building_shape = “”
      building_size_description = “”
      building_contour_center = []
      building_contour_area = 0
      building_contour_upper_left = []
      building_contour_lower_right = []

      #Get the integer value from the integer value campus image.
      count = 0
      xValue = 0
      yValue = 0
      integer_value = 0
      for demensions in cnt:
            for values in demensions:
                  integer_value +=
                  integer_value_campus.getpixel((int(values[0]),
                  int(values[1])))
                  xValue += values[0]
                  yValue += values[1]
                  count += 1
      xCenter = xValue / count
      yCenter = yValue / count

      building_contour_center = [xCenter, yCenter]

      x,y,w,h = cv2.boundingRect(cnt)

      building_contour_upper_left = [x, y]
      building_contour_lower_right = [x+w, y+h]
```

```
      #Get the name of the building from the hash table.
      building_name = label[integer_value / count]
      #Compute the contour area.
      building_contour_area = cv2.contourArea(cnt)

      #Compare with the coutours in the shape list and match shape
      description
      shape_name = shapes[0][1]
      shape_value = cv2.matchShapes(shapes[0][0], cnt,
      cv2.cv.CV_CONTOURS_MATCH_I1, 0)
      for item in shapes:
            if shape_value > cv2.matchShapes(item[0], cnt,
            cv2.cv.CV_CONTOURS_MATCH_I1, 0):
                  shape_value = cv2.matchShapes(item[0], cnt,
                  cv2.cv.CV_CONTOURS_MATCH_I1, 0)
                  shape_name = item[1]
      building_shape = shape_name

      building_properties.append([building_name,
      building_contour_center, building_contour_area,
      building_contour_upper_left, building_contour_lower_right,
      building_shape])

for building in building_properties:
      for building_sub in building_properties:
            if building[0] != building_sub[0]:
                  if (isLeft(building[1], building[3], building_sub[1])
                  == False) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == True):
                        building_relations[building[0], building_sub[0],
                        "relation"] = "north"
                         building_distance[building[0], building_sub[0],
                        "relation"] = math.sqrt((building[1][0] -
                        building_sub[1][0])*(building[1][0] -
                        building_sub[1][0]) + (building[1][1] -
                        building_sub[1][1])*(building[1][1] -
                        building_sub[1][1]))
                         if math.sqrt((building_sub[3][1] -
                        building[4][1])*(building_sub[3][1] -
                        building[4][1])) <= 6 * building[2] / 2500:
                              building_relations[building[0],
                              building_sub[0], "near"] = "near"
                         else:
                              building_relations[building[0],
                              building_sub[0], "near"] =
                              math.sqrt((building_sub[3][1] -
                              building[4][1])*(building_sub[3][1] -
                              building[4][1]))
                  if (isLeft(building[1], building[3], building_sub[1])
                  == True) and (isLeft(building[1], [building[4][0],
                  building[3][1]], building_sub[1]) == False):
                        building_relations[building[0], building_sub[0],
                        "relation"] = "south"
                         building_distance[building[0], building_sub[0],
                        "relation"] = math.sqrt((building[1][0] -
                        building_sub[1][0])*(building[1][0] -
                        building_sub[1][0]) + (building[1][1] -
```

```
     building_sub[1][1])*(building[1][1] -
     building_sub[1][1]))
      if math.sqrt((building[3][1] -
     building_sub[4][1])*(building[3][1] -
     building_sub[4][1])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
if (isLeft(building[1], building[3], building_sub[1])
== True) and (isLeft(building[1], [building[4][0],
building[3][1]], building_sub[1]) == True):
     building_relations[building[0], building_sub[0],
     "relation"] = "west"
      building_distance[building[0], building_sub[0],
     "relation"] = math.sqrt((building[1][0] -
     building_sub[1][0])*(building[1][0] -
     building_sub[1][0]) + (building[1][1] -
     building_sub[1][1])*(building[1][1] -
     building_sub[1][1]))
      if math.sqrt((building_sub[3][0] -
     building[4][0])*(building_sub[3][0] -
     building[4][0])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
if (isLeft(building[1], building[3], building_sub[1])
== False) and (isLeft(building[1], [building[4][0],
building[3][1]], building_sub[1]) == False):
     building_relations[building[0], building_sub[0],
     "relation"] = "east"
      building_distance[building[0], building_sub[0],
     "relation"] = math.sqrt((building[1][0] -
     building_sub[1][0])*(building[1][0] -
     building_sub[1][0]) + (building[1][1] -
     building_sub[1][1])*(building[1][1] -
     building_sub[1][1]))
      if math.sqrt((building[3][0] -
     building_sub[4][0])*(building[3][0] -
     building_sub[4][0])) <= 6 * building[2] / 2500:
            building_relations[building[0],
            building_sub[0], "near"] = "near"
      else:
            building_relations[building[0],
            building_sub[0], "near"] =
            math.sqrt((building_sub[3][1] -
            building[4][1])*(building_sub[3][1] -
            building[4][1]))
```

```
#Reduce the number of spatial relations.
a = 1
while a < 27 :
      b = 1
      target_building_N = ""
      distance_count_N = math.sqrt(495*495+275*275)
      testing_building_N = ""
      building_near_N = ""
      target_building_S = ""
      distance_count_S = math.sqrt(495*495+275*275)
      testing_building_S = ""
      building_near_S = ""
      target_building_W = ""
      distance_count_W = math.sqrt(495*495+275*275)
      testing_building_W = ""
      building_near_W = ""
      target_building_E = ""
      distance_count_E = math.sqrt(495*495+275*275)
      testing_building_E = ""
      building_near_E = ""
      while b < 27 :
            if label[a] != label[b]:
                  if building_relations[label[a], label[b], "relation"]
                  == "north":
                        if distance_count_N >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_N =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_N = label[b]
                              testing_building_N = label[a]
                              if building_relations[label[a], label[b],
                              "near"] == "near":
                                    building_near_N = "near"
                  if building_relations[label[a], label[b], "relation"]
                  == "south":
                        if distance_count_S >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_S =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_S = label[b]
                              testing_building_S = label[a]
                              if building_relations[label[a], label[b],
                              "near"] == "near":
                                    building_near_S = "near"
                  if building_relations[label[a], label[b], "relation"]
                  == "west":
                        if distance_count_W >=
                        building_distance[label[a], label[b],
                        "relation"]:
                              distance_count_W =
                              building_distance[label[a], label[b],
                              "relation"]
                              target_building_W = label[b]
                              testing_building_W = label[a]
```

```
                        if building_relations[label[a], label[b],
                        "near"] == "near":
                              building_near_W = "near"
            if building_relations[label[a], label[b], "relation"]
            == "east":
                  if distance_count_E >=
                  building_distance[label[a], label[b],
                  "relation"]:
                        distance_count_E =
                        building_distance[label[a], label[b],
                        "relation"]
                        target_building_E = label[b]
                        testing_building_E = label[a]
                        if building_relations[label[a], label[b],
                        "near"] == "near":
                              building_near_E = "near"
      b += 1
a += 1

#Further reduce the number of spatial relations.
temp_list = []
target_building = ""
if distance_count_N != math.sqrt(495*495+275*275):
      if building_near_N == "near":
            temp_list.append(["south" , target_building_N,
            "near"])
      else:
            temp_list.append(["south" , target_building_N, "not
            near"])
      target_building = testing_building_N
if distance_count_S != math.sqrt(495*495+275*275):
      if building_near_S == "near":
            temp_list.append(["north" , target_building_S,
            "near"])
      else:
            temp_list.append(["north" , target_building_S, "not
            near"])
      target_building = testing_building_S
if distance_count_W != math.sqrt(495*495+275*275):
      if building_near_W == "near":
           temp_list.append(["east" , target_building_W, "near"])
      else:
            temp_list.append(["east" , target_building_W, "not
            near"])
      target_building = testing_building_W
if distance_count_E != math.sqrt(495*495+275*275):
      if building_near_E == "near":
           temp_list.append(["west" , target_building_E, "near"])
      else:
            temp_list.append(["west" , target_building_E, "not
            near"])
      target_building = testing_building_E
#Save reduced spatial relations into the reduced_relations list.
if target_building != "":
      if not reduced_relations:
           reduced_relations.append([target_building,  temp_list])
      else:
            existance = False
```

```
for item in reduced_relations:
      if item[0] == target_building:
            existance = True
if existance == False:
      reduced_relations.append([target_building,
      temp_list])
```

The above code builds a further reduced relation list. I use the following testing code to test the reduced relations:

```
for item in reduced_relations:
      print item
      print ""
```

The printed results are:

```
dyn-160-39-29-75:assignment3 puconghan$ python creativity.py
['Pupin', [['south', 'Physical Fitness Center', 'not near'], ['east', 'Schapiro CEPSR', 'not near']]]

['Schapiro CEPSR', [['south', 'Gymnasium & Uris', 'not near'], ['east', 'Mudd, Engineering Terrace, Fairchild & Computer Science', 'not near'
], ['west', 'Pupin', 'not near']]]

['Mudd, Engineering Terrace, Fairchild & Computer Science', [['south', 'Schermerhorn', 'not near'], ['west', 'Schapiro CEPSR', 'not near']]]

['Physical Fitness Center', [['south', 'Computer Center', 'not near'], ['north', 'Pupin', 'near'], ['east', 'Gymnasium & Uris', 'near'], ['we
st', 'Chandler & Havemeyer', 'not near']]]

['Gymnasium & Uris', [['south', 'Computer Center', 'not near'], ['north', 'Schapiro CEPSR', 'near'], ['east', 'Schermerhorn', 'near'], ['west
', 'Physical Fitness Center', 'near']]]

['Schermerhorn', [['south', 'Fayerweather', 'near'], ['north', 'Mudd, Engineering Terrace, Fairchild & Computer Science', 'not near'], ['west
', 'Gymnasium & Uris', 'near']]]

['Chandler & Havemeyer', [['south', 'Mathematics', 'not near'], ['north', 'Physical Fitness Center', 'not near'], ['east', 'Computer Center',
 'not near']]]

['Computer Center', [['south', 'Low Library', 'not near'], ['north', 'Physical Fitness Center', 'not near'], ['east', 'Gymnasium & Uris', 'no
t near'], ['west', 'Chandler & Havemeyer', 'not near']]]

