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

# Advanced Session 2: Functions in PhysiCell

Heber L. Rocha
@HeberLRocha

## PhysiCell Project

August 7, 2023

# Goals

- Introduced the full modeling workflow
- Typical form / syntax / purpose of PhysiCell functions
- Learn about the available customizable functions in `cell.functions`
- Learn how to assign new functions to a cell definition
- Learn how to create add-on function

- **Examples:**
  - short example: update_phenotype function
  - full model: antitumor immunity model

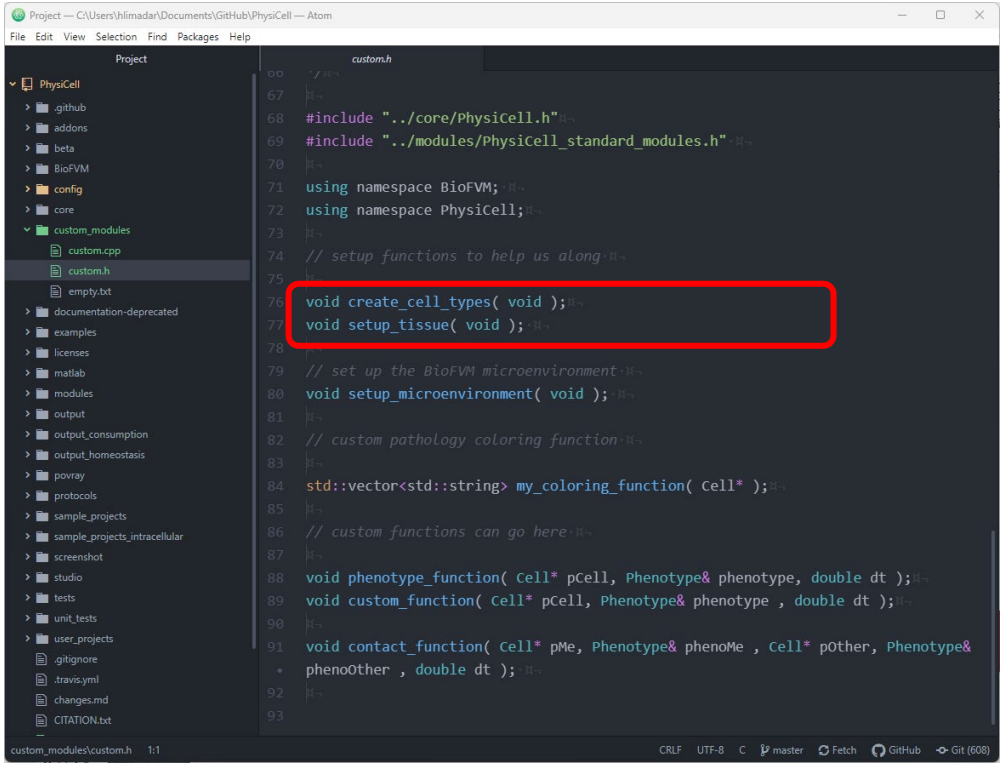
# Full modeling workflow

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

- Plan the model
- Populate a project
- Edit configuration file (.xml)
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - Add custom variables
  - Add custom parameters
- Define rules
- Edit initial cell placement
  - PhysiCell Studio (ICs tab)
  - Coding the `setup_tissue()` function
- **Edit custom modules**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- Build
- Run
- View results

