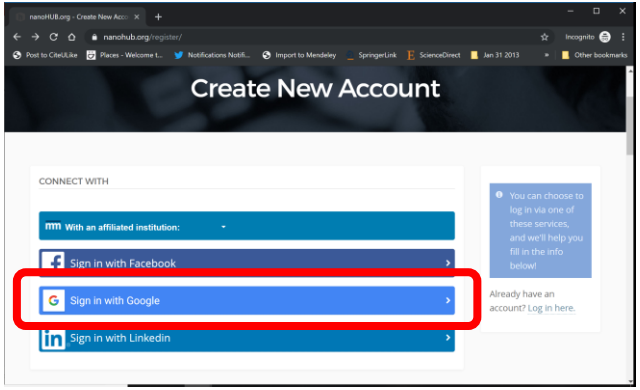
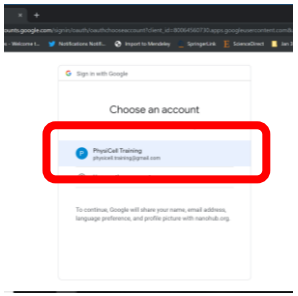
nanoHUB Account

- These tutorials use cloud-hosted PhysiCell models on nanoHUB.org.
- nanoHUB is **free**, but it requires a one-time registration.
- **Steps**:
  1. Visit https://nanohub.org/register
  2. Choose "Sign in with Google"
  3. Choose a Google account
  4. Click "No" (so it doesn't try to associate with some other nanoHIB account)
  5. Finish filling in details, and you're done!
  6. Use your google account to sign in in the future.

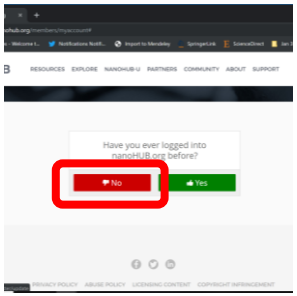
2



3



4



Agent-based modeling of multicellular systems and cancer in PhysiCell

Part 2: Creating models

Get lectures and materials here!

github.com/physicell-training/CAMBAM_2020

Paul Macklin, Ph.D.

Intelligent Systems Engineering
Indiana University

August 13, 2020

Let's download PhysiCell

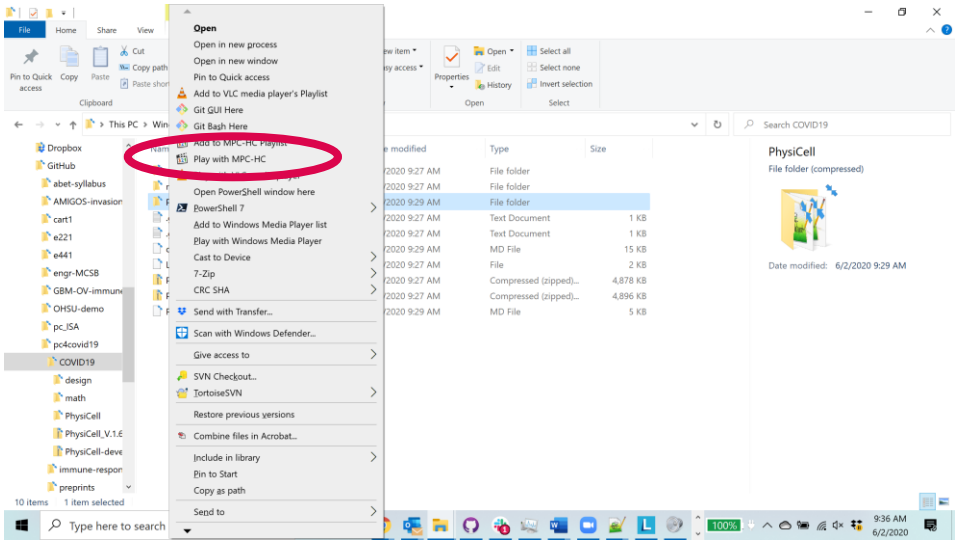
- Two download options to get the latest numbered release:
  - **GitHub:**
    - https://github.com/MathCancer/PhysiCell/releases/latest

  - **SourceForge:**
    - https://sourceforge.net/projects/physicell/files/latest/download

- Unzip the download, and enter the PhysiCell root directory
- Of particular note, go to ./documentation and open the User_Guide.pdf

Open a terminal in your code directory

- Open the extracted code. Browse until you can see a PhysiCell directory.



- Open a terminal window in the PhysiCell directory
  - **Windows:**
    - <shift>-<right click>
    - Open power shell (or command) window here
  - **Mac:** Sorry, ask a Mac person.

Compiling, running, and visualizing your first project

Sample projects

- It's inefficient (and a little insane) to code new projects *entirely* from scratch.
- So, we provide sample projects:
  - 2D and 3D template projects
  - Cancer models
  - Synthetic multicellular systems
  - Viral dynamics in tissue

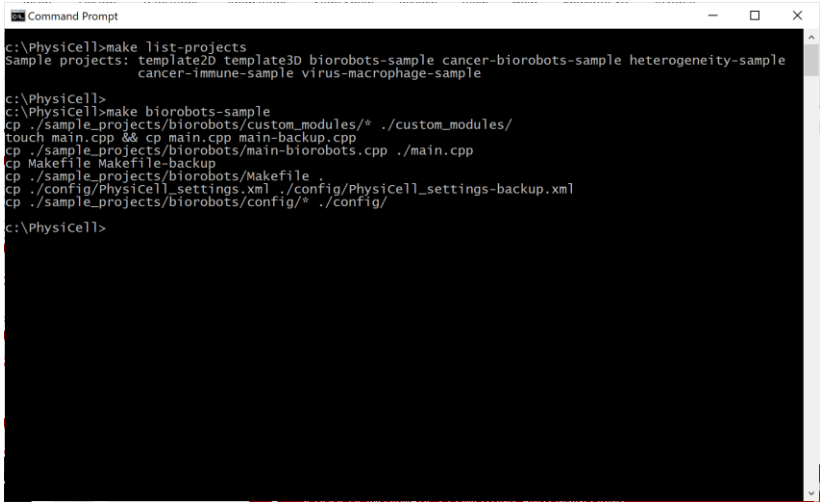
- **make [project-name]:** populate a sample project
  - Then use **make** to compile it
- **make data-cleanup**: clean up the output data
- **make reset**: return to a "clean slate" (depopulate the project)
- **make list-projects:** display all available sample projects

Documentation: User Guide Sections 6, 7.

