Lab 5: Hands-on with PhysiCell (Day 1)

Get lectures and materials here!

Paul Macklin, Ph.D.

Intelligent Systems Engineering
Indiana University

January 27, 2020

LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Macklin Lab
@MathCancer
MathCancer.org

Agenda (first half):

Part 1: Getting familiar with PhysiCell

- Download PhysiCell
- Work with Projects (pre-bundled)
- Build and run a first pre-bundled sample project
- Quick visualization with SVG files
- Create simple animated GIFs from simulation data
- Clean up data and reset to a blank slate

Part 2: Modifying projects in PhysiCell

- Explore project structure
- Define the microenvironment (XML)
- Define and access custom parameters (XML)
- Introduce key cell data
- Introduce cell definitions
- **Exercise 1:** Modify parameters and default cell definition in cancer-biorobots project

BREAK

Agenda (second half):

Part 3: Making new projects in PhysiCell

- Introduce general modeling workflow
- Work with cell definitions
- Introduce functions in PhysiCell
- Attach functions to Cell Definitions
- Add custom data to cells
- Setup routines
- Place new cells
- Coloring functions (for SVG)
- **Exercise 2:** Make a 3D project with basic metabolism and energy-dependent cycling and death

Part 4: Special topics

- Introduce tools ecosystem (and povwriter)
- Create mpeg4 movies

Part 1
Getting familiar with PhysiCell

Agenda (first half):

Part 1: Getting familiar with PhysiCell

- Download PhysiCell
- Work with Projects (pre-bundled)
- Build and run a first pre-bundled sample project
- Quick visualization with SVG files
- Create simple animated GIFs from simulation data
- Clean up data and reset to a blank slate

Part 2: Modifying projects in PhysiCell

- Explore project structure
- Define the microenvironment (XML)
- Define and access custom parameters (XML)
- Introduce key cell data
- Introduce cell definitions
- **Exercise 1:** Modify parameters and default cell definition in cancer biorobots project

BREAK

Agenda (first half):

Part 1: Getting familiar with PhysiCell

- Download PhysiCell
- Work with Projects (pre-bundled)
- Build and run a first pre-bundled sample project
- Quick visualization with SVG files
- Create simple animated GIFs from simulation data
- Clean up data and resetting

Part 2: Modifying projects in PhysiCell

- Explore project structure
- Define the microenvironment (XML)
- Define and access custom parameters (XML)
- Introduce key cell data
- Introduce cell definitions
- **Exercise 1:** Modify parameters and default cell definition in cancer biorobots project

BREAK

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

Sample projects

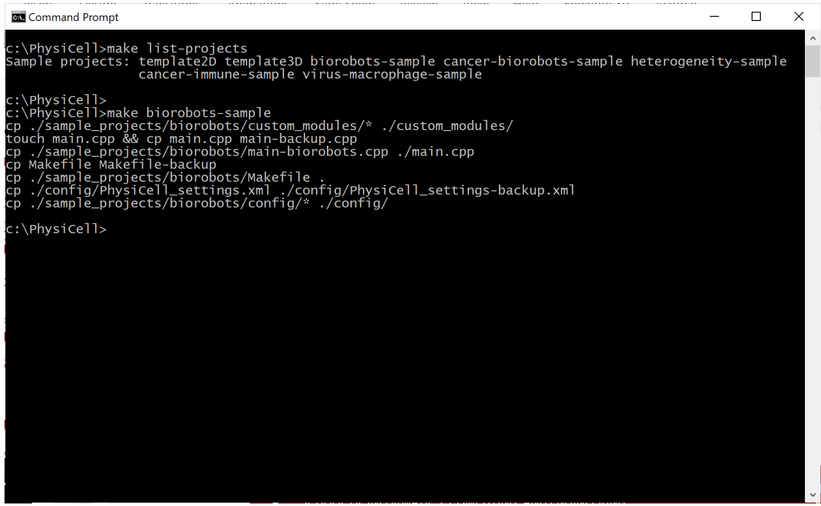
- It's inefficient (and a little insane) to code new projects *entirely* from scratch.
- So, we provide sample projects:
  - 2D and 3D template projects
  - Cancer models
  - Synthetic multicellular systems
  - Viral dynamics in tissue

- **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



Command Prompt

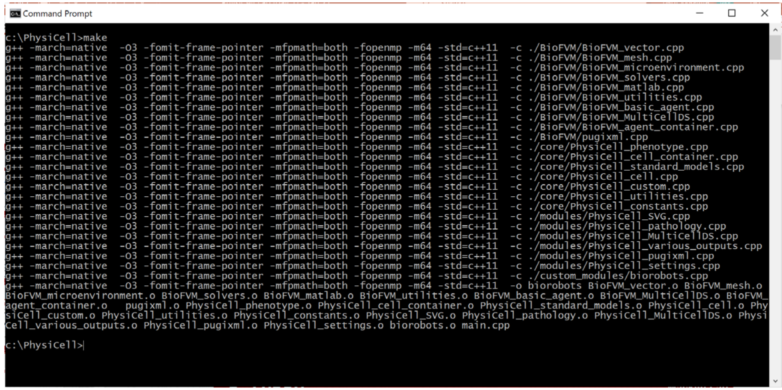
```
c:\PhysiCell>make list-projects
Sample projects: template2D template3D biorobots-sample cancer-biorobots-sample heterogeneity-sample
                 cancer-immune-sample virus-macrophage-sample

c:\PhysiCell>
c:\PhysiCell>make biorobots-sample
cp ./sample_projects/biorobots/custom_modules/* ./custom_modules/
touch main.cpp && cp main.cpp main-backup.cpp
cp ./sample_projects/biorobots/main-biorobots.cpp ./main.cpp
cp Makefile Makefile-backup
cp ./sample_projects/biorobots/Makefile .
cp ./config/PhysiCell_settings.xml ./config/PhysiCell_settings-backup.xml
cp ./sample_projects/biorobots/config/* ./config/

c:\PhysiCell>
```

PhysiCell Project Essentials (2)

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

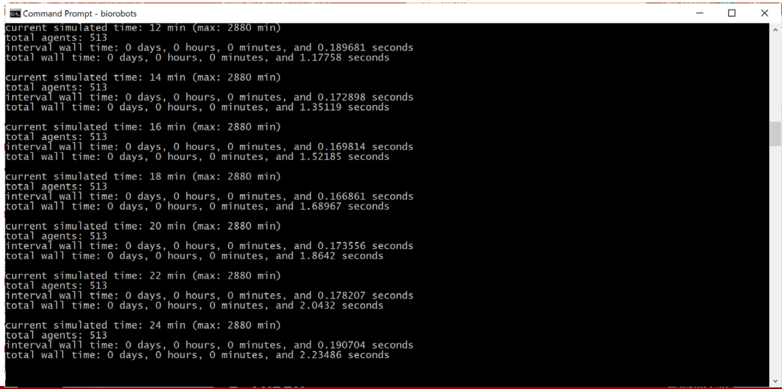
- Then, **run the project**
  - `./biorobots` (Linux, OSX)
  - `biorobots.exe` (Windows)

- This should take about 5 minutes



Command Prompt

```
c:\PhysiCell>make
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_vector.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_mesh.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_microenvironment.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_solvers.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_matlab.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_utilities.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_basic_agent.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_MultiCellDS.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/BioFVM_agent_container.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./BioFVM/pugixml.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_phenotype.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_cell_container.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_standard_models.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_cell.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_custom.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_utilities.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./core/PhysiCell_constants.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_SVG.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_pathology.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_MultiCellDS.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_various_outputs.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_pugixml.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./modules/PhysiCell_settings.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./custom_modules/biorobots.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -o biorobots BioFVM_vector.o BioFVM_mesh.o
BioFVM_microenvironment.o BioFVM_solvers.o BioFVM_matlab.o BioFVM_utilities.o BioFVM_basic_agent.o BioFVM_MultiCellDS.o BioFVM_
agent_container.o   pugixml.o PhysiCell_phenotype.o PhysiCell_cell_container.o PhysiCell_standard_models.o PhysiCell_cell.o Phy
siCell_custom.o PhysiCell_utilities.o PhysiCell_constants.o PhysiCell_SVG.o PhysiCell_pathology.o PhysiCell_MultiCellDS.o Physi
Cell_various_outputs.o PhysiCell_pugixml.o PhysiCell_settings.o biorobots.o main.cpp

c:\PhysiCell>
```

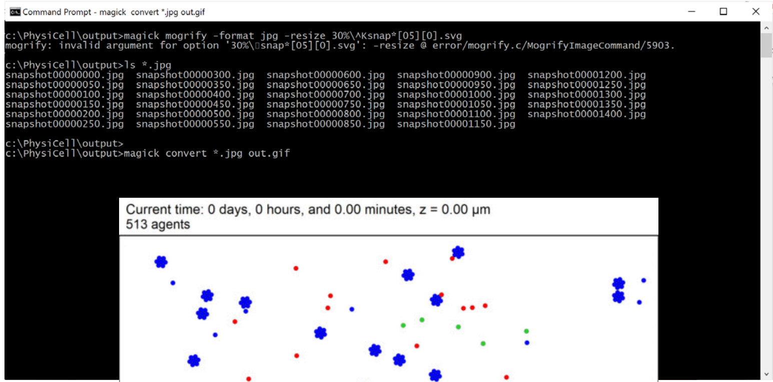


Command Prompt - biorobots

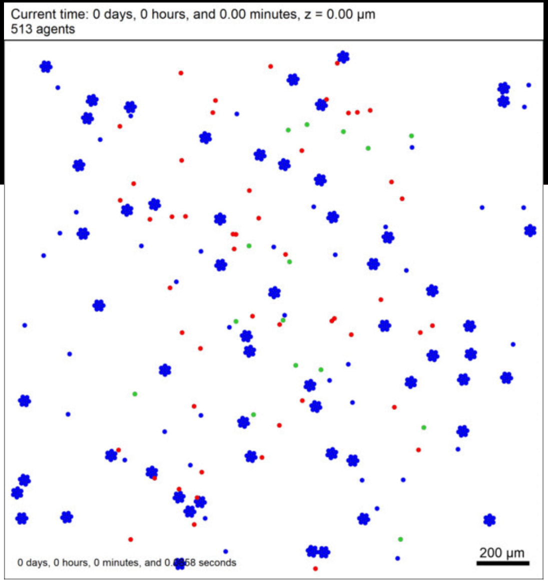
```
current simulated time: 12 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.189681 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 1.17758 seconds

current simulated time: 14 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.172898 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 1.35119 seconds

current simulated time: 16 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.169814 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 1.52185 seconds

current simulated time: 18 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.166861 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 1.68967 seconds

current simulated time: 20 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.173556 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 1.8642 seconds

current simulated time: 22 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.178207 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 2.0432 seconds

current simulated time: 24 min (max: 2880 min)
total agents: 513
interval wall time: 0 days, 0 hours, 0 minutes, and 0.190704 seconds
total wall time: 0 days, 0 hours, 0 minutes, and 2.23486 seconds
```

