Program for drone surveillance

Objective

- Generate a sample target locations from grass area.
- Establish a 10x10 aerial surveillance grid.
- Using building data to check if a target location is visible from a particular surveillance point.
- Generating binary values that can be used to minimize the number of nodes a drone visits.

In [1]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import veroviz as vrv
```

Importing Satelite Image

In [2]:

```
campus = cv2.imread('campus2.png')

plt.imshow(campus)
plt.show()
```

Image format conversion

In [3]:

```
campus = cv2.cvtColor(campus, cv2.COLOR_BGR2RGB)
plt.imshow(campus)
plt.show()
```

Mask Application

In [4]:

```
gold1 = (220, 220, 160)
gold2 = (240, 240, 180)

mask = cv2.inRange(campus, gold1, gold2)

campus_filter = cv2.bitwise_and(campus, campus, mask = mask)

print(mask)
```

```
[[255 255 255 ...   0   0   0]
 [255 255 255 ...   0   0   0]
 [255 255 255 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```

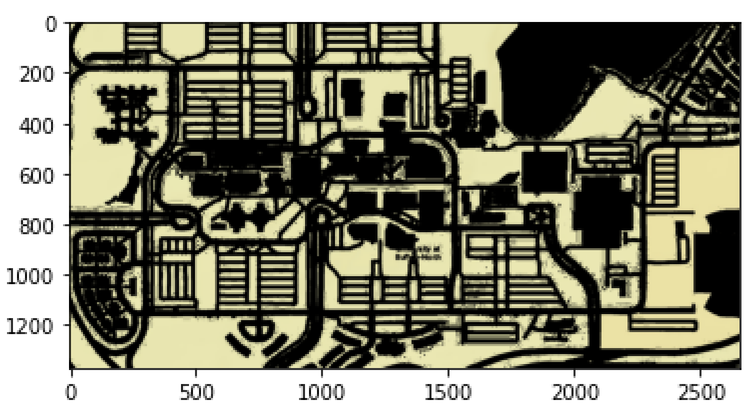
In [5]:

```
print(campus[0,0])
```

```
[233 230 177]
```

In [6]:

```
plt.imshow(campus_filter)
plt.show()
```



Sampling from Grass Area

In [7]:

```
campus_grass = np.argwhere(campus_filter > 1)
N = 80


random_indices = np.arange(0, campus_grass.shape[0])    # array of all indices
np.random.shuffle(random_indices)                         # shuffle the array
grass_samp = campus_grass[random_indices[:N]]  # get N samples without replacement

grass_samp = grass_samp[:, 0:2]
```

In [8]:

```
print(type(grass_samp))
```

```
<class 'numpy.ndarray'>
```

Function for x,y conversion to latitude and longitudes

Note - We know the coordinates of corner points of image.

In [9]:

```
nodesArray = [
    {'lat': 43.00406652249587, 'lon': -78.7773370742798},
    {'lat': 42.99689481000902, 'lon': -78.79634857177736}]

NE = nodesArray[0]
SW = nodesArray[1]
NW = {'lat': NE['lat'], 'lon': SW['lon']}
SE = {'lat': SW['lat'], 'lon': NE['lon']}


def xy2latlon(x, y, img):
    '''
    (0, 0) is in NW corner of image,
    y values INCREASE going DOWN,
    x values increase going right.
    '''

    lon = NW['lon'] + (x/img.shape[1])*(SE['lon']-NW['lon'])
    lat = NW['lat'] + (y/img.shape[0])*(SE['lat']-NW['lat'])

    return {'lat': lat, 'lon': lon}
```

In [10]:

```
SE['lat'] - NW['lat']
print(campus.shape)
print(SE['lat'])
```

```
(1370, 2658, 3)
42.99689481000902
```

Sample conversion to lat,lon

In [11]:

```
samp_loc = []
```

```
for i in range(10):
    latlon_dict = xy2latlon(grass_samp[i, 1], grass_samp[i, 0], campus)
    samp_loc.append([latlon_dict['lat'], latlon_dict['lon'], 0])
```

In [12]:

```
print(samp_loc)
```

```
[[43.00036550006652, -78.77946853637697, 0], [42.998863104822924, -78.7925148010254, 0], [43.00398276526975, -78.78378868103029, 0], [43.00622006571525, -78.79626989364625, 0], [43.0000095318555, -78.77947568893434, 0], [42.99789466189587, -78.7918782234192, 0], [42.99756486781801, -78.79189968109132, 0], [42.99782660914965, -78.7785029411316, 0], [42.99788942706924, -78.77773046493532, 0], [42.99965356364447, -78.78750801086427, 0]]
```