PhysiCell Project Essentials (1)

- Each PhysiCell release includes sample projects. To list them:
  - **`make list-projects`**

- Your first step is to **populate a project.**
  - **`make <project_name>`**
  - Let's use biorobots-sample:
    - **`make biorobots-sample`**
  - This copies source code, a tailored make file, and configuration files

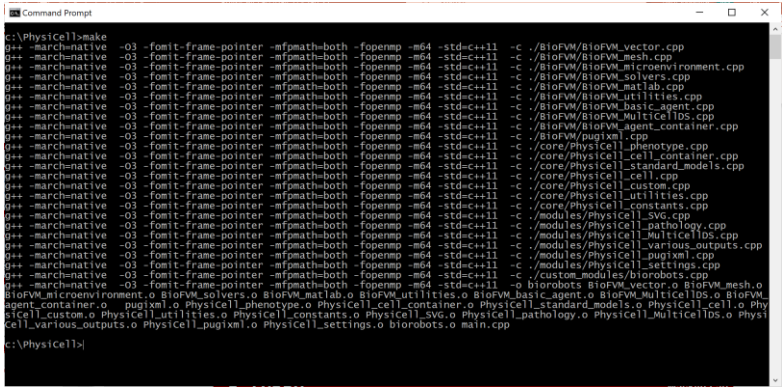


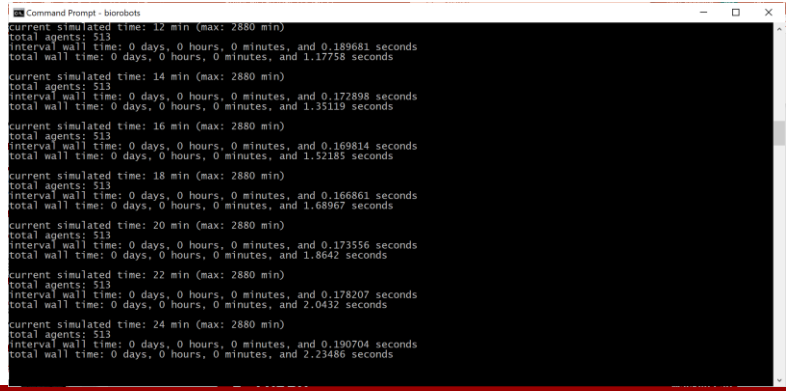
PhysiCell Project Essentials (2)

- Now, **compile the project**
  - `make`

- Then, **run the project**
  - `./biorobots` (Linux, OSX)
  - `biorobots.exe` (Windows)

- This should take about 5 minutes



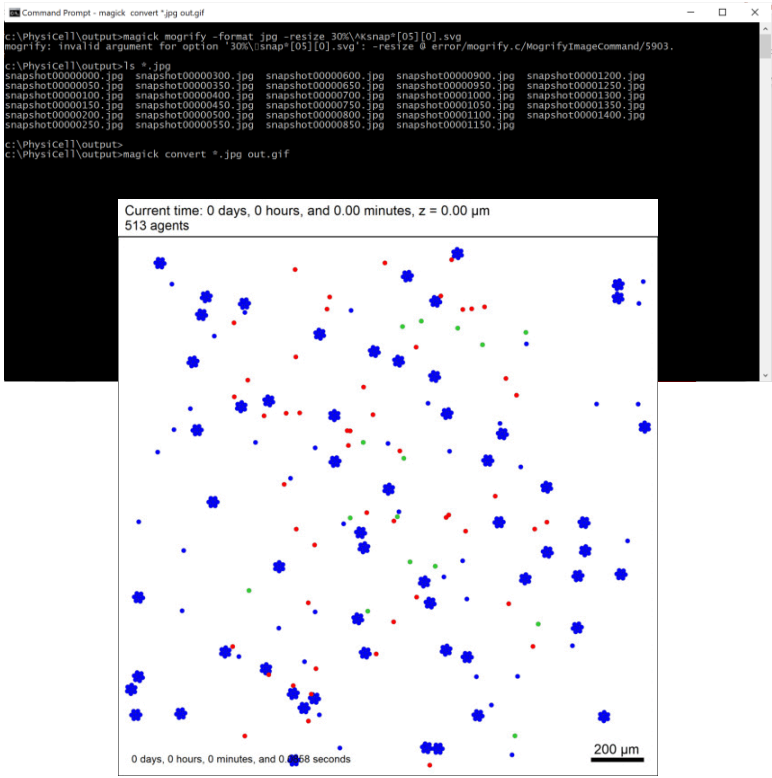


PhysiCell Project Essentials (3)

- **Look at saved data**
  - Most projects save data to ./output
    - XML files give metadata, mesh, and substrate info
    - MAT file save (compressed) substrate and cell data
    - SVG files are visual quick snapshots
    - More on loading XML / MAT files in Python later

- Let's convert SVG to rescaled JPEG
  - `magick mogrify -format jpg -resize 30% snap*.svg`
    - Convert snapshot00000000.svg, snapshot00000001.svg, ...
  - `magick mogrify -format jpg -resize 30% snap*[05][0].svg`
    - Convert snapshot00000000.svg, snapshot00000050.svg, ...

- Now, let's create an animated GIF
  - `magick convert *.jpg out.gif`



Reference: working with the images

- To convert all the SVG files to PNG format

```
magick mogrify -format png snap*.svg
```

- To convert every SVG file ending in 0 or 5 to JPG format

```
magick mogrify -format jpg snap*[05].svg
```

- To convert the JPG files to an animated GIF

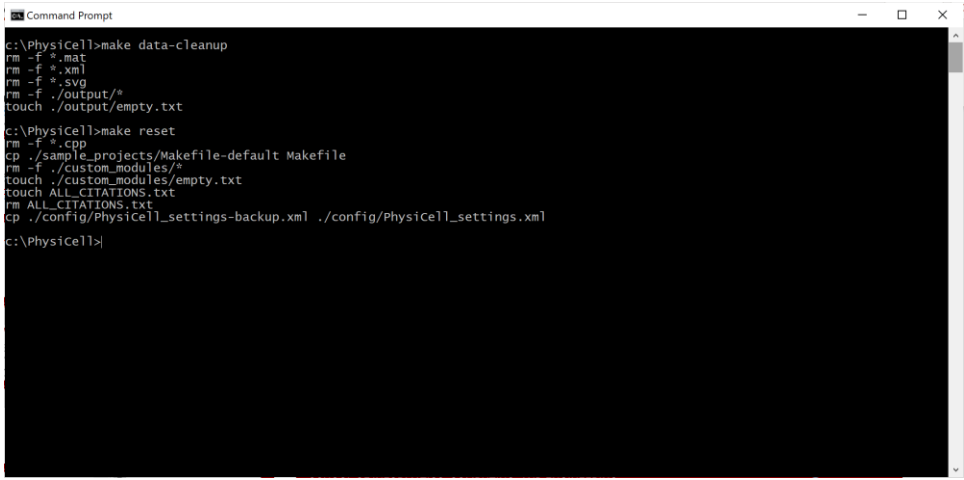
```
magick convert *.jpg out.gif
```

- To create an mp4 movie:

```
ffmpeg -r 24 -f image2 -i snapshot%08d.jpg -vcodec libx264 -pix_fmt yuv420p -strict -2 -tune
animation -crf 15 -acodec aac out.mp4
```

PhysiCell Project Essentials (4)

- **Data cleanup**
  - Clean up data to get ready for another run
  - `make data-cleanup`

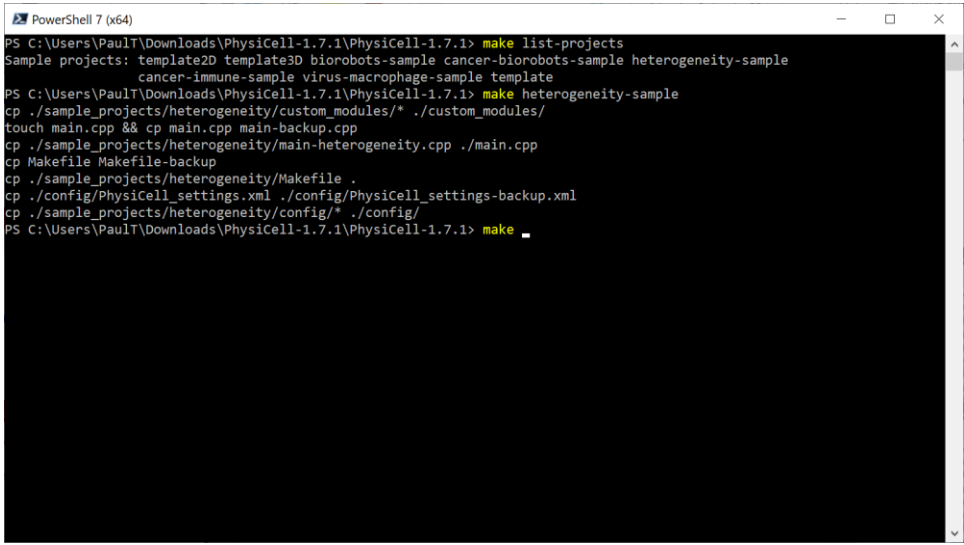


- **Reset** to a clean slate
  - De-populate the project
  - Get ready for another project
  - `make reset`

