Slides, videos, links and more:
https://github.com/physicell-training/ws2021

Session 7: Functions in PhysiCell

Paul Macklin, Ph.D.

@MathCancer

PhysiCell Project

July 27, 2021

LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Goals

- PhysiCell full modeling workflow
- Handy C++ Tidbits for Cell Agents
- Typical form / syntax / purpose of PhysiCell functions
- The customizable functions in Cell.functions
- How to assign new functions to a cell definition
- Sampling the microenvironment at Cell locations
- Example: Oxygen-based cell birth, death, and motility
- **Stretch goal:** Controlling initial cell placement
- **Stretch goal:** Custom coloring functions

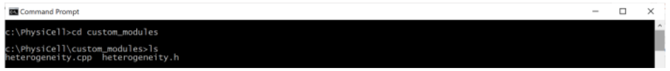
Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

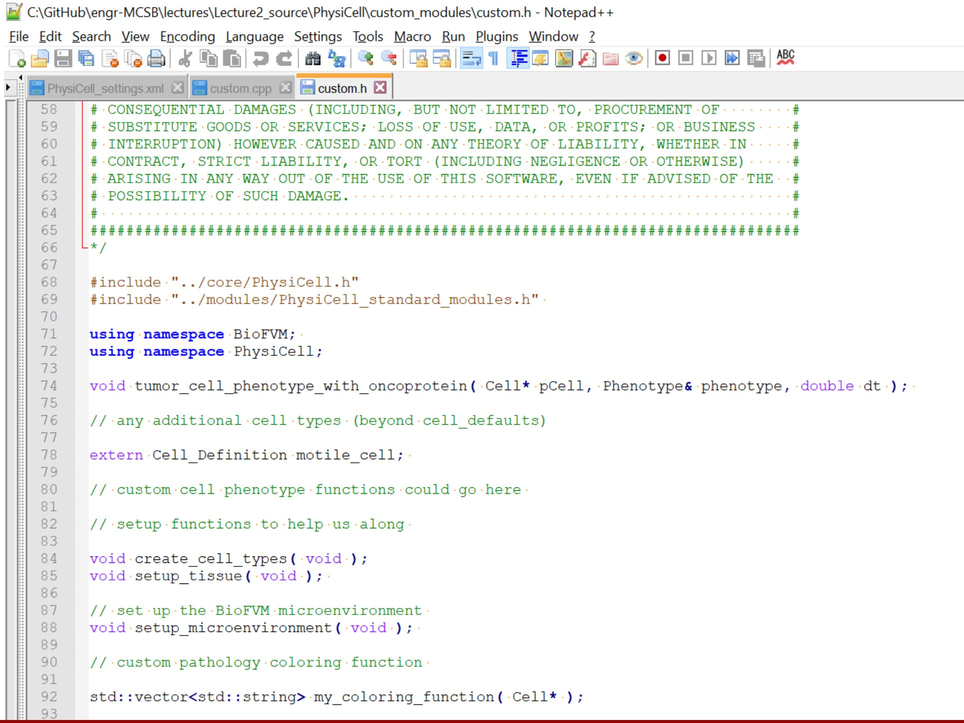
- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results

Project structure: custom modules

- Custom Modules
  - Any user-defined globals (at top)
  - 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



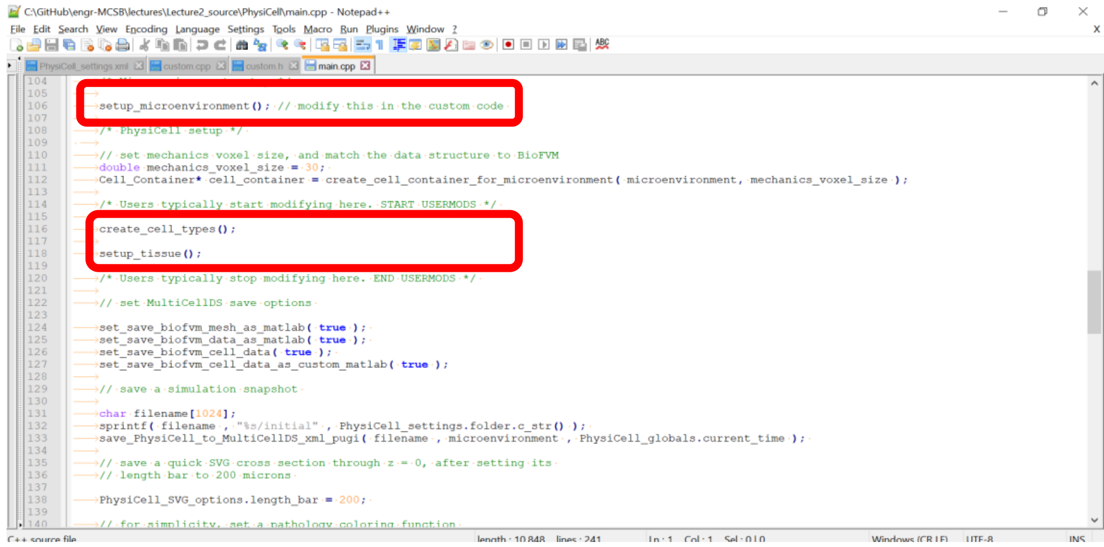
```
c:\PhysiCell>cd custom_modules

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


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

```
# 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

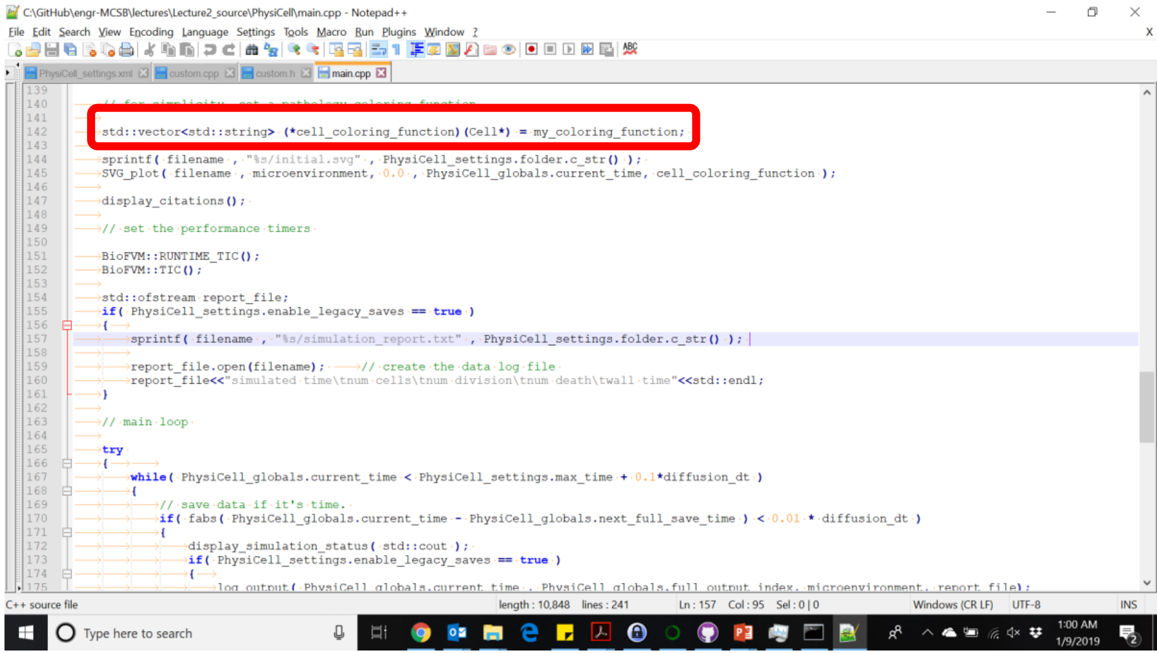
- **main.cpp**
  - (in the root directory)
  - calls the setup functions


C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\main.cpp - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PhysiCell_settings.xml
custom.cpp
custom.h
main.cpp
C++ source file
length : 10,848 lines : 241
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

```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;

    // for simplicity, set a pathology coloring function
```

Project structure: main.cpp (continued)