['Avery', [['south', 'Buell & Maison Francaise', 'not near'], ['north', 'Schermerhorn', 'not near'], ['east', "St. Paul's Chapel", 'not near'
], ['west', 'Low Library', 'not near']]]

['Fayerweather', [['south', 'Philosophy', 'not near'], ['north', 'Schermerhorn', 'not near'], ['west', "St. Paul's Chapel", 'not near']]]

['Mathematics', [['south', 'Lewisohn', 'not near'], ['north', 'Chandler & Havemeyer', 'not near'], ['east', 'Earl Hall', 'near']]]

['Low Library', [['south', 'Alma Mater', 'not near'], ['north', 'Computer Center', 'not near'], ['east', 'Avery', 'not near'], ['west', 'Earl
 Hall', 'not near']]]

["St. Paul's Chapel", [['south', 'Buell & Maison Francaise', 'not near'], ['north', 'Avery', 'not near'], ['west', 'Low Library', 'not near']
]]

['Earl Hall', [['south', 'Lewisohn', 'near'], ['north', 'Mathematics', 'not near'], ['east', 'Low Library', 'not near']]]

['Lewisohn', [['south', 'Journalism & Furnald', 'not near'], ['north', 'Mathematics', 'not near'], ['east', 'Dodge', 'near']]]

['Philosophy', [['south', 'Hamilton, Hartley, Wallach & John Jay', 'not near'], ['north', 'Fayerweather', 'not near'], ['west', 'Kent', 'not
near']]]

['Buell & Maison Francaise', [['south', 'Kent', 'not near'], ['north', "St. Paul's Chapel", 'not near'], ['east', 'Philosophy', 'not near'],
['west', 'Alma Mater', 'not near']]]

['Alma Mater', [['south', 'College Walk', 'not near'], ['north', 'Low Library', 'not near'], ['east', 'Buell & Maison Francaise', 'not near']
, ['west', 'Dodge', 'not near']]]

['Dodge', [['south', 'Journalism & Furnald', 'near'], ['north', 'Lewisohn', 'near'], ['east', 'Kent', 'not near']]]

['Kent', [['south', 'College Walk', 'near'], ['north', 'Philosophy', 'near'], ['west', 'Dodge', 'not near']]]

['College Walk', [['south', "Lion's Court", 'near'], ['north', 'Alma Mater', 'not near']]]

['Journalism & Furnald', [['south', 'Lerner Hall', 'not near'], ['north', 'Dodge', 'not near'], ['east', 'College Walk', 'not near']]]

['Hamilton, Hartley, Wallach & John Jay', [['north', 'Kent', 'not near'], ['west', "Lion's Court", 'not near']]]

["Lion's Court", [['north', 'Kent', 'not near'], ['east', 'Hamilton, Hartley, Wallach & John Jay', 'not near'], ['west', 'Butler Library', 'n
ot near']]]

['Lerner Hall', [['north', 'Journalism & Furnald', 'not near'], ['east', 'Butler Library', 'not near']]]

['Butler Library', [['north', 'College Walk', 'not near'], ['east', "Lion's Court", 'near'], ['west', 'Lerner Hall', 'near']]]
```

Figure 19: Printed testing results.

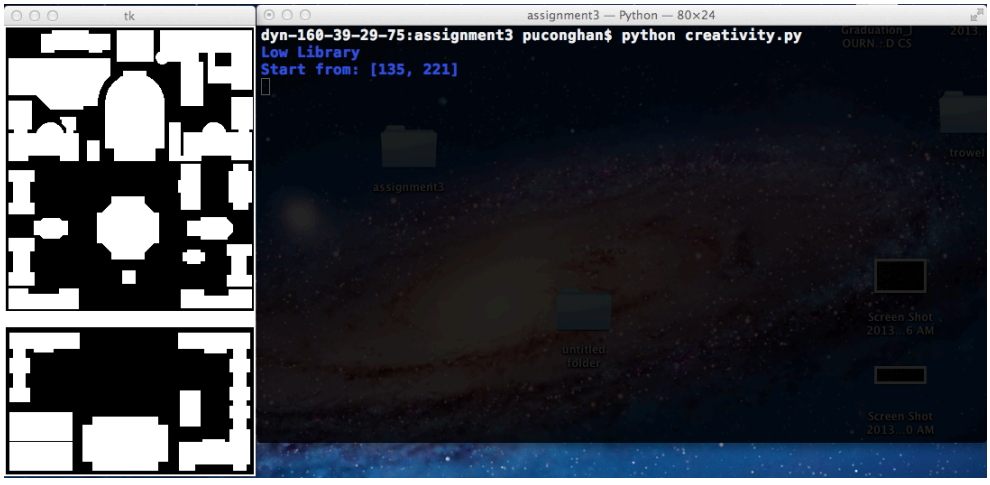
As we can see from Figure 19, there are less than four relations describing each of the 27-targeted buildings on campus. The first element of the reduced_relations list is the targeted building name. The second element of the list is a sub-list containing (less than or equal to four) sub-lists to describe spatial relations toward other buildings. This reduced_relations list is the ground of my creativity.py application. Further analyses take advantages of these results.

After building this reduced spatial relations, I implemented a GUI interface using the Tkinter GUI library. The GUI interface implementation is from step 3:

```
# Importing Tkinter GUI library
from Tkinter import *

master = Tk()

# Callback function for mouse clicks
def callback(event):
      ***

def callback2(event):
      ***

def key(event):
      ***

photo = PhotoImage(file="ass3-campus.pgm")
w = Label(None, image=photo)
w.photo = photo
w.bind("<Return>", key)
w.focus()
w.bind("<Button-1>", callback)
w.bind("<Button-2>", callback2)
w.pack()
```

The graphic user interface looks like the one in Figure 10. The first callback function captures left mouse click. The second callback function captures right mouse click. The key callback function captures the return button on keyboard.

Similar to the first callback function in step 3, the first callback function, callback(event), for creativity.py finds the most immediate relational descriptions of the point (x and y values) triggered by a left mouse click. If a building gesture on the campus map is selected, my program will display the location and the name of the building in the console, as shown in Figure 20. If a point on the street is selected, all pixels in the equivalence class are colored in green, as shown in Figure 20.



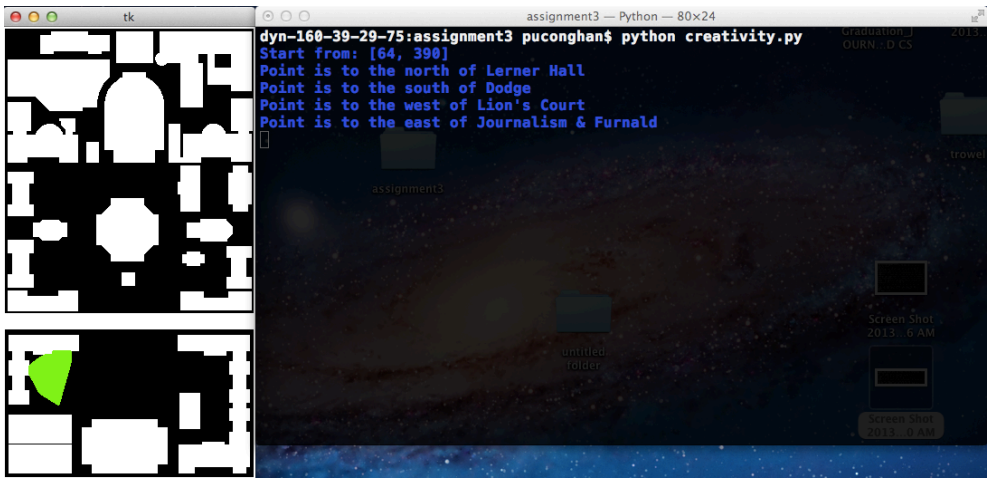


Figure 20: Left mouse click selects a building gesture or a point on the street.

Here are my implementations for the first callback function:

```
#Callback function for mouse left clicks.
def callback(event):
      #Reset the image in a new click.
      #Open the integer value campus image.
      campus_region = cv2.imread("ass3-campus.pgm")
      #Save the modified image to image_with_points.png file.
      cv2.imwrite('image_with_points.png', campus_region)

      #The following code computes the descriptions for the locations.
      contour_fall = ""
      contour_name = ""
      descritions = []
      for h,cnt in enumerate(contours):
            #If the point is in any building contours, codes within
            this if-statement will print the building name.
            if cv2.pointPolygonTest(cnt,(event.x, event.y),True) >= 0:
                  contour_fall = cnt
                  count = 0
                  integer_value = 0
                  for demensions in cnt:
                        for values in demensions:
                              integer_value +=
                              integer_value_campus.getpixel((int(values
                              [0]), int(values[1])))
                              count += 1
                              #Get the name of the building by passing
                              the variable to the hashtable label.
                              contour_name = label[integer_value /
                              count]
                  print colored(contour_name, 'blue')
                  for item in building_properties:
                        if contour_name == item[0]:
                              start_and_destination[0] = [item[1][0],
                              item[1][1]]
                              print colored("Start from: " +
                              str(start_and_destination[0]), 'blue')
```

```
#If the point is not in any building contours, codes within this
if-statement compute the location description of the point.
if contour_fall == "":
      start_and_destination[0] = [event.x, event.y]
      print colored("Start from: " +
      str(start_and_destination[0]), 'blue')
      #Storing spatial relations of the point to all buildings.
      This is a temporary variable.
      spatial_relations = []
      #This for-loop goes over all buildings in the
      building_properties variable.
      for building in building_properties:
            #These if-and-else-statement checks and store
            building names and relations to the points to the
            spatial_relations list variable.
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == True):
                  if math.sqrt((event.y -
                  building[4][1])*(event.y - building[4][1])) <=
                  10 * building[2] / 2500:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == False):
                  if math.sqrt((building[3][1] -
                  event.y)*(building[3][1] - event.y)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == True):
                  if math.sqrt((event.x -
                  building[4][0])*(event.x - building[4][0])) <=
                  10 * building[2] / 2500:
```