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`


Command Prompt - magick convert *.jpg out.gif

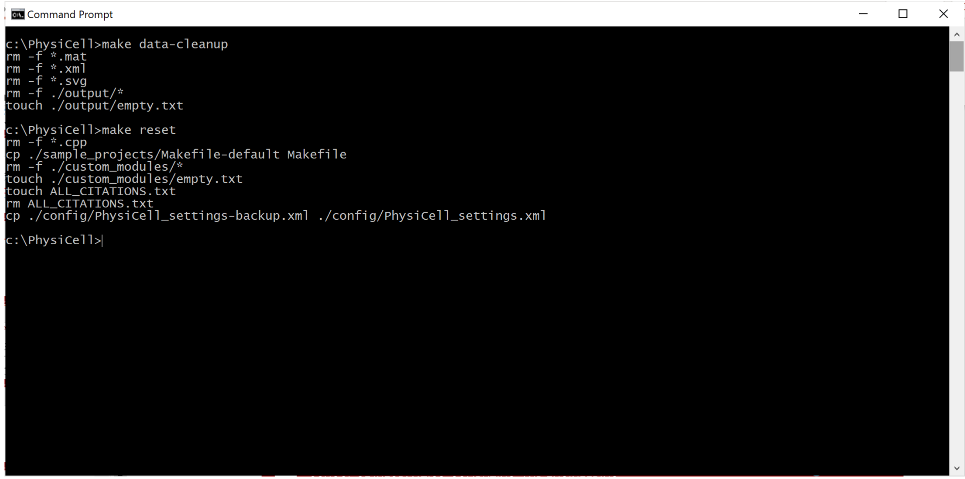
```
c:\PhysiCell\output>magick mogrify -format jpg -resize 30%\^Ksnap*[05][0].svg
mogrify: invalid argument for option '30%\snap*[05][0].svg': -resize @ error/mogrify.c/MogrifyImageCommand/5903.

c:\PhysiCell\output>ls *.jpg
snapshot00000000.jpg  snapshot00000300.jpg  snapshot00000600.jpg  snapshot00000900.jpg  snapshot00001200.jpg
snapshot00000050.jpg  snapshot00000350.jpg  snapshot00000650.jpg  snapshot00000950.jpg  snapshot00001250.jpg
snapshot00000100.jpg  snapshot00000400.jpg  snapshot00000700.jpg  snapshot00001000.jpg  snapshot00001300.jpg
snapshot00000150.jpg  snapshot00000450.jpg  snapshot00000750.jpg  snapshot00001050.jpg  snapshot00001350.jpg
snapshot00000200.jpg  snapshot00000500.jpg  snapshot00000800.jpg  snapshot00001100.jpg  snapshot00001400.jpg
snapshot00000250.jpg  snapshot00000550.jpg  snapshot00000850.jpg  snapshot00001150.jpg

c:\PhysiCell\output>
c:\PhysiCell\output>magick convert *.jpg out.gif
```



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`


Command Prompt

```
c:\PhysiCell>make data-cleanup
rm -f *.mat
rm -f *.xml
rm -f *.svg
rm -f ./output/*
touch ./output/empty.txt

c:\PhysiCell>make reset
rm -f *.cpp
cp ./sample_projects/Makefile-default Makefile
rm -f ./custom_modules/*
touch ./custom_modules/empty.txt
touch ALL_CITATIONS.txt
rm ALL_CITATIONS.txt
cp ./config/PhysiCell_settings-backup.xml ./config/PhysiCell_settings.xml

c:\PhysiCell>
```

Part 2
Modifying projects in PhysiCell

Agenda (first half):

Part 1: Getting familiar with PhysiCell

- Download PhysiCell
- Work with Projects (pre-bundled)
- Build and run a first pre-bundled sample project
- Quick visualization with SVG files
- Create simple animated GIFs from simulation data
- Clean up data and resetting

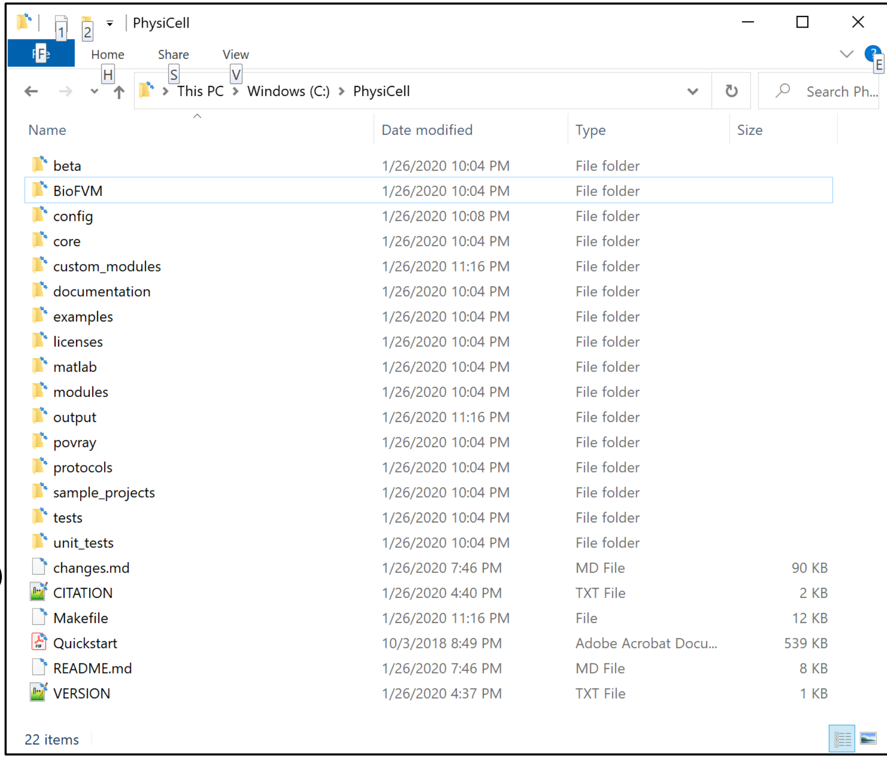
Part 2: Modifying projects in PhysiCell

- Explore project structure
- Define the microenvironment (XML)
- Define and access custom parameters (XML)
- Introduce key cell data
- Introduce cell definitions
- **Exercise 1:** Modify parameters and default cell definition in cancer biorobots project

BREAK

Project structure: overview

- (key) directories:
  - **./ (root):** main source, Makefile, and executable go here
  - **./beta**: for beta-testing (don't use)
  - **./BioFVM:** diffusion solver
  - **./config:** configuration files
  - **./core:** PhysiCell core functions
  - **./custom_modules:** put custom code for your project here.
  - **./documentation:** user guide, etc.
  - **./examples:** deprecated
  - **./licenses:** yep
  - **./matlab:** scripts and functions to load data in matlab
  - **./modules:** standard add-ons for PhysiCell
  - **./output:** where data are stored (by default, but can be changed)
  - **./povray:** deprecated
  - **./protocols:** instructions mostly for maintainers (e.g., release protocols)
  - **./sample_projects:** where we add sample projects
  - **./tests:** for automated testing (WIP
  - **./unit_tests:** for automated testing (WIP)
- **Most of your work will be in the red directories**



PhysiCell

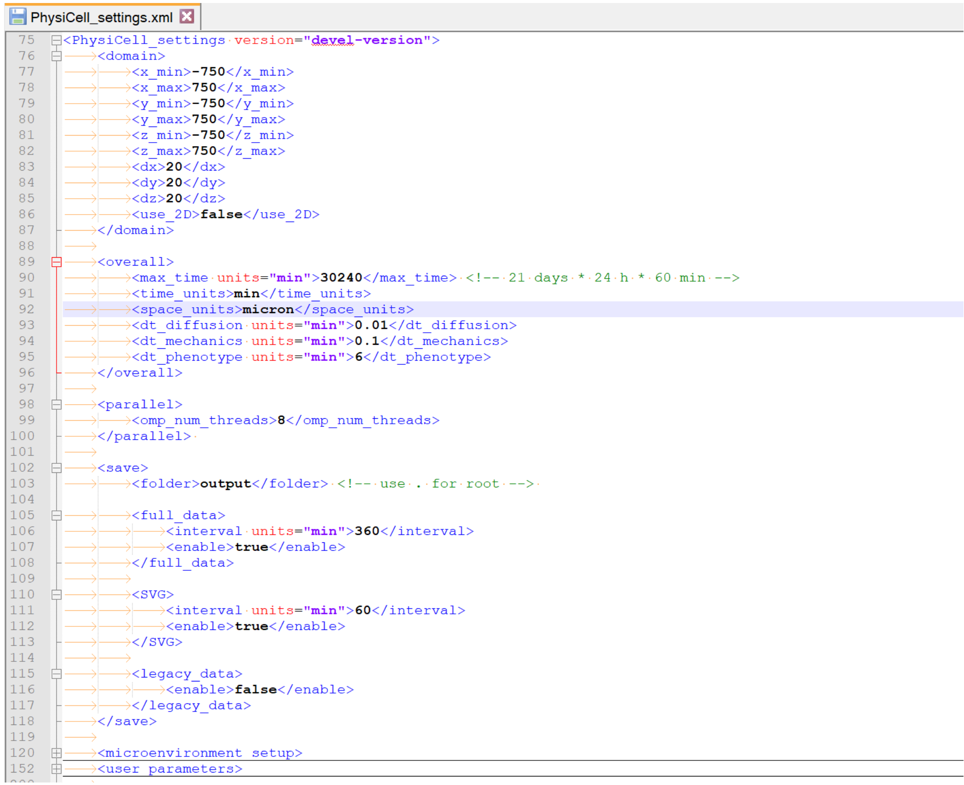
This PC > Windows (C:) > PhysiCell

Name	Date modified	Type	Size
beta	1/26/2020 10:04 PM	File folder	
BioFVM	1/26/2020 10:04 PM	File folder	
config	1/26/2020 10:08 PM	File folder	
core	1/26/2020 10:04 PM	File folder	
custom_modules	1/26/2020 11:16 PM	File folder	
documentation	1/26/2020 10:04 PM	File folder	
examples	1/26/2020 10:04 PM	File folder	
licenses	1/26/2020 10:04 PM	File folder	
matlab	1/26/2020 10:04 PM	File folder	
modules	1/26/2020 10:04 PM	File folder	
output	1/26/2020 11:16 PM	File folder	
povray	1/26/2020 10:04 PM	File folder	
protocols	1/26/2020 10:04 PM	File folder	
sample_projects	1/26/2020 10:04 PM	File folder	
tests	1/26/2020 10:04 PM	File folder	
unit_tests	1/26/2020 10:04 PM	File folder	
changes.md	1/26/2020 7:46 PM	MD File	90 KB
CITATION	1/26/2020 4:40 PM	TXT File	2 KB
Makefile	1/26/2020 11:16 PM	File	12 KB
Quickstart	10/3/2018 8:49 PM	Adobe Acrobat Docu...	539 KB
README.md	1/26/2020 7:46 PM	MD File	8 KB
VERSION	1/26/2020 4:37 PM	TXT File	1 KB