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



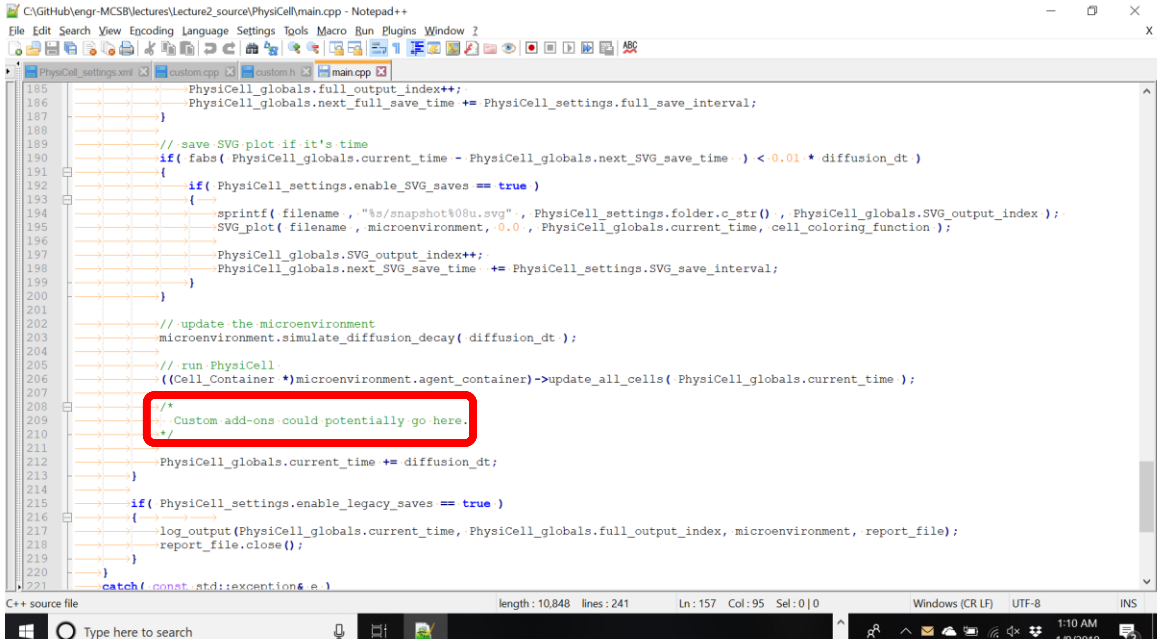
C:\GitHub\engr-MCSB\lectures\Lecture2_source\PhysiCell\main.cpp - Notepad++

```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`**
  - main loop:
    - ♦ **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 )
```

C++ source file
length : 10,848 lines : 241
Ln : 157 Col : 95 Sel : 0 | 0
Windows (CR LF) UTF-8 INS
Type here to search
1:10 AM

Summary: Where things will go

- Declare custom functions in **./custom_modules/custom.h**
- Implement these functions in **./custom_modules/custom.cpp**
- Assign custom functions to cell definitions in custom.cpp in **create_cell_types()**;
- Declare any cell parameters needed for custom functions in the **custom_data** part of a cell definition in the XML configuration file
- Declare any parameters need to set up the simulation in the **user_parameters** part of the XML config file

Handy C++ Helpers (Part 1)

Handy C++ Tidbits: Random Numbers

- `double UniformRandom( void );`
  - Get a uniformly distributed number in U(0,1)
- `double NormalRandom( double mean, double standard_deviation );`
  - Get a normally distributed number in N(mean, standard_deviation)
- `std::vector<double> UniformOnUnitCircle( void );`
  - Get a uniformly random point on the Unit Circle
- `std::vector<double> UniformOnUnitSphere( void );`
  - Get a uniformly random point on (not in!) the unit sphere.
- `int choose_event( std::vector<double>& probabilities );`
  - Given a vector of probabilities $(p_0, p_1, \ldots, p_{n-1})$, choose an integer in [0,*n*-1] with the given probabilities.
  - The probabilities must sum to 1.

These use the STL 64-bit Mersenne Twister in C++11.

Handy C++ Tidbits: Vectors

- `std::vector<double> normalize( std::vector<double>& v );`
  - Return a normalized vector. (Convention: return (0,0,0) for small vectors)
- `void normalize( std::vector<double>* v );`
  - Directly normalize the vector at v. (Convention: return (0,0,0) for small vectors)
- `double norm_squared( const std::vector<double>& v );`
  - Return a norm squared. (Handy and avoids an expensive square root.)
- `double norm( const std::vector<double>& v );`
  - Returns standard Euclidean (ℓ_2) norm.
- `double maxabs( const std::vector<double>& v );`
  - Returns the maximum absolute value in the vector. i.e., this is the ℓ_∞ norm.
- `void csv_to_vector( const char* buffer , std::vector<double>& vect );`
  - Starting with a character string, separates (by a comma delimiter), converts to doubles, and stores result in the vector.
- `char* vector_to_csv( const std::vector<double>& vect );`
  - Turns a vector into a comma-separated string. I should probably modernize this with std::string.

We have also defined expected vector operations for std::vector<double>:
+, -, *, /, +=, -=, *=, /=

Handy C++ tidbits: forced birth and death

- **Cell** methods for forcing cell birth and death
  - `void flag_for_division( void );`
    - ♦ Use this if you want to force the cell to divide at the next opportunity
  - `void start_death( int death_model_index );`
    - ♦ Use this to trigger a specific death model (at the next opportunity)
  - `void lyse_cell( void );`
    - ♦ Trigger ***immediate death*** that sets volume to 0, deactivates all functions, and detaches the cell from all other linked cells. Cell will be deleted at next dt_cell step.
    - ♦ Safer than flag_for_removal()

Handy C++ tidbits: Boolean flags

- The **Cell** class has a few useful member data:
  - `bool is_out_of_domain;`
    - Set this to **true** if the cell is out of the domain.
    - I'm not 100% sure this is maintained up-to-date by the code. (It's on my to do list!)
  - `bool is_movable;`
    - This is ordinarily **true**, and allows the cell to be pushed by other cells.
    - Set this to **false** if you want the cell to exert forces on other cells, but don’t want it to move (i.e., behave as a rigid barrier)

PhysiCell Cell Functions

Functions in PhysiCell

- In PhysiCell, almost all cell functions have the following form:

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

• pCell :	pointer to a cell. Can be NULL
• phenotype:	a cell phenotype. Usually pCell->phenotype.
• dt:	how far the function / model should be advanced in time.

- These functions can access:
  - Cell **state** via `pCell->state`
  - Cell **custom** data via `pCell->custom_data`
  - Cell **functions** via `pCell->functions`
  - Cell **phenotype** via phenotype
  - nearby **microenvironment** via `pCell->nearest_density_vector()` & `pCell->nearest_gradient_vector()`

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` (defaults to a built-in model)
- `update_migration_bias` (default NULL unless you enabled chemotaxis)
- `custom_cell_rule` (default NULL, evaluated at each mechanics time step)
- `update_phenotype` (default NULL, evaluated at each phenotype time step)
- `update_velocity` (defaults to a built-in model with potentials)
- `set_orientation` (automatically set as needed)
- `contact_function` (default NULL, evaluated at each mechanics time step)
  - We'll spend more time on this in Session 10

Purpose of the Functions

- **volume_update_function**
  - Dynamically grow / shrink cells towards "target" values. Usually you can stick with the default function.
- **update_migration_bias ****
  - Used whenever a cell chooses a new migration bias direction.
- **custom_cell_rule ****
  - A catch-all customization that's evaluated at each mechanics time step. (0.1 min)
  - Use this for rules that need frequent evaluation.
- **update_phenotype ****
  - The general purpose rule to set phenotype parameters at each cell temp step. (6 min)
  - Generally where you spend the majority of your (implementation) time in a modeling project.
- **update_velocity**
  - Sets the cell velocity based on interaction potentials.
  - The custom rule and motility functions are automatically evaluated as well. Usually you can stick with the default function.
- **set_orientation**
  - Used during cell division to choose the division plane (a random plane through this vector).
  - We set this to (0,0,1) for 2-D simulations to ensure division in the xy-plane
- **contact_function**
  - A newer addition for cell-cell contact interactions such as adding/removing spring links. Evaluated at each mechanics step. More tomorrow.