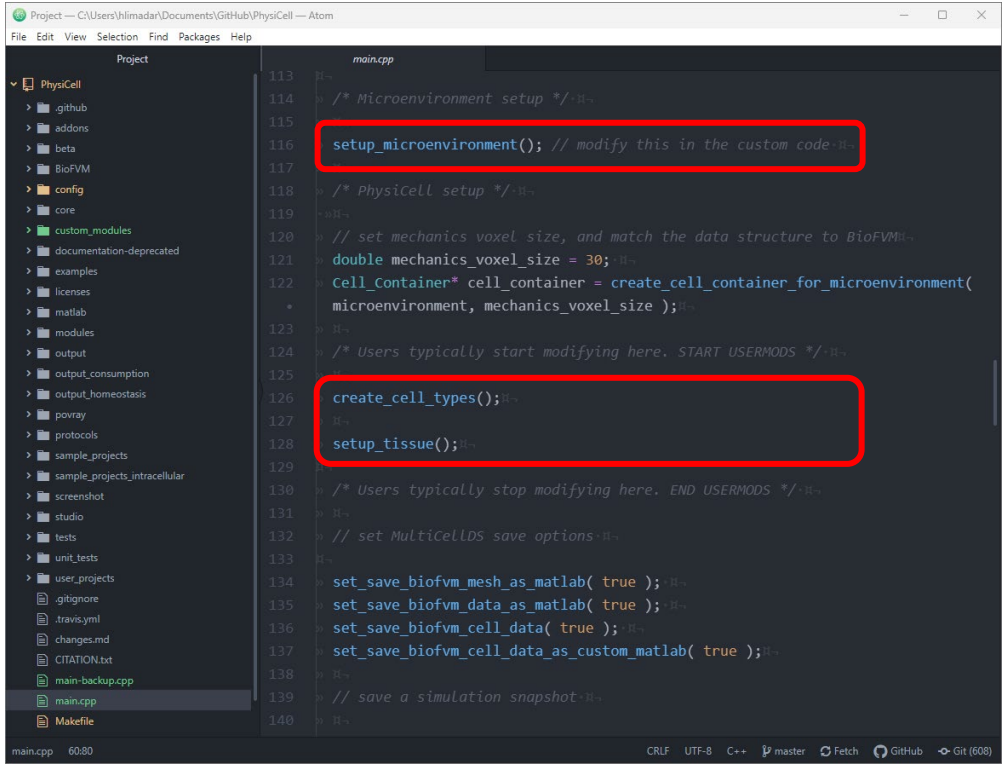
# Project structure: custom modules

- 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
        - ○ Assign functions
    - ♦ **setup_tissue()**
      - » Place initial cells in microenvironment
      - » Modify each cell as needed
  - Custom functions
  - any other modeling
  - Custom coloring functions