22 items

Project structure: config files

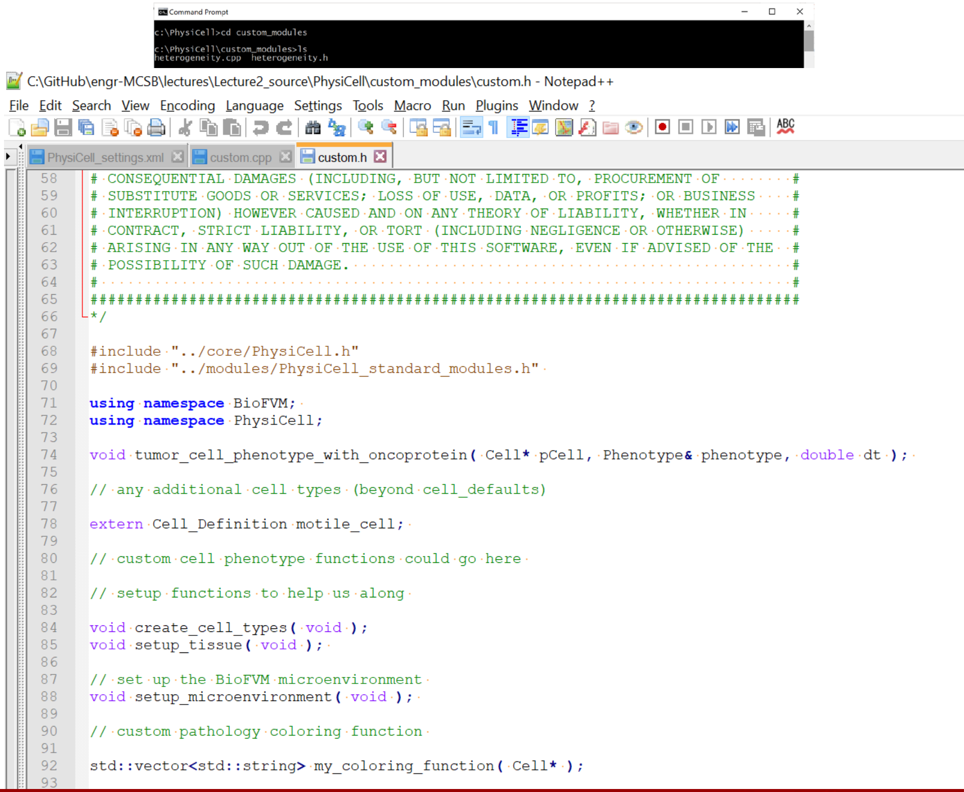
- Configuration files (XML)
  - **domain:** domain size and resolution
  - **overall:** general options
    - Final simulation time
    - Time step sizes
  - **parallel:** parallelization options
    - Number of threads
  - **save:** save options
    - Save where?
    - Save SVGs? (how often?)
    - Save full data? (how often?)
    - Save legacy data (don't)
  - **microenvironment_setup:** diffusion settings
    - more later
  - **user_parameters:** simulation-specific settings
    - more later



```
<PhysiCell_settings version="devel-version">
    <domain>
        <x_min>-750</x_min>
        <x_max>750</x_max>
        <y_min>-750</y_min>
        <y_max>750</y_max>
        <z_min>-750</z_min>
        <z_max>750</z_max>
        <dx>20</dx>
        <dy>20</dy>
        <dz>20</dz>
        <use_2D>false</use_2D>
    </domain>

    <overall>
        <max_time units="min">30240</max_time> <!-- 21 days * 24 h * 60 min -->
        <time_units>min</time_units>
        <space_units>micron</space_units>
        <dt_diffusion units="min">0.01</dt_diffusion>
        <dt_mechanics units="min">0.1</dt_mechanics>
        <dt_phenotype units="min">6</dt_phenotype>
    </overall>

    <parallel>
        <omp_num_threads>8</omp_num_threads>
    </parallel>

    <save>
        <folder>output</folder> <!-- use . for root -->

        <full_data>
            <interval units="min">360</interval>
            <enable>true</enable>
        </full_data>

        <SVG>
            <interval units="min">60</interval>
            <enable>true</enable>
        </SVG>

        <legacy_data>
            <enable>false</enable>
        </legacy_data>
    </save>

    <microenvironment_setup>
    <user_parameters>
```

Project structure: custom modules

- Custom Modules
  - Setup functions
  - Cell definitions
  - Custom functions
  - any other modeling
  - Custom coloring functions

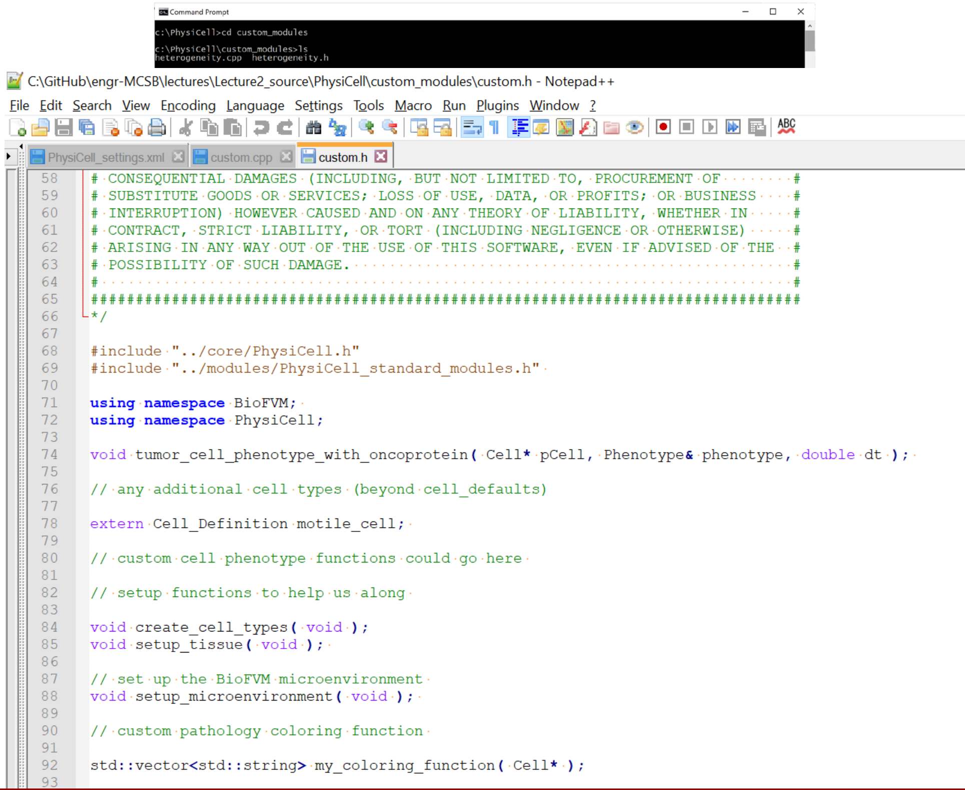

Command Prompt
C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\custom_modules\custom.h - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PhysiCell_settings.xml
custom.cpp
custom.h

```
c:\PhysiCell>cd custom_modules

c:\PhysiCell\custom_modules>ls
heterogeneity.cpp  heterogeneity.h
```

```
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF        #
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS    #
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN     #
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)     #
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  #
# POSSIBILITY OF SUCH DAMAGE.                                                 #
#                                                                             #
###############################################################################
*/

#include "../core/PhysiCell.h"
#include "../modules/PhysiCell_standard_modules.h"

using namespace BioFVM;
using namespace PhysiCell;

void tumor_cell_phenotype_with_oncoprotein( Cell* pCell, Phenotype& phenotype, double dt );

// any additional cell types (beyond cell_defaults)

extern Cell_Definition motile_cell;

// custom cell phenotype functions could go here

// setup functions to help us along

void create_cell_types( void );
void setup_tissue( void );

// set up the BioFVM microenvironment
void setup_microenvironment( void );

// custom pathology coloring function

std::vector<std::string> my_coloring_function( Cell* );
```

Project structure: custom modules

- Custom Modules
  - Any user-defined globals (at top)
    - Declared cell types
  - Setup functions
    - `create_cell_types()`
      - Do all setup on all cell types
        - Adjust phenotype
        - Add / adjust custom data
        - Set functions
    - `setup_tissue()`
      - Place initial cells in microenvironment
      - Modify each cell as needed
  - Custom functions
  - any other modeling
  - Custom coloring functions

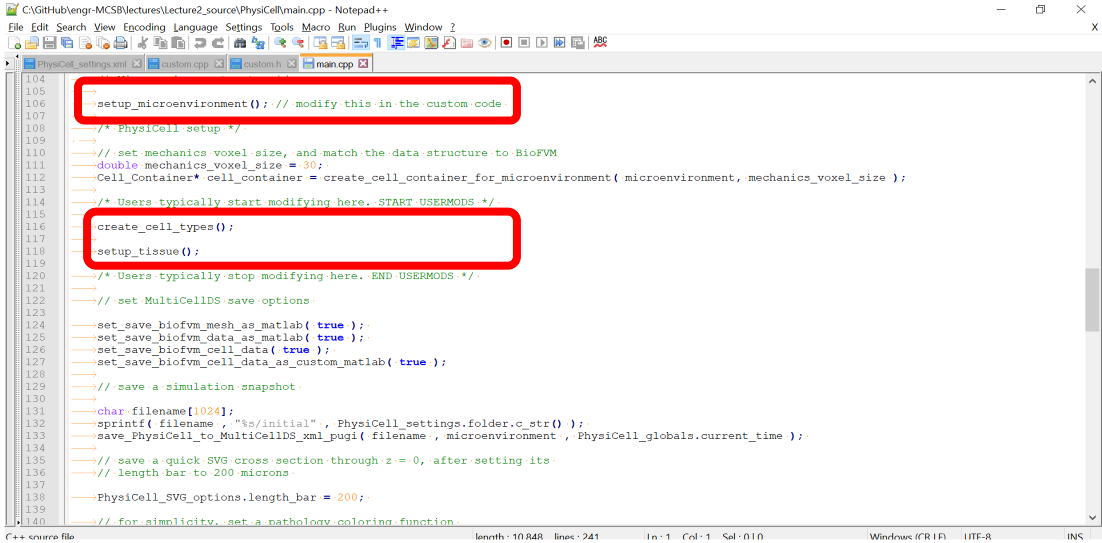

Command Prompt
C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\custom_modules\custom.h - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PhysiCell_settings.xml
custom.cpp
custom.h

```
c:\PhysiCell>cd custom_modules

c:\PhysiCell\custom_modules>ls
heterogeneity.cpp  heterogeneity.h
```

```
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF       #
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS   #
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN    #
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)    #
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE #
# POSSIBILITY OF SUCH DAMAGE.                                                #
#                                                                            #
##############################################################################
*/

#include "../core/PhysiCell.h"
#include "../modules/PhysiCell_standard_modules.h"

using namespace BioFVM;
using namespace PhysiCell;

void tumor_cell_phenotype_with_oncoprotein( Cell* pCell, Phenotype& phenotype, double dt );

// any additional cell types (beyond cell_defaults)

extern Cell_Definition motile_cell;

// custom cell phenotype functions could go here

// setup functions to help us along

void create_cell_types( void );
void setup_tissue( void );

// set up the BioFVM microenvironment
void setup_microenvironment( void );

// custom pathology coloring function

std::vector<std::string> my_coloring_function( Cell* );

```

Project structure: main.cpp

- `main.cpp`
  - (in the root directory)
  - calls the setup functions

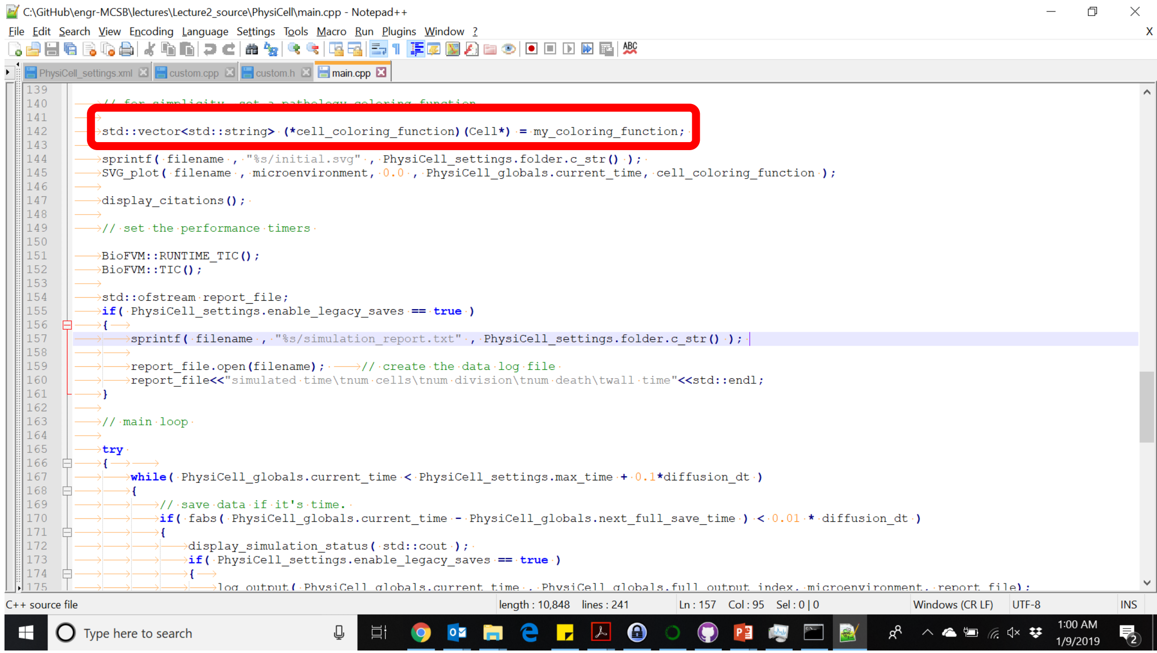

C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\main.cpp - Notepad++

```cpp

	setup_microenvironment(); // modify this in the custom code 

	/* PhysiCell setup */ 
	
	// set mechanics voxel size, and match the data structure to BioFVM
	double mechanics_voxel_size = 30; 
	Cell_Container* cell_container = create_cell_container_for_microenvironment( microenvironment, mechanics_voxel_size );
	
	/* Users typically start modifying here. START USERMODS */ 
	
	create_cell_types();
	
	setup_tissue();

	/* Users typically stop modifying here. END USERMODS */ 
	
	// set MultiCellDS save options 

	set_save_biofvm_mesh_as_matlab( true ); 
	set_save_biofvm_data_as_matlab( true ); 
	set_save_biofvm_cell_data( true ); 
	set_save_biofvm_cell_data_as_custom_matlab( true );
	
	// save a simulation snapshot 
	
	char filename[1024];
	sprintf( filename , "%s/initial" , PhysiCell_settings.folder.c_str() ); 
	save_PhysiCell_to_MultiCellDS_xml_pugi( filename , microenvironment , PhysiCell_globals.current_time ); 
	
	// save a quick SVG cross section through z = 0, after setting its 
	// length bar to 200 microns 

	PhysiCell_SVG_options.length_bar = 200; 
```