```
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
            if (isLeft(building[1], building[3], [event.x,
            event.y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [event.x, event.y])
            == False):
                  if math.sqrt((building[3][0] -
                  event.x)*(building[3][0] - event.x)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "near"])
                  else:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        event.x)*(building[1][0] - event.x) +
                        (building[1][1] - event.y)*(building[1][1]
                        - event.y)), "not near"])
#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""
#This for-loop goes over the spatial_relations and stores
the most immediate relation (nearest/ north/ south/ west/
east) of a point (with shortest distance to a building
gesture) to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
```

```
                near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check all other points
with same relation descriptions.
point_stack = ""

#These if-statements update and save the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture) from
temporary variables to descriptions list and point_stack
string variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            descritions.append("Point is near and is to the
            north of " + building_N)
            point_stack = point_stack + "near_north" +
            building_N
      else:
            descritions.append("Point is to the north of "
            + building_N)
            point_stack = point_stack + "north" +
            building_N
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            descritions.append("Point is near and is to the
            south of " + building_S)
            point_stack = point_stack + "near_south" +
            building_S
      else:
            descritions.append("Point is to the south of "
            + building_S)
            point_stack = point_stack + "south" +
            building_S
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
            descritions.append("Point is near and is to the
            west of " + building_W)
            point_stack = point_stack + "near_west" +
            building_W
      else:
            descritions.append("Point is to the west of " +
            building_W)
            point_stack = point_stack + "west" + building_W
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
            descritions.append("Point is near and is to the
            east of " + building_E)
```

```
                point_stack = point_stack + "near_east" +
                building_E
        else:
                descritions.append("Point is to the east of " +
                building_E)
                point_stack = point_stack + "east" + building_E
#This for-loop prints all immediate relation (nearest/
north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from the descriptions list.
for item in descritions:
        print colored(item, 'blue')

# The following code finds and draws all points sharing
similar location descriptions.

#Improve running efficiency using the following loops. They
go over all point within a region of interest 100 pixel *
100 pixel (mouse pixel in the middle)
x = event.x - 50
while x <= event.x + 50:
        y = event.y - 50
        while y <= event.y + 50:
                #Similar to the previous portion, these loops
                and if-statement compute all relational
                descriptions of a point to all building
                contours.
                spatial_relations = []
                for building in building_properties:
                        if (isLeft(building[1], building[3], [x,
                        y]) == False) and (isLeft(building[1],
                        [building[4][0], building[3][1]], [x, y])
                        == True):
                                if math.sqrt((y -
                                building[4][1])*(y -
                                building[4][1])) <= 10 * building[2]
                                / 2500:
                                        spatial_relations.append(["so
                                        uth", building[0],
                                        math.sqrt((building[1][0] -
                                        x)*(building[1][0] - x) +
                                        (building[1][1] -
                                        y)*(building[1][1] - y)),
                                        "near"])
                                else:
                                        spatial_relations.append(["so
                                        uth", building[0],
                                        math.sqrt((building[1][0] -
                                        x)*(building[1][0] - x) +
                                        (building[1][1] -
                                        y)*(building[1][1] - y)),
                                        "not near"])
                        if (isLeft(building[1], building[3], [x,
                        y]) == True) and (isLeft(building[1],
                        [building[4][0], building[3][1]], [x, y])
                        == False):
                                if math.sqrt((building[3][1] -
                                y)*(building[3][1] - y)) <= 10 *
                                building[2] / 2500:
```

```
                        spatial_relations.append(["no
                        rth", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["no
                        rth", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])
            if (isLeft(building[1], building[3], [x,
            y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [x, y])
            == True):
                  if math.sqrt((x -
                  building[4][0])*(x -
                  building[4][0])) <= 10 * building[2]
                  / 2500:
                        spatial_relations.append(["ea
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["ea
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])
            if (isLeft(building[1], building[3], [x,
            y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [x, y])
            == False):
                  if math.sqrt((building[3][0] -
                  x)*(building[3][0] - x)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
```

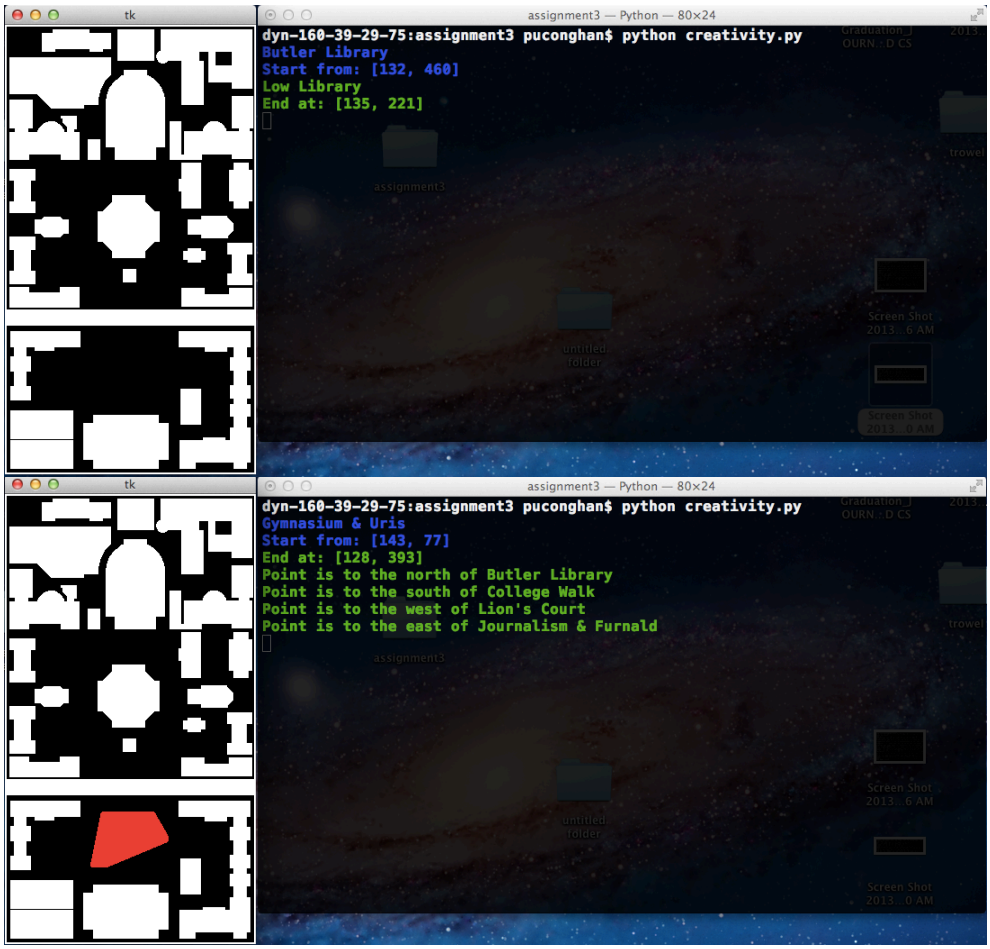
```
                    y)*(building[1][1] - y)),
                    "not near"])

#These temporary variables help to find the
most immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations
and stores the most immediate relation
(nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture)
to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check points
with same relation descriptions.
new_point_stack = ""

#These if-statements update and save the most
immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture) from temporary variables
to descriptions list and point_stack string
variable.
if distance_N != math.sqrt(495*495+275*275):
```

```
                        if near_N == "near":
                              new_point_stack = new_point_stack +
                              "near_north" + building_N
                        else:
                              new_point_stack = new_point_stack +
                              "north" + building_N
                  if distance_S != math.sqrt(495*495+275*275):
                        if near_S == "near":
                              new_point_stack = new_point_stack +
                              "near_south" + building_S
                        else:
                              new_point_stack = new_point_stack +
                              "south" + building_S
                  if distance_W != math.sqrt(495*495+275*275):
                        if near_W == "near":
                              new_point_stack = new_point_stack +
                              "near_west" + building_W
                        else:
                              new_point_stack = new_point_stack +
                              "west" + building_W
                  if distance_E != math.sqrt(495*495+275*275):
                        if near_E == "near":
                              new_point_stack = new_point_stack +
                              "near_east" + building_E
                        else:
                              new_point_stack = new_point_stack +
                              "east" + building_E

                  #This if-statement will check points with same
                  relation descriptions to the mouse click point
                  and draw red circles on a temporary campus
                  image.
                  if new_point_stack == point_stack:
                        overlap = False
                        for h,cnt in enumerate(contours):
                              if cv2.pointPolygonTest(cnt,(x,
                              y),True) >= 0:
                                    overlap = True
                        if overlap == False:
                              cv2.circle(campus_region,(x,
                              y),1,[0,255,0],-1)
                  y += 1
            x += 1
      #Save the modified image to image_with_points.png file.
      cv2.imwrite('image_with_points.png', campus_region)

#Read the modified image and replace the label image of the GUI.
As a result, the red circles will be displayed on the map.
photo = PhotoImage(file='image_with_points.png')
w.config(image=photo)
w.photo = photo
```

My implementations for the first callback function check whether the point is within a building contour or on the street. If the point is within a building contour, the name of the building will be displayed in the console. My first call back function computes the relation descriptions of the point to all building contours in four directions, and then reduced the size of the relations. The spatial results will be displayed in the console. The

start point location will be stored in the global start_and_destination 2D list. By reading documentations, I learned that in Python lists declared at the beginning of the program are global variables that are accessible by the Tkinter callback functions.

The second callback function, callback2(event), for creativity.py is similar to the first callback function finds the most immediate relational descriptions of the point (x and y values) triggered by a right mouse click. If a building gesture on the campus map is selected, my program will display the location and the name of the building in the console, as shown in Figure 21. If a point on the street is selected, all pixels in the equivalence class are colored in red, as shown in Figure 21.



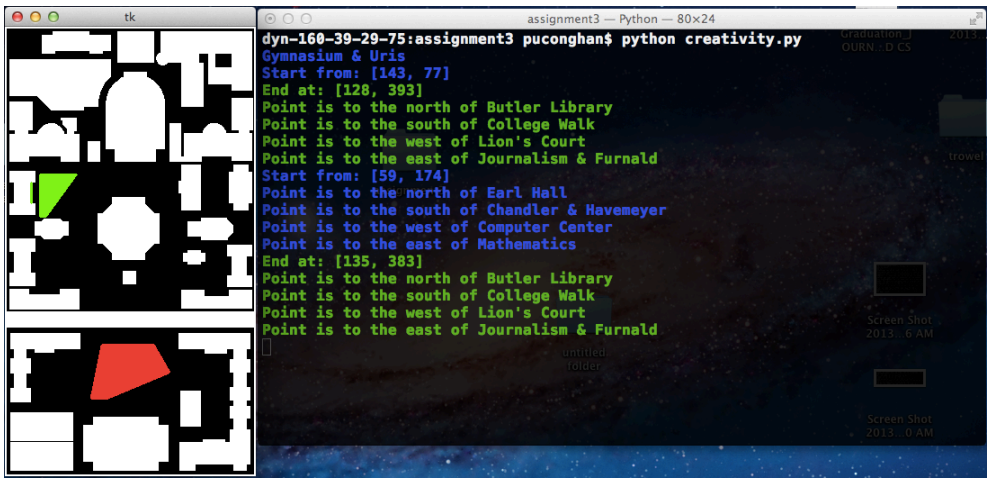


Figure 21: Right mouse clicks selects a building gesture or a point on the street.

Here are my implementations for the second callback function:

