OPTIMUS
Open Teaching Platform for Teaching Integrated Modeling and Urban Simulation

TUTORIAL 1

This document will walk you through the process of experimenting with OPTIMUS using a set of example files. It assumes you have an understanding of the basic python and OPTIMUS architecture.

For this first tutorial, we're going to develop a simulation involving pollution-creating factories, where wind affects the pollution movement, and where there are agents known as verdes who do everything they can to move away from highly polluted areas to less polluted areas.

GET THE FILES YOU NEED

Download the latest version of OPTIMUS from the project website:
http://ccnmtl.columbia.edu/projects/optimus

(if you need to install python or any of its libraries, there is a python installer and associated instructions included in the OPTIMUS download.)

EXPLORE THE FILE STRUCTURE

Find your way to your OPTIMUS directory (or folder). Open it up.

Open the Simulations folder. In it you will see a directory called 'tutorial'. Open this directory.

(There is a completed tutorial directory (tutorial_completed) as well so you can check your work later if necessary. No cheating now though.)

In the tutorial directory you will see directories called Pollution and Wind. We have made these 2 federates to get you started. We're going to create two federates in addition to the two you see here. If you'd like, go ahead and open these directories and examine the files contained within.

MAKING YOUR FIRST FEDERATE: FACTORY

We're going to start by creating a Factory federate that will emit pollution following the rules of the pollution federate we've already created for you.

Make a directory called 'factory' within the 'tutorial' directory.

Within that directory create a file called federate.xml. Save it.

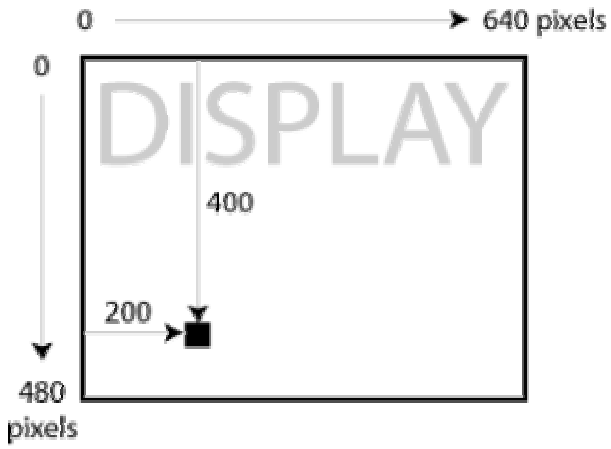
All federates in OPTIMUS begin with this heading where name and system are the name of the directory you created for the federate:

```
<?xml version="1.0"?>
<federate name="factory" system="factory">
```

The next section of a federate is its attributes. For factories, all we need to specify is where they are located. They will be fixed in place. Rather than code the locations into the federate, we will have OPTIMUS refer to an external CVS file for the locations. Here is how it should look, where factories.csv is the external data file:

```
<attributes>
  <attribute name="factories_file" varname="filename" type="text"
default="factories.csv"/>
</attributes>
```

Create another file called factories.csv which will contain the x,y coordinates of each factory you want to include as well as how many units of pollution they will emit. The origin of the display is in the upper left so a x,y of 200, 400 would appear here:



Our completed CSV file looks like this, but you can use whatever numbers you like keeping in mind that the display size is 640 x 480.

```
x,y,emission
200,200,5.0
300,300,5.0
400,400,10.0
100,100,5.0
100,200,5.0
10,300,5.0
```

Now it's time to return to our federate.xml file to add event handlers. We need two events to occur, the init.py and something called the tick.py. OPTIMUS is designed to run init.py files when federates are initially (get it? init – initially, ok) loaded. OPTIMUS also is preset to run tick.py filesl each time a time step passes (tick tick tick...asleep yet?).

Handlers look like this and follow the attributes:

```
<handlers>
  <event type="init" file="init.py"/>
  <event type="tick" file="tick.py"/>
</handlers>
```

Then while we're here, let's finish off the file with a federate endtag:

```
</federate>
```

So the complete federate xml file for your factories should look like this:

```
<?xml version="1.0"?>
<federate name="factory" system="factory">
<attributes>
  <attribute name="factories_file" varname="filename" type="text"
default="factories.csv"/>
</attributes>
<handlers>
  <event type="init" file="init.py"/>
  <event type="tick" file="tick.py"/>
</handlers>
</federate>
```

Save it.

Now let's make the init.py file which will tell OPTIMUS what to do with the factory information that we have in our CSV file:

```
data = self.csv2numeric(self.filename,(int,int,float))

self.x = data['x']
self.y = data['y']
self.emission = data['emission']
```

The first line will load in the CSV data for the factory and that it should expect 2 integers for the x and y and then a float (aka decimal number) for the emission units per unit time for each factory.