Changing settings in a project

First, populate the cancer heterogeneity project

- List all available sample projects
- Populate the cancer heterogeneity project
- Build the project
- Change some settings (next slide)



How to change settings in XML

- Open config/PhysiCell_settings.xml
- Major sections:
  - **domain** -- how big of a region to simulate
  - **overall** -- how long to simulate, time step sizes
  - **parallel** -- OpenMP settings
  - **save** -- how often to save SVG images and full data
  - **microenvironment** -- settings on diffusing substrates
  - **user_parameters** -- model-specific settings

Future versions:

- **cell_definitions** -- set default cell properties

Exercise: change settings and run

- Let's set the maximum simulation time to 2160 minutes
- Let's set the domain to [-500,500] x [-500,500] to speed it up
- Let's set the oncoprotein standard deviation to 3
- Let's set the max oncoprotein to 9 (3 standard deviations)
- Compile and run as before.

Let's set options and run (1)

- Open `./config/PhysiCell-settings.xml`
- Let's set the domain size in the **domain** block
  - Switch to [-500,500] x [-500,500] x [-10,10] to speed it up

```
<PhysiCell_settings version="devel-version">
        <domain>
                <x_min>-500</x_min>
                <x_max>500</x_max>
                <y_min>-500</y_min>
                <y_max>500</y_max>
                <z_min>-10</z_min>
                <z_max>10</z_max>
                <dx>20</dx>
                <dy>20</dy>
                <dz>20</dz>
                <use_2D>true</use_2D>
        </domain>
```

Let's set options and run (2)

- Let's also look at the **user_parameters** block
  - Let's change the oncoprotein standard deviation (`oncoprotein_sd`) to 3 (more variation)
  - Let's change the max oncoprotein (`oncoprotein_max`) to mean + 3 sds = 1 + 9 = 10

```
<user_parameters>
  <tumor_radius type="double" units="micron">250.0</tumor_radius>
  <oncoprotein_mean type="double" units="dimensionless">
    1.0</oncoprotein_mean>
  <oncoprotein_sd type="double" units="dimensionless">3.0</oncoprotein_sd>
  <oncoprotein_min type="double" units="dimensionless">0.0</oncoprotein_min>
  <oncoprotein_max type="double" units="dimensionless">10</oncoprotein_max>
  <random_seed type="int" units="dimensionless">0</random_seed>
</user_parameters>
```

Let's set options and run (3)

- Let's look at the **overall** block
  - Set max time to 1.5 days = 1.5 x 24 x 60 = 2160 minutes

```
<overall>
        <max_time units="min">2160</max_time> <!-- 36 h * 60 min -->
        <time_units>min</time_units>
        <space_units>micron</space_units>
```

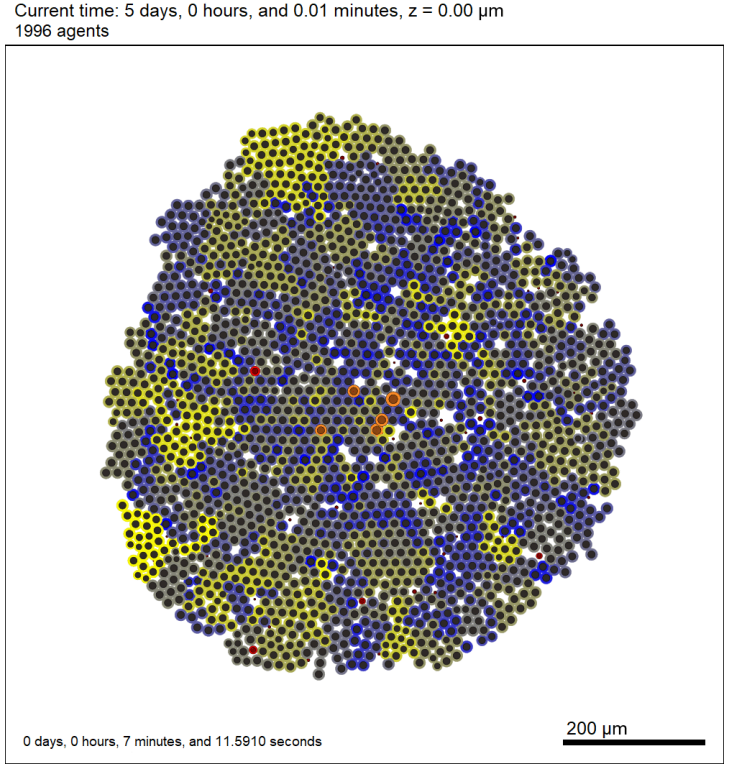
- Let's look at the **save** block
  - Set the full save interval to 6 hours = 360 minutes

```
<save>
        <folder>output</folder> <!-- use . for root -->
        <full_data>
                <interval units="min">360</interval>
                <enable>true</enable>
        </full_data>
```

- Now, run! (`./heterogeneity`)

Let's do a quick visualization

- `magick mogrify -format jpg *.svg`
- `magick convert *.svg out.gif`



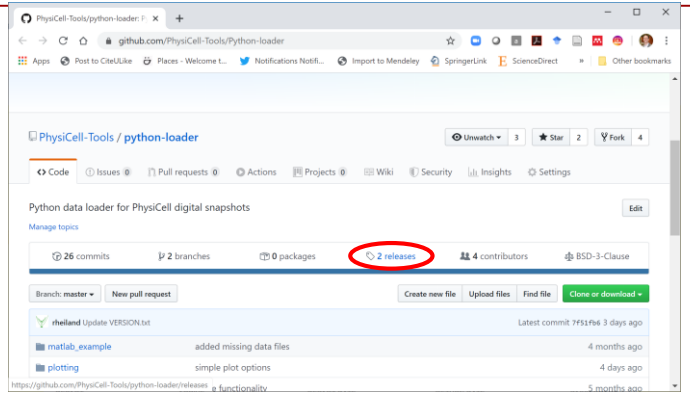
- We can see that the yellow cells eventually "win": they grow faster and form microcolonies within the tumor

- The effect is greatest on the outside edge: They have access to more O_2 here

Let's load some data!