Generating Aerial Grid Survelience Points

In [13]:

```
from itertools import product

air_grid_NROW = 265
air_grid_NCOL = 137

air_rows = np.arange(0, 2658, air_grid_NROW)
air_cols = np.arange(0, 1370, air_grid_NCOL)
air_grid = list(product(air_rows, air_cols))
air_grid = [list(ele) for ele in air_grid]
air_arr = np.array(air_grid)

grid_loc = []

for i in range(100):
    latlon_dict = xy2latlon(air_arr[i, 1], air_arr[i, 0], campus)
    grid_loc.append([latlon_dict['lat'], latlon_dict['lon'], 25])

print(grid_loc)
```

```
[[43.00406652249587, -78.79634857177736, 25], [43.00406652249587, -78.79536867141725, 25], [43.00406652249587, -78.79438877105714, 25], [43.00406652249587, -78.79340887069704, 25], [43.00406652249587, -78.79242897033693, 25], [43.00406652249587, -78.79144906997682, 25], [43.00406652249587, -78.79046916961671, 25], [43.00406652249587, -78.7894892692566, 25], [43.00406652249587, -78.7885093688965, 25], [43.00406652249587, -78.78752946853639, 25], [43.00267929343819, -78.79634857177736, 25], [43.00267929343819, -78.79536867141725, 25], [43.00267929343819, -78.79438877105714, 25], [43.00267929343819, -78.79340887069704, 25], [43.00267929343819, -78.79242897033693, 25], [43.00267929343819, -78.79144906997682, 25], [43.00267929343819, -78.79046916961671, 25], [43.00267929343819, -78.7894892692566, 25], [43.00267929343819, -78.7885093688965, 25], [43.00267929343819, -78.78752946853639, 25], [43.001292064380515, -78.79634857177736, 25], [43.001292064380515, -78.79536867141725, 25], [43.001292064380515, -78.79438877105714, 25], [43.001292064380515, -78.79340887069704, 25], [43.001292064380515, -78.79242897033693, 25], [43.001292064380515, -78.79144906997682, 25], [43.001292064380515, -78.79046916961671, 25], [43.001292064380515, -78.7894892692566, 25], [43.001292064380515, -78.7885093688965, 25], [43.001292064380515, -78.78752946853639, 25], [42.999904835322845, -78.79634857177736, 25], [42.999904835322845, -78.79536867141725, 25], [42.999904835322845, -78.79438877105714, 25], [42.999904835322845, -78.79340887069704, 25], [42.999904835322845, -78.79242897033693, 25], [42.999904835322845, -78.79144906997682, 25], [42.999904835322845, -78.79046916961671, 25], [42.999904835322845, -78.7894892692566, 25], [42.999904835322845, -78.7885093688965, 25], [42.999904835322845, -78.78752946853639, 25], [42.99851760626517, -78.79634857177736, 25], [42.99851760626517, -78.79536867141725, 25], [42.99851760626517, -78.794388
```

```
77105714, 25], [42.99851760626517, -78.79340887069704, 25], [42.99851760626517, -78.7924
2897033693, 25], [42.99851760626517, -78.79144906997682, 25], [42.99851760626517, -78.79
046916961671, 25], [42.99851760626517, -78.7894892692566, 25], [42.99851760626517, -78.7
885093688965, 25], [42.99851760626517, -78.78752946853639, 25], [42.99713037720749, -78.
79634857177736, 25], [42.99713037720749, -78.79536867141725, 25], [42.99713037720749, -7
8.79438877105714, 25], [42.99713037720749, -78.79340887069704, 25], [42.99713037720749,
-78.79242897033693, 25], [42.99713037720749, -78.79144906997682, 25], [42.9971303772074
9, -78.79046916961671, 25], [42.99713037720749, -78.7894892692566, 25], [42.997130377207
49, -78.7885093688965, 25], [42.99713037720749, -78.78752946853639, 25], [42.99574314814
981, -78.79634857177736, 25], [42.99574314814981, -78.79536867141725, 25], [42.995743148
14981, -78.79438877105714, 25], [42.99574314814981, -78.79340887069704, 25], [42.9957431
4814981, -78.79242897033693, 25], [42.99574314814981, -78.79144906997682, 25], [42.99574
314814981, -78.79046916961671, 25], [42.99574314814981, -78.7894892692566, 25], [42.9957
4314814981, -78.7885093688965, 25], [42.99574314814981, -78.78752946853639, 25], [42.994
355919092136, -78.79634857177736, 25], [42.994355919092136, -78.79536867141725, 25], [4
2.994355919092136, -78.79438877105714, 25], [42.994355919092136, -78.79340887069704, 2
5], [42.994355919092136, -78.79242897033693, 25], [42.994355919092136, -78.7914490699768
2, 25], [42.994355919092136, -78.79046916961671, 25], [42.994355919092136, -78.789489269
2566, 25], [42.994355919092136, -78.7885093688965, 25], [42.994355919092136, -78.7875294
6853639, 25], [42.992968690034466, -78.79634857177736, 25], [42.992968690034466, -78.795
36867141725, 25], [42.992968690034466, -78.79438877105714, 25], [42.992968690034466, -7
8.79340887069704, 25], [42.992968690034466, -78.79242897033693, 25], [42.99296869003446
6, -78.79144906997682, 25], [42.992968690034466, -78.79046916961671, 25], [42.9929686900
34466, -78.7894892692566, 25], [42.992968690034466, -78.7885093688965, 25], [42.99296869
0034466, -78.78752946853639, 25], [42.99158146097679, -78.79634857177736, 25], [42.99158
146097679, -78.79536867141725, 25], [42.99158146097679, -78.79438877105714, 25], [42.991
58146097679, -78.79340887069704, 25], [42.99158146097679, -78.79242897033693, 25], [42.9
9158146097679, -78.79144906997682, 25], [42.99158146097679, -78.79046916961671, 25], [4
2.99158146097679, -78.7894892692566, 25], [42.99158146097679, -78.7885093688965, 25], [4
2.99158146097679, -78.78752946853639, 25]]
```