The next 3 lines of the init file assign the csv data to 3 arrays (x,y, and emission) that will be picked up by the tick file for passing to the pollution federate for each time step.

Save this file.

And now create the tick.py file:

```
self.organizer.generate_event(OptimusEvent("emit_pollution",
                                            {'x' : self.x,
                                             'y' : self.y,
                                             'pollution' : self.emission}))
```

For each timestep, OPTIMUS will perform an event called "emit pollution" at each factory's x,y point using the two arrays (x and y) from the init.py file with the pollution units stored in the emission array from the init.py file. The pollution we made for you will pick this information up and use it to calculation the pollution levels and dispersal taking wind into account (we'll get to that in a minute).

Now eventually we'll make a user interface that will generate these federate files without you having to create them by hand, but for now, congratulations, you just made your first federate! Feeling special? Let's check your work and make sure that feeling is justified...

GET READY TO TEST YOUR WORK

Go up 2 directory levels so you are back in the OPTIMUS home directory. Open the file called config.txt. It should look like this:

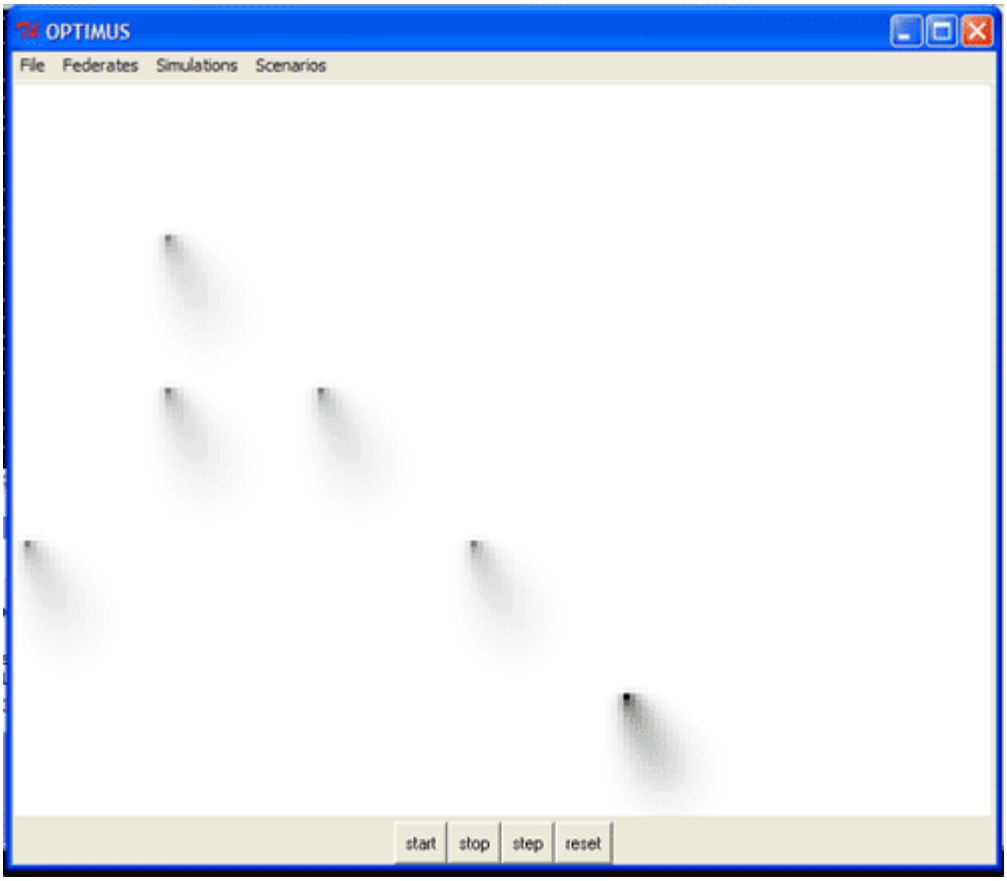
```
file:simulations/tutorial/wind/federate.xml
file:simulations/tutorial/pollution/federate.xml
```

This file tells OPTIMUS which federates to load when you launch the OPTIMUS.py file (located in this same directory). If you were to launch OPTIMUS now, I'm afraid it would be more exciting watching the paint dry on your dorm walls. The wind will blow and if there were any pollution to be calculated and displayed it would show up, but unfortunately there are no pollution emitting federates included, so let's add this line and resave:

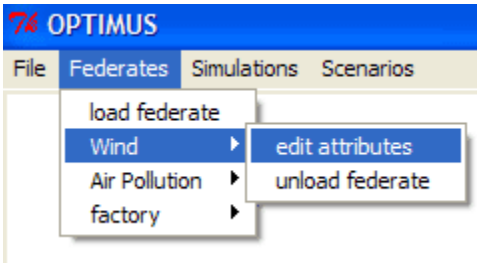
```
file:simulations/tutorial/factory/federate.xml
```

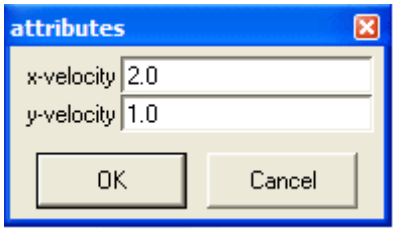
RUN OPTIMUS

Now try running OPTIMUS by opening the Optimus.py file, hit the Start button and see what you've got. You should see something resembling this once you let it run for a bit:



Based on the origin of the pollution you can see where the factories are located and you can probably guess the wind's default vectors. Stop the simulation for a moment. Try the menu up top and edit the wind direction and check the result:





You also have the skills to go into the wind federate directory and change the default – see if you can do it.

MAKING SPATIALLY AWARE AGENT FEDERATES: VERDES

To make things a little more exciting we're going to make a federate that will examine the pollution being emitted and try to move away from it every timestep.

Start by creating a directory called 'verdes' in the Tutorial directory.

Create a federate.xml file with the same formatting and tags as the factory federate minus the contents of the attributes and handlers:

```
<?xml version="1.0"?>
<federate name="verdes" system="verdes">

<attributes>
</attributes>

<handlers>
</handlers>

</federate>
```

Now instead of specifying where the verdes will start with a CSV as with the factories, we're going to generate their locations randomly. First we need to decide how many there will be and list that as an attribute within our attribute tags:

```
<attribute name="number" varname="number" default="50" type="int"
min="0" max="1000" label="number of verdes" />
```

As you can see, we're going to start with 50 verdes as a default but allow anywhere from 0-1000.

To generate the random locations, we're going to use the init.py file, so we'll need the following in as a handler (just as in the factory federate):

```
<event type="init" file="init.py"/>
```

Go ahead and create that file, again in the same directory as the federate.xml file we are working on.

To generate the random locations we're going to use a Python library called MLab:

```
import MLab
self.x = MLab.rand(self.number) * 640
self.y = MLab.rand(self.number) * 480
```

This will generate 2 arrays of random numbers with as many numbers as we specified in our attribute above (in our case 50), one for the x coordinate and one for the y. Since the generated numbers are all between 0 and 1 we will cleverly multiple them all by the dimensions of our display to allow the equal possibility of verdes appearing anywhere on the display.

Now what is going to be displayed? Unlike our factories, where we could infer the location of the factories based on the pollution being emitted, we need to have OPTIMUS create a display element for each verde so we can watch them move since we have no reference points based on them emitting anything.

To do this we're going to add a second piece to the init.py file that will plot each verde's location using a dot:

```
self.dots = []
for i in range(self.number):
    self.dots.append(self.canvas.create_rectangle(self.x[i] -
1,self.y[i] - 1,self.x[i] + 1,self.y[i] + 1,

fill="#006600",outline="#006600",width=0))
```

So for every verde we're going to use the canvas create rectangle around the location of each x,y one pixel in each direction and color it with the hex #006600 (aka green – hey! green – verde – get it?)

Save your init.py file. It's done for now.

Now we need a tick file, so add that handler event:

```
<event type="tick" file="tick.py"/>
```

While we're there, also add a display event which will allow us to update the display with the verdes new location after each tick.

```
<event type="display" file="display.py"/>
```

You're federate.xml file is done... should look like this:

```
<?xml version="1.0"?>

<federate name="verdes" system="verdes">

<attributes>
  <attribute name="number" varname="number" default="50" type="int"
min="0" max="1000"
          label="number of verdes" />
</attributes>
```

```
<handlers>
  <event type="init" file="init.py"/>
  <event type="tick" file="tick.py"/>
  <event type="display" file="display.py"/>
</handlers>

</federate>
```

Now the display.py file is straightforward, we just have to repeat what we had in the init file to draw the tiny green rectangles (try making them bigger if you want):

```
for i in range(self.number):
    self.canvas.coords(self.dots[i],self.x[i] - 1,self.y[i] -1,
                       self.x[i] + 1,self.y[i] + 1)
```

Save that file my friend.

Now the tick.py file is a little more involved as we have to use some techniques that will allow each verde to “look” around and decide how to move away from any pollution that might be near it at every time step.