```
def callback2(event):
      #The following code computes the descriptions for the locations.
      contour_fall = ""
      contour_name = ""
      descritions = []
      for h,cnt in enumerate(contours):
            #If the point is in any building contours, codes within
            this if-statement will print the building name.
            if cv2.pointPolygonTest(cnt,(event.x, event.y),True) >= 0:
                  contour_fall = cnt
                  count = 0
                  integer_value = 0
                  for demensions in cnt:
                        for values in demensions:
                              integer_value +=
                              integer_value_campus.getpixel((int(values
                              [0]), int(values[1])))
                              count += 1
                              #Get the name of the building by passing
                              the variable to the hashtable label.
                              contour_name = label[integer_value /
                              count]
                  print colored(contour_name, 'green')
                  for item in building_properties:
                        if contour_name == item[0]:
                              start_and_destination[1] = [item[1][0],
                              item[1][1]]
                              print colored("End at: " +
                              str(start_and_destination[1]), 'green')
      #If the point is not in any building contours, codes within this
      if-statement compute the location description of the point.
      if contour_fall == "":
            start_and_destination[1] = [event.x, event.y]
            print colored("End at: " + str(start_and_destination[1]),
            'green')
            #Storing spatial relations of the point to all buildings.
            This is a temporary variable.
```

```
spatial_relations = []
#This for-loop goes over all buildings in the
building_properties variable.
for building in building_properties:
      #These if-and-else-statement checks and store
      building names and relations to the points to the
      spatial_relations list variable.
      if (isLeft(building[1], building[3], [event.x,
      event.y]) == False) and (isLeft(building[1],
      [building[4][0], building[3][1]], [event.x, event.y])
      == True):
            if math.sqrt((event.y -
            building[4][1])*(event.y - building[4][1])) <=
            10 * building[2] / 2500:
                  spatial_relations.append(["south",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["south",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
      if (isLeft(building[1], building[3], [event.x,
      event.y]) == True) and (isLeft(building[1],
      [building[4][0], building[3][1]], [event.x, event.y])
      == False):
            if math.sqrt((building[3][1] -
            event.y)*(building[3][1] - event.y)) <= 10 *
            building[2] / 2500:
                  spatial_relations.append(["north",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["north",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
      if (isLeft(building[1], building[3], [event.x,
      event.y]) == True) and (isLeft(building[1],
      [building[4][0], building[3][1]], [event.x, event.y])
      == True):
            if math.sqrt((event.x -
            building[4][0])*(event.x - building[4][0])) <=
            10 * building[2] / 2500:
                  spatial_relations.append(["east",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["east",
                  building[0], math.sqrt((building[1][0] -
```

```
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
      if (isLeft(building[1], building[3], [event.x,
      event.y]) == False) and (isLeft(building[1],
      [building[4][0], building[3][1]], [event.x, event.y])
      == False):
            if math.sqrt((building[3][0] -
            event.x)*(building[3][0] - event.x)) <= 10 *
            building[2] / 2500:
                  spatial_relations.append(["west",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "near"])
            else:
                  spatial_relations.append(["west",
                  building[0], math.sqrt((building[1][0] -
                  event.x)*(building[1][0] - event.x) +
                  (building[1][1] - event.y)*(building[1][1]
                  - event.y)), "not near"])
#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""
#This for-loop goes over the spatial_relations and stores
the most immediate relation (nearest/ north/ south/ west/
east) of a point (with shortest distance to a building
gesture) to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
```

```
                building_E = item[1]
                distance_E = item[2]
                near_E = item[3]

#This temporary variable used to check all other points
with same relation descriptions.
point_stack = ""

#These if-statements update and save the most immediate
relation (nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture) from
temporary variables to descriptions list and point_stack
string variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            descritions.append("Point is near and is to the
            north of " + building_N)
            point_stack = point_stack + "near_north" +
            building_N
      else:
            descritions.append("Point is to the north of "
            + building_N)
            point_stack = point_stack + "north" +
            building_N
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            descritions.append("Point is near and is to the
            south of " + building_S)
            point_stack = point_stack + "near_south" +
            building_S
      else:
            descritions.append("Point is to the south of "
            + building_S)
            point_stack = point_stack + "south" +
            building_S
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
            descritions.append("Point is near and is to the
            west of " + building_W)
            point_stack = point_stack + "near_west" +
            building_W
      else:
            descritions.append("Point is to the west of " +
            building_W)
            point_stack = point_stack + "west" + building_W
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
            descritions.append("Point is near and is to the
            east of " + building_E)
            point_stack = point_stack + "near_east" +
            building_E
      else:
            descritions.append("Point is to the east of " +
            building_E)
            point_stack = point_stack + "east" + building_E
#This for-loop prints all immediate relation (nearest/
north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from the descriptions list.
```

```
for item in descritions:
      print colored(item, 'green')

# The following code finds and draws all points sharing
similar location descriptions.

#Open the integer value campus image
campus_region = cv2.imread("image_with_points.png")

#Improve running efficiency using the following loops. They
go over all point within a region of interest 100 pixel *
100 pixel (mouse pixel in the middle)
x = event.x - 50
while x <= event.x + 50:
      y = event.y - 50
      while y <= event.y + 50:
            #Similar to the previous portion, these loops
            and if-statement compute all relational
            descriptions of a point to all building
            contours.
            spatial_relations = []
            for building in building_properties:
                  if (isLeft(building[1], building[3], [x,
                  y]) == False) and (isLeft(building[1],
                  [building[4][0], building[3][1]], [x, y])
                  == True):
                        if math.sqrt((y -
                        building[4][1])*(y -
                        building[4][1])) <= 10 * building[2]
                        / 2500:
                              spatial_relations.append(["so
                              uth", building[0],
                              math.sqrt((building[1][0] -
                              x)*(building[1][0] - x) +
                              (building[1][1] -
                              y)*(building[1][1] - y)),
                              "near"])
                        else:
                              spatial_relations.append(["so
                              uth", building[0],
                              math.sqrt((building[1][0] -
                              x)*(building[1][0] - x) +
                              (building[1][1] -
                              y)*(building[1][1] - y)),
                              "not near"])
                  if (isLeft(building[1], building[3], [x,
                  y]) == True) and (isLeft(building[1],
                  [building[4][0], building[3][1]], [x, y])
                  == False):
                        if math.sqrt((building[3][1] -
                        y)*(building[3][1] - y)) <= 10 *
                        building[2] / 2500:
                              spatial_relations.append(["no
                              rth", building[0],
                              math.sqrt((building[1][0] -
                              x)*(building[1][0] - x) +
                              (building[1][1] -
```

```
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["no
                        rth", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])
            if (isLeft(building[1], building[3], [x,
            y]) == True) and (isLeft(building[1],
            [building[4][0], building[3][1]], [x, y])
            == True):
                  if math.sqrt((x -
                  building[4][0])*(x -
                  building[4][0])) <= 10 * building[2]
                  / 2500:
                        spatial_relations.append(["ea
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["ea
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])
            if (isLeft(building[1], building[3], [x,
            y]) == False) and (isLeft(building[1],
            [building[4][0], building[3][1]], [x, y])
            == False):
                  if math.sqrt((building[3][0] -
                  x)*(building[3][0] - x)) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "near"])
                  else:
                        spatial_relations.append(["we
                        st", building[0],
                        math.sqrt((building[1][0] -
                        x)*(building[1][0] - x) +
                        (building[1][1] -
                        y)*(building[1][1] - y)),
                        "not near"])

      #These temporary variables help to find the
      most immediate relation (nearest/ north/ south/
```

```
west/ east) of a point (with shortest distance
to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations
and stores the most immediate relation
(nearest/ north/ south/ west/ east) of a point
(with shortest distance to a building gesture)
to appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#This temporary variable used to check points
with same relation descriptions.
new_point_stack = ""

#These if-statements update and save the most
immediate relation (nearest/ north/ south/
west/ east) of a point (with shortest distance
to a building gesture) from temporary variables
to descriptions list and point_stack string
variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            new_point_stack = new_point_stack +
            "near_north" + building_N
      else:
```

```
                        new_point_stack = new_point_stack +
                        "north" + building_N
                  if distance_S != math.sqrt(495*495+275*275):
                        if near_S == "near":
                              new_point_stack = new_point_stack +
                              "near_south" + building_S
                        else:
                              new_point_stack = new_point_stack +
                              "south" + building_S
                  if distance_W != math.sqrt(495*495+275*275):
                        if near_W == "near":
                              new_point_stack = new_point_stack +
                              "near_west" + building_W
                        else:
                              new_point_stack = new_point_stack +
                              "west" + building_W
                  if distance_E != math.sqrt(495*495+275*275):
                        if near_E == "near":
                              new_point_stack = new_point_stack +
                              "near_east" + building_E
                        else:
                              new_point_stack = new_point_stack +
                              "east" + building_E