A short example

- In custom.h, declare your new function;

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

- In custom.cpp, write the code:

```
void my_phenotype_function( Cell* pCell, Phenotype& phenotype, double dt )
{
        // get a rate from cell's custom data
        double rate = pCell->custom_data["rate"];
        // change a cell's apoptosis rate
        phenotype.death.rates[0] = rate;
        return;
}
```

- Use the function

```
cell_defaults.functions.update_phenotype = my_phenotype_function;
```

  - The best place to do this is in create_cell_types() in custom.cpp

Handy C++ Helpers (Part 2)

Cell methods to access the microenvironment

- `int` **`get_current_voxel_index`**`( void )`
  - Gives the index of the voxel that contains the agent's center.
  - More on this when we discuss diffusion

- `std::vector<double>&` **`nearest_density_vector`**`( void )`
  - a vector for all the substrate density values (stuff/volume) in the cell's voxel
  - allows the user to directly access (i.e., sample or modify) the vector of substrates at the cell's position
  - useful building functions that alter cell phenotype based on the microenvironment.

- `std::vector<double>&` **`nearest_gradient`**`( int substrate_index )`
  - for the substrate with index *i,* gives the gradient $\left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right] \rho_i$ in the cell's voxel
  - useful for things like chemotaxis.

- `std::vector<gradient>&` **`nearest_gradient_vector`**`( void )`
  - gives a vector of ***all*** the gradients in the cell's voxel returns

More key functions

- `Cell_Definition*` **`find_cell_definition`**`( std::string )`
  - Get a pointer to a cell definition by searching for its name.
- `Cell_Definition*` **`find_cell_definition`**`( int )`
  - Get a pointer to a cell definition by searching for its integer type.
  - Since cells keep their `type_ID`, this can be quite handy for phenotype functions.
- `int Microenvironment::`**`find_density_index`**`( std::string )`
  - Search for the index in for a specific ucstrate
  - Useful for phenotype functions.
  - The default microenvironment in PhysiCell is named **microenvironment.**

Full Model Workflow: Example

Scenario: Oxygen-dependent cells

- Let's illustrate these with an example:
  - tumor cells:
    - Cycle entry proportional to local pO2
    - Necrosis probability increases below a pO2 threshold
  - motile tumor cells:
    - Same as tumor cells, but:
      - 1/10 cycling rate
      - 1/10 apoptosis rate
      - a more advanced chemotaxis up oxygen gradients
      - migration slows as oxygen increases.

Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results

Planning (1)

- Microenvironment
  - [-400,400] x [-400,400], 2160 minutes max time.
  - Oxygen with default parameters, boundary and initial conditions to 38 mmHg
  - Enable virtual wall

- Custom cell data (known once you have planned your cell functions)

pO2_proliferation_saturation	(max proliferation rate above this value)
pO2_proliferation_threshold	(no proliferation below this value)
pO2_necrosis_threshold	(necrosis starts at this value)
pO2_necrosis_saturation	(necrosis at max value below this value)
max_necrosis_rate	(max necrotic death rate for very low pO2)
pO2_half_max	(for Hill function)
pO2_hill_power	(for Hill function)

- Cell definitions
  - tumor
  - motile tumor

Planning (2)

- Tumor cell proliferation (σ = pO_2) with the simpler **live** cycle model.

$$r_{00} = \bar{r}_{00}\left(\frac{\sigma - \sigma_{\text{p\_threshold}}}{\sigma_{\text{p\_saturation}} - \sigma_{\text{p\_threshold}}}\right)$$

  - $\sigma_{\text{p\_saturation}} = 38 \text{ mmHg } (5\%)$
  - $\sigma_{p\_\text{threshold}} = 5 \text{ mmHg } (0.65\%)$
  - $\bar{r}_{00} = 0.00072 \ \text{min}^{-1}$

- Tumor cell necrosis (σ = pO_2)

$$r_N = \bar{r}_N\left(\frac{\sigma_{\text{n\_threshold}} - \sigma}{\sigma_{\text{n\_threshold}} - \sigma_{\text{n\_saturation}}}\right)$$

  - $\sigma_{\text{n\_threshold}} = 5 \text{ mmHg } (0.65\%)$
  - $\sigma_{\text{n\_saturation}} = 2.5 \text{ mmHg } (0.32\%)$
  - $\bar{r}_N = 0.0028 \ \text{min}^{-1}$

Planning (3)

- Tumor cell motility (σ = pO_2)
  - Let's use a basic Hill function to modulate speed and bias in (already) motile tumor cells
  - Assume as pO2 increases, the "signal" is stronger for less random and faster migration.

$$\mathbf{d}_{\text{bias}} = \frac{\nabla\sigma}{|\nabla\sigma|}$$

$$\text{speed} = \bar{s}\left(1 - \frac{s}{1+s}\right), \qquad \text{where} \quad s = \left(\frac{\sigma}{\sigma_{\text{HM}}}\right)^{\text{hp}}$$

$$\text{bias} = \left(\frac{s}{1+s}\right)$$

  - $\bar{s} = 1\ \mu\text{m/min}$
  - $\sigma_{\text{HM}} = 4\ \text{mmHg}\ (1\%)$
  - $\text{hp} = 2$
  - set persistence time to 15 min

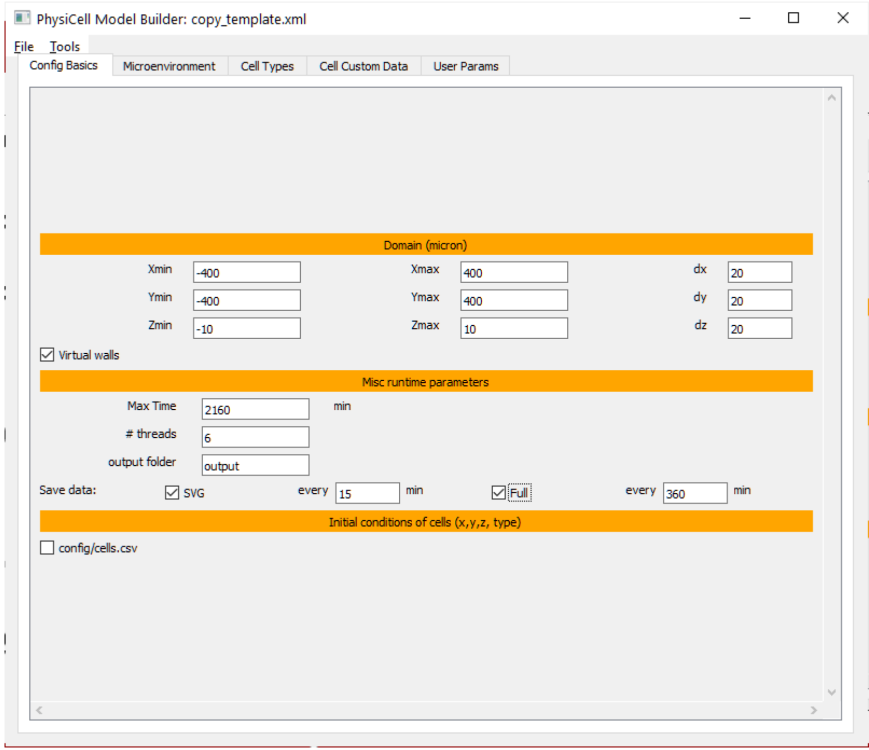
Start modeling!