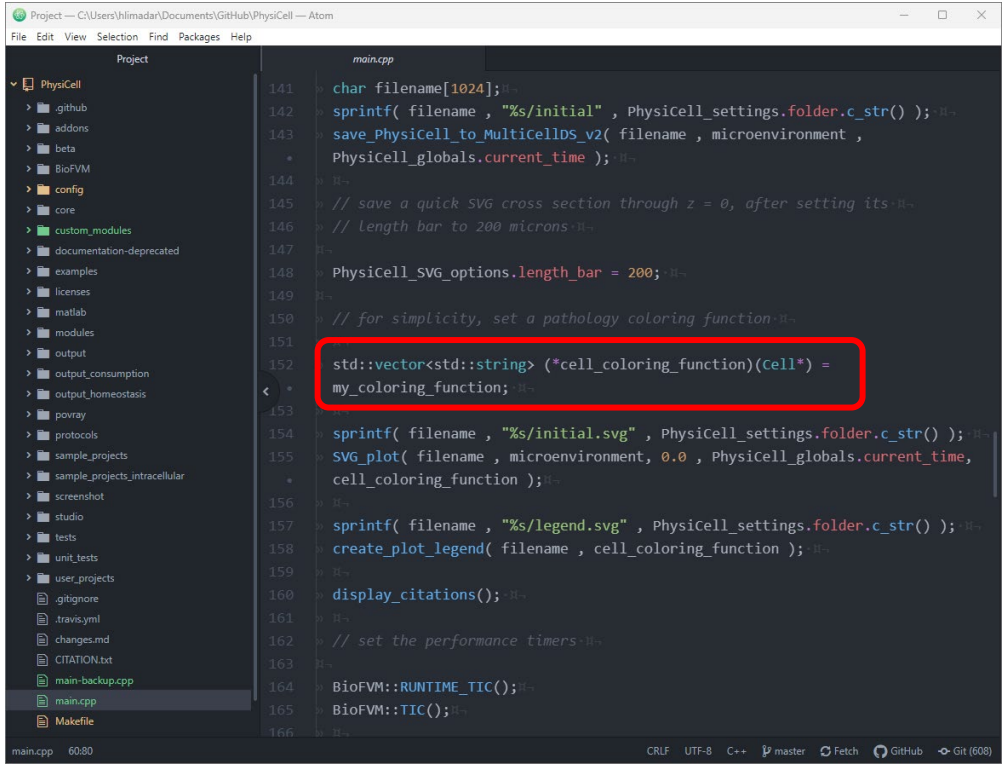
# Project structure: main.cpp

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



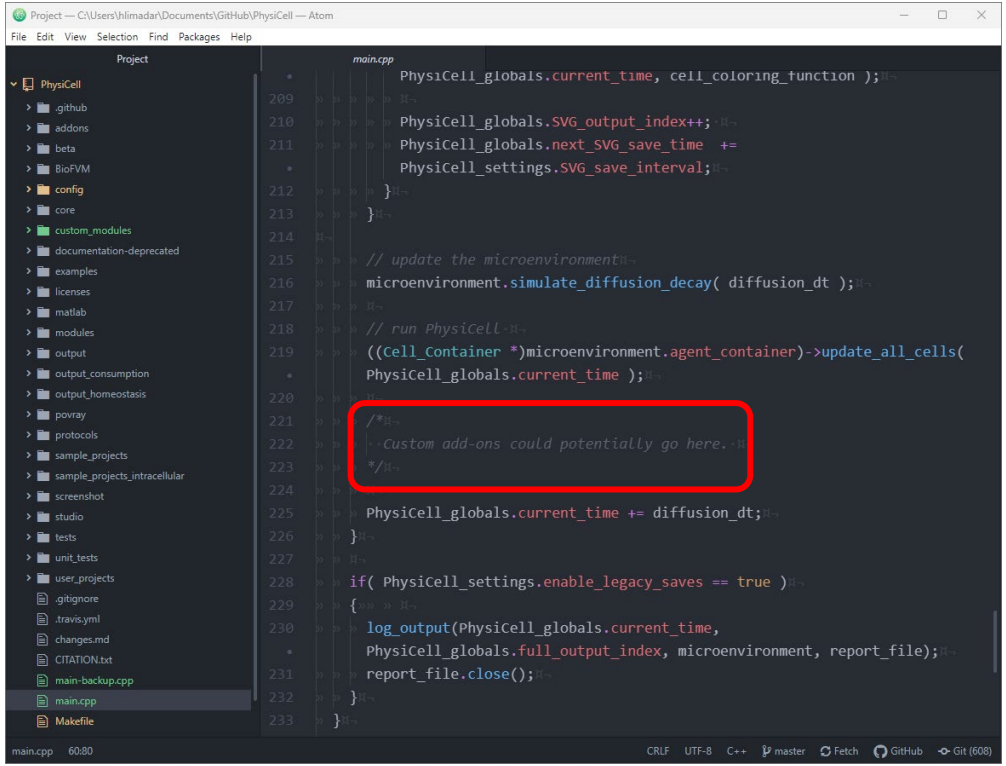
# Project structure: main.cpp

- main.cpp
  - Set coloring function (Advanced Session 3)



# Project structure: main.cpp

- main.cpp
  - Main loop:
    - This would be a good place to put extensions



# 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

# PhysiCell Cell Functions

# Cell functions

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**: `pCell->state`
- Cell **custom data :** `get_single_behavior( pCell, "custom:data_name" );`
  `set_single_behavior( pCell, "custom:data_name" );`
- Cell **functions :** `pCell->functions`
- Cell **phenotype :** `get_single_behavior( pCell, "behavior_name" );`
  `set_single_behavior( pCell, "behavior_name" , new_value);`
- Reference **phenotype:** `get_single_base_behavior( pCell, "behavior_name" );`
- nearby **microenvironment:**
  - `get_single_signal( pCell, "substrate_name");` extracellular value at cell location
  - `get_single_signal( pCell, "intracellular substrate_name");` intracellular value at cell location
  - `get_single_signal( pCell, "substrate_namegradient");` slope of substrate at cell location

# Cell functions (continued)

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)

# Purpose of the Cell Functions

- **volume_update_function**
  - Dynamically grow / shrink cells towards "target" values
- **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. (default 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. (default 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.
- **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.

# 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 = get_single_behavior( pCell, "custom:rate" );
      // change a cell's apoptosis rate
      set_single_behavior( pCell, "apoptosis", 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++ Functions

# Handy C++ tidbits: finding cell definitions

- `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.