                  #This if-statement will check points with same
                  relation descriptions to the mouse click point
                  and draw red circles on a temporary campus
                  image.
                  if new_point_stack == point_stack:
                        overlap = False
                        for h,cnt in enumerate(contours):
                              if cv2.pointPolygonTest(cnt,(x,
                              y),True) >= 0:
                                    overlap = True
                        if overlap == False:
                              cv2.circle(campus_region,(x,
                              y),1,[0,0,255],-1)
                  y += 1
            x += 1
      #Save the modified image to image_with_points.png file.
      cv2.imwrite('image_with_points.png', campus_region)

#Read the modified image and replace the label image of the GUI.
As a result, the red circles will be displayed on the map.
photo = PhotoImage(file='image_with_points.png')
w.config(image=photo)
w.photo = photo
```

My implementation for the second callback function checks whether the point is within a building contour or on the street. If the point is within a building contour, the name of the building will be displayed in the console. My second call back function computes the relation descriptions of the point to all building contours in four directions, and then reduced the size of the relations. The spatial results will be displayed in the console. The end point location will be stored in the global start_and_destination 2D list.

The keyboard callback function computes the shortest routes from the start point (captured by the left mouse click) to the end point (captured by the right mouse click). It calls one helper function called findRoute(start, end, description_start, description_end) and another recursive function called findEachRoute(start, end, short, cornor, orientation, route) to recursively find and display each route.

Here are my implementations for the keyboard callback function:

```
def key(event):
    description_start = []
    description_end = []
    route = []
    if start_and_destination[0] == []:
          print colored("Please select a start point from the campus
          map using a left click", "red")
    if start_and_destination[1] == []:
          print colored("Please select a end point from the campus map
          using a right click", "red")
    else:
          findRoute(start_and_destination[0], start_and_destination[1],
          description_start, description_end)

      if description_start == description_end:
            print colored("Start point and destination point are in the
            same region or building gesture", "red")
      else:
            if start_and_destination[1][0] >=
            start_and_destination[0][0] and start_and_destination[1][1]
            <= start_and_destination[0][1]:
                  for item in description_start:
                        if item[1] == "south":
                              for building in building_properties:
                                    if building[0] == item[2]:
                                          extrema = building[7]
                                         size_description  =  building[6]
                              route.append(["north", item[3], extrema,
                              size_description, item[2]])
                              findEachRoute(item[2],
                              description_end[0][0], item[3],
                              "upper_right", "south", route)
                        if item[1] == "west":
                              for building in building_properties:
                                    if building[0] == item[2]:
                                          extrema = building[7]
                                         size_description  =  building[6]
                              route.append(["east", item[3], extrema,
                              size_description, item[2]])
                              findEachRoute(item[2],
                              description_end[0][0], item[3],
                              "upper_right", "west", route)
            if start_and_destination[1][0] >=
            start_and_destination[0][0] and
            start_and_destination[1][1] >= start_and_destination[0][1]:
                  for item in description_start:
                        if item[1] == "north":
                              for building in building_properties:
```

```
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["south", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "lower_right", "north", route)
            if item[1] == "west":
                  for building in building_properties:
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["east", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "lower_right", "west", route)
if start_and_destination[1][0] <=
start_and_destination[0][0] and start_and_destination[1][1]
<= start_and_destination[0][1]:
      for item in description_start:
            if item[1] == "south":
                  for building in building_properties:
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["north", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "upper_left", "south", route)
            if item[1] == "east":
                  for building in building_properties:
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["west", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "upper_left", "east", route)
if start_and_destination[1][0] <=
start_and_destination[0][0] and
start_and_destination[1][1] >= start_and_destination[0][1]:
      for item in description_start:
            if item[1] == "north":
                  for building in building_properties:
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["south", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "lower_left", "north", route)
            if item[1] == "east":
                  for building in building_properties:
```

```
                        if building[0] == item[2]:
                              extrema = building[7]
                             size_description = building[6]
                  route.append(["west", item[3], extrema,
                  size_description, item[2]])
                  findEachRoute(item[2],
                  description_end[0][0], item[3],
                  "lower_left", "east", route)
```

The return key triggers this keyboard callback function. It reads the start point and the end point from the global list variable start_and_destination[]. This function calls the findRoute(start_and_destination[0], start_and_destination[1], description_start, description_end) to compute descriptions for the start point and the end point.

Here are my implementations for the findRoute() function:

```
def findRoute(start, end, description_start, description_end):
      #If the start point fall into a building contour, this variable
      stores the name of the building.
      start_building_name = ""
      #If the end point fall into a building contour, this variable
      stores the name of the building.
      end_building_name = ""

      #This for-loop checks whether the start and end points fall into
      a building contour. If they are the name of the building will be
      stored to start_building_name and end_building_name variable
      accordingly.
      for h,cnt in enumerate(contours):
            #If the start point is in any building contours, store the
            name in the start_building_name variable.
           if cv2.pointPolygonTest(cnt,(start[0], start[1]),True) >= 0:
                 count = 0
                 integer_value = 0
                 for demensions in cnt:
                       for values in demensions:
                             integer_value +=
                             integer_value_campus.getpixel((int(values
                             [0]), int(values[1])))
                             count += 1
                             #Get the name of the building by passing
                             the variable to the hashtable label.
                             start_building_name = label[integer_value
                             / count]
            #If the end point is in any building contours, store the
            name in the end_building_name variable.
            if cv2.pointPolygonTest(cnt,(end[0], end[1]),True) >= 0:
                 count = 0
                 integer_value = 0
                 for demensions in cnt:
                       for values in demensions:
                             integer_value +=
                             integer_value_campus.getpixel((int(values
                             [0]), int(values[1])))
                             count += 1
```

```
                        #Get the name of the building by passing
                        the variable to the hashtable label.
                        end_building_name = label[integer_value /
                        count]
# Following codes find spatial relations for the start point.
spatial_relations = []
#This for-loop goes over all buildings in the building_properties
variable.
for building in building_properties:
      if building[0] != start_building_name:
            #These if-and-else-statement checks and store
            building names and relations to the points to the
            spatial_relations list variable.
            if (isLeft(building[1], building[3], start) == False)
            and (isLeft(building[1], [building[4][0],
            building[3][1]], start) == True):
                  if math.sqrt((start[1] -
                  building[4][1])*(start[1] - building[4][1])) <=
                  10 * building[2] / 2500:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "near"])
                  else:
                        spatial_relations.append(["south",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "not near"])
            if (isLeft(building[1], building[3], start) == True)
            and (isLeft(building[1], [building[4][0],
            building[3][1]], start) == False):
                  if math.sqrt((building[3][1] -
                  start[1])*(building[3][1] - start[1])) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "near"])
                  else:
                        spatial_relations.append(["north",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "not near"])
            if (isLeft(building[1], building[3], start) == True)
            and (isLeft(building[1], [building[4][0],
            building[3][1]], start) == True):
                  if math.sqrt((start[0] -
                  building[4][0])*(start[0] - building[4][0])) <=
                  10 * building[2] / 2500:
```

```
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "near"])
                  else:
                        spatial_relations.append(["east",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "not near"])
            if (isLeft(building[1], building[3], start) == False)
            and (isLeft(building[1], [building[4][0],
            building[3][1]], start) == False):
                  if math.sqrt((building[3][0] -
                  start[0])*(building[3][0] - start[0])) <= 10 *
                  building[2] / 2500:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "near"])
                  else:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        start[0])*(building[1][0] - start[0]) +
                        (building[1][1] -
                        start[1])*(building[1][1] - start[1])),
                        "not near"])

#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point (with
shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations and stores the
most immediate relation (nearest/ north/ south/ west/ east) of a
point (with shortest distance to a building gesture) to
appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
```

```
                near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#These if-statements update and save the most immediate relation
(nearest/ north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from temporary variables to
descriptions list and point_stack string variable.
if distance_N != math.sqrt(495*495+275*275):
      if near_N == "near":
            description_start.append([start_building_name,
            "north", building_N, "near"])
      else:
            description_start.append([start_building_name,
            "north", building_N, "not near"])
if distance_S != math.sqrt(495*495+275*275):
      if near_S == "near":
            description_start.append([start_building_name,
            "south", building_S, "near"])
      else:
            description_start.append([start_building_name,
            "south", building_S, "not near"])
if distance_W != math.sqrt(495*495+275*275):
      if near_W == "near":
           description_start.append([start_building_name, "west",
           building_W, "near"])
      else:
           description_start.append([start_building_name, "west",
           building_W, "not near"])
if distance_E != math.sqrt(495*495+275*275):
      if near_E == "near":
           description_start.append([start_building_name, "east",
           building_E, "near"])
      else:
           description_start.append([start_building_name, "east",
           building_E, "not near"])

# Following codes find spatial relations for the end point.
spatial_relations = []
#This for-loop goes over all buildings in the building_properties
variable.
for building in building_properties:
      if building[0] != end_building_name:
```

```
#These if-and-else-statement checks and store
building names and relations to the points to the
spatial_relations list variable.
if (isLeft(building[1], building[3], end) == False)
and (isLeft(building[1], [building[4][0],
building[3][1]], end) == True):
      if math.sqrt((end[1] - building[4][1])*(end[1]
      - building[4][1])) <= 10 * building[2] / 2500:
            spatial_relations.append(["south",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "near"])
      else:
            spatial_relations.append(["south",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "not near"])
if (isLeft(building[1], building[3], end) == True)
and (isLeft(building[1], [building[4][0],
building[3][1]], end) == False):
      if math.sqrt((building[3][1] -
      end[1])*(building[3][1] - end[1])) <= 10 *
      building[2] / 2500:
            spatial_relations.append(["north",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "near"])
      else:
            spatial_relations.append(["north",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "not near"])
if (isLeft(building[1], building[3], end) == True)
and (isLeft(building[1], [building[4][0],
building[3][1]], end) == True):
      if math.sqrt((end[0] - building[4][0])*(end[0]
      - building[4][0])) <= 10 * building[2] / 2500:
            spatial_relations.append(["east",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "near"])
      else:
            spatial_relations.append(["east",
            building[0], math.sqrt((building[1][0] -
            end[0])*(building[1][0] - end[0]) +
            (building[1][1] - end[1])*(building[1][1]
            - end[1])), "not near"])
if (isLeft(building[1], building[3], end) == False)
and (isLeft(building[1], [building[4][0],
building[3][1]], end) == False):
      if math.sqrt((building[3][0] -
      end[0])*(building[3][0] - end[0])) <= 10 *
      building[2] / 2500:
```