- populate and build the template project
  - `make template`
  - `make`

- Open Model Builder GUI
  - enter the `./PhysiCell/config` directory
  - `python ../../PhysiCell-model-builder/bin/gui4xml.py`

Edit the model: domain

- Go to "config basics" tab
- Xmin = -400, Xmax = 400
- Ymin = -400, Ymax = 400
- max time = 2160
- full output every 360 min
- SVG every 15 min
- activate "virtual wall"
  - keep cells from leaving the domain



Domain (micron)

Xmin	-400	Xmax	400	dx	20
Ymin	-400	Ymax	400	dy	20
Zmin	-10	Zmax	10	dz	20

☑ Virtual walls

Misc runtime parameters

Max Time 2160 min
# threads 6
output folder output
Save data: ☑ SVG every 15 min ☑ Full every 360 min

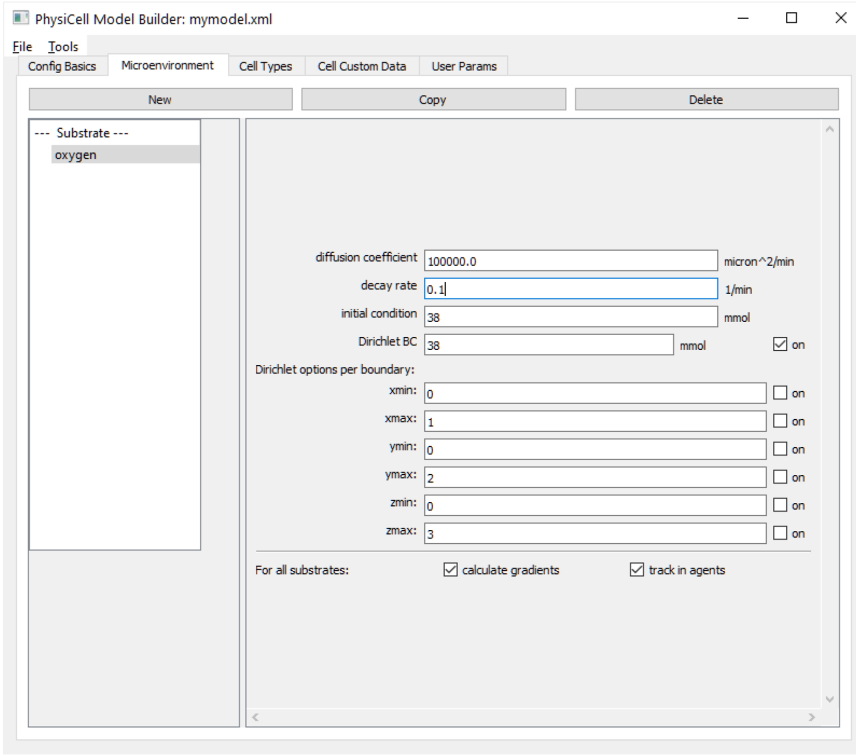
Initial conditions of cells (x,y,z, type)

☐ config/cells.csv

Edit the model: microenvironment

- Go to "microenvironment" tab

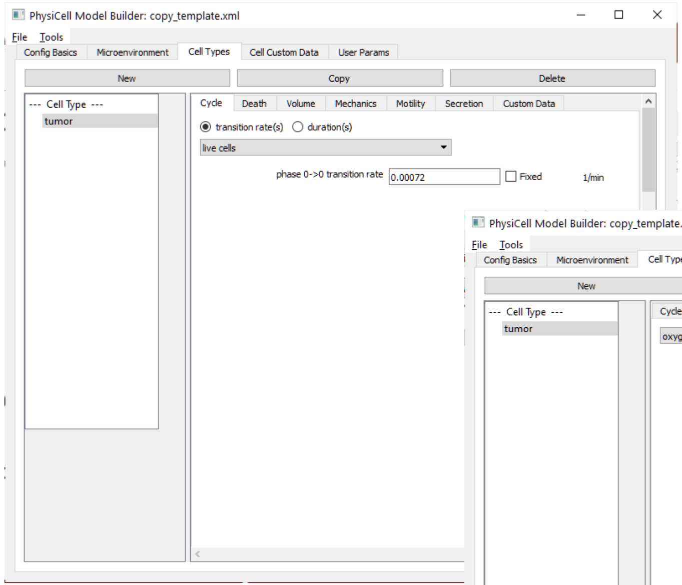
- double-click "substrate"
  - rename it oxygen, with units mmHg
  - reduce decay rate to 0.1
  - set Dirichlet BC to 38 (mmHg)
  - enable the Dirichlet BC
  - set initial value to 38 (mmHg)



diffusion coefficient 100000.0 micron^2/min
decay rate 0.1 1/min
initial condition 38 mmol
Dirichlet BC 38 mmol ☑ on
Dirichlet options per boundary:
xmin: 0 ☐ on
xmax: 1 ☐ on
ymin: 0 ☐ on
ymax: 2 ☐ on
zmin: 0 ☐ on
zmax: 3 ☐ on
For all substrates: ☑ calculate gradients ☑ track in agents

Edit the model: cell definitions (1)

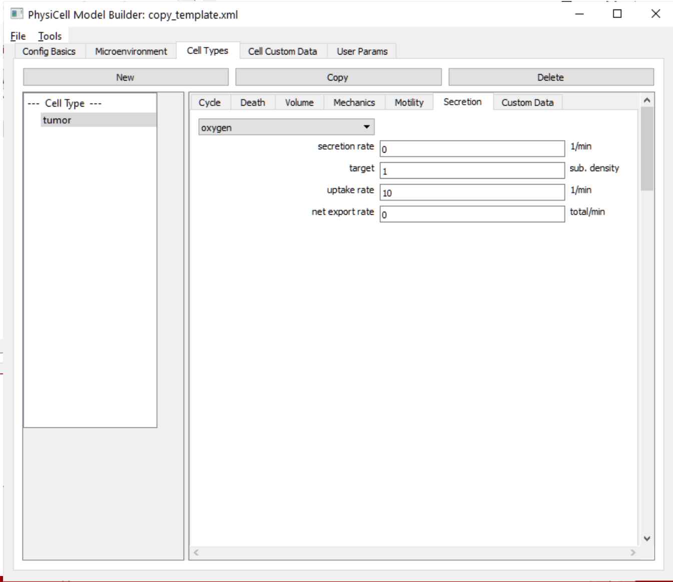
- Go to "cell types" tab
- double-click "default"
  - rename it "tumor"
  - edit its phenotype:
    - ♦ click "cycle" subtab
      - » choose live cycle model
      - » select "transition rate(s)"
      - » set 0→0 transition to 0.00072
    - ♦ click "secretion" subtab
      - » Choose "oxygen" from dropdown
        - ○ set uptake rate to 10


PhysiCell Model Builder: copy_template.xml
File Tools
Config Basics Microenvironment Cell Types Cell Custom Data User Params
New Copy Delete
--- Cell Type ---
tumor
Cycle Death Volume Mechanics Motility Secretion Custom Data

(●) transition rate(s) ( ) duration(s)

live cells

phase 0->0 transition rate 0.00072 [ ] Fixed 1/min

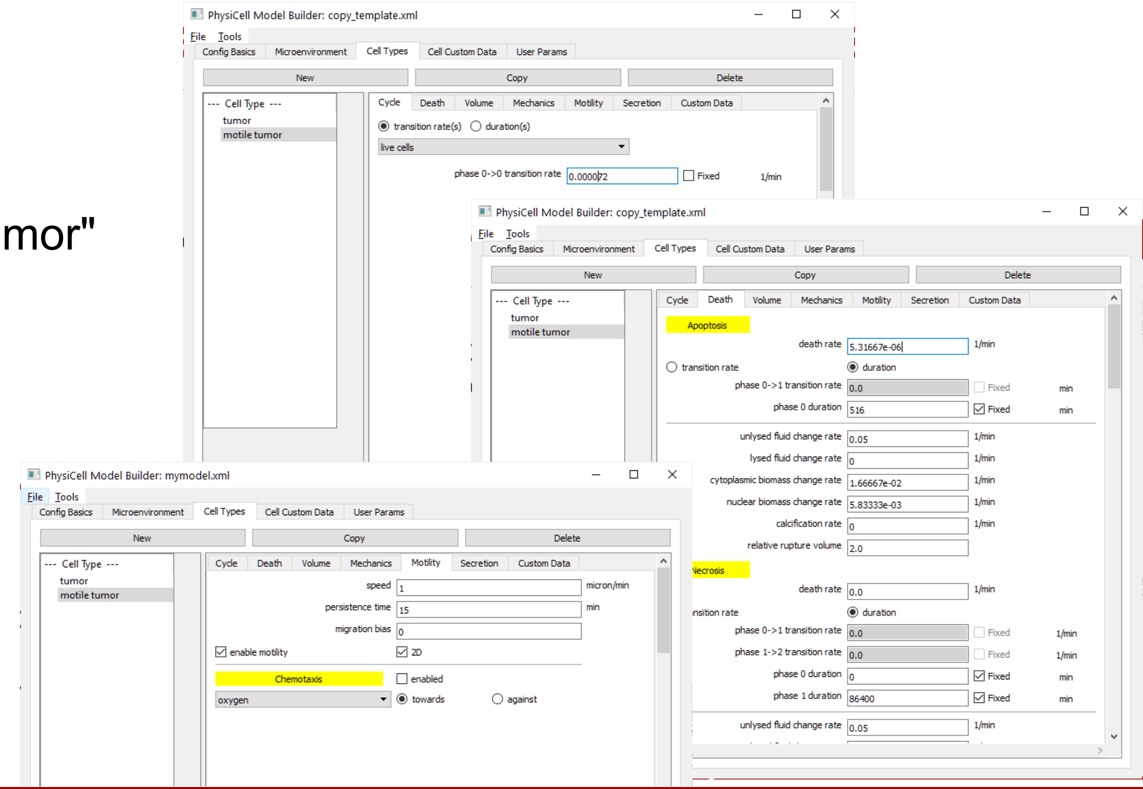

PhysiCell Model Builder: copy_template.xml
File Tools
Config Basics Microenvironment Cell Types Cell Custom Data User Params
New Copy Delete
--- Cell Type ---
tumor
Cycle Death Volume Mechanics Motility Secretion Custom Data

oxygen

secretion rate	0	1/min
target	1	sub. density
uptake rate	10	1/min
net export rate	0	total/min