C++ source file | length : 10,848 lines : 241 | Ln : 1 Col : 1 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

Project structure: main.cpp (continued)

- **`main.cpp`**
  - set coloring function



C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\main.cpp - Notepad++

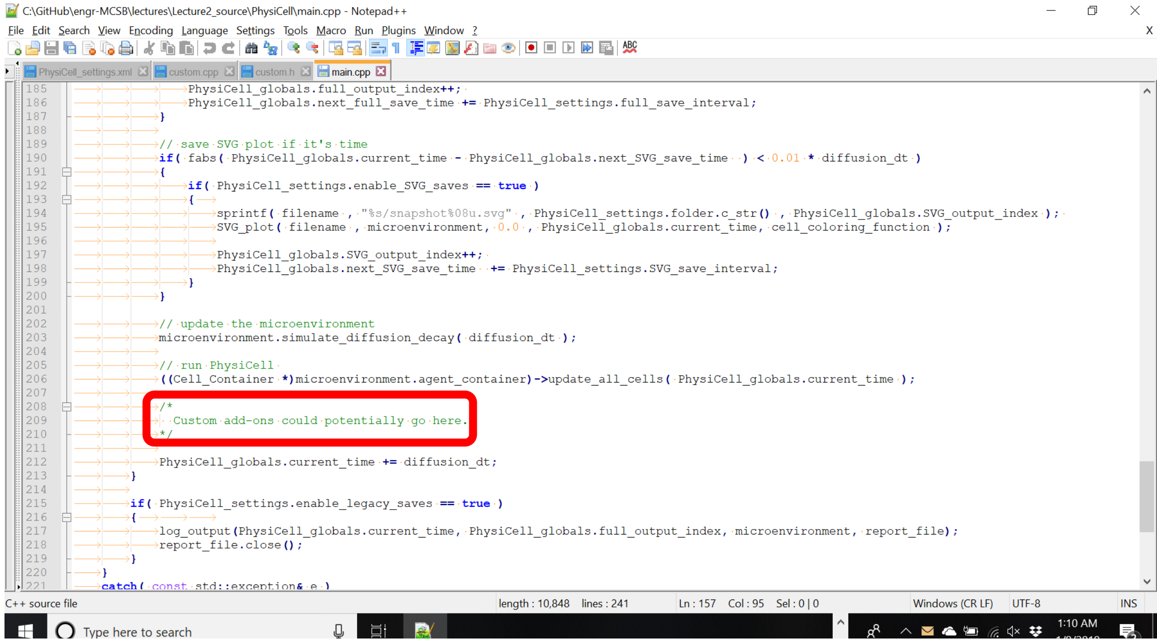
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PhysiCell_settings.xml
custom.cpp
custom.h
main.cpp

```
	// for simplicity, set a pathology coloring function 
	
	std::vector<std::string> (*cell_coloring_function)(Cell*) = my_coloring_function;
	
	sprintf( filename , "%s/initial.svg" , PhysiCell_settings.folder.c_str() ); 
	SVG_plot( filename , microenvironment, 0.0 , PhysiCell_globals.current_time, cell_coloring_function );
	
	display_citations(); 
	
	// set the performance timers 

	BioFVM::RUNTIME_TIC();
	BioFVM::TIC();
	
	std::ofstream report_file;
	if( PhysiCell_settings.enable_legacy_saves == true )
	{	
		sprintf( filename , "%s/simulation_report.txt" , PhysiCell_settings.folder.c_str() ); 
		
		report_file.open(filename); 	// create the data log file 
		report_file<<"simulated time\tnum cells\tnum division\tnum death\twall time"<<std::endl;
	}
	
	// main loop 
	
	try 
	{		
		while( PhysiCell_globals.current_time < PhysiCell_settings.max_time + 0.1*diffusion_dt )
		{
			// save data if it's time. 
			if( fabs( PhysiCell_globals.current_time - PhysiCell_globals.next_full_save_time ) < 0.01 * diffusion_dt )
			{
				display_simulation_status( std::cout ); 
				if( PhysiCell_settings.enable_legacy_saves == true )
				{	
					log_output( PhysiCell_globals.current_time , PhysiCell_globals.full_output_index, microenvironment, report_file);
```

C++ source file   length : 10,848   lines : 241   Ln : 157   Col : 95   Sel : 0 | 0   Windows (CR LF)   UTF-8   INS

Project structure: main.cpp (continued)

- **`main.cpp`**
  - insert custom routines
  - **This would be a good place to put extensions.**


C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\main.cpp - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PhysiCell_settings.xml
custom.cpp
custom.h
main.cpp

```cpp
                PhysiCell_globals.full_output_index++; 
                PhysiCell_globals.next_full_save_time += PhysiCell_settings.full_save_interval;
            }
            
            // save SVG plot if it's time
            if( fabs( PhysiCell_globals.current_time - PhysiCell_globals.next_SVG_save_time  ) < 0.01 * diffusion_dt )
            {
                if( PhysiCell_settings.enable_SVG_saves == true )
                {   
                    sprintf( filename , "%s/snapshot%08u.svg" , PhysiCell_settings.folder.c_str() , PhysiCell_globals.SVG_output_index ); 
                    SVG_plot( filename , microenvironment, 0.0 , PhysiCell_globals.current_time, cell_coloring_function );
                    
                    PhysiCell_globals.SVG_output_index++; 
                    PhysiCell_globals.next_SVG_save_time  += PhysiCell_settings.SVG_save_interval;
                }
            }

            // update the microenvironment
            microenvironment.simulate_diffusion_decay( diffusion_dt );
            
            // run PhysiCell 
            ((Cell_Container *)microenvironment.agent_container)->update_all_cells( PhysiCell_globals.current_time );
            
            /*
              Custom add-ons could potentially go here.
            */
            
            PhysiCell_globals.current_time += diffusion_dt;
        }
        
        if( PhysiCell_settings.enable_legacy_saves == true )
        {       
            log_output(PhysiCell_globals.current_time, PhysiCell_globals.full_output_index, microenvironment, report_file);
            report_file.close();
        }
    }
    catch( const std::exception& e )
```

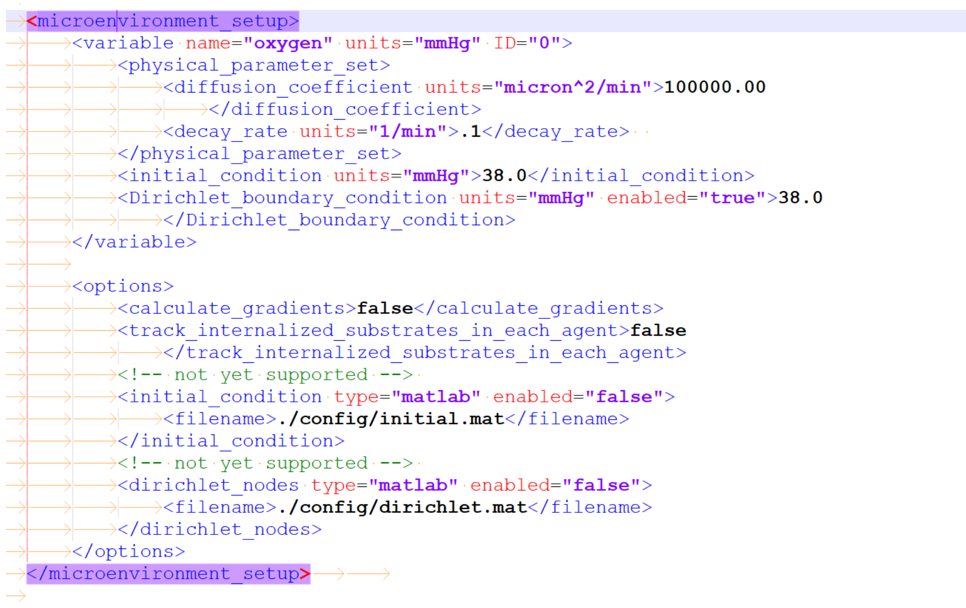
Let's modify the microenvironment

- Open ./config/PhysiCell_settings.xml
- Browse to **microenvironment_setup** block
- Modify the first **variable**
  - Change Dirichlet conditions
  - Modify diffusion and decay coefficients
  - Turn on/off gradient calculations
  - Turn on/off per-cell tracking of substrates
- Copy / paste to create more variables.

Documentation: User Guide Sec. 13.4

Tutorial:

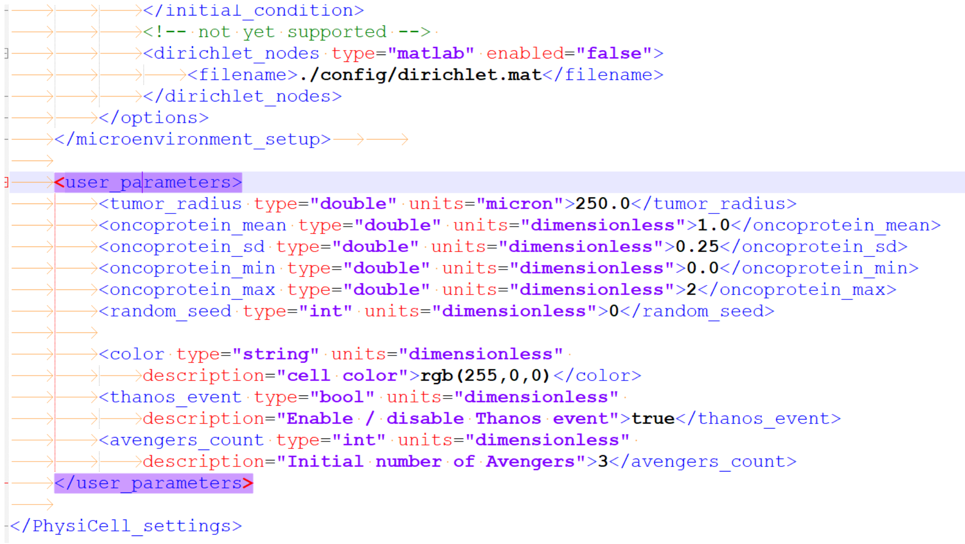
http://www.mathcancer.org/blog/setting-up-the-physicell-microenvironment-with-xml/



```
<microenvironment_setup>
	<variable name="oxygen" units="mmHg" ID="0">
		<physical_parameter_set>
			<diffusion_coefficient units="micron^2/min">100000.00
				</diffusion_coefficient>
			<decay_rate units="1/min">.1</decay_rate>
		</physical_parameter_set>
		<initial_condition units="mmHg">38.0</initial_condition>
		<Dirichlet_boundary_condition units="mmHg" enabled="true">38.0
			</Dirichlet_boundary_condition>
	</variable>

	<options>
		<calculate_gradients>false</calculate_gradients>
		<track_internalized_substrates_in_each_agent>false
			</track_internalized_substrates_in_each_agent>
		<!-- not yet supported -->
		<initial_condition type="matlab" enabled="false">
			<filename>./config/initial.mat</filename>
		</initial_condition>
		<!-- not yet supported -->
		<dirichlet_nodes type="matlab" enabled="false">
			<filename>./config/dirichlet.mat</filename>
		</dirichlet_nodes>
	</options>
</microenvironment_setup>
```