```
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        end[0])*(building[1][0] - end[0]) +
                        (building[1][1] - end[1])*(building[1][1]
                        - end[1])), "near"])
                  else:
                        spatial_relations.append(["west",
                        building[0], math.sqrt((building[1][0] -
                        end[0])*(building[1][0] - end[0]) +
                        (building[1][1] - end[1])*(building[1][1]
                        - end[1])), "not near"])

#These temporary variables help to find the most immediate
relation (nearest/ north/ south/ west/ east) of a point (with
shortest distance to a building gesture).
building_N = ""
distance_N = math.sqrt(495*495+275*275)
near_N = ""
building_S = ""
distance_S = math.sqrt(495*495+275*275)
near_S = ""
building_W = ""
distance_W = math.sqrt(495*495+275*275)
near_W = ""
building_E = ""
distance_E = math.sqrt(495*495+275*275)
near_E = ""

#This for-loop goes over the spatial_relations and stores the
most immediate relation (nearest/ north/ south/ west/ east) of a
point (with shortest distance to a building gesture) to
appropriate temporary variables.
for item in spatial_relations:
      if item[0] == "north":
            if distance_N >= item[2]:
                  building_N = item[1]
                  distance_N = item[2]
                  near_N = item[3]
      if item[0] == "south":
            if distance_S >= item[2]:
                  building_S = item[1]
                  distance_S = item[2]
                  near_S = item[3]
      if item[0] == "west":
            if distance_W >= item[2]:
                  building_W = item[1]
                  distance_W = item[2]
                  near_W = item[3]
      if item[0] == "east":
            if distance_E >= item[2]:
                  building_E = item[1]
                  distance_E = item[2]
                  near_E = item[3]

#These if-statements update and save the most immediate relation
(nearest/ north/ south/ west/ east) of a point (with shortest
distance to a building gesture) from temporary variables to
descriptions list and point_stack string variable.
```

```
      if distance_N != math.sqrt(495*495+275*275):
            if near_N == "near":
                  description_end.append([end_building_name, "north",
                  building_N, "near"])
            else:
                  description_end.append([end_building_name, "north",
                  building_N, "not near"])
      if distance_S != math.sqrt(495*495+275*275):
            if near_S == "near":
                  description_end.append([end_building_name, "south",
                  building_S, "near"])
            else:
                  description_end.append([end_building_name, "south",
                  building_S, "not near"])
      if distance_W != math.sqrt(495*495+275*275):
            if near_W == "near":
                  description_end.append([end_building_name, "west",
                  building_W, "near"])
            else:
                  description_end.append([end_building_name, "west",
                  building_W, "not near"])
      if distance_E != math.sqrt(495*495+275*275):
            if near_E == "near":
                  description_end.append([end_building_name, "east",
                  building_E, "near"])
            else:
                  description_end.append([end_building_name, "east",
                  building_E, "not near"])
```

The results of this helper function are saved into two variables: description_start and description_end. These two variables allow the keyboard function to check whether the two points are in the same region. If the two points are in the same region or same gesture, my program will print a notifincation to the user and stop further analyses. If the two points are in different regions or gestures, the keyboard callback function will take advantages of the findEachRoute() recursive function:

```
#Recursive function for finding and displaying routes.
def findEachRoute(start, end, short, cornor, orientation, route):
      if start != end:
            for item in reduced_relations:
                  if item[0] == start:
                        #This flag allow the function to pop unrelated
                        routes.
                        havetarget = False
                        for related_item in item[1]:
                              #If the building name exists in the list,
                              it will not be considered again.
                              existance = False
                              for item in route:
                                    if item[4] == related_item[1]:
                                          existance = True
                              if existance == False:
                                    if cornor == "upper_right":
                                          if related_item[0] == "north":
                                                for building in
                                                building_properties:
```

```
                        if building[0] ==
                        related_item[1]:
                              extrema =
                              building[7]
                              size_descri
                              ption =
                              building[6]
                        route.append(["north",
                        related_item[2],
                        extrema,
                        size_description,
                        related_item[1]])
                        findEachRoute(related_i
                        tem[1], end,
                        related_item[2],  cornor,
                        "north", route)
                        havetarget = True
                  if related_item[0] == "east":
                        for building in
                        building_properties:
                              if building[0] ==
                              related_item[1]:
                                    extrema =
                                    building[7]
                                    size_descri
                                    ption =
                                    building[6]
                        route.append(["east",
                        related_item[2],
                        extrema,
                        size_description,
                        related_item[1]])
                        findEachRoute(related_i
                        tem[1], end,
                        related_item[2],  cornor,
                        "east", route)
                        havetarget = True
            if cornor == "lower_right":
                  if related_item[0] == "south":
                        for building in
                        building_properties:
                              if building[0] ==
                              related_item[1]:
                                    extrema =
                                    building[7]
                                    size_descri
                                    ption =
                                    building[6]
                        route.append(["south",
                        related_item[2],
                        extrema,
                        size_description,
                        related_item[1]])
                        findEachRoute(related_i
                        tem[1], end,
                        related_item[2],  cornor,
                        "south", route)
                        havetarget = True
```

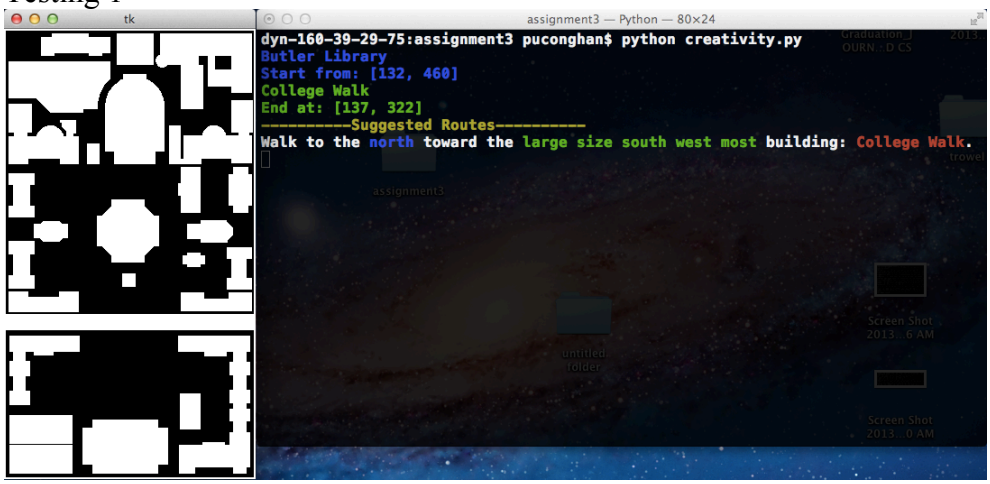
```
            if related_item[0] == "east":
                  for building in
                  building_properties:
                        if building[0] ==
                        related_item[1]:
                              extrema =
                              building[7]
                              size_descri
                              ption =
                              building[6]
                  route.append(["east",
                  related_item[2],
                  extrema,
                  size_description,
                  related_item[1]])
                  findEachRoute(related_i
                  tem[1], end,
                  related_item[2],  cornor,
                  "east", route)
                  havetarget = True
      if cornor == "upper_left":
           if related_item[0] == "north":
                  for building in
                  building_properties:
                        if building[0] ==
                        related_item[1]:
                              extrema =
                              building[7]
                              size_descri
                              ption =
                              building[6]
                  route.append(["north",
                  related_item[2],
                  extrema,
                  size_description,
                  related_item[1]])
                  findEachRoute(related_i
                  tem[1], end,
                  related_item[2],  cornor,
                  "north", route)
                  havetarget = True
            if related_item[0] == "west":
                  for building in
                  building_properties:
                        if building[0] ==
                        related_item[1]:
                              extrema =
                              building[7]
                              size_descri
                              ption =
                              building[6]
                  route.append(["west",
                  related_item[2],
                  extrema,
                  size_description,
                  related_item[1]])
                  findEachRoute(related_i
                  tem[1], end,
```