This first thing to do is create each verde’s sight ability. This first section will give us blank arrays for the point where each verde is currently located as well as one coordinate to its north, south, east and west.

```
c = zeros(self.number).astype('f')
n = zeros(self.number).astype('f')
s = zeros(self.number).astype('f')
e = zeros(self.number).astype('f')
w = zeros(self.number).astype('f')
```

Now we have to pull in the data that each verde is ‘looking’ for - the pollution emission from the pollution federate:

```
self.pollution =
self.organizer.get_var("air_pollution","full",ones((160,120)))
```

Since the pollution dispersion is actually ¼ the size of the display we have to scale it up, convert it into integers and then load all the numbers into our arrays from above:

```
for i in range(self.number):
    x = self.x[i] / 4
    y = self.y[i] / 4
    x = int(x + 1)
    y = int(y + 1)
    c[i] = self.pollution[x,y]
    n[i] = self.pollution[x,y - 1]
    s[i] = self.pollution[x,y + 1]
    e[i] = self.pollution[x + 1,y]
    w[i] = self.pollution[x - 1,y]
```

Now comes the fun part, where each verde is going to go through the following steps to determine which direction to move. Each verde will have to compare all 4 directions and its current spot:

```
dx = zeros(self.number)

dx = where(w < c, dx - 1,dx)
dx = where(e < c, dx + 1,dx)
dx = where(dx == 0, where(e < w, dx + 1, dx - 1), dx)

dy = zeros(self.number)

dy = where(n < c, dy - 1,dy)
dy = where(s < c, dy + 1,dy)
dy = where(dy == 0, where(n < s, dy - 1, dy + 1), dy)
```

Then we add the result of the comparison to each verde's current location:

```
self.x = self.x + dx
self.y = self.y + dy
```

Then we add a caveat where if the verde's are within one pixel of the edge of the display they stay in the same place (so the 'looking' function does not fail on the next tick and so they don't move off the display):

```
self.x = where(self.x > 639, 639, self.x)
self.x = where(self.x < 1, 1, self.x)

self.y = where(self.y > 479, 479, self.y)
self.y = where(self.y < 1, 1, self.y)
```

Save this file.

GET READY TO TEST YOUR WORK

Go up a 2 directory levels so you are back in the OPTIMUS home directory. Open the file called config.txt. It should look like this:

```
file:simulations/tutorial/wind/federate.xml
file:simulations/tutorial/pollution/federate.xml
file:simulations/tutorial/factory/federate.xml
```

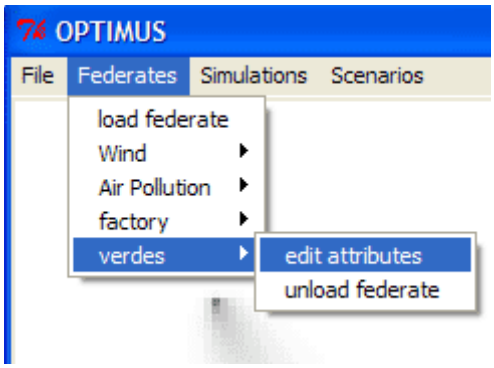
Add this line to include verdes in your simulation and resave:

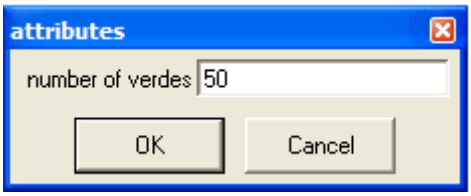
```
file:simulations/tutorial/verdes/federate.xml
```

RUN OPTIMUS

Open Optimus.py, start the simulation and see what you've got... you should see something like This with factories emitting pollution as before and little green rectangles making a run for 'cleaner'areas of the display.

Stop the simulation. Go to the menu and edit the number of verdes and run it again. Notice how you can increase them up to 1000 as we specified in the federate attribute. If you like, go back and try making this maximum number MUCH larger and see what happens.





You've completed this tutorial.

Extension exercises that you might want to try:

- modify verde behavior so they won't bother moving unless a neighboring cell is better than their current location by a certain threshold amount (ie, make them lazy).
- modify verde behavior so not more than one can occupy a given cell.
- if east and west are both better than the current location but are equal to each other, pick one randomly instead of defaulting to one or the other as it currently does. (same for north/south too)
- extend the range of the verdes' "vision" so they can see more than one cell in each direction.
- let the verde's move more than one cell. perhaps make the distance they travel proportional to the pollution gradient.
- let the verdes look diagonally. ie, include ne, nw, se, and sw in the equation.
- make factories randomly located