Sample Visualization

In [14]:

```
locs = [[NW['lat'], NW['lon']], [SE['lat'], SE['lon']]]

myNodes = vrv.createNodesFromLocs(locs     = locs,
                                  nodeType = "corners",
                                  leafletColor = 'Red',
                                  initNodes = None)

myNodes = vrv.createNodesFromLocs(locs     = samp_loc,
                                  nodeType = "corners",
                                  leafletColor = 'Green',
                                  initNodes = myNodes)

'''
myLocs = []
for asphalt locations:
    myLocs.append([lat, lon, 0])

myNodes = vrv.createNodesFromLocs(locs     = [[43,-73,0]],
                                  nodeType = "asphalt",
                                  initNodes = myNodes)

myNodes = vrv.createNodesFromLocs(locs     = [[42,-73,50]],
                                  nodeType = "drone",
                                  initNodes = myNodes)
```

```
myNodes
'''

vrv.createLeaflet(nodes         = myNodes,
                  mapBackground = 'Openstreetmap')
                  #mapBackground = 'Arcgis Roadmap')
```

```
Warning: 'id' in `initNodes` is already larger than `startNode`.  Overriding `startNode`
with maximum `id` + 1.
```

Out[14]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [15]:

```
myNodes
```

Out[15]:

	id	lat	lon	altMeters	nodeName	nodeType	popupText	leafletIconPrefix	leafletIco
0	1	43.004067	-78.796349	0	None	corners	1	glyphicon	inf
1	2	42.996895	-78.777337	0	None	corners	2	glyphicon	inf
2	3	43.000366	-78.779469	0	None	corners	3	glyphicon	inf
3	4	42.998863	-78.792515	0	None	corners	4	glyphicon	inf
4	5	43.003983	-78.783789	0	None	corners	5	glyphicon	inf
5	6	43.000622	-78.796270	0	None	corners	6	glyphicon	inf
6	7	43.000010	-78.779476	0	None	corners	7	glyphicon	inf
7	8	42.997895	-78.791878	0	None	corners	8	glyphicon	inf
8	9	42.997565	-78.791900	0	None	corners	9	glyphicon	inf
9	10	42.997827	-78.778503	0	None	corners	10	glyphicon	inf
10	11	42.997889	-78.777730	0	None	corners	11	glyphicon	inf

	id	lat	lon	altMeters	nodeName	nodeType	popupText	leafletIconPrefix	leafletIco
11	12	42.999654	-78.787508	0	None	corners	12	glyphicon	inf

Reading Building Data

- Data generated from OsmBuildings
- Further processed to find convex hulls and surface planes of each building.
- Campus_building_data
  - Planes : nested list in cartesian3 form first column being center point of plane and second column being direction of normal.
  - Coordinates - In form of longitude latitude.

In [16]:
```
import pandas as pd
import ast
import csv
cbf = pd.read_csv('campus_building_data.csv')
cbf['Coordinates']=cbf['Coordinates'].apply(lambda x: ast.literal_eval(x))
cbf['Planes'] = cbf['Planes'].apply(lambda x: ast.literal_eval(x))
```

Function

- to convert a list of lon_lat to lat_lon

In [17]:
```
def flip(list):
    lat_lon = []
    for i in list:
        lat_lon.append([i[1],i[0]])
    return lat_lon
```

Function

isIntersectSimple

- It is a function which takes data of Building Footprints
- The surveillance grid points and target points.
- For a pair of target points and projection of aerial surveillance grid points(i.e. altitude = 0).
- It checks if line connecting two such points crosses a footprint of building or not.