Edit the model: cell definitions (2)

- Go to "cell types" tab
- click "copy" to duplicate the tumor type
  - double click, rename to "motile tumor"
  - edit its phenotype:
    - ♦ click on "cycle"
      - » set phase 00 transition to 0.000072
    - ♦ click on "death"
      - » set apoptosis rate to 5.31667e-06
    - ♦ click on "motility"
      - » set speed to 1
      - » set persistence time to 15
      - » set bias to 0 (we'll handle in-function later)
      - » check to enable


PhysiCell Model Builder: copy_template.xml

Cell Type: tumor, motile tumor (selected)

Cycle: [✓ transition rate(s)] duration(s); live cells

phase 0->0 transition rate 0.000072 Fixed 1/min

PhysiCell Model Builder: copy_template.xml

Cell Type: tumor, motile tumor (selected)

Death — Apoptosis

death rate	5.31667e-06		1/min
transition rate [✓ duration]			
phase 0->1 transition rate	0.0	Fixed	min
phase 0 duration	516	✓ Fixed	min
unlysed fluid change rate	0.05		1/min
lysed fluid change rate	0		1/min
cytoplasmic biomass change rate	1.66667e-02		1/min
nuclear biomass change rate	5.83333e-03		1/min
calcification rate	0		1/min
relative rupture volume	2.0		

Necrosis

death rate	0.0		1/min
transition rate [✓ duration]			
phase 0->1 transition rate	0.0	Fixed	1/min
phase 1->2 transition rate	0.0	Fixed	1/min
phase 0 duration	0	✓ Fixed	min
phase 1 duration	86400	✓ Fixed	min
unlysed fluid change rate	0.05		1/min

PhysiCell Model Builder: mymodel.xml

Cell Type: tumor, motile tumor (selected)

Motility

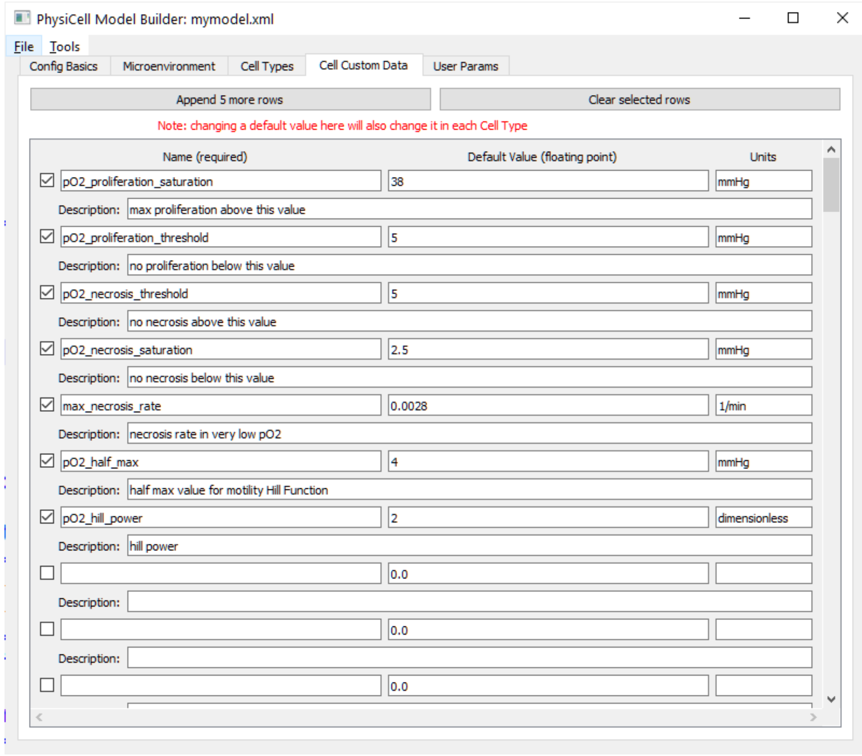
speed	1	micron/min
persistence time	15	min
migration bias	0	

✓ enable motility ✓ 2D

Chemotaxis: enabled (unchecked); oxygen; [✓ towards] against

Edit the model: custom cell data

- Go to "cell custom data" tab
  - add a new data called "pO2_proliferation_saturation"
    - ♦ Fill out the default value, units, and description
    - ♦ Check the box on left to make sure it's copied to all cell definitions
  - Add pO2_proliferation_threshold
  - Add pO2_necrosis_threshold
  - Add pO2_necrosis_saturation
  - Add max_necrosis_rate
  - Add pO2_half_max
  - Add pO2_hill_power

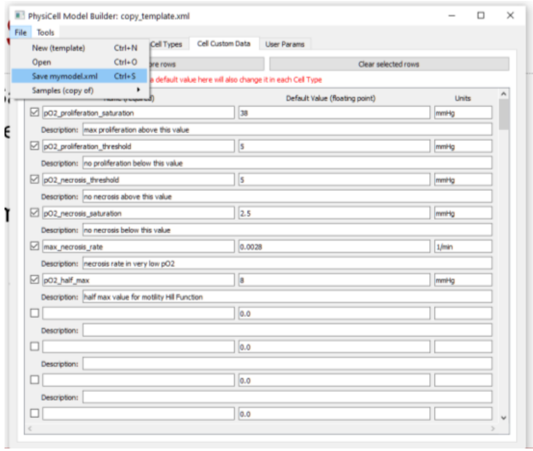


Note: changing a default value here will also change it in each Cell Type

	Name (required)	Default Value (floating point)	Units
☑	pO2_proliferation_saturation	38	mmHg
	Description: max proliferation above this value		
☑	pO2_proliferation_threshold	5	mmHg
	Description: no proliferation below this value		
☑	pO2_necrosis_threshold	5	mmHg
	Description: no necrosis above this value		
☑	pO2_necrosis_saturation	2.5	mmHg
	Description: no necrosis below this value		
☑	max_necrosis_rate	0.0028	1/min
	Description: necrosis rate in very low pO2		
☑	pO2_half_max	4	mmHg
	Description: half max value for motility Hill Function		
☑	pO2_hill_power	2	dimensionless
	Description: hill power		
☐		0.0	
	Description:		
☐		0.0	
	Description:		
☐		0.0	

Save to the project

- Go to "File", then "Save mymodel.xml"
  - This saves to wherever we ran PhysiCell Model Builder
- If needed, copy mymodel.xml to ./PhysiCell/config/
- Since we ran inside the config directory, it's already there!



a default value here will also change it in each Cell Type

	Default Value (floating point)	Units
☑ pO2_proliferation_saturation	38	mmHg
Description: max proliferation above this value		
☑ pO2_proliferation_threshold	5	mmHg
Description: no proliferation below this value		
☑ pO2_necrosis_threshold	5	mmHg
Description: no necrosis above this value		
☑ pO2_necrosis_saturation	2.5	mmHg
Description: no necrosis below this value		
☑ max_necrosis_rate	0.0028	1/min
Description: necrosis rate in very low pO2		
☑ pO2_half_max	8	mmHg
Description: half max value for motility Hill Function		
☐	0.0	
Description:		
☐	0.0	
Description:		
☐	0.0	
Description:		
☐	0.0	

Unzip Session7_checkpoint1.zip in ./PhysiCell to get this code.

