**Slides, videos, links and more:**
https://github.com/physicell-training/ws2023

# Advanced Session 1:

## Boundary Conditions, Custom Variables, User Parameters, and C++ searches

Aneequa Sundus
@AneequaSundus

# PhysiCell Project

August 7, 2023

# Agenda

- Working with C++ code
  - Access user parameters
  - Define Custom Cell variables
  - Query  cell definitions
  - Query for microenvironment
  - Boundary conditions
  - Dictionaries for signals and behaviors

# Need and Files to edit

- Advanced behaviors that can not be achieved by defaults
- Custom Functions
- File to edit
  - `custom.cpp`
  - `custom.h`
  - `PhysiCell_settings.xml`

# User Parameters

- Define
  - PhysiCell Studio (recommended)
  - Add in XML file
- Access

- `parameters.doubles("oncoprotein_mean");`
- `parameters.ints("random_seed");`

```
<initial_conditions>
    <cell_positions type="csv" enabled="false">
        <folder>./config</folder>
        <filename>cells.csv</filename>
    </cell_positions>
</initial_conditions>

<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)">0</number_of_cells>
    <tumor_radius type="double" units="micron" description="">250</tumor_radius>
    <oncoprotein_mean type="double" units="" description="">1</oncoprotein_mean>
    <oncoprotein_sd type="double" units="" description="">0.25</oncoprotein_sd>
    <oncoprotein_min type="double" units="" description="">0.0</oncoprotein_min>
    <oncoprotein_max type="double" units="" description="">2</oncoprotein_max>
</user_parameters>
```

# Custom Variables

- How?
  - PhysiCell Studio (recommended)
  - Add in C++ file (`custom_modules/custom.cpp`)
- Define :
- `cell_defaults.custom_data.add_variable( "energy", "dimensionless" , 0.5);`
- `cell_defaults.custom_data.add_variable( "energy", "dimensionless" , parameters.doubles("cell_default_inital_energy") );`
- `cell_defaults.custom_data.add_variable( "alpha", "none" , parameters.doubles("cell_default_aplha") );`
- How?
- `static int nE = pCell->custom_data.find_variable_index( "energy" );`
- `pCell->custom_data[nE]`

# Cell definition search

- By index

```
Cell_Definition* pCD = cell_definitions_by_index[n];
```

- By Human readable name

```
pCD = find_cell_definition("blood vessel");
```

# Boundary conditions microenvironment

# Sampling the microenvironment (1)

- There is a global microenvironment called **microenvironment.** You can access it anywhere from inside a PhysiCell model.

- Each cell is in some computational voxel in the microenvironment.

  `pCell->get_current_voxel_index( void );` Get the index of the voxel

- You can query the microenvironment to determine which index corresponds to a variable.
  - microenvironment.find_density_index( "resource" ); // Find the index of "resource".
    - This function returns -1 if it can't find your substrate.

- Each cell can access the vector of chemical substrates in its voxel
  - pCell->nearest_density_vector();
    - Vector of all the substrates
  - pCell->nearest_density_vector()[2]
    - substrate with index 2 in the cell's voxel
    - often, you'll want to use the search above to figure out which index

# Sampling the microenvironment (2)

- Each cell can access the gradients of the substrates in its voxel
  - `pCell->nearest_gradient(2);`
    - gradient of substrate #2

- We can access the mesh
  - `microenvironment.mesh`
  - `microenvironment.mesh.voxels`

- We can iterate through all voxels

```
for( int i=0; i < microenvironment.mesh.voxels.size() ; i++ )
{ std::cout << microenvironment.mesh.voxels[i].center << std::endl; }
```

# Dirichlet's nodes

- add voxels to act as source or sink

- `void Microenvironment::add_Dirichlet_node( int voxel index, std::vector<double>& value )`
- `microenvironment.add_dirichlet_node( n,bc_vector);`

- Where n is the voxel number and bc_vector is double vector of size n where n is number of substrates in the environment.

# Signal and Behavior Dictionaries

- a "dictionary" of standard signals
  - inputs to intracellular and rule-based models.


- A "dictionary" of standard behaviors that can be used as outputs to intracellular and rule-based models.
- This dictionaries are automatically constructed at the start of each simulation based upon the combinations of signaling substrates and cell types.

# Common Signals

- extracellular and intracellular substrate concentrations
- substrate gradients
- contact with dead cells
- contact with cells (of type X)
- damage
- Pressure
- Use display_signal_dictionary() to quickly display a list of available signals.

Windows PowerShell

```
    initial condition: 38 dimensionless
    boundary condition: 38 dimensionless (enabled: true)

virtual_wall_at_domain_edge: enabled
Pre-processing type 0 named cancer cell
Processing cancer cell ...
Note: setting cell definition to 2D based on microenvironment do
main settings ...
  --------- attachment_elastic_constant = 0.01
Signals:
=======
0 : oxygen
1 : intracellular oxygen
2 : oxygen gradient
3 : pressure
4 : volume
5 : contact with cancer cell
6 : contact with live cell
7 : contact with dead cell
8 : contact with basement membrane
9 : damage
10 : dead
11 : total attack time
12 : time
13 : custom:oncoprotein
14 : apoptotic
15 : necrotic


Behaviors:
=========
0 : oxygen secretion
```