Let's get ready to load in Python

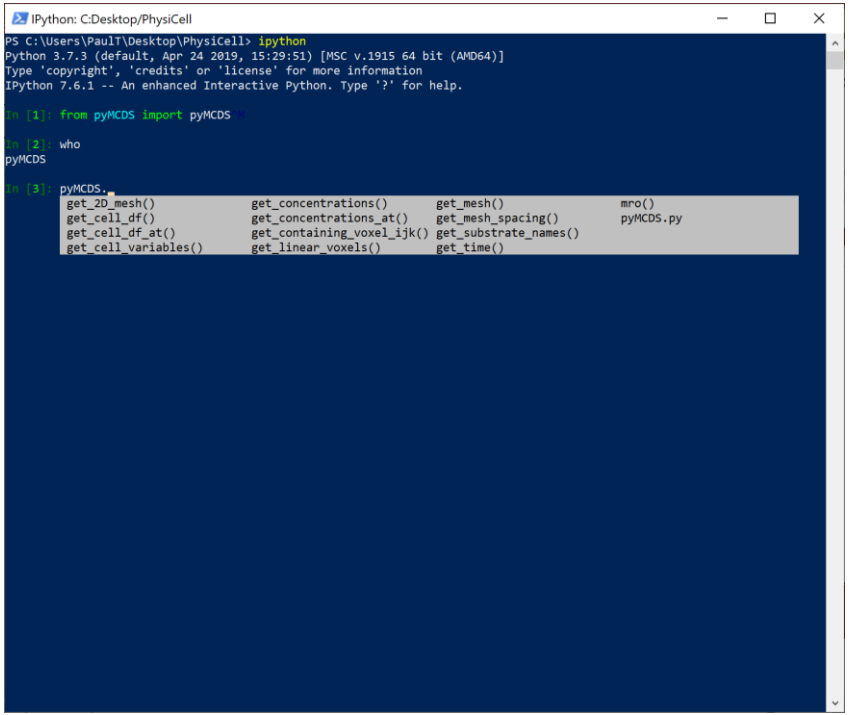
- We'll go to Python-loader and get the source:
  - https://github.com/PhysiCell-Tools/Python-loader



- Get the latest release:
  - Click on "releases" link
  - Click the green "clone or download" button
    - (For simplicity, I'm using "download ZIP" option)
- Copy the following Python file (end in .py) to the root of PhysiCell
  - `pyMCDS`

Let's get started in ipython

- Start ipython (interactive python)
  - `ipython`
- Import the python loader:
  - `from pyMCDS import pyMCDS`
- Import other useful things
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
- Let's see what is available.
  - Type `pyMCDS.`
  - Hit "tab" to autocomplete
- Historical note:
  - MCDS = MultiCellDS, our multicellular data standard



Let's load a single saved time

- Syntax: result = pyMCDS(filename , directory):

```
mcds = pyMCDS('output00000000.xml', 'output')
```

- Let's get some basic info on the snapshot:

```
print(mcds.get_time())  # what simulation time is saved here?
print(mcds.get_cell_variables()) # what data are saved in the cells?
print(mcds.get_substrate_names())  # what diffusing substrates?
```

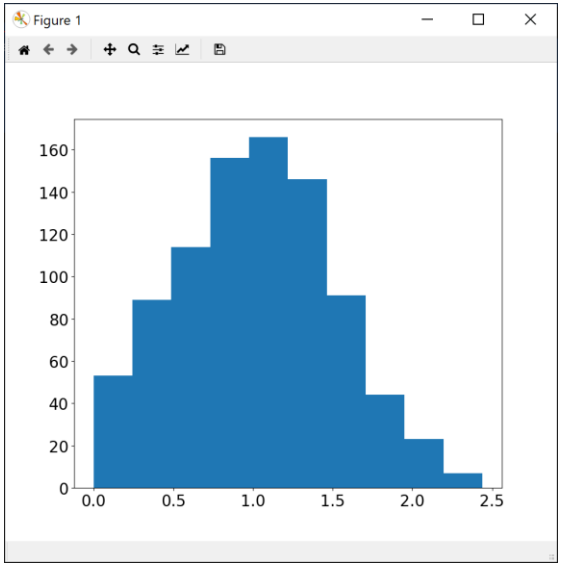
- mcds.data is a dict. Let's see what's available:

```
mcds.data.keys()
Out[41]: dict_keys(['metadata', 'mesh', 'continuum_variables', 'discrete_cells'])
```

Let's access some cell data

- First, let's find out the mean value of the oncoprotein
  - np.mean(mcds.data['discrete_cells']['oncoprotein'])

```
Out[61]: 1.017719876837275
```



- Let's make sure matplotlib doesn't use small fonts

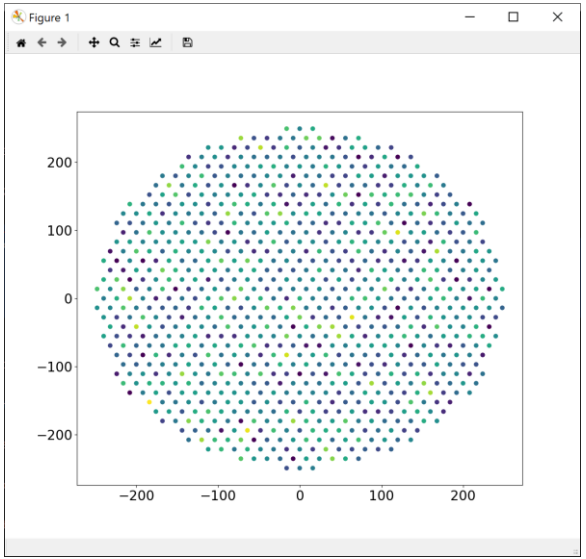
```
import matplotlib
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
```

- Now, let's plot a histogram
  - `plt.hist( mcds.data['discrete_cells']['oncoprotein'] )`

Let's plot the cells

- We'll do a scatter plot of the cells, and color by oncoprotein.
- First, let's grab the data to make our typing easier

```
cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
op = mcds.data['discrete_cells']['oncoprotein']
```



- Now, a scatter plot.
  - Note: these are not plotting by the ***physical*** cell size

```
plt.scatter(cx,cy,c=op)
```

- This plot is pretty ugly. let's improve it.

Improving the plot scatter plot

- Let's replot with bigger dots

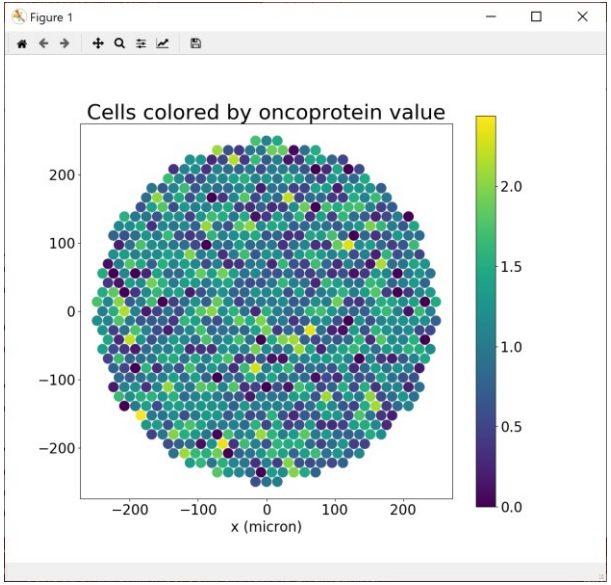
```
plt.clf()
plt.scatter( cx , cy, c=op, s=40 )
```