# 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

# Handy C++ Tidbits: Random Numbers

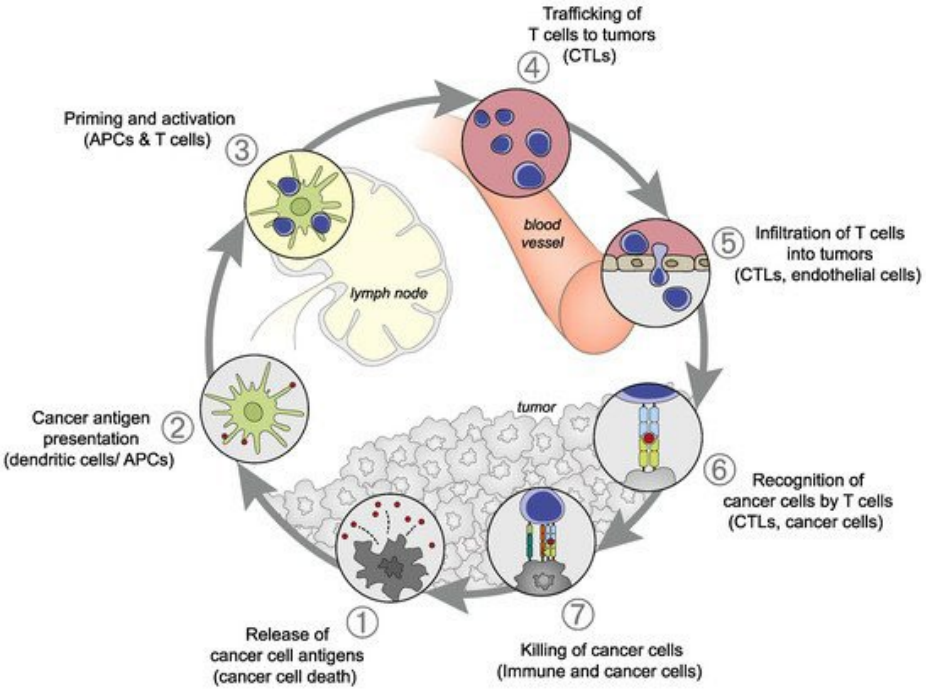
- `double UniformRandom( void );`
  - Get a uniformly distributed number in U(0,1)
- `double NormalRandom( double mean, double standard_deviation);`
  - Get a normally distributed number in N(mean, standard_deviation)
- `std::vector<double> UniformOnUnitCircle( void );`
  - Get a uniformly random point 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 $p0,p1,...,pn-1$, 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.**

# Full Model Workflow: Example

# Scenario: simple antitumor immunity

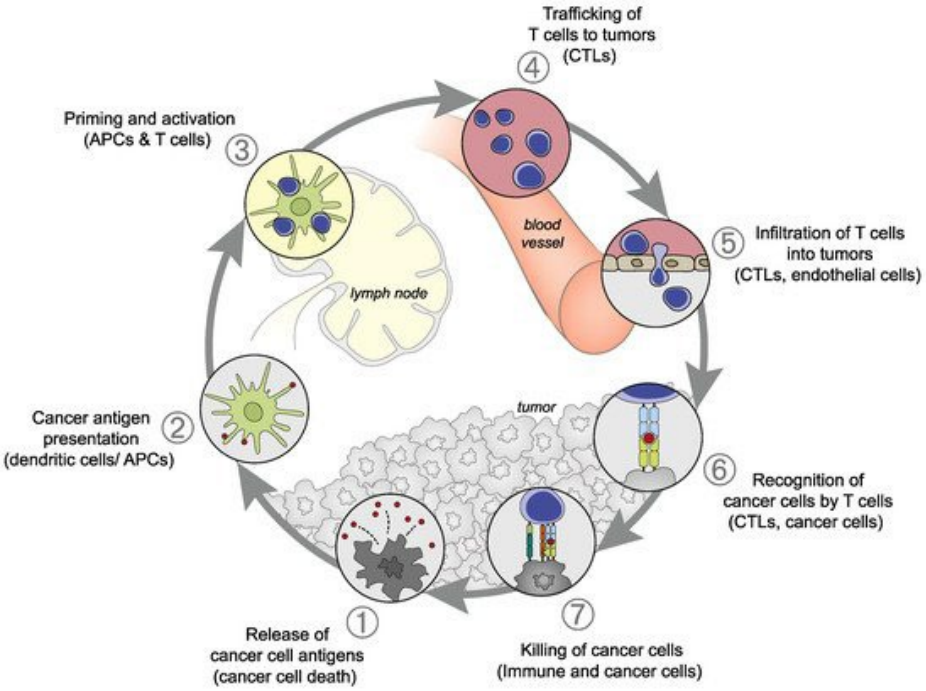
- Let's illustrate these with an example:
  - Cancer cells:
    - Mechanical pressure decrease cell cycle entry
    - Dead cells release debris
  - Antigen presenting cells:
    - Chemotaxis towards debris
    - Uptake debris
    - Activating APCs by contact with cancer cells
      - » increase cell cycle entry
      - » release pro-inflammatory factor
    - **Migration to lymph node (intravasation)**