```
                                    related_item[2],  cornor,
                                    "west", route)
                                    havetarget = True
                              if cornor == "lower_left":
                                   if related_item[0] == "south":
                                    for building in
                                    building_properties:
                                          if building[0] ==
                                          related_item[1]:
                                                extrema =
                                                building[7]
                                                size_descri
                                                ption =
                                                building[6]
                                    route.append(["south",
                                    related_item[2],
                                    extrema,
                                    size_description,
                                    related_item[1]])
                                    findEachRoute(related_i
                                    tem[1], end,
                                    related_item[2],  cornor,
                                    "south", route)
                                    havetarget = True
                               if related_item[0] == "west":
                                    for building in
                                    building_properties:
                                          if building[0] ==
                                          related_item[1]:
                                                extrema =
                                                building[7]
                                                size_descri
                                                ption =
                                                building[6]
                                    route.append(["west",
                                    related_item[2],
                                    extrema,
                                    size_description,
                                    related_item[1]])
                                    findEachRoute(related_i
                                    tem[1], end,
                                    related_item[2],  cornor,
                                    "west", route)
                                    havetarget = True
                        #Pop unrelated routes if the flag is false (no
                        routes are found).
                        if havetarget == False:
                              route.pop()
      else:
            print colored("----------Suggested Routes----------",
            "yellow")
            for item in route:
                  if item[1] == "near":
                        print "Walk to the " + colored(item[0], "blue")
                        + " toward the nearby " + colored(item[3],
                        "green") + " " + colored(item[3], "green") + "
                        building: " + colored(item[4], "red") + "."
                  else:
```

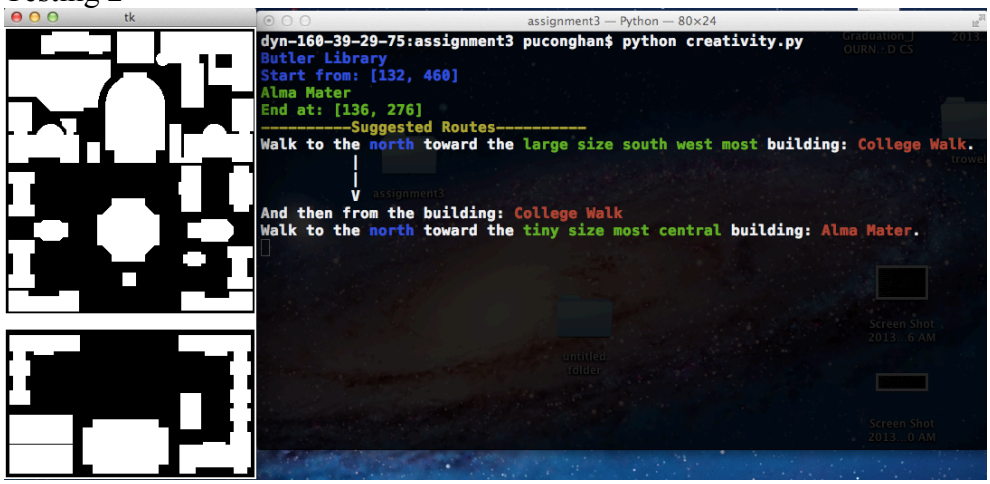
```
            #Recursively print the routes in the console.
            print "Walk to the " + colored(item[0], "blue")
            + " toward the " + colored(item[3], "green") +
            " " + colored(item[2], "green") + " building: "
            + colored(item[4], "red") + "."
      if item != route[len(route) - 1]:
            print "          |"
            print "          |"
            print "          V"
            print "And then from the building: " +
            colored(item[4], "red")
```