- Make sure aspect ratio is right:

```
plt.axis( 'image' )
```

- Now, let's add a colorbar

```
plt.colorbar()
```

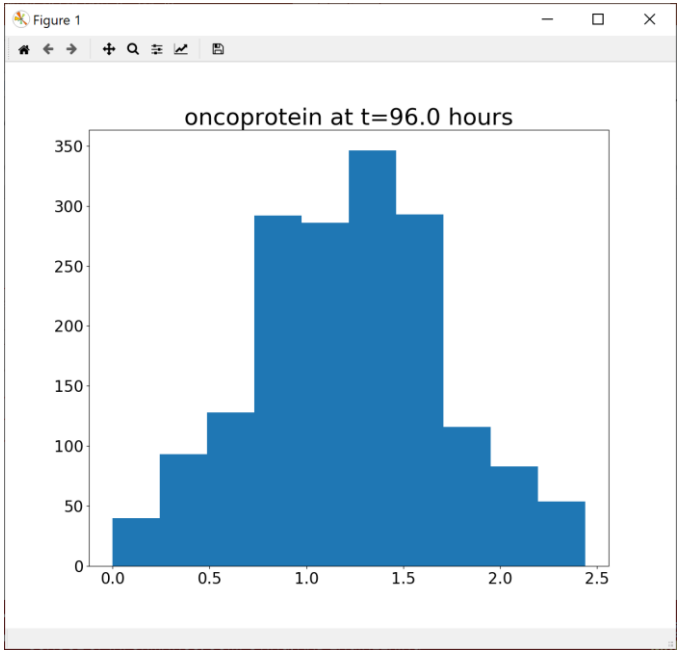


- Now, let's add labels

```
plt.title( 'Cells colored by oncoprotein value' , size=30)
plt.xlabel( 'x' , size=20 )
plt.ylabel( 'y', size=20 )
```

Let's load another time

```
mcds = pyMCDS('output00000006.xml', 'output')
t=mcds.get_time()
cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
op = mcds.data['discrete_cells']['oncoprotein']
plt.clf()
plt.hist( op )
plt.title( 'oncoprotein at t=' + \
str(t/60) + ' hours' , size=30)
```



Let's find live and dead cells

- Each cycle model has a unique code
  - Codes ≥ 100 denote death cycles

- Let's get the cycle code of each cell, and convert to integers

```
cycle = mcds.data['discrete_cells']['cycle_model']
cycle = cycle.astype( int )
```

- Let's find the live cells

```
live = np.argwhere( cycle < 100 ).flatten()
dead = np.argwhere( cycle >= 100 ).flatten()
```

Let's work with these

- Live and dead cell counts

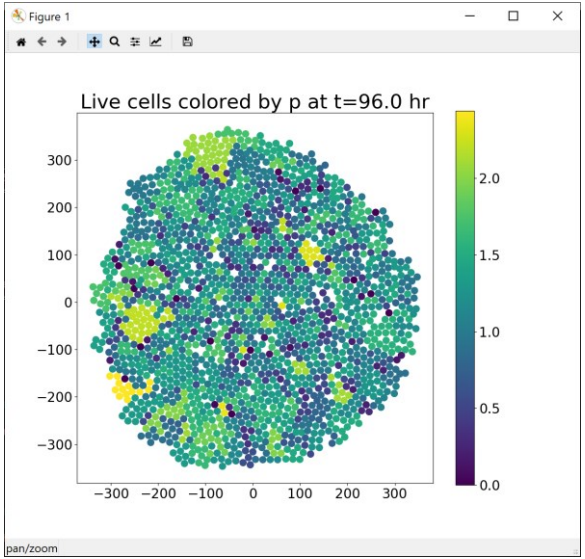
```
n_live = len( live )
n_dead = len( dead )
```

- Mean oncoprotein in live cells only

```
np.mean( op[live] )
```

- Let's scatter plot of only live cells

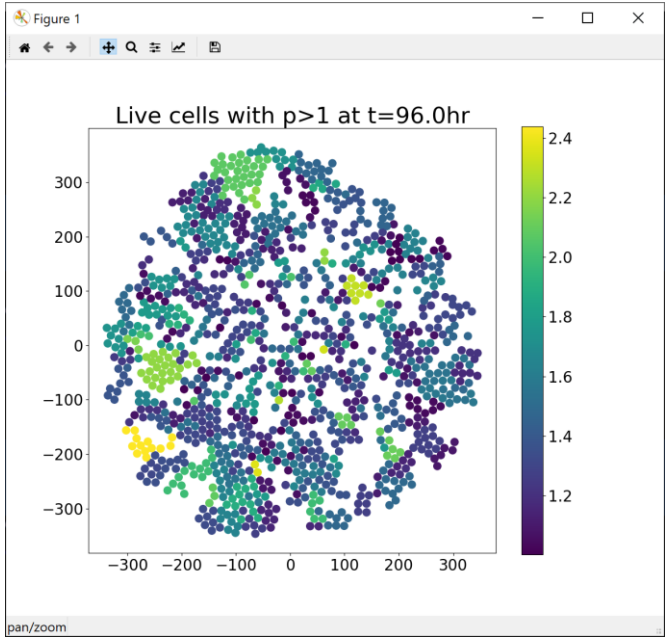
```
plt.clf()
plt.scatter( cx[live],cy[live],c=op[live],s=40);
plt.colorbar()
plt.axis('image')
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr', size=30)
```



Let's do a fancier search

- Only plot live cells with $p > 4$:

```
ind = np.argwhere( (cycle<100) & (op>4) ) .flatten()
plt.clf()
plt.scatter( cx[ind], cy[ind], c=op[ind], s=40 )
plt.title( 'Live cells with p>4 at t='\
+str(t/60) + 'hr', size=30)
plt.axis('image')
plt.colorbar()
```



- **Note:** The circle size (s=40) will vary based on your desktop resolution and window size. You will need to experiment.

Now let's plot the oxygen

```
plt.clf()
mcds.get_substrate_names();

o2 = mcds.get_concentrations( 'oxygen' );
X,Y = mcds.get_2D_mesh();

plt.clf()
plt.contourf(X,Y,o2[:,:,0]);
```

Now let's plot the oxygen with cells

```
circle_size = 30

plt.clf()
mcds.get_substrate_names();

o2 = mcds.get_concentrations( 'oxygen' );
X,Y = mcds.get_2D_mesh();
plt.contourf(X,Y,o2[:,:,0],cmap='spring');

plt.scatter( cx[live],cy[live],c=op[live],s=circle_size);
plt.colorbar()
plt.axis('image')
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr', size=30)

# let's plot dead cells as black
plt.scatter( cx[dead],cy[dead],c='k',s=circle_size );
```

Now, let's do some time series analysis

- Let's get live and dead cell counts, mean *p* (in live cells). We need to loop overall simulation times

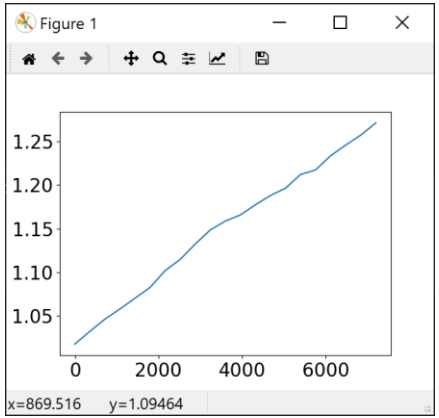
```
last_index = 6;
live_count = np.zeros( last_index+1 );
dead_count = np.zeros( last_index+1 );
mean_p = np.zeros( last_index+1 );
std_p = np.zeros( last_index+1 );
times = np.zeros( last_index+1 );
for n in range( 0,last_index+1 ):
     filename='output'+"%08i"%n+'.xml'
     mcds=pyMCDS(filename,'output')
     times[n]= mcds.get_time()
     cycle=mcds.data['discrete_cells']['cycle_model']
     p = mcds.data['discrete_cells']['oncoprotein']
     live = np.argwhere(cycle<100).flatten()
     dead = np.argwhere(cycle>=100).flatten()
     live_count[n] = len(live)
     dead_count[n] = len(dead)
     mean_p[n] = np.mean( p[live] )
     std_p[n] = np.std( p[live] )
```