User-defined parameters

- Open ./config/PhysiCell_settings.xml
- Browse to **user_parameters** block
- Each parameter has:
  - name (based on tag name)
  - type
    - double, int, bool, string
  - units
    - no auto conversions in PhysiCell -- for your own QC
  - description (optional)
    - Helps in auto-generating GUIs (later!)



```
                </initial_condition>
                <!-- not yet supported -->
                <dirichlet_nodes type="matlab" enabled="false">
                    <filename>./config/dirichlet.mat</filename>
                </dirichlet_nodes>
            </options>
        </microenvironment_setup>

        <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">0.25</oncoprotein_sd>
            <oncoprotein_min type="double" units="dimensionless">0.0</oncoprotein_min>
            <oncoprotein_max type="double" units="dimensionless">2</oncoprotein_max>
            <random_seed type="int" units="dimensionless">0</random_seed>

            <color type="string" units="dimensionless"
                description="cell color">rgb(255,0,0)</color>
            <thanos_event type="bool" units="dimensionless"
                description="Enable / disable Thanos event">true</thanos_event>
            <avengers_count type="int" units="dimensionless"
                description="Initial number of Avengers">3</avengers_count>
        </user_parameters>

</PhysiCell_settings>
```

Documentation: User Guide Sec. 13.5

Tutorial:

http://www.mathcancer.org/blog/user-parameters-in-physicell/

Accessing user-defined parameters

- In PhysiCell projects, the parameter values can be accessed via:
  - parameters.doubles(name)
    - `double d_temp = parameters.doubles( "tumor_radius" );`
  - parameters.ints(name)
    - `int n_temp = pararameters.ints( "avengers_count" );`
  - parameters.bools(name)
    - `bool b_temp = parameters.bools( "thanos_event" );`
  - parameters.strings(name)
    - `std::string sz_temp = parameters.strings( "color" );`
- In PhysiCell projects, the parameter data structures can be accessed via:
  - parameters.doubles[name]
    - `std::cout << parameters.doubles[ "tumor_radius" ].units << std::endl;`

Documentation: User Guide Sec. 13.5.2

Tutorial:

http://www.mathcancer.org/blog/user-parameters-in-physicell/

Key cell information

- Each cell agent is part of the `Cell` class.
- Some key data:
  - `std::string type_name` // human-readable name of cell type
  - `int type` // machine-readable unique integer identifier for cell type

  - `int ID` // cell agent's unique integer identifier. (different for each cell)
  - `int index` // cell's unique position in `std::vector<Basic_Agent*> all_basic_agents`

  - `custom_data` // custom scalar and vector data (more later)
  - `parameters` // oxygen and other parameters, and pointer to a reference phenotype
  - `functions` // list of key cell functions (more later)
  - `cell_state` // things like size (more later)
  - `phenotype` // behavioral properties / state (more later)

Cell Definitions

- A **Cell Definition** is a convenient way to set the parameters and functions for a whole class of cells
  - Users can instantiate cells of a specific type using `create_cell()`
  - With no argument, new cells use the `cell_defaults` definition

- Best practice: set up the `cell_defaults` definition with the correct custom data and functions, then copy this to make new cell definitions

- Tip: Refer back to the phenotype in your agent's cell definition as a reference parameter set (i.e., to get the initial parameter values)

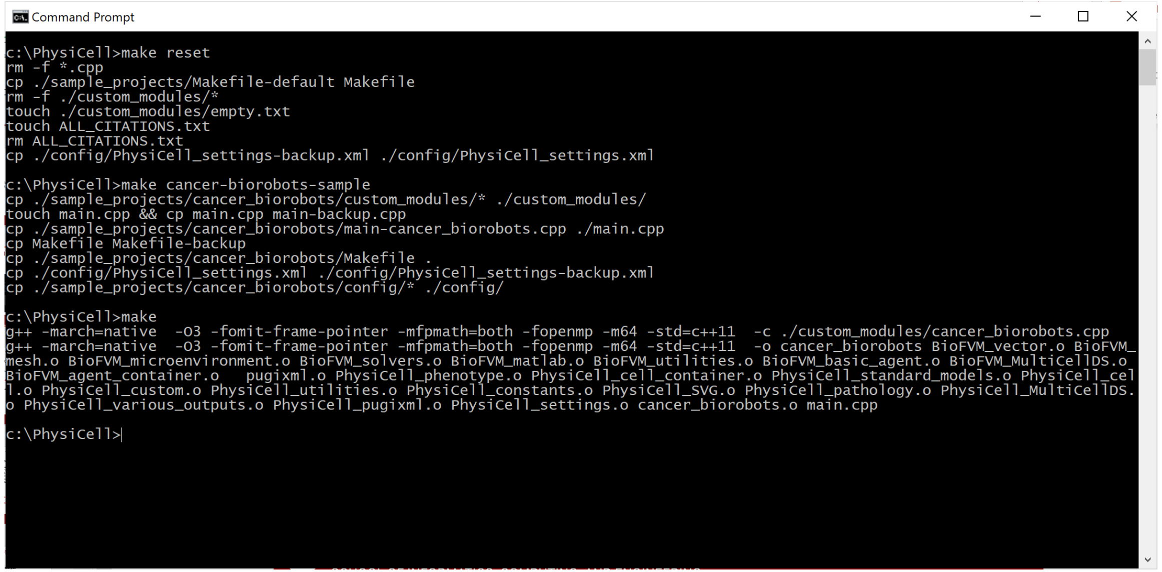
Documentation: User Guide Sec. 9.4.5

Exercise 1

- Populate cancer-biorobots project
- Modify configuration file
  - Set oxygen boundary value at 40 mmHg
  - Enable gradient calculations
  - Set max run time to 3 days (4320 minutes)
  - Add a custom user parameter "speed" at 1 micron/min
  - Add a custom user parameter "motile_fraction", initially 0.5
  - Add a custom user parameter "max_cycle_entry_rate", initially 0.25 1/hr
  - Set therapy activation time to never (a really really big number)
- Modify default cell definition
  - Set the speed according to your parameter
  - Make default definition non-motile (for now!)
  - Set the cycle entry rate to the parameter rate
- Modify the tissue setup
  - With motile_fraction probability, set some cells motile

Populate and initially build project

- `make reset`
- `make heterogeneity-sample`
- `make`


Command Prompt

```
c:\PhysiCell>make reset
rm -f *.cpp
cp ./sample_projects/Makefile-default Makefile
rm -f ./custom_modules/*
touch ./custom_modules/empty.txt
touch ALL_CITATIONS.txt
rm ALL_CITATIONS.txt
cp ./config/PhysiCell_settings-backup.xml ./config/PhysiCell_settings.xml

c:\PhysiCell>make cancer-biorobots-sample
cp ./sample_projects/cancer_biorobots/custom_modules/* ./custom_modules/
touch main.cpp && cp main.cpp main-backup.cpp
cp ./sample_projects/cancer_biorobots/main-cancer_biorobots.cpp ./main.cpp
cp Makefile Makefile-backup
cp ./sample_projects/cancer_biorobots/Makefile .
cp ./config/PhysiCell_settings.xml ./config/PhysiCell_settings-backup.xml
cp ./sample_projects/cancer_biorobots/config/* ./config/

c:\PhysiCell>make
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -c ./custom_modules/cancer_biorobots.cpp
g++ -march=native  -O3 -fomit-frame-pointer -mfpmath=both -fopenmp -m64 -std=c++11  -o cancer_biorobots BioFVM_vector.o BioFVM_
mesh.o BioFVM_microenvironment.o BioFVM_solvers.o BioFVM_matlab.o BioFVM_utilities.o BioFVM_basic_agent.o BioFVM_MultiCellDS.o
BioFVM_agent_container.o   pugixml.o PhysiCell_phenotype.o PhysiCell_cell_container.o PhysiCell_standard_models.o PhysiCell_cel
l.o PhysiCell_custom.o PhysiCell_utilities.o PhysiCell_constants.o PhysiCell_SVG.o PhysiCell_pathology.o PhysiCell_MultiCellDS.
o PhysiCell_various_outputs.o PhysiCell_pugixml.o PhysiCell_settings.o cancer_biorobots.o main.cpp

c:\PhysiCell>
```

Modify configuration file (1)

- **Modify the microenvironment setup** (./config/PhysiCell_settings.xml)

```
<microenvironment_setup>
        <variable name="oxygen" units="mmHg" ID="0">
                <physical_parameter_set>
                        <diffusion_coefficient units="micron^2/min">100000.00</diffusion_coefficient>
                        <decay_rate units="1/min">.1</decay_rate>
                </physical_parameter_set>
                <initial_condition units="mmHg">38.0</initial_condition>
                <Dirichlet_boundary_condition units="mmHg"
                        enabled="true">40.0</Dirichlet_boundary_condition>
        </variable>
        // ...
        <options>
                <calculate_gradients>true</calculate_gradients>
                <track_internalized_substrates_in_each_agent>false
                        </track_internalized_substrates_in_each_agent>
                // ...
        </options>
</microenvironment_setup>
```

Modify configuration file (2)

- **Modify time options**

```
<overall>
      <max_time units="min">4320</max_time>
      <time_units>min</time_units>
      <space_units>micron</space_units>

      <dt_diffusion units="min">0.01</dt_diffusion>
      <dt_mechanics units="min">0.1</dt_mechanics>
      <dt_phenotype units="min">6</dt_phenotype>
</overall>
```

Modify configuration file (3)

- **Modify user-defined parameters**
  - Disable therapy
  - Add custom parameters
    - Careful on units!

```
<user_parameters>
        <random_seed type="int" units="dimensionless">0</random_seed>
        <!-- for main -->
        <therapy_activation_time type="double" units="min">9e99</therapy_activation_time>
                <!-- activate in 7 days -->
        <save_interval_after_therapy_start type="double" units="min">3</save_interval_after_therapy_start>

        <!-- for tumor cells -->
        <speed type="double" units="micron/min"
                description="tumor cell speed">1.0</speed>
        <motile_fraction type="double" units="dimensionless"
                description="initial fraction of motile cells">0.5</motile_fraction>
        <max_cycle_entry_rate type="double" units="1/min"
                description="max cycle entry rate">0.00417</max_cycle_entry_rate>
        // ...
```

Modify the default cell definition

- Open ./custom_modules/cancer-biorobots.cpp
- Open documentation: ./documentation/User_Guide.pdf
- Find **`create_cell_types()`**
- Edit the default cell type
  - Edit the motile speed, but le
  - We use the live cell cycle: only one phase (0th phase)
    - Edit the 0 → 0 transition rate

```
// ...
// further edit the default cell type here
cell_defaults.phenotype.motility.is_motile = false;
       // set motile speed
cell_defaults.phenotype.motility.migration_speed =
       parameters.doubles("speed");
       // set cycle entry rate (0 --> 0)
cell_defaults.phenotype.cycle.data.transition_rate(0,0) =
       parameters.doubles( "max_cycle_entry_rate" );
return;
```