In [18]:
```
def isIntersectSimple(ground_point, air_point, build_foot):
    '''
    ground_point = list of form lat, lon
    air_point = list of form lat, lon, alt
```

```
    build_foot =

    returns 0 if no conflict
    returns 1 if there is a definite conflict
    returns -1 if there may be a conflict


    '''
    path = [ground_point[:2], air_point[:2]]


    myReturn = 0
    Building_Index = []

    for k in build_foot.index:
        poly = flip(build_foot.iloc[k]['Coordinates'])
        if vrv.isPathCrossPoly(path, poly):
            Building_Index.append(k)
            if air_point[2] <= build_foot.iloc[k]['Height']:
                return (1,[k])
            else:
                myReturn = -1
    return myReturn,Building_Index
```

In [19]:

```
from collections import defaultdict
```

Cesium functions converted from JavaScript to Python

In [20]:

```
import math

class Ellipsoid:
    # def __init__(self):
    #
    # def WGS84():


    def _oneOverRadiiSquared(cartesian):
        result = Cartesian3()

        if (cartesian.x == 0):
            result.x = 0
        else:
            result.x = 1.0 / (cartesian.x**2)

        if (cartesian.y == 0):
            result.y = 0
        else:
            result.y = 1.0 / (cartesian.y**2)

        if (cartesian.z == 0):
            result.z = 0
        else:
            result.z = 1.0 / (cartesian.z**2)
```

```
        return result

    def geodeticSurfaceNormal(cartesian):
        result = Cartesian3.multiplyComponents(cartesian, Ellipsoid._oneOverRadiiSquare
        return Cartesian3.normalize(result)

    def  scaleToGeodeticSurface(cartesian,oneOverRadii,oneOverRadiiSquared, centerToler
        positionX = cartesian.x
        positionY = cartesian.y
        positionZ = cartesian.z

        oneOverRadiiX = oneOverRadii.x
        oneOverRadiiY = oneOverRadii.y
        oneOverRadiiZ = oneOverRadii.z

        x2 = positionX * positionX * oneOverRadiiX * oneOverRadiiX
        y2 = positionY * positionY * oneOverRadiiY * oneOverRadiiY
        z2 = positionZ * positionZ * oneOverRadiiZ * oneOverRadiiZ

        #Compute the squared ellipsoid norm.
        squaredNorm = x2 + y2 + z2
        ratio = Math.sqrt(1.0 / squaredNorm)

class CesiumMath:
    EPSILON1 = 0.1
    EPSILON2 = 0.01
    EPSILON3 = 0.001
    EPSILON4 = 0.0001
    EPSILON5 = 0.00001
    EPSILON6 = 0.000001
    EPSILON7 = 0.0000001
    EPSILON8 = 0.00000001
    EPSILON9 = 0.000000001
    EPSILON10 = 0.0000000001
    EPSILON11 = 0.00000000001
    EPSILON12 = 0.000000000001

    RADIANS_PER_DEGREE = math.pi / 180.0
    DEGREES_PER_RADIAN = 180.0 / math.pi

    def toRadians(degrees):
        return degrees * CesiumMath.RADIANS_PER_DEGREE

    def sign(value):
        if (value == 0):
            return 0
        elif (value > 0):
            return 1
        else:
            return -1

class Cartesian3:

    def __init__(self, x=0.0, y=0.0, z=0.0):
        self.x = x
        self.y = y
        self.z = z

    def clone(cartesian):
        result = Cartesian3(cartesian.x, cartesian.y, cartesian.z)
```

```
        return result

    def divideByScalar(cartesian, scalar):
        '''
        Cartesian3.multiplyByScalar = function (cartesian, scalar, result) {
            result.x = cartesian.x * scalar;
            result.y = cartesian.y * scalar;
            result.z = cartesian.z * scalar;
            return result;
            };
        '''

        result = Cartesian3()
        result.x = cartesian.x / scalar
        result.y = cartesian.y / scalar
        result.z = cartesian.z / scalar
        return result

    def add(left, right):
        result = Cartesian3()
        result.x = left.x + right.x
        result.y = left.y + right.y
        result.z = left.z + right.z
        return result


    def subtract(left, right):
        result = Cartesian3()
        result.x = left.x - right.x
        result.y = left.y - right.y
        result.z = left.z - right.z
        return result

    def multiplyByScalar(cartesian, scalar):
        '''
        Cartesian3.multiplyByScalar = function (cartesian, scalar, result) {
            result.x = cartesian.x * scalar;
            result.y = cartesian.y * scalar;
            result.z = cartesian.z * scalar;
            return result;
            };
        '''

        result = Cartesian3()
        result.x = cartesian.x * scalar
        result.y = cartesian.y * scalar
        result.z = cartesian.z * scalar
        return result

    def multiplyComponents(left, right):
        result = Cartesian3()
        result.x = left.x * right.x
        result.y = left.y * right.y