Declare custom functions

- In `./custom_modules/custom.h`, declare:

```
void tumor_phenotype( Cell* pC, Phenotype& p, double dt);

void motile_tumor_phenotype( Cell* pC, Phenotype& p, double dt);

void motility_rule( Cell* pC, Phenotype& p, double dt );
```

Custom phenotype rule (1)

```
void tumor_phenotype( Cell* pC, Phenotype& p, double dt)
{
   // find my cell definition
   // find index of O2 in the microenvironment
   // find index of necrosis death model

   // sample O2

   // set birth rate
      // get base rate from cell definition
      // set multiplier to 1.0
      // sample pO2. if pO2 < pO2_proliferation_saturation:
         // multiplier = (pO2 -pO2_proliferation_threshold)
         //   /(pO2_proliferation_saturation-pO2_proliferation_threshold)
      // if pO2 < pO2_proliferation_threshold, set multipler = 0.0
      // transition rate = base_rate * multiplier

   // set necrosis rate
      // multiplier = 0.0
      // if pO2 < pO2_necrosis_threshold
         // multipler = (pO2_necrosis_threshold - pO2)
         //   /(pO2_necrosis_threshold-pO2_necrosis_saturation)
      // if pO2 < pO2_necrosis_saturation
         // multipler = 1
      // necrosis rate = max_necrosis_rate * multiplier

   // if dead, set secretion / uptake rates to zero

   // trick: if dead, overwrite with NULL function pointer.
}
```

Custom phenotype rule (2)

```
void tumor_phenotype( Cell* pC, Phenotype& p, double dt)
{
   // find my cell definition
   Cell_Definition* pCD = find_cell_definition( pC->type );

   // find index of O2 in the microenvironment
   static int nO2 = microenvironment.find_density_index( "oxygen" );

   // find index of necrosis death model
   static int nNecro = p.death.find_death_model_index( "Necrosis" );

   // sample O2
   double pO2 = pC->nearest_density_vector()[nO2];

   // set birth rate
      // get base rate from cell definition
   double base_rate = pCD->phenotype.cycle.data.transition_rate(0,0);
      // set multiplier to 1.0
   double multiplier = 1.0;
```

Custom phenotype rule (3)

```
   // sample pO2. if pO2 < pO2_proliferation_saturation:
if( pO2 < pC->custom_data["pO2_proliferation_saturation"] )
{
     // multiplier = (pO2 -pO2_proliferation_threshold)
     //    /(pO2_proliferation_saturation-pO2_proliferation_threshold)

   multiplier = (pO2 - pC->custom_data["pO2_proliferation_threshold"] )
       /( pC->custom_data["pO2_proliferation_saturation"]
         -pC->custom_data["pO2_proliferation_threshold"] );
}

   // if pO2 < pO2_proliferation_threshold, set multipler = 0.0
if( pO2 < pC->custom_data["pO2_proliferation_threshold"] )
{ multiplier = 0.0; }

   // transition rate = base_rate * multiplier
p.cycle.data.transition_rate(0,0) = base_rate * multiplier;
```

Custom phenotype rule (4)

```
// set necrosis rate
   // multiplier = 0.0
multiplier = 0.0;

   // if pO2 < pO2_necrosis_threshold
if( pO2 < pC->custom_data["pO2_necrosis_threshold"] )
{
      // multipler = (pO2_necrosis_threshold - pO2)
      //    /(pO2_necrosis_threshold-pO2_necrosis_saturation)

   multiplier = (pC->custom_data["pO2_necrosis_threshold"] - pO2 )
        / ( pC->custom_data["pO2_necrosis_threshold"]
           -pC->custom_data["pO2_necrosis_saturation"] );
}

   // if pO2 < pO2_necrosis_saturation
      // multipler = 1
if( pO2 < pC->custom_data["pO2_necrosis_saturation"] )
{ multiplier = 1.0; }

   // necrosis rate = max_necrosis_rate * multiplier
p.death.rates[nNecro] = pC->custom_data["max_necrosis_rate"] * multiplier;
```

Custom phenotype rule (5)

```
   // if dead, set secretion / uptake rates to zero
   // trick: if dead, overwrite with NULL function pointer.
   if( p.death.dead == true )
   {
     p.secretion.set_all_secretion_to_zero();
     p.secretion.set_all_uptake_to_zero();
     pC->functions.update_phenotype = NULL;
   }

   return;
}

void motile_tumor_phenotype( Cell* pC, Phenotype& p, double dt)
{ return tumor_phenotype(pC, p, dt); }
```

Unzip Session7_checkpoint2.zip in ./PhysiCell to get this code.

Custom motility rule (1)

```
void motility_rule( Cell* pC, Phenotype& p, double dt )
{
   // find my cell definition
   // find index of O2 in the microenvironment

   // sample pO2
   // sample pO2 gradient

   // check against pO2_half_max; set motility false and exit if needed

   // set migration bias direction to grad(pO2)
   // normalize

   // set the Hill multiplier
      // s = (pO2/pO2_half_max)^pO2_hill_power
      // hill = s / ( 1+s );

   // set speed
      // speed = base_speed * (1-hill)

   // migration bias
      // bias = hill

   // trick: if dead, overwrite with NULL function pointer.
}
```

Custom motility rule (2)

```
void motility_rule( Cell* pC, Phenotype& p, double dt )
{
   // find my cell definition
   Cell_Definition* pCD = find_cell_definition( pC->type );

   // find index of O2 in the microenvironment
   static int nO2 = microenvironment.find_density_index( "oxygen" );

   // sample pO2
   double pO2 = pC->nearest_density_vector()[nO2];

   // sample pO2 gradient
   // set migration bias direction to grad(pO2)
   p.motility.migration_bias_direction = pC->nearest_gradient(nO2);

   // normalize
   normalize( &(p.motility.migration_bias_direction) );
```

Custom motility rule (3)

```
   // set the Hill multiplier
      // s = (pO2/pO2_half_max)^pO2_hill_power
      // hill = s / ( 1+s );
   double temp = pow( pO2 / pC->custom_data["pO2_half_max"] , pC->custom_data["pO2_hill_power"] );
   double hill = temp / (1.0 + temp );

   // set speed
      // speed = base_speed * hill
   p.motility.migration_speed = pCD->phenotype.motility.migration_speed * (1-hill);

   // migration bias
      // bias = multiplier
   p.motility.migration_bias = hill;


   // trick: if dead, overwrite with NULL function pointer.
   if( p.death.dead == true )
   { pC->functions.update_migration_bias = NULL; }
}
```

Assign the functions

```
// in create_cell_types():

   /*
      Put any modifications to individual cell definitions here.

      This is a good place to set custom functions.
   */

   cell_defaults.functions.update_phenotype = phenotype_function;
   cell_defaults.functions.custom_cell_rule = custom_function;
   cell_defaults.functions.contact_function = contact_function;

   Cell_Definition* pCD = find_cell_definition( "tumor" );
   pCD->functions.update_phenotype = tumor_phenotype;

   pCD = find_cell_definition( "motile tumor" );
   pCD->functions.update_phenotype = motile_tumor_phenotype;
   pCD->functions.update_migration_bias = motility_rule;

   /*
      This builds the map of cell definitions and summarizes the setup.
   */

   // ...
```

Let's modify the setup

- Let's start with 200 of each cell type. Open `./config/mymodel.xml`:
- Scroll down to `user_parameters`

```
<user_parameters>
  <random_seed type="int" units="dimensionless"
    description="">0</random_seed>
  <number_of_cells type="int" units="none"
    description="initial number of cells
    (for each cell type)">200</number_of_cells>
</user_parameters>
```

Unzip Session7_checkpoint3.zip in ./PhysiCell to get this code.