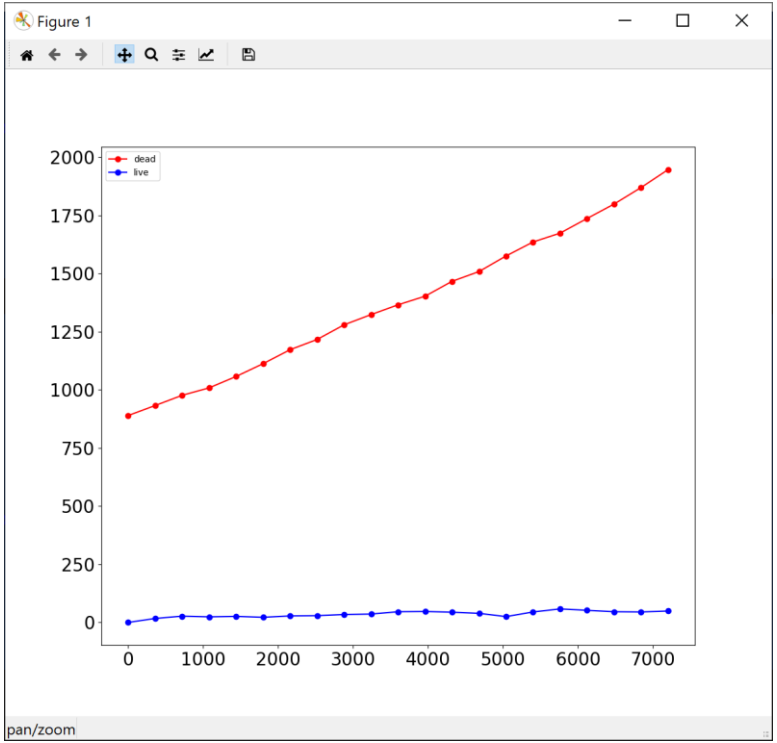
Let's plot and get growth rates

```
plt.clf()
plt.plot( times, live_count , 'r-o' )
plt.plot( times, dead_count , 'b-o' );
plt.legend( {'live', 'dead' } )
```

```
poly=np.polyfit( times,np.log(live_count),1)
# growth rate is 0th element
# in units of 1/min
```

```
plt.clf()
plt.plot(times,mean_p);
```





More data loading (on your own)

Let's work on data with multiple types

- Let's go and run the biorobots sample

```
make data-cleanup
make reset
make biorobots-sample
make
```

- Edit the config file to only run to 1440 min, and save every 240 min

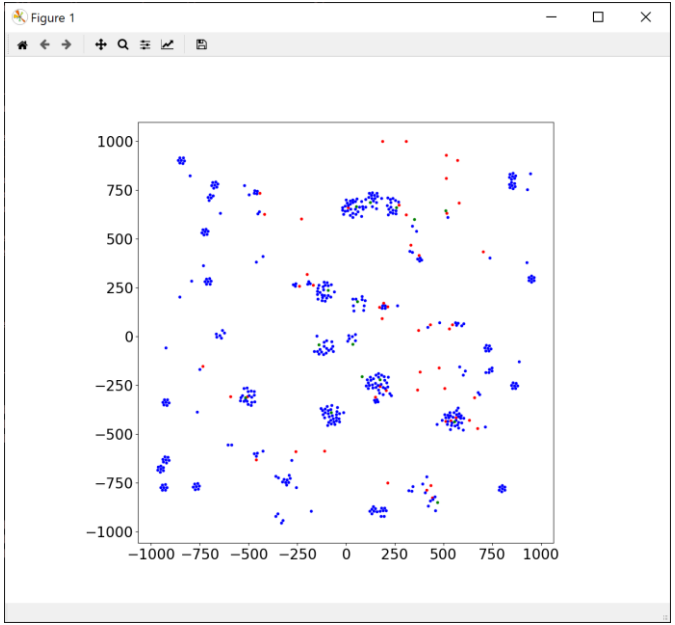
```
./biorobots
```

Let's load an intermediate time

```
n = 3
filename='output'+"%08i"%n+'.xml'
mcds=pyMCDS(filename,'output')
t = mcds.get_time()
cell_type=mcds.data['discrete_cells']['cell_type']
cell_type=cell_type.astype(int)

ind0 = np.argwhere(cell_type==0).flatten();
ind1 = np.argwhere(cell_type==1).flatten();
ind3 = np.argwhere(cell_type==3).flatten();

cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
```

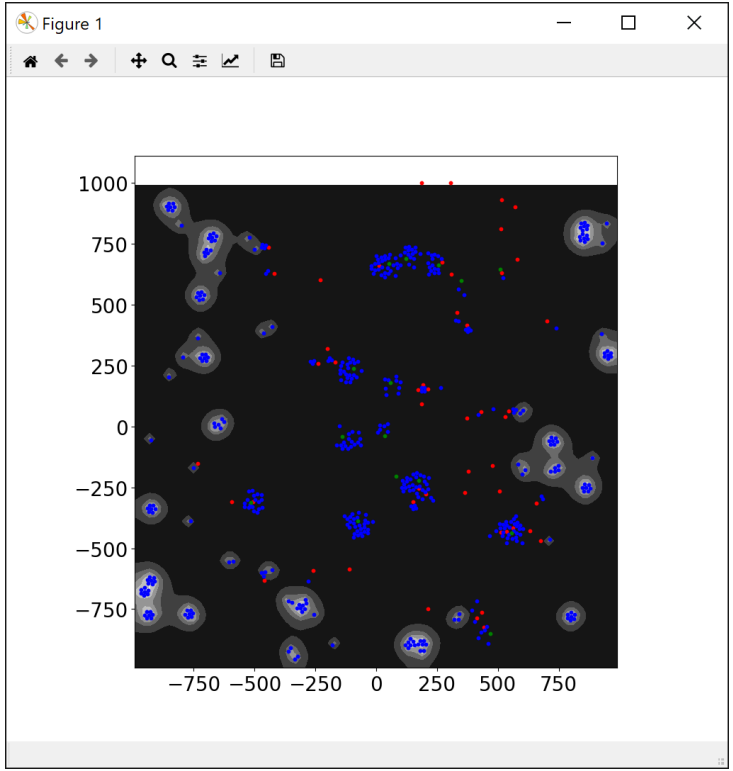
Let's plot each type a different color

```
circle_size=10

plt.clf()
plt.scatter(cx[ind0],cy[ind0],c='r',s=circle_size)
plt.scatter(cx[ind1],cy[ind1],c='b',s=circle_size)
plt.scatter(cx[ind3],cy[ind3],c='g',s=circle_size)
plt.axis('image');
```



Overlay on cargo signal

```
mcds.get_substrate_names();

cs = mcds.get_concentrations( 'cargo signal' );
X,Y = mcds.get_2D_mesh();

plt.clf()
plt.contourf(X,Y,cs[:,:,0],cmap='gray');


plt.scatter(cx[ind0],cy[ind0],c='r',s=circle_size)
plt.scatter(cx[ind1],cy[ind1],c='b',s=circle_size)
plt.scatter(cx[ind3],cy[ind3],c='g',s=circle_size)
plt.axis('image');
```



Let's build a new model!