        result.z = left.z * right.z
        return result

    def magnitudeSquared(cartesian):
        return (cartesian.x**2 + cartesian.y**2 + cartesian.z**2)

    def magnitude(cartesian):
```

```
        return math.sqrt(Cartesian3.magnitudeSquared(cartesian))

    def normalize(cartesian):
        magnitude = Cartesian3.magnitude(cartesian)

        result = Cartesian3()
        result.x = cartesian.x / magnitude
        result.y = cartesian.y / magnitude
        result.z = cartesian.z / magnitude

        return result

    def fromDegrees(longitude, latitude, height=0.0, ellipsoid=None):
        longitude = CesiumMath.toRadians(longitude)
        latitude = CesiumMath.toRadians(latitude)

        return Cartesian3.fromRadians(longitude, latitude, height)

    def fromRadians(longitude, latitude, height=0.0, ellipsoid=None):
        radiiSquared = wgs84RadiiSquared

        cosLatitude = math.cos(latitude)

        scratchN = Cartesian3()
        scratchN.x = cosLatitude * math.cos(longitude)
        scratchN.y = cosLatitude * math.sin(longitude)
        scratchN.z = math.sin(latitude)
        scratchN = Cartesian3.normalize(scratchN)

        scratchK = Cartesian3()
        scratchK = Cartesian3.multiplyComponents(radiiSquared, scratchN)
        gamma = math.sqrt(Cartesian3.dot(scratchN, scratchK))
        scratchK = Cartesian3.divideByScalar(scratchK, gamma)
        scratchN = Cartesian3.multiplyByScalar(scratchN, height)

        return Cartesian3.add(scratchK, scratchN)

    def dot(left, right):
        return left.x * right.x + left.y * right.y + left.z * right.z

Cartesian3.UNIT_X = Cartesian3(1.0, 0.0, 0.0)
Cartesian3.UNIT_Y = Cartesian3(0.0, 1.0, 0.0)
Cartesian3.UNIT_Z = Cartesian3(0.0, 0.0, 1.0)

Cartesian3.UNIT_Z

class IntersectionTests:
    # def __init__(self):
    #     nothing to see here...

    def lineSegmentPlane(endPoint0, endPoint1, plane):
        '''
        Find the intersection of the line segment from p0 to p1 and the tangent plane a
        * intersection = IntersectionTests.lineSegmentPlane(p0, p1, plane)
        '''

        result = Cartesian3()

        difference = Cartesian3.subtract(endPoint1, endPoint0)
```

```
        normal = plane.normal
        nDotDiff = Cartesian3.dot(normal, difference)

        # check if the segment and plane are parallel
        if (abs(nDotDiff) < CesiumMath.EPSILON6):
            return None    # FIXME

        nDotP0 = Cartesian3.dot(normal, endPoint0)
        t = -(plane.distance + nDotP0) / nDotDiff

        # intersection only if t is in [0, 1]
        if ((t < 0.0) or (t > 1.0)):
            return None    # FIXME

        # intersection is endPoint0 + t * (endPoint1 - endPoint0)
        result = Cartesian3.multiplyByScalar(difference, t)
        result = Cartesian3.add(endPoint0, result)

        return result

class Plane:
    def __init__(self, normal, distance):
        '''
        A plane in Hessian Normal Form defined by
        * ax + by + cz + d = 0
        where (a, b, c) is the plane's <code>normal</code>, d is the signed
        * <code>distance</code> to the plane, and (x, y, z) is any point on
        * the plane.
        '''

        self.normal = Cartesian3.clone(normal)
        self.distance = distance

    def fromPointNormal(point, normal):
        distance = -Cartesian3.dot(normal, point)

        result = Plane(normal, distance)

        result.normal = Cartesian3.clone(normal)
        result.distance = distance
        return result

'''
DON'T NEED THIS, BUT AFRAID TO DELETE JUST YET

class cartesian:
    def __init__(self, x=None, y=None, z=None):
        self.x = x;
        self.y = y;
        self.z = z;
'''

wgs84OneOverRadii = Cartesian3( 1.0 / 6378137.0, 1.0 / 6378137.0,  1.0 / 6356752.314245

wgs84RadiiSquared = Cartesian3(6378137.0 * 6378137.0,
                               6378137.0 * 6378137.0,
                               6356752.3142451793 * 6356752.3142451793)
wgs84OneOverRadiiSquared = Cartesian3(
  1.0 / (6378137.0 * 6378137.0),
  1.0 / (6378137.0 * 6378137.0),
```

```
  1.0 / (6356752.3142451793 * 6356752.3142451793)
)

wgs84CenterToleranceSquared = CesiumMath.EPSILON1


origin = Cartesian3.fromDegrees(-75.59777, 40.03883, 50)
normal = Ellipsoid.geodeticSurfaceNormal(origin)

plane = Plane.fromPointNormal(origin, normal)

p0 = Cartesian3.fromDegrees(-75.59777, 40.03883, 150)
p1 = Cartesian3.fromDegrees(-75.59777, 40.03883, 0)

# find the intersection of the line segment from p0 to p1 and the tangent plane at orig
intersection = IntersectionTests.lineSegmentPlane(p0, p1, plane)