Kunimasa, Kei, and Taichiro Goto. "Immunosurveillance and immunoediting of lung cancer: current perspectives and challenges." *International Journal of Molecular Sciences* 21.2 (2020): 597.

# Scenario: simple antitumor immunity

- Let's illustrate these with an example:
  - Effector T cells:
    - **Migration from lymph node (extravasation)**
    - Chemotaxis towards pro-inflammatory factor
    - Uptake pro-inflammatory factor
    - Attack cancer cells




Kunimasa, Kei, and Taichiro Goto. "Immunosurveillance and immunoediting of lung cancer: current perspectives and challenges." *International Journal of Molecular Sciences* 21.2 (2020): 597.

# Run the example

- Download (click here) and extract the project in `user_projects` folder
- Load and build
  - `make load PROJ=antiTumor_immunity`
  - `make`
- Open PhysiCell Studio
  - `python ../PhysiCell-model-builder/bin/ -c config/PhysiCell_settings.xml -e project.exe`
- Run and view results

# Digging into the code

- Phenotype function for APCs:
  - declared the function in `custom.h`
  - write the function in `custom.cpp`
    - getting intravasation rate
    - increment the intravasation count (needs to be thread-safe)
    - remove cell (needs to be thread-safe)
  - assign the function to cell definition:

**`create_cell_types()`**

```
Cell_Definition* pCD = find_cell_definition( "APC" );
pCD->functions.update_phenotype = APC_phenotype_function;
```

custom.cpp — C:\Users\hlimadar\Documents\GitHub\PhysiCell — Atom

File Edit View Selection Find Packages Help

Project: PhysiCell, .github, addons, beta, BioFVM, config (cell_rules.csv, cells.csv, PhysiCell_settings.xml, rules0.csv, template.xml), core (PhysiCell_basic_signaling.cpp, PhysiCell_basic_signaling.h, PhysiCell_cell_container.cpp, PhysiCell_cell_container.h, PhysiCell_cell.cpp, PhysiCell_cell.h, PhysiCell_constants.cpp, PhysiCell_constants.h, PhysiCell_custom.cpp, PhysiCell_custom.h, PhysiCell_digital_cell_line.cpp, PhysiCell_digital_cell_line.h, PhysiCell_phenotype.cpp, PhysiCell_phenotype.h, PhysiCell_rules.cpp, PhysiCell_rules.h, PhysiCell_signal_behavior.cpp, PhysiCell_signal_behavior.h, PhysiCell_standard_models.cpp, PhysiCell_standard_models.h)