Rebuild and run the project

- Open the XML config file and set max simulation time to 3600 minutes

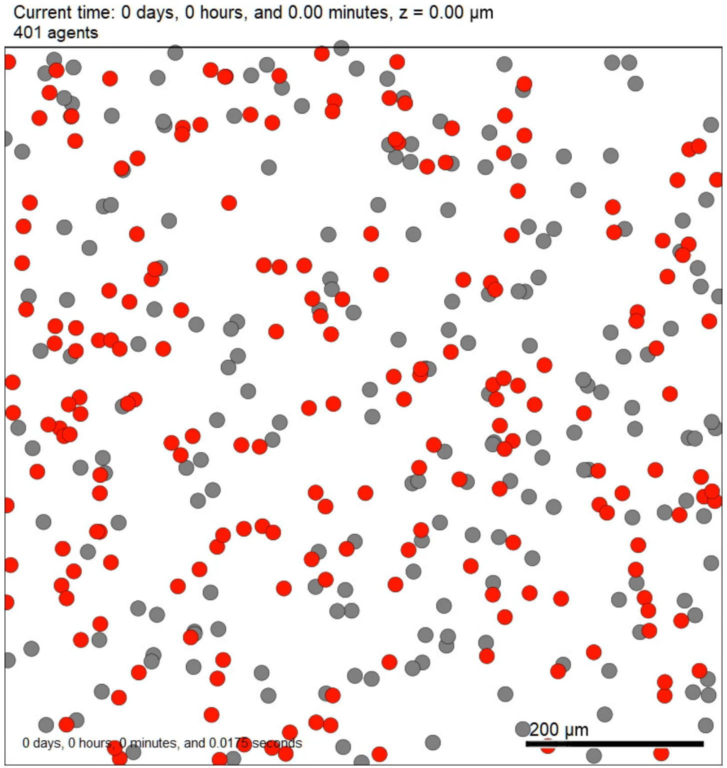
- rebuild:
  - `make`

- run:
  - `./project ./config/mymodel.xml   (linux, MacOS)`
  - `project ./config/mymodel.xml     (Windows)`

View results!

- make movie
  - `make jpeg && make movie`

● tumor
● motile tumor



- Expected behavior:
  - gray tumor cells proliferate
    - ♦ fastest near higher pO2 on boundary
  - gray cells proliferate faster than red
  - red motile tumor cells migrate towards outer boundary
  - red migration slows down as they near boundary

Handy C++ Helpers (Part 3)

Handy C++ tidbits: creating cells

- Functions to help (properly) create and place new cells
  - `Cell* create_cell( void );`
    - ♦ Create a new **Cell** using the default cell definition (cell_defaults: has ID 0)
    - ♦ Returns a pointer to the cell, allowing you to further access and modify it
  - `Cell* create_cell( Cell_Definition& cd );`
    - ♦ Create a new **Cell** using supplied cell definition
    - ♦ Returns a pointer to the cell, allowing you to further access and modify it
  - `bool assign_position( std::vector<double> new_position);`
    - ♦ Use this if you want to manually set the cell's position.
    - ♦ Fully compatible with BioFVM and its data structures
    - ♦ Useful for initialization

Example: Creating a new (default) cell

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

// 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;
```

Example: Creating and placing a new (custom) cell

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

// find the cell definition for fibroblasts
Cell_Definition* pCD = find_cell_definition( "fibroblast" );

// create a cell (of type *pCD)
pCell = create_cell( *pCD );

// choose a random point on the circle of radius 3 centered at (4,-2,0)
std::vector<double> position = UniformOnUnitCircle();
position *= 3.0; position += {4,-2,0};
pCell->assign_position( position );
```

Handy C++ tidbits: accessing all cells

- `all_cells` is a pointer to a vector of all cell agents.

- Here's the syntax to use it to traverse all cell agents:

```
for( int i=0 ; i < (*all_cells).size() ; i++ )
{
 Cell* pCell = (*all_cells)[i];
 std::cout << "cell " << i << " at " << pCell
    << " is type " << pCell->type
    << " and death status is " << pCell->phenotype.death.dead
    << std::endl;
}
```

Full Model Workflow: Example 2

Scenario: Oxygen-dependent cells v2

- Let's illustrate these with an example:
  - tumor cells:
    - ♦ Cycle entry proportional to local pO2
    - ♦ Necrosis probability increases below a pO2 threshold
  - motile tumor cells:
    - ♦ Same as tumor cells, but:
      - » 1/10 cycling rate
      - » 1/10 apoptosis rate
      - » a more advanced chemotaxis up oxygen gradients
      - » stop migrating above a threshold value
  - **Specify how many of each cell type, and place them closer to origin**

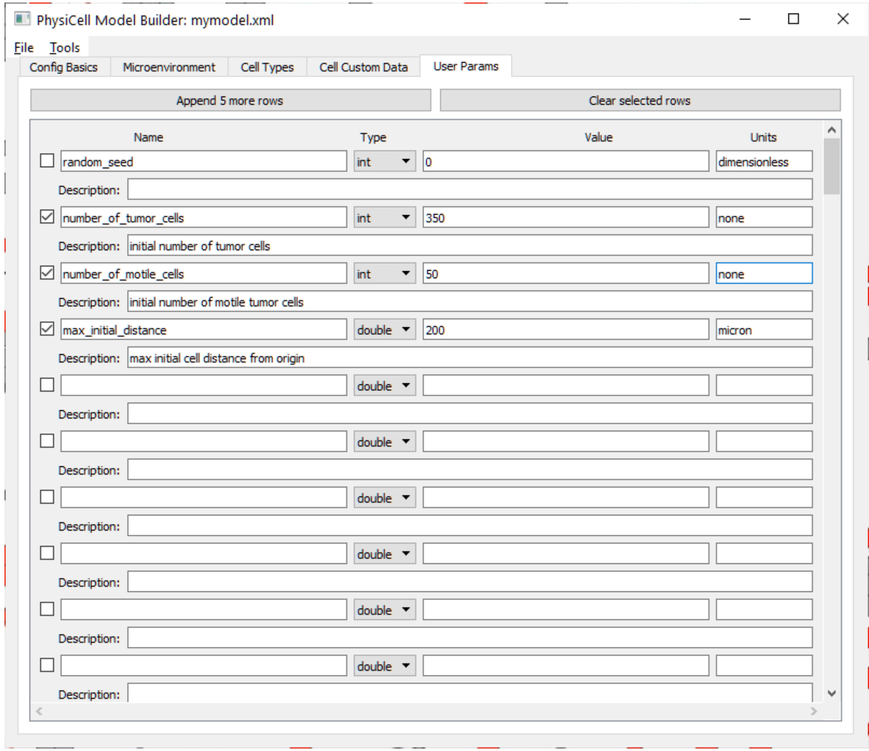
Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results

Add new user parameters

- Go to "user parameters" in Model Builder
  - rename "number_of_cells" to "number_of_tumor_cells"
    - ♦ Set its value to 750
    - ♦ Change its description to "initial number of tumor cells"
  - add another parameter called "number_of_motile_cells"
    - ♦ set its type to int
    - ♦ set its value to 50
    - ♦ set units and description
  - add another parameter called "max_initial_distance"
    - ♦ keep its type as double
    - ♦ set its value and units to 250 micron
    - ♦ set description to "max initial cell distance from origin"
- Resave mymodel.xml



	Name	Type	Value	Units
☐	random_seed	int	0	dimensionless
	Description:			
☑	number_of_tumor_cells	int	350	none
	Description:	initial number of tumor cells		
☑	number_of_motile_cells	int	50	none
	Description:	initial number of motile tumor cells		
☑	max_initial_distance	double	200	micron
	Description:	max initial cell distance from origin		
☐		double		
	Description:			
☐		double		
	Description:			
☐		double		
	Description:			
☐		double		
	Description:			
☐		double		
	Description:			
☐		double		
	Description:			

LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project
PhysiCell.org
@PhysiCell

Edit setup_tissue (1)

- In **./custom_modules/custom.cpp** find **setup_tissue**
- Comment out the current placement code:

```
/*
   for( int k=0; k < cell_definitions_by_index.size() ; k++ )
   {
      Cell_Definition* pCD = cell_definitions_by_index[k];
      std::cout << "Placing cells of type " << pCD->name << " ... " << std::endl;
      for( int n = 0 ; n < parameters.ints("number_of_cells") ; n++ )
      {
         std::vector<double> position = {0,0,0};
         position[0] = Xmin + UniformRandom()*Xrange;
         position[1] = Ymin + UniformRandom()*Yrange;
         position[2] = Zmin + UniformRandom()*Zrange;

         pC = create_cell( *pCD );
         pC->assign_position( position );
      }
   }
*/
```