print(intersection.x, intersection.y, intersection.z)


'''
var cartographic =

Cesium.Cartographic.fromCartesian(cartesian);
console.log(
    'lon ' + Cesium.Math.toDegrees(cartographic.longitude) + ', ' +
    'lat ' + Cesium.Math.toDegrees(cartographic.latitude) + ', ' +
    'alt ' + cartographic.height);
'''


def scaleToGeodeticSurface(cartesian, oneOverRadii, oneOverRadiiSquared, centerToleranc

    positionX = cartesian.x
    positionY = cartesian.y
    positionZ = cartesian.z

    oneOverRadiiX = oneOverRadii.x
    oneOverRadiiY = oneOverRadii.y
    oneOverRadiiZ = oneOverRadii.z

    x2 = positionX * positionX * oneOverRadiiX * oneOverRadiiX
    y2 = positionY * positionY * oneOverRadiiY * oneOverRadiiY
    z2 = positionZ * positionZ * oneOverRadiiZ * oneOverRadiiZ

    #Compute the squared ellipsoid norm.
    squaredNorm = x2 + y2 + z2
    ratio = math.sqrt(1.0 / squaredNorm)

    #As an initial approximation, assume that the radial intersection is the projection
    intersection = Cartesian3.multiplyByScalar(cartesian, ratio)

    #If the position is near the center, the iteration will not converge.
    if (squaredNorm < centerToleranceSquared):
        if  math.isfinite(ratio):
            return Cartesian3.clone(intersection)
        else:
            return None
```

```
    oneOverRadiiSquaredX = oneOverRadiiSquared.x
    oneOverRadiiSquaredY = oneOverRadiiSquared.y
    oneOverRadiiSquaredZ = oneOverRadiiSquared.z

    #Use the gradient at the intersection point in place of the true unit normal.
    #The difference in magnitude will be absorbed in the multiplier.
    gradient = Cartesian3()
    gradient.x = intersection.x * oneOverRadiiSquaredX * 2.0
    gradient.y = intersection.y * oneOverRadiiSquaredY * 2.0
    gradient.z = intersection.z * oneOverRadiiSquaredZ * 2.0

    #Compute the initial guess at the normal vector multiplier, lambda.
    lamda = ((1.0 - ratio) * Cartesian3.magnitude(cartesian)) / (0.5 * Cartesian3.magni
    correction = 0.0

    func = CesiumMath.EPSILON11
    while (abs(func) > CesiumMath.EPSILON12):
        lamda -= correction
        zMultiplier = 1.0 / (1.0 + (lamda * oneOverRadiiSquaredZ))
        yMultiplier = 1.0 / (1.0 + (lamda * oneOverRadiiSquaredY))
        xMultiplier = 1.0 / (1.0 + (lamda * oneOverRadiiSquaredX))

        xMultiplier2 = xMultiplier * xMultiplier
        yMultiplier2 = yMultiplier * yMultiplier
        zMultiplier2 = zMultiplier * zMultiplier

        xMultiplier3 = xMultiplier2 * xMultiplier
        yMultiplier3 = yMultiplier2 * yMultiplier
        zMultiplier3 = zMultiplier2 * zMultiplier

        func = x2 * xMultiplier2 + y2 * yMultiplier2 + z2 * zMultiplier2 - 1.0

        #\"denominator\" here refers to the use of this expression in the velocity and
        #computations in the sections to follow.
        denominator = (x2 * xMultiplier3 * oneOverRadiiSquaredX + y2 * yMultiplier3 * o

        derivative = -2.0 * denominator
        correction = func / derivative

    return Cartesian3(positionX * xMultiplier, positionY * yMultiplier, positionZ * zMu

class Cartographic:
    def __init__(self, longitude=0.0, latitude=0.0, height=0.0):
        self.longitude = longitude
        self.latitude = latitude
        self.height = height

    def fromCartesian(cartesian, ellipsoid=None):
        # FIXME -- Need to find these values\n"
        oneOverRadii           = wgs84OneOverRadii
        oneOverRadiiSquared    = wgs84OneOverRadiiSquared
        centerToleranceSquared = wgs84CenterToleranceSquared

        p = scaleToGeodeticSurface(cartesian, oneOverRadii, oneOverRadiiSquared, center

        if p is None:
            return None
        n = Cartesian3.multiplyComponents(p, oneOverRadiiSquared)
        n = Cartesian3.normalize(n)
```

```
        h = Cartesian3.subtract(cartesian, p)

        # lon/lat are in radians
        longitude = math.atan2(n.y, n.x)
        latitude = math.asin(n.z)
        height = CesiumMath.sign(Cartesian3.dot(h, cartesian)) * Cartesian3.magnitude(h

        result = Cartographic()
        result.longitude = longitude
        result.latitude = latitude
        result.height = height

        return result



# To Do:
# - Need `scaleToGeodeticSurface()` function converted to Python
#
# - Finish/Fix/Debug `Cartographic.fromCartesian()` function
#
# - Display plane in Cesium
# - Display line segment in Cesium
#
#
# - Verify that our intersection code seems reasonable.
#
# ---
#
# - We need road network data (Thursday installation of pgRouting or OSRM)
#     - Interections
```

```
1216271.2721645187 -4736298.548879281 4081319.5905806036
```