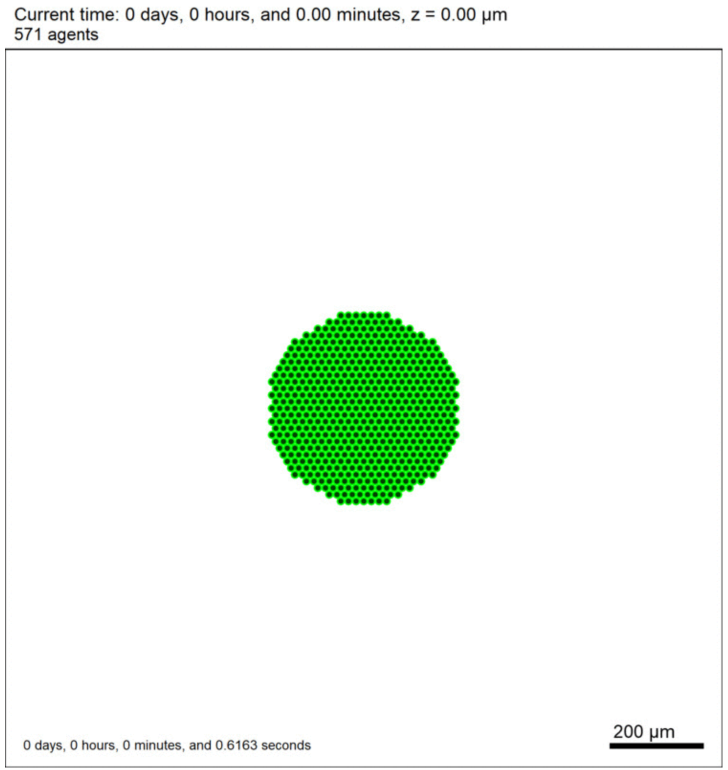
Modify the tissue setup

- Find `setup_tissue()`
- Let's just cycle through the list of all cells
  - Check if it'sa tumor cell by comparing IDs
  - If so, then with the probability in your config file, set motility on

```
        // iterate through all cells
        // if it's a tumor cell, try to make it motile
        for( int n = 0; n < (*all_cells).size(); n++ )
        {
                pCell = (*all_cells)[n] ;
                if( pCell->type == cell_defaults.type )
                {
                        if( UniformRandom() <= parameters.doubles( "motile_fraction" ) )
                        { pCell->phenotype.motility.is_motile = true; }
                }
        }
        return;
}
```

Build and run the project

- `make`
- `cancer-biorobots`
- Let it run about 10 minutes
- Then, visualize
  - `cd output`
  - `magick mogrify -format jpg -resize 50% snap*.svg`
  - `magick convert *.jpg out.gif`


Current time: 0 days, 0 hours, and 0.00 minutes, z = 0.00 μm
571 agents
0 days, 0 hours, 0 minutes, and 0.6163 seconds
200 μm

Break

Part 3
Making new projects in PhysiCell

Agenda (second half):

Part 3: Making new projects in PhysiCell

- Introduce general modeling workflow
- Work with cell definitions
- Introduce functions in PhysiCell
- Attach functions to Cell Definitions
- Add custom data to cells
- Setup routines
- Place new cells
- Coloring functions (for SVG)
- **Exercise 2:** Modify a 3D project with basic metabolism and energy-dependent cycling and death

Part 4: Special topics

- Introduce tools ecosystem (and povwriter)
- Create mpeg4 movies

Typical modeling workflow

- Plan!
- Create a new project (start with 2D or 3D template)
- Set up the XML (./config/PhysiCell_settings.xml).
  - Set up the chemical microenvironment
  - Set up user-defined parameters
- Define cells (create_cell_types())
  - Add custom data to the default cell definition
  - Create any needed custom function (for model rules)
  - Create and configure cell definitions
- Place cells (setup_tissue())
  - Place different cell types in their initial positions
- Compile and run

Reminder: phenotype-centric programming

- The core cell behaviors are implemented:
  - Cell cycling (with user-selectable models)
  - Cell death
  - Cell adhesion / repulsion
  - Cell motility
  - Cell secretion / uptake

- Modelers can focus on writing functions that control these behaviors.
- This is **phenotype-centric programming.**

Reminder: cell definitions

- A **Cell Definition** is a convenient way to set the parameters and functions for a whole class of cells
  - Users can instantiate cells of a specific type using create_cell()
  - With no argument, new cells use the **cell_defaults** definition

- Best practice: set up the **cell_defaults** definition with the correct custom data and functions, then copy this to make new cell definitions

- Tip: Refer back to the phenotype in your agent's cell definition as a reference parameter set (i.e., to get the initial parameter values)

Documentation: User Guide Sec. 9.4.5

Custom cell data

- Each cell can have scalar and vector **custom data**.
    - We'll eventually merge these, so I suggest focusing on scalars.
- Use these for cell-specific models, or to track new things

Add a custom variable:

```
// Custom_Data::add_variable( name, units , default_value );
cell_defaults.custom_data.add_variable( "oxygen_AUC", "mmHg * min", 0.0 );
```

Access a custom variable:

```
cell_defaults.custom_data["oxygen_AUC"] += dt*O2;
pCell->custom_data["oxygen_AUC"] += dt*O2;
```

- Best practice: set up the **`cell_defaults`** definition with the correct custom data. Then copy this to make new cell definitions

Documentation: User Guide Sec. 9.4.1

Functions in PhysiCell

- Almost all functions in PhysiCell have this form:

```
void my_function( Cell* pCell, Phenotype& phenotype, double dt );
```

All cells have the following key functions (`in pCell->`**`functions`**):

- `volume_update_function`
- `update_migration_bias`
- `custom_cell_rule` (default NULL, evaluated at each mechanics time step)
- `update_velocity`
- `set_orientation`

Documentation: User Guide Sections 9.4.3, 19.1

Example function: chemotaxis

```
void chemotaxis( Cell* pCell, Phenotype* phenotype, double dt )
{
   static int o2_index = microenvironment.find_density_index( "oxygen" );
   // enable it
   phenotype.motility.is_motile = true;

   // set bias
   phenotype.motility.migration_bias = 1.0;

   // set direction
   phenotype.motility.migration_bias_direction =
      pCell->nearest_gradient(o2_index);

   // normalize
   normalize( &( phenotype.motility.migration_bias_direction ) );

   // set persistence time
   phenotype.motility.persistence_time = 1.0;


   return;
}
```

How to attach a function to a cell definition

- Suppose we want to attach chemotaxis to the default celltype

```
cell_defaults.functions.update_migration_bias = chemotaxis;
```

How to place a new cell (default)

```
// declare a cell pointer
Cell* pCell;

// create a cell
pCell = create_cell();

// assign its position
std::vector<double> position = {0,0,0};
pCell->assign_position( position );

// set any other state variables or properties
pCell->phenotype.motility.is_motile = false;
pCell->custom_data[ "damage" ] = 0.0;
```

How to place a new cell (custom type)

```
// declare a cell pointer
Cell* pCell;

// create a cell
pCell = create_cell( Thanos );

// assign its position
std::vector<double> position = {0,0,0};
pCell->assign_position( position );

// set any other state variables or properties
pCell->phenotype.motility.is_motile = false;
pCell->custom_data[ "damage" ] = 0.0;
```

Custom coloring functions for SVGs (1)

Declare the function in the custom header file

```
std::vector<std::string> my_coloring_function( Cell* );
```

Create it in the custom cpp file

```
std::vector<std::string> my_coloring_function( Cell* pCell )
{
   // color 0: cytoplasm fill
   // color 1: outer outline
   // color 2: nuclear fill
   // color 3: nuclear outline

   // start black
   std::vector< std::string > output( 4, "black" );
   // make the cytoplasm red if it's not dead
   if( pCell->phenotype.death.dead == false )
   { output[0] = "red"; }
   return output;
}
```

Tell PhysiCell to use your coloring function

In main.cpp

```
std::vector<std::string> (*cell_coloring_function)(Cell*) =
   my_coloring_function;
```

Colors follow the W3C standards for SVG files. Names, RGB values, etc.

https://www.w3.org/TR/SVG11/types.html#ColorKeywords

User Guide: Section 14.2

Exercise 2: new 3D project

- Let's make a 3D simulation
- Tumor cells uptake oxygen and glucose to make energy
- Cells make waste
- Cell proliferate more with more energy
- Cells necrose with very low energy
- Cells apoptosis with increasing waste
- Later: Some cells are motile in low O2 (chemotactic), but that consumes energy

Modeling workflow

- Plan!
- Create a new project (start with 2D or 3D template)
- Set up the XML (./config/PhysiCell_settings.xml).
  - Set up the chemical microenvironment
  - Set up user-defined parameters
- Define cells (create_cell_types())
  - Add custom data to the default cell definition
  - Create any needed custom function (for model rules)
  - Create and configure cell definitions
- Place cells (setup_tissue())
  - Place different cell types in their initial positions
- Compile and run

Planning

- What microenvironment variables?
  - Oxygen, glucose, waste
- What cell types?
  - Just a tumor cell type for now
- What custom data?
  - Store energy
  - Parameters for metabolism model
  - Parameters for birth, apoptosis, necrosis
  - Parameters for motility
- What functions?
  - metabolism model
  - set birth, death, and motility based on energy and/or O2

Planning: metabolism

- Cells can use two types of "metabolism"

$$\frac{dE}{dt} = \alpha\sigma g + \beta g - uE$$

- Cells consume supplies and make waste

$$U_\sigma = 10\alpha$$
$$U_g = 0.2(\alpha + \beta)$$
$$S_w = 10r$$

- Cell behaviors consume more energy:

$$u = \mu(0.1 + \alpha + \beta + 2r + 5s)$$

  - s = motile speed

- Let's give each cell independent "gene expressions" for α, β, r

- Suppose cell cycling increases with energy:

$$b = \begin{cases} 0 & E < 0.1 \\ b_M\left(\frac{E-0.1}{0.9-0.1}\right) & 0.1 < E < 0.9 \\ b_M & 0.9 < E \end{cases}$$

- Necrotic death increases below E = 0.1

$$d_N = d_{\text{nec}}\left(\frac{0.1-E}{0.1}\right)^+$$

- Apoptotic death increases with waste:
  - But *r* gives tolerance (resistance) to the waste

$$d_A = d_{\text{apop}}\big(1 + 9w(1-r)\big)$$

What we'll need

- Modify the microenvironment
  - oxygen, glucose, waste
- Change the custom variables in cell defaults
  - alpha, beta, resistance, energy, use_rate
  - Let's leave all the birth and death rates relatively untouched.
- A new phenotype function
  - void energy_based_cell_phenotype(Cell* , Phenotype& , dt)
- Modify the cell_defaults with:
  - Custom data and the new phenotype function.
- Modify the setup_tissue() to choose random values of alpha, beta, resistance for each cell.
- New coloring function (to tailor the SVGs)
  - energy_coloring_function(Cell*)

Create a new project

```
make data-cleanup

make reset

make template3D
```

Set up the XML

Let's modify the microenvironment

- Open ./config/PhysiCell_settings.xml
- Browse to **microenvironment_setup** block
- Modify the first **variable**
- Now, copy it to make **glucose** and **waste**

Documentation: User Guide Sec. 13.4

Tutorial:

http://www.mathcancer.org/blog/setting-up-the-physicell-microenvironment-with-xml/

Modifying the microenvironment (1)

Modify oxygen to be dimensionless, set initial and boundary values

```
<variable name="oxygen" units="dimensionless" ID="0">
  <physical_parameter_set>
    <diffusion_coefficient units="micron^2/min">100000.00</diffusion_coefficient>
    <decay_rate units="1/min">.1</decay_rate>
  </physical_parameter_set>
  <initial_condition units="mmHg">1</initial_condition>
  <Dirichlet_boundary_condition units="mmHg"
    enabled="true">1</Dirichlet_boundary_condition>
</variable>
```

Modifying the microenvironment (2)