Edit setup_tissue (2)

```
            pC->assign_position( position );
        }
    }
*/

    // place tumor cells
    double max_distance = parameters.doubles("max_initial_distance");

    Cell_Definition* pCD = find_cell_definition( "tumor" );
    std::cout << "Placing cells of type " << pCD->name << " ... " << std::endl;
    for( int k=0 ; k < parameters.ints( "number_of_tumor_cells" ); k++ )
    {
        std::vector<double> position = {0,0,0};
        double r = sqrt(UniformRandom())* max_distance;
        double theta = UniformRandom() * 6.2831853;
        position[0] = r*cos(theta);
        position[1] = r*sin(theta);

        pC = create_cell( *pCD );
        pC->assign_position( position );
    }
```

Edit setup_tissue (3)

```
   // place motile tumor cells
   pCD = find_cell_definition( "motile tumor" );

   std::cout << "Placing cells of type " << pCD->name << " ... " << std::endl;
   for( int k=0 ; k < parameters.ints( "number_of_motile_cells" ); k++ )
   {
      std::vector<double> position = {0,0,0};
      double r = sqrt(UniformRandom()) * max_distance;
      double theta = UniformRandom() * 6.2831853;
      position[0] = r*cos(theta);
      position[1] = r*sin(theta);

      pC = create_cell( *pCD );
      pC->assign_position( position );
   }

   std::cout << std::endl;

   // load cells from your CSV file (if enabled)
   load_cells_from_pugixml();

   return;
}
```

Unzip Session7_checkpoint4.zip in ./PhysiCell to get this code.

Rebuild and run the project

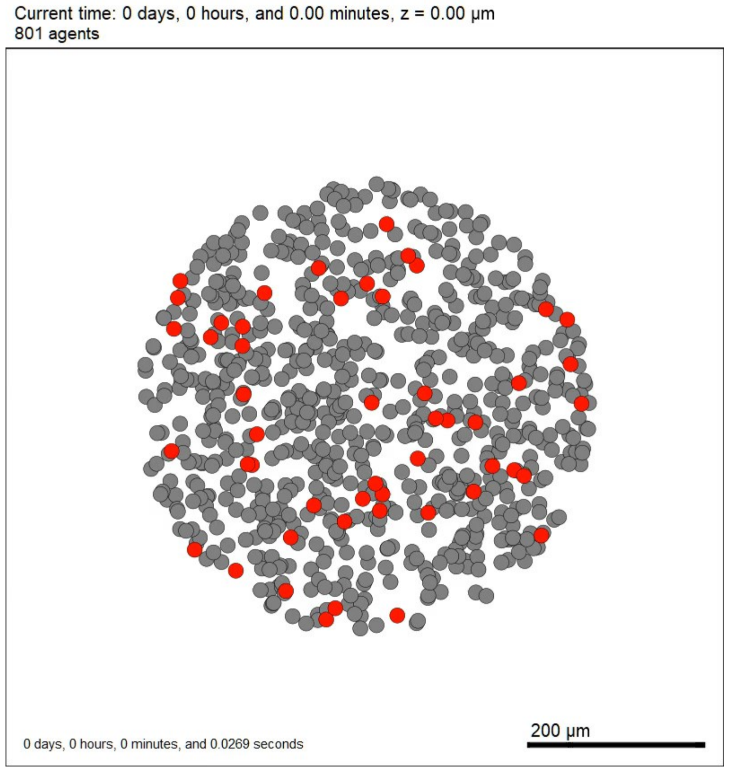
- Open the XML config file and set max simulation time to 3600 minutes
- rebuild:
  - `make`
- run:
  - `./project ./config/mymodel.xml   (linux, MacOS)`
  - `project ./config/mymodel.xml     (Windows)`

View results!

- make movie
  - **`make jpeg && make movie`**

●	tumor
🔴	motile tumor

- Expected behavior:
  - gray tumor cells proliferate
    - fastest near higher pO2 on boundary
  - gray cells proliferate faster than red
  - red motile tumor cells migrate towards outer boundary
  - red migration slows down as they near boundary



Full Model Workflow: Example 3

Scenario: Oxygen-dependent cells v3

- Let's illustrate these with an example:
  - tumor cells:
    - Cycle entry proportional to local pO2
    - Necrosis probability increases below a pO2 threshold

  - motile tumor cells:
    - Same as tumor cells, but:
      - 1/10 cycling rate
      - 1/10 apoptosis rate
      - a more advanced chemotaxis up oxygen gradients
      - stop migrating above a threshold value

  - Specify how many of each cell type, and place them closer to origin
  - Dial initial and boundary pO2 down to 15 mmHg
  - **Color non-motile tumor cells based on current cycling rate**

Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results

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

Let's make our shade from blue (zero prolif rate) to yellow (max prolif rate)

Coloring function (1)

```
// declare new coloring in custom.h

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

// start work in custom.cpp

std::vector<std::string> custom_coloring_function( Cell* pC )
{
   // start with paint-by-numbers
   std::vector<std::string> output = paint_by_number_cell_coloring(pC);

   // get tumor cell def
   static Cell_Definition* pTC = find_cell_definition( "tumor" );

   // return if not live tumor cell
   if( pC->phenotype.death.dead || pC->type != pTC->type )
   { return output; }
```

Coloring function (2)

```
   // get relative birth rate
   double s = pC->phenotype.cycle.data.transition_rate(0,0) /
      pTC->phenotype.cycle.data.transition_rate(0,0);

   // make color
   int color = (int) round( 255.0 * s );
   char szColor [1024];

   // blue to yellow
   sprintf( szColor, "rgb(%u,%u,%u)",color,color,255-color );

   // modify output
   output[0] = szColor;
   output[2] = szColor;
   output[3] = szColor;

   return output;
}
```

Tell PhysiCell to use your coloring function

In main.cpp

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

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

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

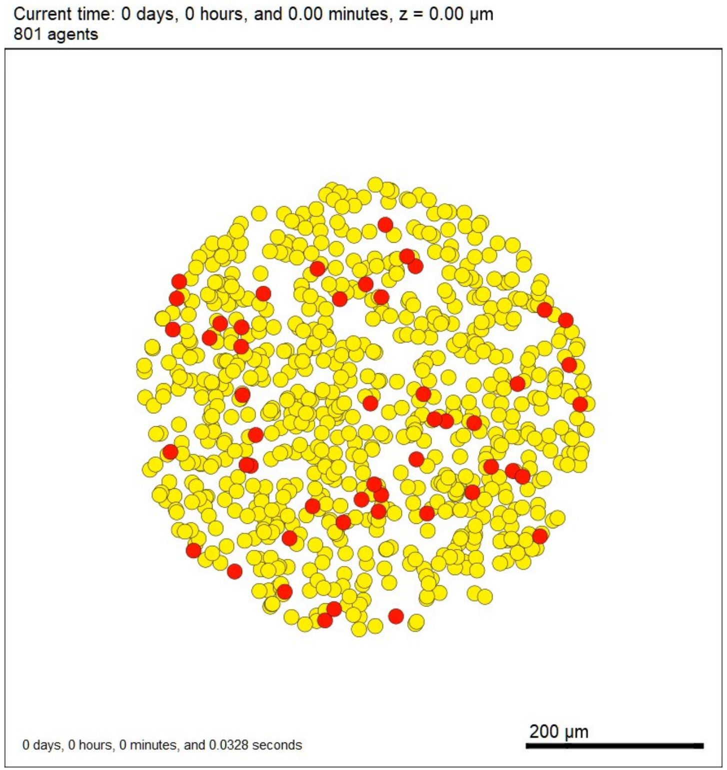
User Guide: Section 14.2

Unzip Session7_checkpoint5.zip in ./PhysiCell to get this code.

Run and View results!

- make && ./project ./config/mymodel.xml

- make movie
  - `make jpeg && make movie`



🟡 tumor
🔴 motile tumor

Funding Acknowledgements

PhysiCell Development:

- Breast Cancer Research Foundation
- Jayne Koskinas Ted Giovanis Foundation for Health and Policy
- National Cancer Institute (U01CA232137)
- National Science Foundation (1720625)

Training Materials:

- Administrative supplement to NCI U01CA232137 (Year 2)