Functions

- **nested_dict** Nested dictionary creation.
- **isIntersectCesium** Checks if a straight line drawn from aerial surveillanace point to ground point intesect any facade of building.

In [21]:

```
def nested_dict():
        return defaultdict(nested_dict)

def isIntersectCesium(ground_point, aerial_point, df, index):
    for i in index:
        gp = Cartesian3.fromDegrees(ground_point[1],ground_point[0], ground_point[2])
        ap = Cartesian3.fromDegrees(aerial_point[1],aerial_point[0], aerial_point[2])
        for j in df['Planes'][i]:
            normal = Cartesian3()
            point = Cartesian3()
            point.x = j[0][0]
            point.y = j[0][1]
            point.z = j[0][2]
            normal.x = j[1][0]
            normal.y = j[1][1]
            normal.z = j[1][2]
```

```
            plane  = Plane.fromPointNormal(point,normal)
            intersection_point = IntersectionTests.lineSegmentPlane(gp,ap, plane)

            if intersection_point is not None:
                #Plane j of buiding i is conflict
                intersection_point_degree  = Cartographic.fromCartesian(intersection_po
                if 0<= intersection_point_degree.height <= df['Height'][i]:
                    return 0

    return 1


    '''

    '''
```

from CesiumPy import Cartesian3 cartesian_samp_loc = [] cartesian_grid_loc = [] for i in range(len(samp_loc)): cartesian_samp_loc.append(Cartesian3.fromDegrees(samp_loc[i][1],samp_loc[i][0],0)) for i in range(len(grid_loc)): cartesian_grid_loc.append(Cartesian3.fromDegrees(grid_loc[i][1],grid_loc[i][0],25))

print(cartesian_samp_loc[0].z) print(samp_loc[0])

Generating suitable values

- Binary values for each possible pair of aerial surveillance points and ground points to represent if a particular target location is vissible or not.

In [22]:
```
'''
Using nested dictionary takes less run time.
'''
c = defaultdict(nested_dict)

for i in range(len(samp_loc)):
    for j in range(len(grid_loc)):
        val,index = isIntersectSimple(samp_loc[i], grid_loc[j], cbf)
        if (val == 1):
            c[i][j] = 0
        elif (val == 0):
            c[i][j] = 1
        else:
            c[i][j] = isIntersectCesium(samp_loc[i], grid_loc[j], cbf, index)
```

In [23]:
```
'''
Using nested dictionary to ease the process of optimization
'''
c_l = []

for i in range(len(samp_loc)):
    c_l.append([])
    for j in range(len(grid_loc)):
        val,index = isIntersectSimple(samp_loc[i], grid_loc[j], cbf)
        if (val == 1):
            c_l[i].append(0)
```

```
        elif (val == 0):
            c_l[i].append(1)
        else:
            c_l[i].append(isIntersectCesium(samp_loc[i], grid_loc[j], cbf, index))
```