Copy oxygen to make glucose and waste.

```
<variable name="glucose" units="dimensionless" ID="1">
   <physical_parameter_set>
      <diffusion_coefficient units="micron^2/min">18000</diffusion_coefficient>
      <decay_rate units="1/min">0</decay_rate>
   </physical_parameter_set>
   <initial_condition units="mmHg">1</initial_condition>
   <Dirichlet_boundary_condition units="mmHg" enabled="true">1</Dirichlet_boundary_condition>
</variable>

<variable name="waste" units="dimensionless" ID="2">
   <physical_parameter_set>
      <diffusion_coefficient units="micron^2/min">100000.00</diffusion_coefficient>
      <decay_rate units="1/min">0</decay_rate>
   </physical_parameter_set>
   <initial_condition units="mmHg">0</initial_condition>
   <Dirichlet_boundary_condition units="mmHg" enabled="true">0</Dirichlet_boundary_condition>
</variable>
```

Modifying the microenvironment (3)

```
<options>
    <calculate_gradients>false</calculate_gradients>
    <track_internalized_substrates_in_each_agent>false</track_internalized_substrates_in_each_agent>
    <!-- not yet supported -->
    <initial_condition type="matlab" enabled="false">
        <filename>./config/initial.mat</filename>
    </initial_condition>
    <!-- not yet supported -->
    <dirichlet_nodes type="matlab" enabled="false">
        <filename>./config/dirichlet.mat</filename>
    </dirichlet_nodes>
</options>
```

Define cells

Let's add new custom variables

in create_cell_types() in custom.cpp

```
// add custom data here, if any

cell_defaults.custom_data.add_variable( "alpha" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "beta" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "resistance" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "use_rate" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "energy" , "dimensionless", 1.0 );
```

Let's set up the default type a bit

```
in custom.cpp, in create_cell_types :

   // set default cell cycle model to live

   cell_defaults.functions.cycle_model = live; // flow_cytometry_separated_cycle_model;

   // Delete out all the junk for motile cells in custom.cpp

In custom.cpp, in setup_tissue :

   // delete anything that seedsa motile cell

In custom.cpp and custom.h :

   // delete declarations of motile cell type
```

Let's make a new phenotype function (1)

First, declare the new function in custom.h

```
// custom cell phenotype functions could go here

void energy_based_cell_phenotype( Cell* pCell, Phenotype& phenotype, double dt );
```

Next, let's implement. In the function in custom.cpp

```
void energy_based_cell_phenotype( Cell* pCell, Phenotype& phenotype, double dt )
{
    // housekeeping: one-time searches for variables
        // for finding the right cycle phases
    static int cycle_start_index = live.find_phase_index( PhysiCell_constants::live );
    static int cycle_end_index = live.find_phase_index( PhysiCell_constants::live );

        // for finding the death model indices
    static int apoptosis_i =
        cell_defaults.phenotype.death.find_death_model_index( PhysiCell_constants::apoptosis_death_model );
    static int necrosis_i =
        cell_defaults.phenotype.death.find_death_model_index( PhysiCell_constants::necrosis_death_model );

        // for accessing the custom variables
    static int energy_i = pCell->custom_data.find_variable_index( "energy" );
    static int alpha_i = pCell->custom_data.find_variable_index( "alpha" );
    static int beta_i = pCell->custom_data.find_variable_index( "beta" );
    static int resistance_i = pCell->custom_data.find_variable_index( "resistance" );
    static int use_rate_i = pCell->custom_data.find_variable_index( "use_rate" );

        // for sampling the microenvironment
    static int oxygen_i = microenvironment.find_density_index( "oxygen" );
    static int glucose_i = microenvironment.find_density_index( "glucose" );
    static int waste_i = microenvironment.find_density_index( "waste" );
```

Let's make a new phenotype function (2)

Next, a fun trick: check to see if the cell's dead. If so, set uptake / secretions rates to zero, and overwrite the function pointer so that we don't keep asking dead cells to do things. :-)

```
    // if I'm dead, set secretoin rates to zero, and tell us not to
    // bother checking ever again.

  if( phenotype.death.dead == true )
  {
    phenotype.secretion.set_all_secretion_to_zero();
    phenotype.secretion.set_all_uptake_to_zero();

    pCell->functions.update_phenotype = NULL;
    return;
  }
```

Let's make a new phenotype function (3)

Let's do the energy model

```
     // do the basic energy model
     // use rate : u = alpha + beta + resistance
   double alpha = pCell->custom_data[alpha_i];
   double beta = pCell->custom_data[beta_i];
   double resistance = pCell->custom_data[resistance_i];

   pCell->custom_data[use_rate_i] = 0.1 + alpha + 2*beta + 2*resistance;
   if( phenotype.motility.is_motile )
   { pCell->custom_data[use_rate_i] += (5.0* phenotype.motility.migration_speed ); }

     // sample the oxygen, glucose, and waste
   double oxygen = pCell->nearest_density_vector()[oxygen_i];
   double glucose = pCell->nearest_density_vector()[glucose_i];
   double waste = pCell->nearest_density_vector()[waste_i];
     // run the ODE. Let's use backwards Euler
   double use_rate = pCell->custom_data[use_rate_i];

   pCell->custom_data[energy_i] +=
     dt*( alpha*oxygen*glucose + beta*glucose );
   pCell->custom_data[energy_i] /= ( 1.0 + dt*use_rate );
```

Let's make a new phenotype function (4)

Let's uptake secretion/ uptake

```
    // set secretion parameters
  phenotype.secretion.secretion_rates[waste_i] = beta * 10.0;
  phenotype.secretion.saturation_densities[waste_i] = 1.0;
    // set uptake rates
  phenotype.secretion.uptake_rates[oxygen_i] = alpha * 10.0;
  phenotype.secretion.uptake_rates[glucose_i] = 0.2*(alpha+beta);
```

Next, the cell cycle rate

```
    // set cycle parameters
  double energy = pCell->custom_data[energy_i];
  double scale = ( energy - 0.1 )/( 0.9 - 0.1 );
  if( scale > 1.0 )
  { scale = 1.0; }
  if( scale  < 0.0 )
  { scale = 0.0; }
  phenotype.cycle.data.transition_rate( cycle_start_index ,cycle_end_index ) =
    6.94e-4 * scale;
  // 1/24 hr^-1 max birth rate, in units of min^-1
```

Let's make a new phenotype function (5)

Lastly, let's set the death rates

```
     // set necrotic death rate
   scale = ( 0.1 - energy )/0.1;
   if( scale < 0.0 )
   { scale = 0.0; }
   if( scale > 1.0 )
   { scale = 1.0; }
   phenotype.death.rates[necrosis_i] = scale * 0.01 ;
     // 100 minute survival time when zero energy;

     // set the apoptotic death rate
   scale = 1.0 + 9.0*(1.0-pCell->custom_data[resistance_i])*waste;
   phenotype.death.rates[apoptosis_i] = scale * 6.94e-6;
   // 1% of the max birth rate

   return;
}
```

Now, in create_cell_types, let's switch to this rule:

```
   cell_defaults.functions.update_phenotype = energy_based_cell_phenotype;
```

Place cells

Modify setup_tissue()

Some bookkeeping

```
    // some bookkeeping
    static int energy_i = cell_defaults.custom_data.find_variable_index( "energy" );
    static int alpha_i = cell_defaults.custom_data.find_variable_index( "alpha" );
    static int beta_i = cell_defaults.custom_data.find_variable_index( "beta" );
    static int resistance_i = cell_defaults.custom_data.find_variable_index("resistance");
    static int use_rate_i = cell_defaults.custom_data.find_variable_index( "use_rate" );
```

and now let's put 750 cells at random within 100 microns of the origin

```
    for( int n=0 ; n < 750 ; n++ )
    {
            Cell* pCell = create_cell();
            std::vector<double> position = {0,0,0};
            position[0] = UniformRandom();
            position[1] = UniformRandom();
            position[2] = UniformRandom();
            normalize( position );
            position *= ( 100.0 * UniformRandom() );
            pCell->assign_position( position );

            // choose random properties
            pCell->custom_data[alpha_i] = UniformRandom();
            pCell->custom_data[beta_i] = UniformRandom();
            pCell->custom_data[resistance_i] = UniformRandom();
   }
```

Make a coloring function

A custom coloring function for SVGs (1)

Declare the function in custom.h (replace my_coloring_function)

```
std::vector<std::string> energy_coloring_function( Cell* );
```

Create it in custom.cpp

```
std::vector<std::string> energy_coloring_function( Cell* pCell )
{
   // color 0: cytoplasm fill
   // color 1: outer outline
   // color 2: nuclear fill
   // color 3: nuclear outline

   // some bookkeeping
   static int energy_i = pCell->custom_data.find_variable_index( "energy" );
   static int alpha_i = pCell->custom_data.find_variable_index( "alpha" );
   static int beta_i = pCell->custom_data.find_variable_index( "beta" );
   static int resistance_i = pCell->custom_data.find_variable_index( "resistance" );
   static int use_rate_i = pCell->custom_data.find_variable_index( "use_rate" );

   // start black
   std::vector< std::string > output( 4, "black" );
```

A custom coloring function for SVGs (2)

Sample cell properties and set colors (if not dead)

```
// nucleus: color by the three "genes"
// red: alpha
// green: beta
// blue: resistance
// cytoplasm: energy (black = 0, white >= 1)
if( pCell->phenotype.death.dead == false )
{
   int red = (int) round( 255.0 * pCell->custom_data[alpha_i] );
   int green = (int) round( 255.0 * pCell->custom_data[beta_i] );
   int blue = (int) round( 255.0 * pCell->custom_data[resistance_i] );
   int grey = (int) round( 255.0 * pCell->custom_data[energy_i] );
   char szTempString [128];
   sprintf( szTempString , "rgb(%u,%u,%u)", red, green, blue );
   output[2].assign( szTempString ); // nucleus by alpha, beta, resistance "genes"
   sprintf( szTempString , "rgb(%u,%u,%u)", grey, grey, grey );
   output[0].assign( szTempString ); // cyto by energy

   return output;
}
```

A custom coloring function for SVGs (3)

Standard dead colors

```
   // if not, dead colors

   if (pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::apoptotic )
      // Apoptotic - Red
   {
      output[0] = "rgb(255,0,0)";
      output[2] = "rgb(125,0,0)";
   }

   // Necrotic - Brown
   if( pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic_swelling ||
      pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic_lysed ||
      pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic )
   {
      output[0] = "rgb(250,138,38)";
      output[2] = "rgb(139,69,19)";
   }

   return output;
}
```

A custom coloring function for SVGs (4)

Select this coloring function in main.cpp

```
   std::vector<std::string> (*cell_coloring_function)(Cell*) = energy_coloring_function;
```

Now, go back to config. Set max time to 7200, save interval to 60 min, and domain size to [-300,300] x [-300,300] x [-300,300]

Now, compile and run! (Can take 20-60 minutes, depending on your machine.)

```
make && ./project3D
```

Part 4
Special topics

Agenda (second half):

Part 3: Making new projects in PhysiCell

- Introduce general modeling workflow
- Work with cell definitions
- Introduce functions in PhysiCell
- Attach functions to Cell Definitions
- Add custom data to cells
- Setup routines
- Place new cells
- Coloring functions (for SVG)
- **Exercise 2:** Modify a 3D project with basic metabolism and energy-dependent cycling and death

Part 4: Special topics

- Introduce tools ecosystem (and povwriter)
- Create mpeg4 movies

PhysiCell-Tools ecosystem

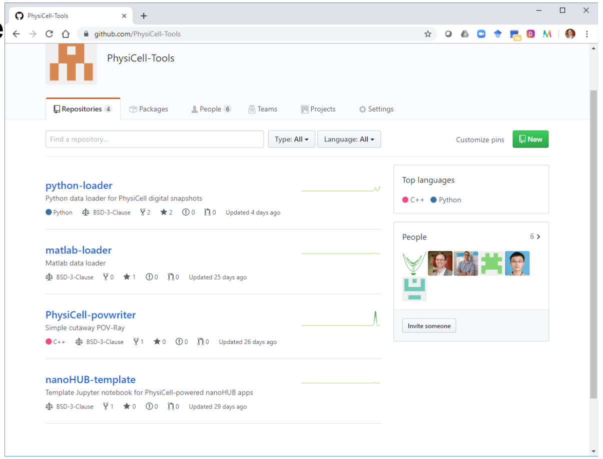
- A collection of stand-alone utilities to smooth use of PhysiCell
- Typically, a blog post or tutorial for each tool
- PhysiCell-Tools GitHub organization stores officially supported tools
  - https://www.github.com/PhysiCell-Tools