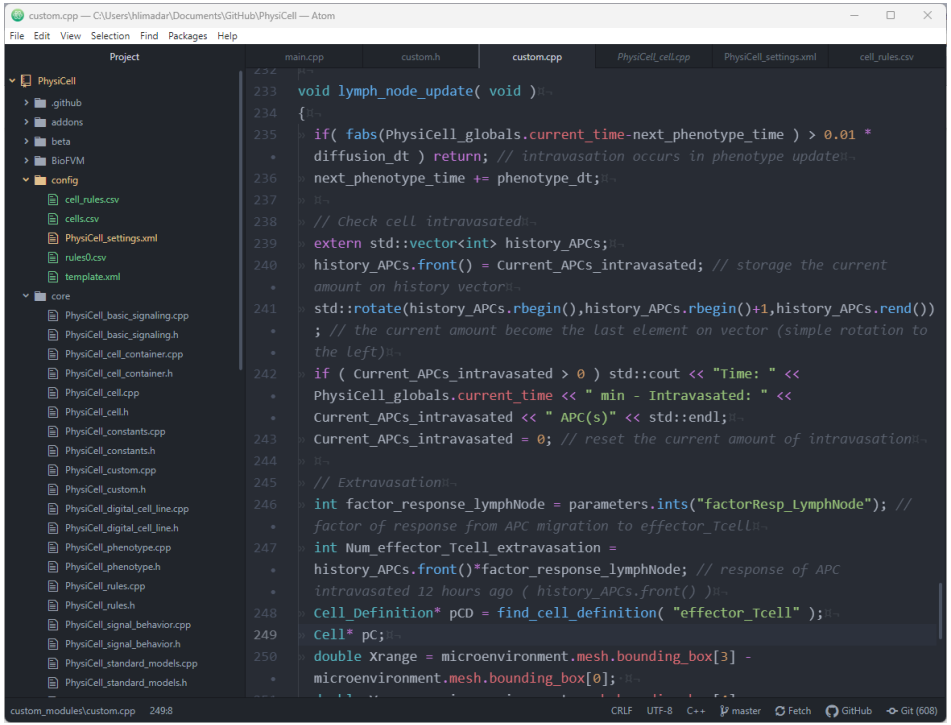
main.cpp | custom.h | custom.cpp | PhysiCell_cell.cpp | PhysiCell_settings.xml | cell_rules.csv

```
std::vector<std::string> my_coloring_function( Cell* pCell )
{ return paint_by_number_cell_coloring(pCell); }

void phenotype_function( Cell* pCell, Phenotype& phenotype, double dt )
{ return; }

void custom_function( Cell* pCell, Phenotype& phenotype , double dt )
{ return; }

void contact_function( Cell* pMe, Phenotype& phenoMe , Cell* pOther, Phenotype&
phenoOther , double dt )
{ return; }

void APC_phenotype_function( Cell* pCell, Phenotype& phenotype, double dt )
{
  double intravasation_rate = get_single_behavior( pCell,
  "custom:intravasation_rate");
  if (UniformRandom() < intravasation_rate*dt){
    #pragma omp critical
    { Current_APCs_intravasated++;}
    // things that have their own thread safety
    pCell->flag_for_removal();
    pCell->remove_all_attached_cells();
    pCell->remove_all_spring_attachments();
  }
  return;
}
```

custom_modules\custom.cpp 228:42 CRLF UTF-8 C++ master Fetch GitHub Git (608)

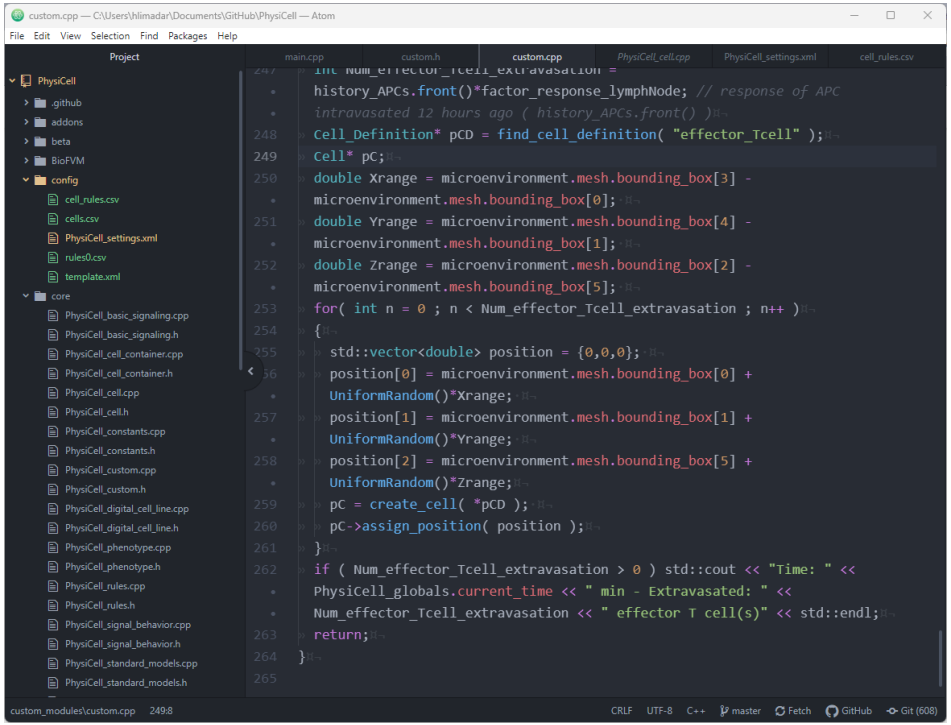
# Digging into the code (2)