In [24]:

```
print(c)
print(c_l)
```

```
defaultdict(<function nested_dict at 0x0000025C8559D360>, {0: defaultdict(<function nest
ed_dict at 0x0000025C8559D360>, {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 1,
9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 0, 21:
0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 29: 0, 30: 0, 31: 0, 32: 0, 33: 0, 3
4: 0, 35: 0, 36: 0, 37: 0, 38: 0, 39: 0, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46:
1, 47: 1, 48: 1, 49: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 5
9: 1, 60: 1, 61: 1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71:
1, 72: 1, 73: 1, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 8
4: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96:
1, 97: 1, 98: 1, 99: 1}), 1: defaultdict(<function nested_dict at 0x0000025C8559D360>,
{0: 1, 1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1, 11: 1, 12: 1, 13: 0,
14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 1, 21: 1, 22: 1, 23: 1, 24: 0, 25: 0, 26:
0, 27: 0, 28: 0, 29: 0, 30: 1, 31: 1, 32: 1, 33: 1, 34: 1, 35: 0, 36: 0, 37: 1, 38: 1, 3
9: 1, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 50: 1, 51:
1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61: 1, 62: 1, 63: 1, 6
4: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 74: 1, 75: 1, 76:
1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 8
9: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1}), 2: default
dict(<function nested_dict at 0x0000025C8559D360>, {0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1,
6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 1
9: 1, 20: 1, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 29: 0, 30: 0, 31:
0, 32: 0, 33: 0, 34: 0, 35: 0, 36: 0, 37: 0, 38: 0, 39: 0, 40: 0, 41: 0, 42: 0, 43: 0, 4
4: 0, 45: 0, 46: 0, 47: 0, 48: 0, 49: 0, 50: 0, 51: 0, 52: 0, 53: 0, 54: 0, 55: 0, 56:
0, 57: 0, 58: 0, 59: 0, 60: 0, 61: 0, 62: 0, 63: 0, 64: 0, 65: 0, 66: 0, 67: 0, 68: 0, 6
9: 1, 70: 0, 71: 0, 72: 0, 73: 0, 74: 0, 75: 0, 76: 0, 77: 0, 78: 0, 79: 0, 80: 0, 81:
0, 82: 0, 83: 0, 84: 0, 85: 0, 86: 0, 87: 0, 88: 1, 89: 0, 90: 0, 91: 0, 92: 0, 93: 0, 9
4: 0, 95: 0, 96: 0, 97: 1, 98: 0, 99: 0}), 3: defaultdict(<function nested_dict at 0x000
0025C8559D360>, {0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11:
1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 1, 2
4: 1, 25: 1, 26: 1, 27: 1, 28: 1, 29: 1, 30: 1, 31: 1, 32: 1, 33: 1, 34: 1, 35: 0, 36:
0, 37: 0, 38: 0, 39: 0, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 4
9: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61:
1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 7
4: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86:
1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 9
9: 1}), 4: defaultdict(<function nested_dict at 0x0000025C8559D360>, {0: 0, 1: 0, 2: 0,
3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16:
0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 2
9: 0, 30: 0, 31: 0, 32: 0, 33: 0, 34: 0, 35: 0, 36: 0, 37: 0, 38: 0, 39: 0, 40: 1, 41:
1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 50: 1, 51: 1, 52: 1, 53: 1, 5
4: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61: 1, 62: 1, 63: 1, 64: 1, 65: 1, 66:
1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 7
9: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91:
1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1}), 5: defaultdict(<function ne
sted_dict at 0x0000025C8559D360>, {0: 1, 1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0,
9: 0, 10: 1, 11: 1, 12: 1, 13: 1, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 1, 20: 1, 21:
1, 22: 1, 23: 1, 24: 0, 25: 0, 26: 0, 27: 0, 28: 1, 29: 0, 30: 1, 31: 1, 32: 1, 33: 1, 3
4: 1, 35: 0, 36: 1, 37: 1, 38: 1, 39: 0, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46:
1, 47: 1, 48: 1, 49: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 5
9: 1, 60: 1, 61: 1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71:
```

```
1, 72: 1, 73: 1, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 8
4: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96:
1, 97: 1, 98: 1, 99: 1}), 6: defaultdict(<function nested_dict at 0x0000025C8559D360>,
{0: 1, 1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1,
14: 0, 15: 0, 16: 0, 17: 0, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 1, 24: 0, 25: 0, 26:
0, 27: 0, 28: 1, 29: 0, 30: 1, 31: 1, 32: 1, 33: 1, 34: 1, 35: 0, 36: 1, 37: 1, 38: 1, 3
9: 1, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 50: 1, 51:
1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61: 1, 62: 1, 63: 1, 6
4: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 74: 1, 75: 1, 76:
1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 8
9: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1}), 7: default
dict(<function nested_dict at 0x0000025C8559D360>, {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0,
6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 1
9: 1, 20: 0, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 29: 0, 30: 1, 31:
1, 32: 1, 33: 1, 34: 0, 35: 0, 36: 1, 37: 1, 38: 1, 39: 1, 40: 1, 41: 1, 42: 1, 43: 1, 4
4: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56:
1, 57: 1, 58: 1, 59: 1, 60: 1, 61: 1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 6
9: 1, 70: 1, 71: 1, 72: 1, 73: 1, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81:
1, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 9
4: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1}), 8: defaultdict(<function nested_dict at 0x000
0025C8559D360>, {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11:
0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 1, 2
4: 0, 25: 0, 26: 0, 27: 0, 28: 0, 29: 0, 30: 1, 31: 1, 32: 1, 33: 1, 34: 0, 35: 0, 36:
1, 37: 1, 38: 1, 39: 1, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 4
9: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61:
1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 7
4: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86:
1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 9
9: 1}), 9: defaultdict(<function nested_dict at 0x0000025C8559D360>, {0: 0, 1: 0, 2: 0,
3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16:
0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 2
9: 0, 30: 0, 31: 0, 32: 0, 33: 0, 34: 0, 35: 0, 36: 0, 37: 0, 38: 0, 39: 1, 40: 0, 41:
0, 42: 0, 43: 0, 44: 0, 45: 0, 46: 0, 47: 0, 48: 1, 49: 1, 50: 0, 51: 0, 52: 0, 53: 0, 5
4: 0, 55: 0, 56: 0, 57: 1, 58: 1, 59: 1, 60: 0, 61: 0, 62: 0, 63: 0, 64: 0, 65: 1, 66:
1, 67: 1, 68: 1, 69: 1, 70: 0, 71: 0, 72: 0, 73: 0, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 7
9: 1, 80: 0, 81: 0, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 0, 91:
1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1})})
[[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1]]
```