Current tools:

- **PhysiCell-povwriter:** Write PhysiCell output to POV-Ray scenes for raytracing
- **Python-loader:** Load PhysiCell simulation output data into a nice Python data structure
- **Matlab-loader:** A basic Matlab data loader
- **nanoHUB-template:** Template Jupyter notebook for nanoHUB apps

Future:

- Migrate scattered internal tools into PhysiCell-Tools
- Develop protocols to officially "adopt" community-contributed tools
- Develop documentation and other requirements for tools
- PhysiCell.org or other registry links external, non-official tools



PhysiCell-Tools

python-loader
Python data loader for PhysiCell digital snapshots
● Python · BSD-3-Clause · 2 · ★ 2 · 0 · 0 · Updated 4 days ago

matlab-loader
Matlab data loader
BSD-3-Clause · 0 · ★ 1 · 0 · 0 · Updated 25 days ago

PhysiCell-povwriter
Simple cutaway POV-Ray
● C++ · BSD-3-Clause · 1 · ★ 0 · 0 · 0 · Updated 26 days ago

nanoHUB-template
Template Jupyter notebook for PhysiCell-powered nanoHUB apps
BSD-3-Clause · 1 · ★ 0 · 0 · 0 · Updated 29 days ago

Top languages
● C++ ● Python

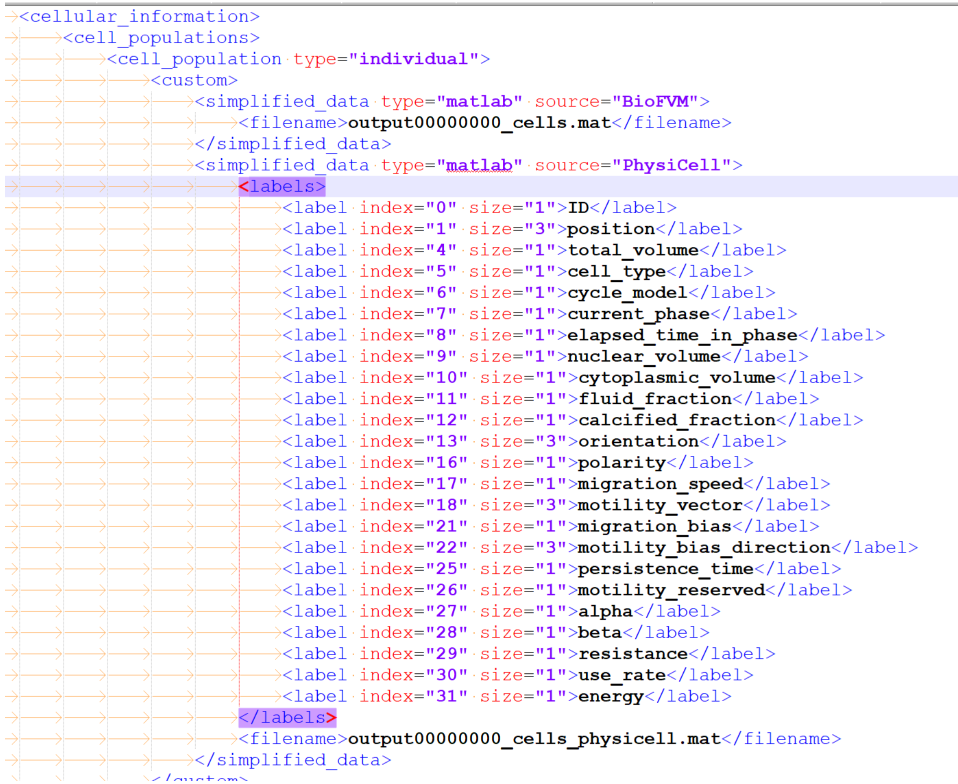
People 6

PhysiCell Povwriter

- POV-Ray is an open source raytracer
- It simulates human vision in a 3D scene by tracing the path of light
- PhysiCell-povwriter converts PhysiCell simulation data to povray scenes
- It has some rudimentary support for:
  - clipping planes (cutaway views)
  - XML-based colors
  - User-defined coloring functions
  - Processing multiple scnee files in parallel
- **Blog tutorial:** http://www.mathcancer.org/blog/povwriter/

Let's try it out on our 3D data

- Download the package
- Unzip somewhere
- open a simulation XML file to see what data fields are available
- look for labels:
  - the **index** is the position of the data field in the associated matlab file
  - Here, we might want to plot energy, for example, or query for the cycle model



```
<cellular_information>
    <cell_populations>
        <cell_population type="individual">
            <custom>
                <simplified_data type="matlab" source="BioFVM">
                    <filename>output00000000_cells.mat</filename>
                </simplified_data>
                <simplified_data type="matlab" source="PhysiCell">
                    <labels>
                        <label index="0" size="1">ID</label>
                        <label index="1" size="3">position</label>
                        <label index="4" size="1">total_volume</label>
                        <label index="5" size="1">cell_type</label>
                        <label index="6" size="1">cycle_model</label>
                        <label index="7" size="1">current_phase</label>
                        <label index="8" size="1">elapsed_time_in_phase</label>
                        <label index="9" size="1">nuclear_volume</label>
                        <label index="10" size="1">cytoplasmic_volume</label>
                        <label index="11" size="1">fluid_fraction</label>
                        <label index="12" size="1">calcified_fraction</label>
                        <label index="13" size="3">orientation</label>
                        <label index="16" size="1">polarity</label>
                        <label index="17" size="1">migration_speed</label>
                        <label index="18" size="3">motility_vector</label>
                        <label index="21" size="1">migration_bias</label>
                        <label index="22" size="3">motility_bias_direction</label>
                        <label index="25" size="1">persistence_time</label>
                        <label index="26" size="1">motility_reserved</label>
                        <label index="27" size="1">alpha</label>
                        <label index="28" size="1">beta</label>
                        <label index="29" size="1">resistance</label>
                        <label index="30" size="1">use_rate</label>
                        <label index="31" size="1">energy</label>
                    </labels>
                    <filename>output00000000_cells_physicell.mat</filename>
                </simplified_data>
```

Pigment and finish function

- In ./custom_modules/povwriter.cpp, we'll modify the custom pigment function:

```
void my_pigment_and_finish_function( Cell_Colorset& colors,
     std::vector<std::vector<double>>& MAT, int i )
```

- Here:
  - **`colors`** is a data structure with two pigments (colors):
    - **`cyto_pigment`** is an RGB vector to color the cytoplasm. (each component is in [0,1])
    - **`nuclear_pigment`** is an RGB vector to color the nucleus. (each component is in [0,1])
  - **`MAT`** is an *fields* x *n_cells* matrix of cell data. Each row is a field. Each column is a cell. Access i^{th} cell's variable in index *k* via **`MAT[k][i]`**
  - **`i`** is the index of the cell you're trying to color

Important tricks (1)

- We can usethis to figure out if a cell is dead or alive

```
// first, some housekeeping
static int type_index = 5; //column that stores cell type (integer)
static int cycle_model_index = 6; // column that stores which cycle (or death) model (integer)
static int data_index = 31; // alpha is stored here


// some simple processing to see if the cell
// is live, apoptotic, or necrotic
bool necrotic = false;
bool apoptotic = false;
bool live = true;
int cycle_model = (int) round( MAT[cycle_model_index][i] );
if( cycle_model == 100 )
{ apoptotic = true; live = false;}
if( cycle_model == 101 )
{ necrotic = true; live = false; }
```

Important tricks (2)

- We can use this to shade from low (blue) to high (yellow) in the **data_index** cell variable.

```
// first, some housekeeping
static int type_index = 5; //column that stores cell type (integer)
static int cycle_model_index = 6; // column that stores which cycle (or death) model (integer)
static int data_index = 31; // alpha is stored here

double my_data = MAT[data_index][i];
if( my_data > 1.0 )
{ my_data = 1.0; }
if( my_data < 0 )
{ my_data =0.0; }

if( live )
{
        colors.cyto_pigment[0] = my_data;
        colors.cyto_pigment[1] = my_data;
        colors.cyto_pigment[2] = 1.0 - my_data;
}
```

Let's make a coloring

- Cytoplasm colored by energy level
- Nucleus colored by (alpha, beta, resistance)
- Dead cells colored based on XML settings

The code (1)

```
void my_pigment_and_finish_function( Cell_Colorset& colors, std::vector<std::vector<double>>& MAT, int i )
{
   // first, some housekeeping
   static int type_index = 5; // stores cell type
   static int cycle_model_index = 6; // that stores which cycle model
   static int data_index = 31; // energy index

   bool necrotic = false;
   bool apoptotic = false;
   bool live = true;
   int cycle_model = (int) round( MAT[cycle_model_index][i] );
   if( cycle_model == 100 )
   { apoptotic = true; live = false; }
   if( cycle_model == 101 )
   { necrotic = true; live = false; }

   // use XML-defined colors for dead cells
   int color_index = 0;
   int cell_type = (int) MAT[type_index][i];
   for( int j=0; j < cell_color_definitions.size(); j++ )
   {
      if( cell_type == cell_color_definitions[j].type )
      { color_index = j; }
   }
```

The code (2)

```
    if( apoptotic == true )
    {
        colors = cell_color_definitions[color_index].apoptotic;
        return;
    }
    if( necrotic == true )
    {
        colors = cell_color_definitions[color_index].necrotic;
        return;
    }
    double my_data = MAT[data_index][i];
    if( my_data > 1.0 )
    { my_data = 1.0; }
    if( my_data < 0 )
    { my_data =0.0; }

    if( live )
    {
        colors.cyto_pigment[0] = my_data;
        colors.cyto_pigment[1] = my_data;
        colors.cyto_pigment[2] = 1.0 - my_data;
        colors.nuclear_pigment[0] = MAT[27][i]; // alpha
        colors.nuclear_pigment[1] = MAT[28][i]; // beta
        colors.nuclear_pigment[2] = MAT[29][i]; // gamma
    }
    return;
}
```

Let's make sure we use it

- Compile and copy povwriter (executable) to the root project directory
- Copy ./config/povwriter-settings.xml to the config directory in your project

- Edit the config file:
  - in **options**, set **use_standard_colors** to false.
  - set the camera position in **camera**:
    - Our scene is [-300,300]x[-300,300]x[-300,300], so let's place the camera 500 microns from origin
    - Also, in **clipping_planes** I suggest we comment out all but (0,0,1,0) (points up).

Create povray files

- Start from XML file 0, increase in increments of 1, end at 120:
- povwriter 0:1:120

- then, open the files up in povwriter (make sure you have a square screen pre-defined), and process the files to make a series of png files

-

Create povray files

- Start from XML file 0, increase in increments of 1, end at 120:
- povwriter 0:1:120

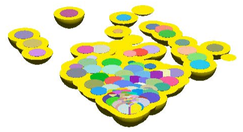
- then, open the files up in povwriter (make sure you have a square screen pre-defined), and process the files to make a series of png files

Create a movie

- Convert from png to jpeg with ImageMagick:
  - magick mogrify -format jpg *.png

- Use mencoder to create the mpeg4 movie
  - mencoder "mf://pov*.jpg" -ovc lavc -lavcopts vcodec=mpeg4:vbitrate=10000:mbd=2:trell -mf fps=24:type=jpg -nosound -o out.avi

- More advanced uses: ImageMagick to apply time labels to frames.



Tomorrow

- More advanced techniques
  - Contact testing
  - Inducing apoptosis
- Thanos vs. the Avengers
- Intro to loading data in Python
- Creating Jupyter GUIs
- Deploying PhysiCell models as cloud-hosted models