This recursive function recursively prints the routes in the console. Here are the practical results tested by three users:

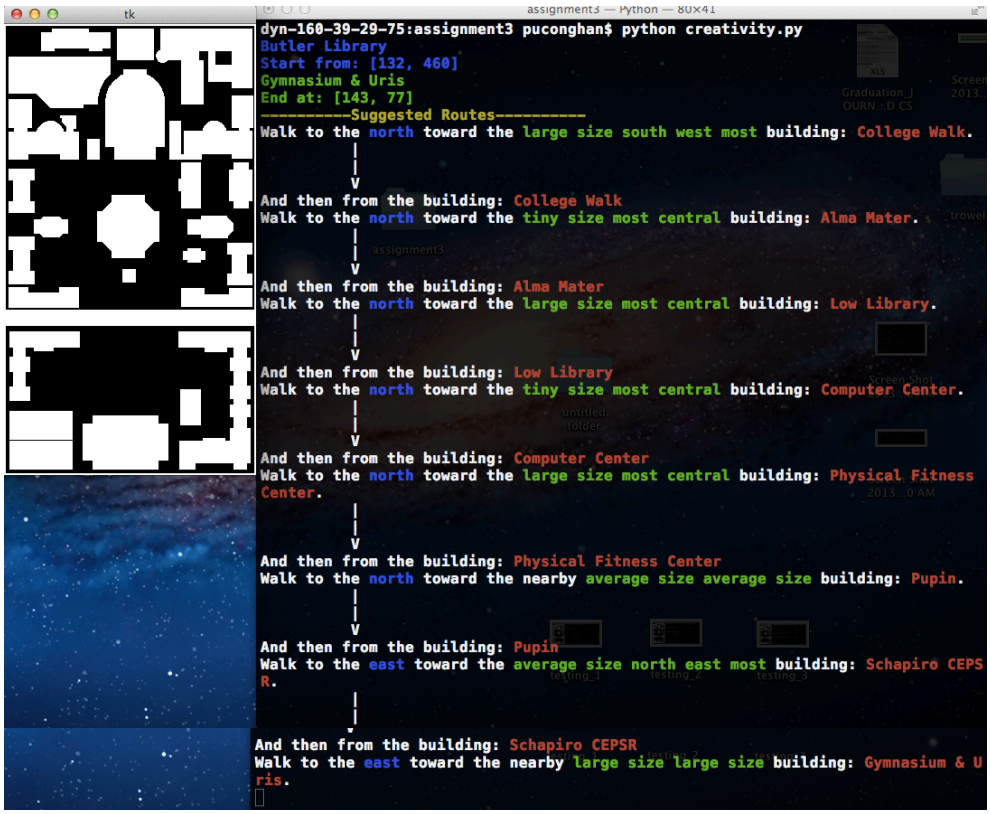
Testing 1



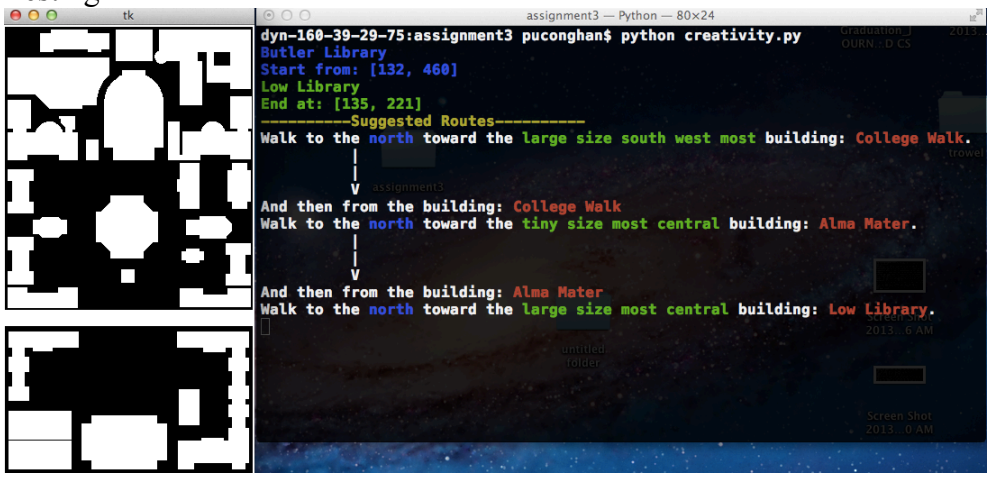
Testing 2



Testing 3



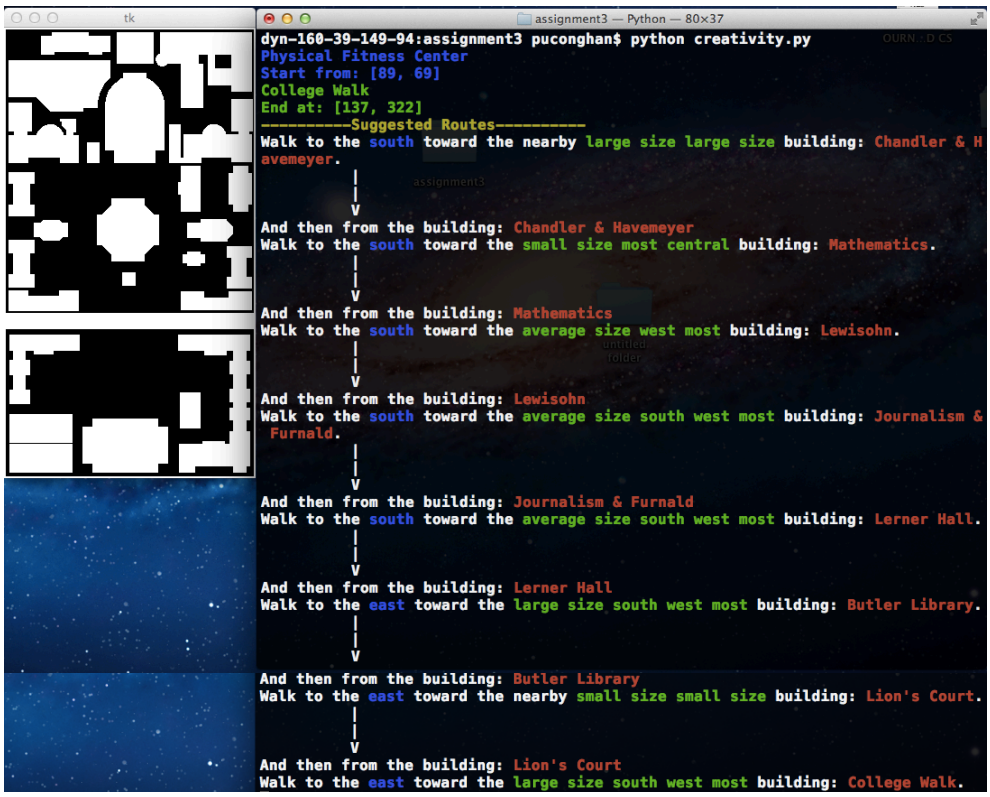
Testing 4



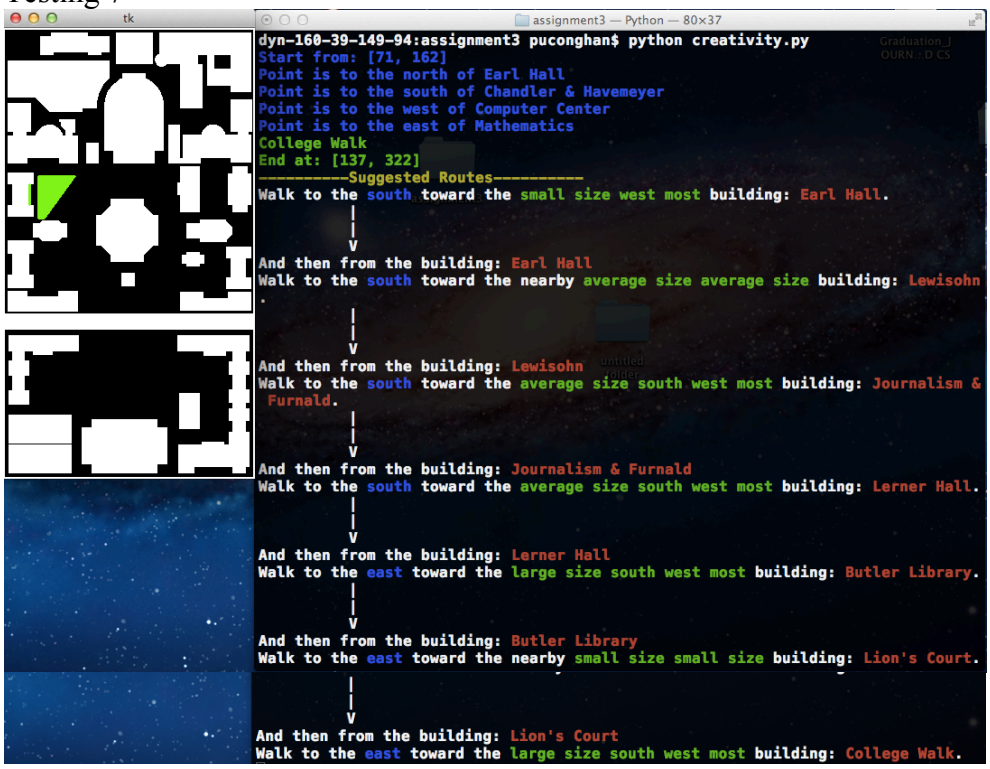
Testing 5


tk
assignment3 — Python — 80×40
dyn-160-39-29-75:assignment3 puconghan$ python creativity.py
Lerner Hall
Start from: [38, 441]
Schermerhorn
End at: [234, 118]
----------Suggested Routes----------
Walk to the north toward the average size south west most building: Journalism & Furnald.
|
|
V
And then from the building: Journalism & Furnald
Walk to the north toward the average size south west most building: Dodge.
|
|
V
And then from the building: Dodge
Walk to the north toward the nearby average size average size building: Lewisohn.
|
|
V
And then from the building: Lewisohn
Walk to the north toward the small size most central building: Mathematics.
|
|
V
And then from the building: Mathematics
Walk to the north toward the large size most central building: Chandler & Havemeyer.
|
|
V
And then from the building: Chandler & Havemeyer
Walk to the north toward the large size most central building: Physical Fitness Center.
|
|
V
And then from the building: Physical Fitness Center
Walk to the north toward the nearby average size average size building: Pupin.
|
|
V
And then from the building: Pupin
Walk to the east toward the average size north east most building: Schapiro CEPSR.
|
|
V
And then from the building: Schapiro CEPSR
Walk to the east toward the nearby large size large size building: Gymnasium & Uris.
|
|
V
And then from the building: Gymnasium & Uris
Walk to the east toward the nearby large size large size building: Schermerhorn.

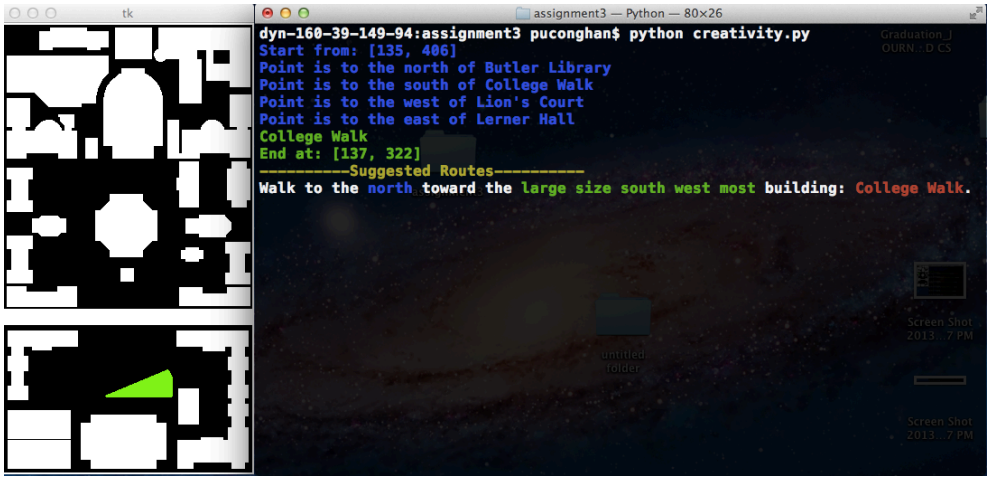
Testing 6



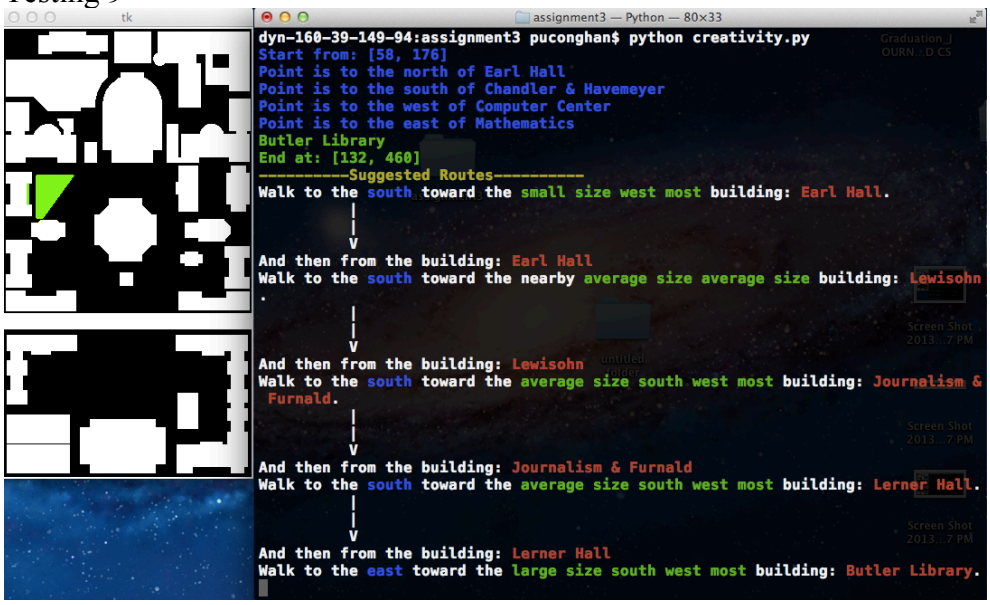
Testing 7



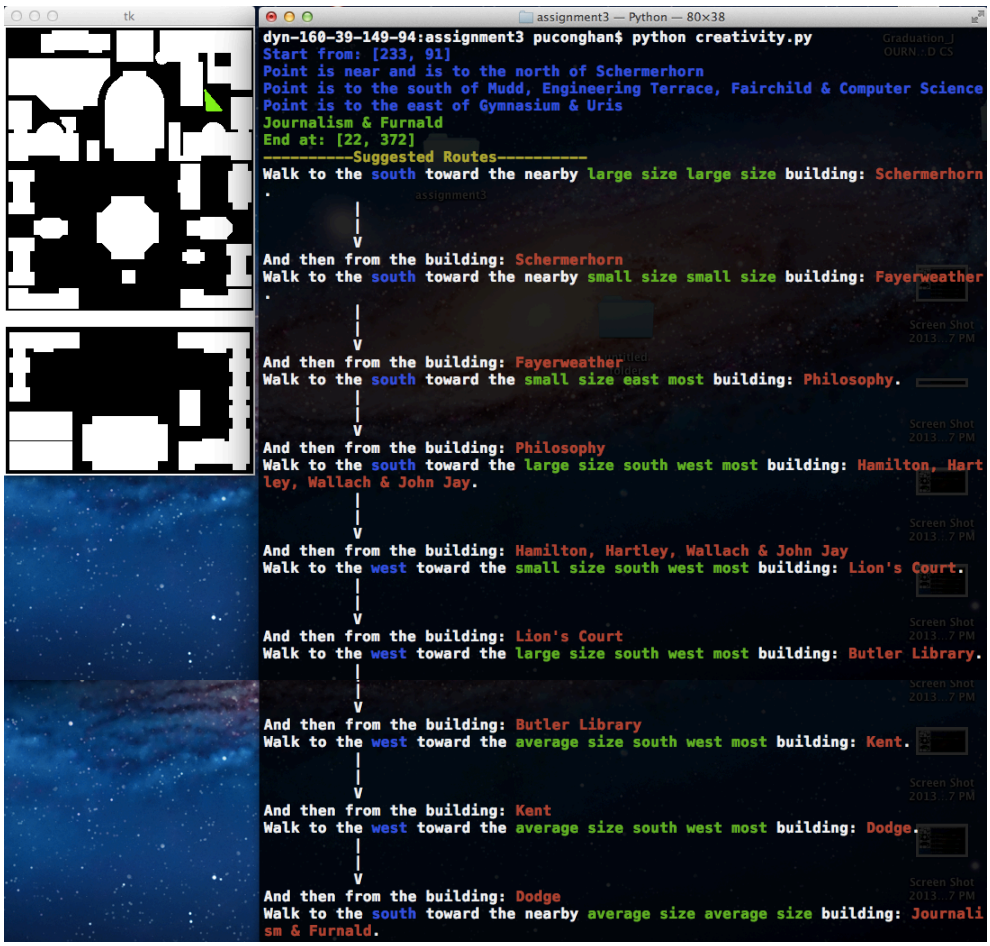
Testing 8



Testing 9



Testing10



As we can see from these ten testing results, about 80 percent of the suggested shortest routes are accurate. In particular, if the start point and the end points are in one direction (north to south or west to east), the suggested route in my program is more accurate. For instance, start points and end points in testing 1, testing 2 and testing 4 are in one direction (north to south). My program suggested the shortest routes with less than three steps. Testing 3, testing 5, testing 8, testing 9 and testing 10 yield accurate step-by-step shortest routes. Their start points and end points' locations are different in both directions (both north to south and west to east).

Testing 6 and testing 7 suggested routes that are more complicated than others. This was because their start points and end points are relatively similar in one direction or close to each other (aline with each other in one direction, but not the other direction). As a result, their locations become ambiguous to my default recursive algorithms. The recursive algorithm might strickly find the next best route in both directions. In fact, the next best route only exist in one direction. Future works are improving my recursive function by allowing it to optimize the choices in two directions.

Work Cited

"Find which side of a line a point is on." *Wiki*. 12 Jun 2010: n. page. Web. 26 Mar. 2013. <http://wiki.processing.org/w/Find_which_side_of_a_line_a_point_is_on>.

"OpenCV Python Tutorials." *OpenCV Python*. Google Blogger. Web. 26 Mar. 2013. <http://opencvpython.blogspot.com/2012/06/hi-this-article-is-tutorial-which-try.html>.

"OpenCV 2.4.3 documentation." *OpenCV Documentation*. OpenCV Developer Team, 26 Dec 2012. Web. 26 Mar. 2013. <http://docs.opencv.org/index.html>.

"TkInter." *Python Wiki*. 2013. <http://wiki.python.org/moin/TkInter>.