Modeling task

- Iteratively build up a cancer model, starting simple and adding more
- **Model 1:**
  - Proliferating cancer cells with motility
  - Oxygen uptake
  - Can mostly be designed in XML
- **Model 2:**
  - Refine model 1 to add an aggressive subclone
    - more motile and proliferative, but higher O2 uptake rate
- **Model 3:**
  - Refine model 2 so that proliferation and death are oxygen-based
  - Add a small probability that

Model 1

- In the XML configuration file:
  - Define diffusing substrates
  - Create cell definition(s)
  - Set time, domain, other settings
  - Custom parameters as needed

- In the custom.cpp file:
  - Add cells in the initial simulation

Use the template project

- clean up date (if any)

```
make data-cleanup
```

- reset to blank slate

```
make reset
```

- populate and compile the (new) template project

```
make template
make
```

XML file: substrates

- Let's do oxygen with standard diffusion coefficients, decay for a 1 mm length scale, and 5% (38 mmHg) Dirichlet boundary condition

$$L = \sqrt{\frac{D}{\lambda + U}}$$

```
<microenvironment_setup>
        <variable name="oxygen" units="mmHg" ID="0">
                <physical_parameter_set>
                        <diffusion_coefficient units="micron^2/min">100000.0</diffusion_coefficient>
                        <decay_rate units="1/min">0.1</decay_rate>
                </physical_parameter_set>
                <initial_condition units="mmHg">38</initial_condition>
                <Dirichlet_boundary_condition units="mmHg" enabled="true">38</Dirichlet_boundary_condition>
        </variable>

        <options>
                <calculate_gradients>true</calculate_gradients>
                <track_internalized_substrates_in_each_agent>true</track_internalized_substrates_in_each_agent>
        </options>
</microenvironment_setup>
```

XML file: default cell definition

- The default cell definition sets default parameters for all cell types.
- It also sets custom variables available for all other cell types.

```
<cell_definitions>
        <cell_definition name="default" ID="0">
                <phenotype/>
                <custom_data/>
        </cell_definition>
```

- If you create a cell in a simulation without choosing a cell definition, it will use the default cell definition.

- I suggest mostly leaving it alone. But make sure the <secretion> section has the same variables as the microenvironment. (Some day we will write an XML validator. It would be a great community project!)
- But definitely add custom variables the default definition so that all cell types have the same custom variables.

- We don't need any for now.

XML file: default cell definition - secretion

- Two gotcha's:
  - Need to make sure that secretion and chemotaxis only reference things that are defined in your microenvironment.

```
<secretion>
        <substrate name="oxygen">
                <secretion_rate units="1/min">0</secretion_rate>
                <secretion_target units="substrate density">1</secretion_target>
                <uptake_rate units="1/min">10</uptake_rate>
                <net_export_rate units="total substrate/min">0</net_export_rate>
        </substrate>
</secretion>
```

XML file: default cell definition - secretion

- Two gotcha's:
  - Need to make sure that secretion and chemotaxis only reference things that are defined in your microenvironment.

```
<motility>
        <speed units="micron/min">1</speed>
        <persistence_time units="min">1</persistence_time>
        <migration_bias units="dimensionless">.5</migration_bias>

        <options>
                <enabled>false</enabled>
                <use_2D>true</use_2D>
                <chemotaxis>
                        <enabled>false</enabled>
                        <substrate>oxygen</substrate>
                        <direction>1</direction>
                </chemotaxis>
        </options>
</motility>
```

XML file: cancer cell definition

- We will create a cancer cell definition that "inherits" all the physical rate parameters from the default definition.
- We only need to define how it differs from default:
  - proliferation
  - death (set apoptosis to zero)
  - O2 uptake (no change from default)
  - motility
- Copy the default cell definition. GIve it a new ID (1)

XML file: cancer cell definition

- Copy the default cell definition.
  - Give it a name (cancer)
  - Give it a new ID (1)
  - Make sure it inherits from the default cell definition
- If it's already there, overwrite it. Delete any other cell defs.

```
<cell_definitions>
      <cell_definition name="cancer" ID="1" parent_type="default">
```

XML file: cancer: cycling

- Let's leave all the cycle parameters alone, *except* that we want cells to spend (on average) 1 hour in the G0/G1 cycle phase.
- Let's edit the <cycle> part of <phenotype>

```
<phenotype>
        <cycle code="6" name="Flow cytometry model (separated)">
                <!-- phases are: G0/G1 , S, G2, M -->
                <!-- use phase_transition_rates OR phase_durations -->
                <phase_durations units="min">
                        <duration index="0" fixed_duration="false">60.0</duration>
                </phase_durations>
        </cycle>
```

- Notice that we only wrote the rate that differs from the default model.

XML file: cancer: cycling (alternative)

- Alternatively, you can set a transition rate from phase 0 (G0/G1) to phase 1 (S). For a mean holding time of 60 min, the rate is 1/60 min^{-1}

```
<phenotype>
  <cycle code="6" name="Flow cytometry model (separated)">
    <!-- phases are: G0/G1 , S, G2, M -->
    <!-- use phase_transition_rates OR phase_durations -->
    <phase_transition_rates units="1/min">
  <rate start_index="0" end_index="1" fixed_duration="false">0.016667</rate>
    </phase_transitions>
  </cycle>
```

- You can use durations ***OR*** rates, but not both within one cell def.

XML file: cancer: death

- Let's set the apoptosis and necrosis rates to zero

```
<death>
        <model code="100" name="apoptosis">
                <death_rate units="1/min">0</death_rate>
        </model>

        <model code="101" name="necrosis">
                <death_rate units="1/min">0.0</death_rate>
        </model>
</death>
```

XML file: cancer: motility

- Let's set biased random migration up oxygen gradients.

```
<motility>
        <speed units="micron/min">1</speed>
        <persistence_time units="min">1</persistence_time>
        <migration_bias units="dimensionless">.1</migration_bias>

        <options>
                <enabled>true</enabled>
                <use_2D>true</use_2D>
                <chemotaxis>
                        <enabled>true</enabled>
                        <substrate>oxygen</substrate>
                        <direction>1</direction>
                </chemotaxis>
        </options>
</motility>
```

XML file: custom parameters

- Let's add a parameters for:
  - initial number of cells
  - max |x| and |y| coordinates

```
<user_parameters>
      <random_seed type="int" units="dimensionless">0</random_seed>
      <!-- example parameters from the template -->

      <number_of_cancer type="int" units="none"
            description="initial number of cancer cells">50</number_of_cancer>
      <coordinate_max type="double" units="micron"
            description="max |x| and |y| coordinates at
            start">100</coordinate_max>

</user_parameters>
```

custom.cpp (1)

- Our cells are set up. Now we need to place some
- Open ./custom_modules/custom.cpp
- Look for a function called setup_tissue

- We'll read our parameter for number of cells, and place them randomly within the coordinate bounds

custom.cpp (2)

```
void setup_tissue( void )
{
   double min = -parameters.doubles( "coordinate_max" );
   double max = parameters.doubles( "coordinate_max" );
   double range = max - min;

   Cell* pC;
   // place cancer cells

   for( int n = 0 ; n < parameters.ints("number_of_cancer") ; n++ )
   {
      std::vector<double> position = {0,0,0};
      position[0] = min + UniformRandom()*range; // choose position
      position[1] = min + UniformRandom()*range;
      pC = create_cell( get_cell_definition("cancer") ); // place a cancer cell
      pC->assign_position( position ); // set its position
   }

   return;
}
```