# Functions to access signals

- `int find_signal_index( std::string signal_name );` get the index of the named signal
- `std::vector<int> find_signal_indices( std::vector<std::string> signal_names );` get a vector of indices for a vector of named signals
- `std::string signal_name( int i );` display the name of the signal with the given index
- `std::vector<double> get_signals( Cell* pCell );` get a vector of all known signals for the cell
- `std::vector<double> get_cell_contact_signals( Cell* pCell );` get a vector of the cell contact associated signals for the cell
- `std::vector<double> get_selected_signals( Cell* pCell , std::vector<int> indices );`

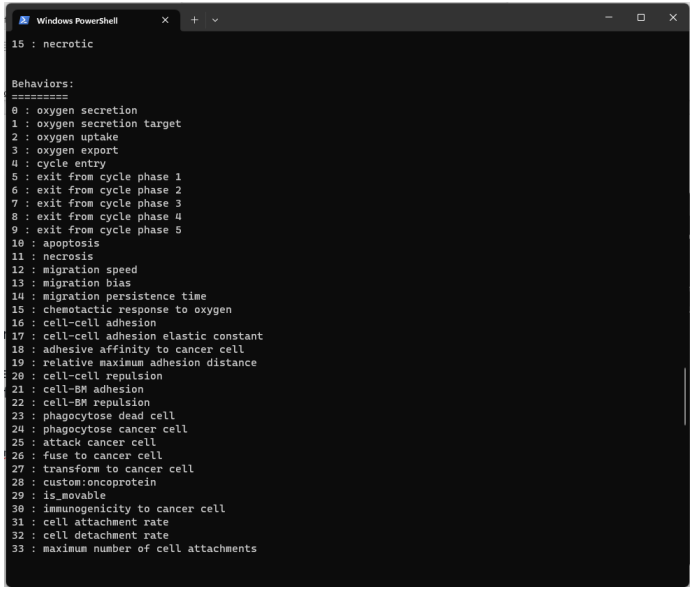
  get a vector of signals for the cell, with the supplied indices
- `std::vector<double> get_selected_signals( Cell* pCell , std::vector<std::string> names );`

  get a vector of signals for the cell, with the supplied human-readable names of the signals
- `double get_single_signal( Cell* pCell, int index );` get a single signal for the cell with the indicated index
- `double get_single_signal( Cell* pCell, std::string name );` get a single signal for the cell with the indicated human-readable name

# Behaviors Dictionary

- We introduced a "dictionary" of standard behaviors that can be used as outputs to intracellular and rule-based models. This dictionary is automatically constructed at the start of each simulation based upon the combinations of signaling substrates and cell types.

- secretion, secretion target, uptake, and export rates
- cycle progression
- attack rates
- fusion rates
- transformation rates
- Use display_behavior_dictionary() to quickly see a list of possible behaviors.



# get and set Functions for behavior

- `int find_behavior_index( std::string response_name ) ;` get the index of the named behavior
- `std::vector<int> find_behavior_indices( std::vector<std::string> behavior_names )`

  get the indices for the given vector of behavior names.
- `std::string behavior_name( int i );` get the name of the behavior with the given index
- `std::vector<double> create_empty_behavior_vector();` create an empty vector for the full set of behaviors
- `void set_behaviors( Cell* pCell , std::vector<double> parameters );`

  write the full set of behaviors to the cell's phenotype
- `void set_selected_behaviors( Cell* pCell , std::vector<int> indices , std::vector<double> parameters );`

  write the selected set of behaviors (with supplied indices) to the cell's phenotype
- `void set_selected_behaviors( Cell* pCell , std::vector<std::string> names , std::vector<double> parameters );`

  write the selected set of behaviors (with supplied names) to the cell's phenotype
- `void set_single_behavior( Cell* pCell, int index , double parameter );`

  write a single behavior (by index) to the cell phenotype
- `void set_single_behavior( Cell* pCell, std::string name , double parameter );`

  write a single behavior (by name) to the cell phenotype

# Example (interaction sampler project)

```
// contact with a differentiated cell reduces proliferation
// high rate of proliferation unless in contact with a differentiated cell
```

```
static double stem_cycling_halfmax = pCD->custom_data["cycling_contact_halfmax"]; // 0.1;

        base_val = pCD->phenotype.cycle.data.exit_rate(0); // 0.002;
        max_val = 0.0;
        signal = num_differentiated;
        half_max = stem_cycling_halfmax; //  0.1;
        hill = Hill_response_function( signal, half_max , 1.5 );
        phenotype.cycle.data.exit_rate(0) = base_val + (max_val-base_val)*hill;
```

# Best Practices

- In any customized cell function, you can access the microenvironment at its location.
- For best practices, you *don't* want to hard-code the index substrate. If somebody adds a substrate to the XML or reorders them, it could break your code.
- Instead, search for the index of the substrate and store the result in a static variable.
- Human readable names to access substrates is even better
- Dictionaries are recommended way to add behaviors to signals

# 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, 1818187)

## Training Materials:

- Administrative supplement to NCI U01CA232137 (Year 2)

## Other Funding:

- NCI / DOE / Frederick National Lab for Cancer Research (21X126F)
- DOD / Defense Threat Reduction Agency (HDTRA12110015)
- NIH Common Fund (3OT2OD026671-01S4)