- Lymph node update:
  - declared the function in `custom.h`
  - write the function in `custom.cpp`
    - update according to `phenotype_dt`
    - saving number of intravasated cells in history vector
    - check number of intravasated cells 12 hours ago (vector length * phenotype_dt)
    - multiply by the response factor
    - create effector T cells randomly in the domain (extravasation)
  - call the function in `main.cpp`



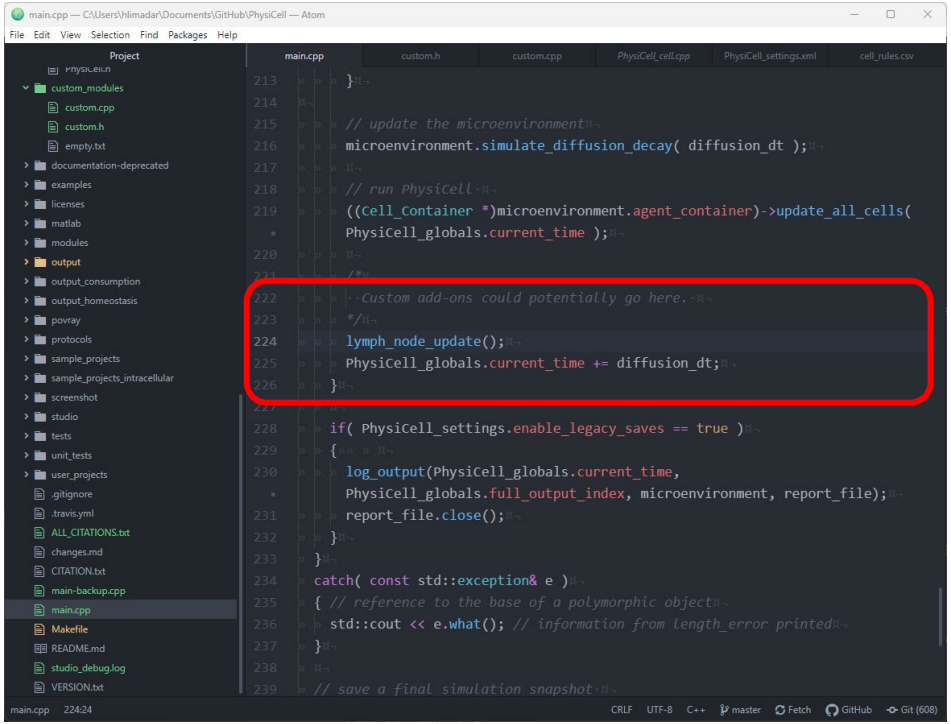
# Digging into the code (3)

- Lymph node update:
  - declared the function in `custom.h`
  - write the function in `custom.cpp`
    - update according to `phenotype_dt`
    - saving number of intravasated cells in history vector
    - check number of intravasated cells 12 hours ago (vector length * phenotype_dt)
    - multiply by the response factor
    - create effector T cells randomly in the domain (extravasation)
  - call the function in `main.cpp`



# Digging into the code (4)

- Lymph node update:
  - declared the function in `custom.h`
  - write the function in `custom.cpp`
    - update according to `phenotype_dt`
    - saving number of intravasated cells in history vector
    - check number of intravasated cells 12 hours ago (vector length * phenotype_dt)
    - multiply by the response factor
    - create effector T cells randomly in the domain (extravasation)
  - call the function in `main.cpp`



# 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)