Modify the color function

```
std::vector<std::string> my_coloring_function( Cell* pCell )
{
   static int cancer_type = get_cell_definition( "cancer" ).type;

   // start with flow cytometry coloring

   std::vector<std::string> output = false_cell_coloring_cytometry(pCell);

   // if it's not a cancer cell, color it black

   if( pCell->phenotype.death.dead == false && pCell->type != cancer_type )
   {
       output[0] = "black";
       output[2] = "black";
   }

   return output;
}
```

One last thing:

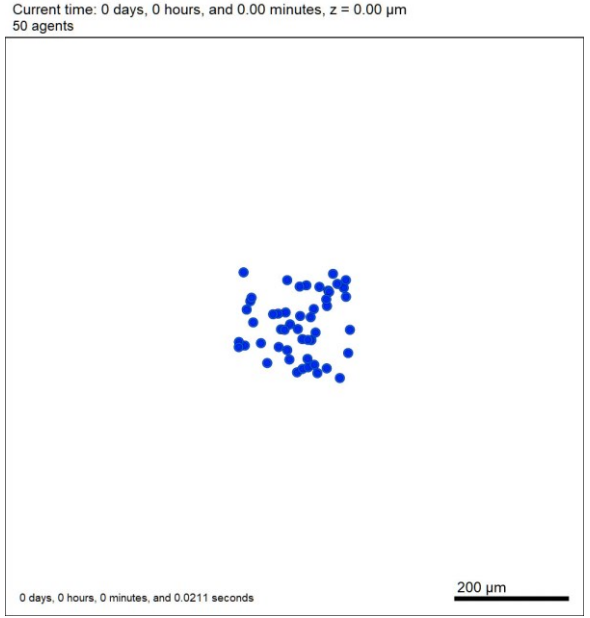
- Make sure there aren't any extra functions in the cell setup
- (Future versions of PhysiCell will have a cleaner template project!)
- Search for the function create_cell_types
- Make sure this section is empty:

```
        /*
           Put any modifications to individual cell definitions here.

           This is a good place to set custom functions.
        */
// nothing here!
        /*
           This builds the map of cell definitions and summarizes the setup.
        */
```

Build and run!

Set the max time to 1 day (1440 minutes), and output every 15 minutes

```
make
./project
```



Next steps

- I'll add Models 2 and 3 as exercises, with code solutions.
- A great way to learn is to look at existing codes. Look at the sample projects. See how they work. Give them a try!
- We are developing a detailed suite of training materials. Stay tuned!
- We are happy to meet one-on-one to consult on possible projects. Just give us a shout, and we'll help set up your project!

Some models to explore

On nanoHUB:

- **pc4heterogen:** heterogeneous cancer growth (https://nanohub.org/tools/pc4heterogen)
- **pc4cancerbots:** use the "biorobots" as a cell-based cancer therapy (https://nanohub.org/tools/pc4cancerbots)
- **pc4livermedium:** tumor-stroma biomechanical feedbacks (https://nanohub.org/tools/pc4livermedium)
- **pc4cancerimmune:** basic cancer immunotherapy model (https://nanohub.org/tools/pc4cancerimmune)
- **pc4covid19:** COVID-19 simulation model (https://nanohub.org/tools/pc4covid19)
- **trmotility**: training on biased random cell migration (https://nanohub.org/tools/trmotility)
- **pc4thanos:** *Avengers Endgame* battle using cell rules (https://nanohub.org/tools/pc4thanos)

Bundled in PhysiCell:

- biorobots, cancer biorobots, heterogeneity, cancer immunotherapy (3D version), virus-macrophage sample, project templates

Further reading (1)

- **BioFVM method paper (3-D diffusion)**
  A. Ghaffarizadeh, S.H. Friedman, and P. Macklin. BioFVM: an efficient, parallelized diffusive transport solver for 3-D biological simulations. *Bioinformatics* 32(8):1256-8, 2016. DOI: 10.1093/bioinformatics/btv730.

- **PhysiCell method paper (agent-based model)**
  A. Ghaffarizadeh, R. Heiland, S.H. Friedman, S.M. Mumenthaler, and P. Macklin. PhysiCell: an open source physics-based cell simulator for 3-D multicellular systems. *PLoS Comput. Biol.* 14(2):e1005991, 2018. DOI: 10.1371/journal.pcbi.1005991.

- **PhysiBoSS (PhysiCell + MaBoSS for Boolean networks)**
  G. Letort, A. Montagud, G. Stoll, R. Heiland, E. Barillot, P. Macklin, A. Zinovyev, and L. Calzone. PhysiBoSS: a multi-scale agent based modelling framework integrating physical dimension and cell signalling. *Bioinformatics* 35(7):1188-96, 2019. DOI: 10.1093/bioinformatics/bty766.

- **xml2jupyter paper (create GUIs for cloud-hosted models)**
  R. Heiland, D. Mishler, T. Zhang, E. Bower, and P. Macklin. xml2jupyter: Mapping parameters between XML and Jupyter widgets. *Journal of Open Source Software* 4(39):1408, 2019. DOI: 10.21105/joss.01408.

- **PhysiCell+EMEWS (high-throughput 3D PhysiCell investigation)**
  J. Ozik, N. Collier, J. Wozniak, C. Macal, C. Cockrell, S.H. Friedman, A. Ghaffarizadeh, R. Heiland, G. An, and P. Macklin. High-throughput cancer hypothesis testing with an integrated PhysiCell-EMEWS workflow. *BMC Bioinformatics* 19:483, 2018. DOI: 10.1186/s12859-018-2510-x.

- **PhysiCell+EMEWS 2 (HPC accelerated by machine learning)**
  J. Ozik, N. Collier, R. Heiland, G. An, and P. Macklin. Learning-accelerated Discovery of Immune-Tumour Interactions. *Molec. Syst. Design Eng.* 4:747-60, 2019. DOI: 10.1039/c9me00036d.

Further reading (2)

- **A review of cell-based modeling (in cancer):**

  J. Metzcar, Y. Wang, R. Heiland, and P. Macklin. A review of cell-based computational modeling in cancer biology. *JCO Clinical Cancer Informatics* 3:1-13, 2019 (invited review). DOI: 10.1200/CCI.18.00069.

- **Progress on multicellular systems biology:**

  P. Macklin, H.B. Frieboes, J.L. Sparks, A. Ghaffarizadeh, S.H. Friedman, E.F. Juarez, E. Jockheere, and S.M. Mumenthaler. "Progress Towards Computational 3-D Multicellular Systems Biology". In: . Rejniak (ed.), *Systems Biology of Tumor Microenvironment*, chap. 12, pp. 225-46, Springer, 2016. ISBN: 978-3-319-42021-9. (invited author: P. Macklin). DOI: 10.1007/978-3-319-42023-3_12.

- **Challenges for data-driven multicellular systems biology**

  P. Macklin. Key challenges facing data-driven multicellular systems biology. *GigaScience* 8(10):giz127, 2019. DOI: 10.1093/gigascience/giz127

- **COVID-19 community preprint**

  Y. Wang et al., Rapid community-driven development of a SARS-CoV-2 tissue simulator. *bioRxiv* 2020.04.02.019075 (2020). DOI: 10.1101/2020.04.02.019075

Some links

- PhysiCell project & downloads: http://PhysiCell.org
- Twitter updates: https://twitter.com/PhysiCell
- Tutorials: http://www.mathcancer.org/blog/physicell-tutorials/
- Tools (in progress): https://github.com/PhysiCell-Tools
- Training (in progress): https://github.com/PhysiCell-Training
- Wiki (in progress): http://PhysiCell.org/wiki