

Slides, videos, links and more:

<https://github.com/physicell-training/ws2023>

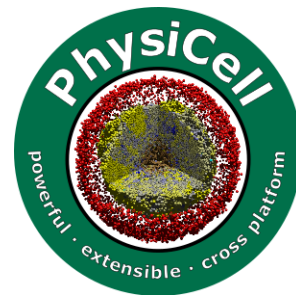
Advanced Session 2: Functions in PhysiCell

Heber L. Rocha

 @HeberLRocha

PhysiCell Project

August 7, 2023



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 @PhysiCell

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

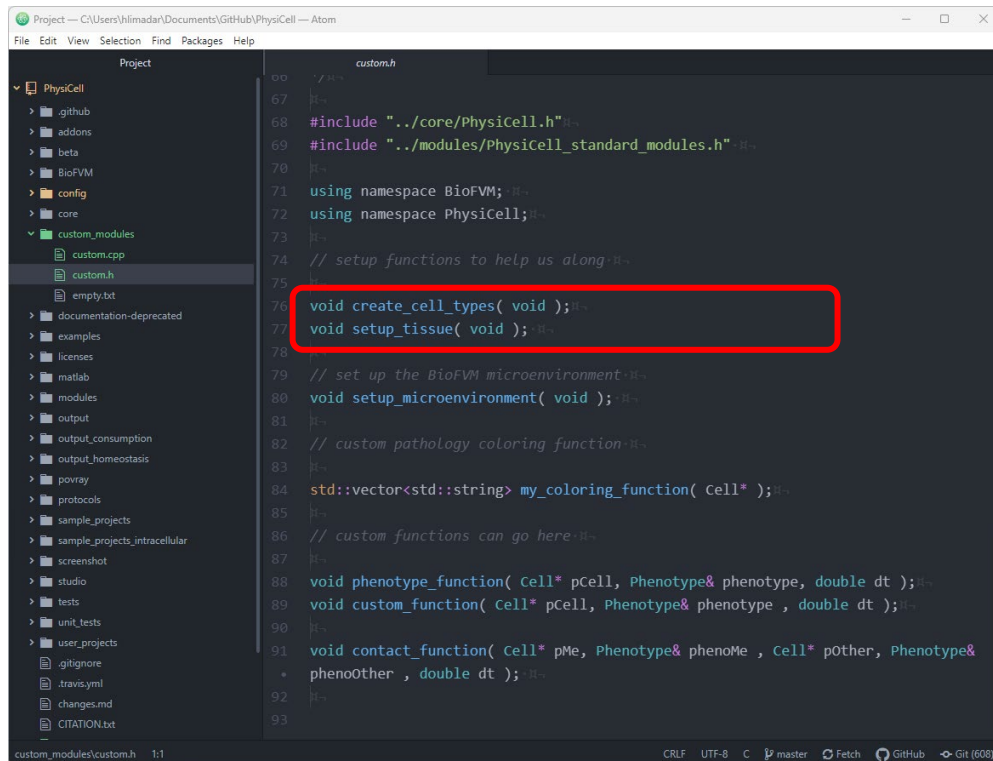
- ♦ **setup_tissue()**

- Place initial cells in microenvironment
 - Modify each cell as needed

- Custom functions

- any other modeling

- Custom coloring functions



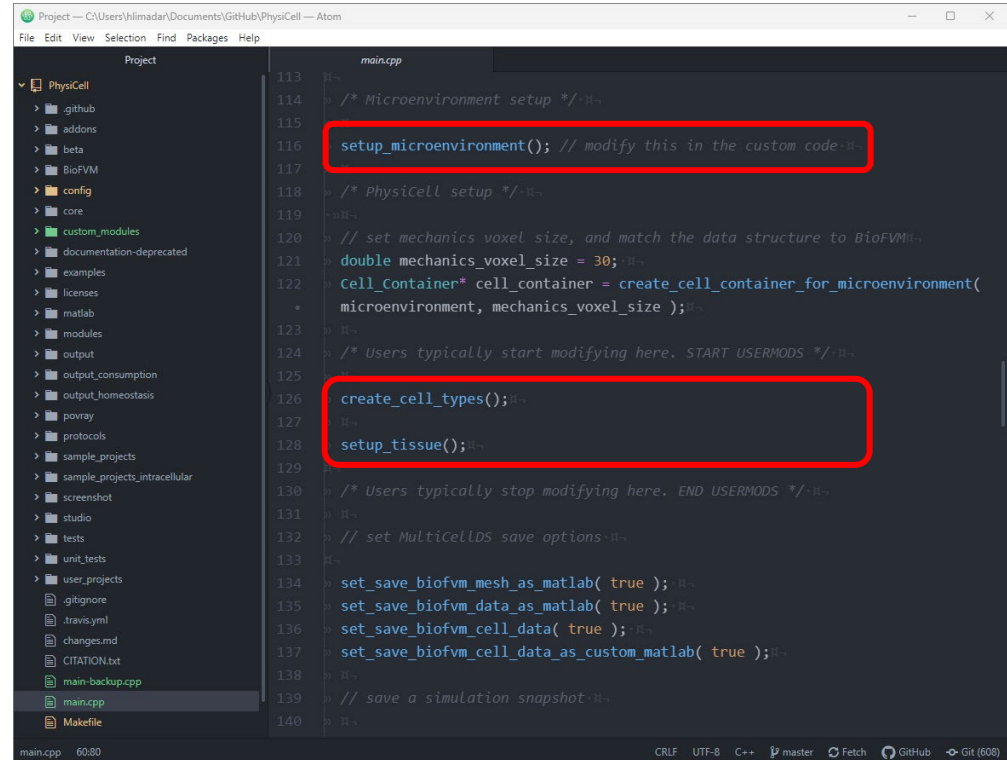
```
Project --- C:\Users\hlimadar\Documents\GitHub\PhysiCell --- Atom
File Edit View Selection Find Packages Help

Project
  PhysiCell
    .github
    addons
    beta
    BioFVM
    config
    core
    custom_modules
      custom.cpp
      custom.h
      empty.txt
    documentation-deprecated
    examples
    licenses
    matlab
    modules
    output
    output_consumption
    output_homeostasis
    povray
    protocols
    sample_projects
    sample_projects_intracellular
    screenshot
    studio
    tests
    unit_tests
    user_projects
    .gitignore
    .travis.yml
    changes.md
    CITATION.txt

custom.h
67 // ...
68 #include "../core/PhysiCell.h"
69 #include "../modules/PhysiCell_standard_modules.h"
70 // ...
71 using namespace BioFVM;
72 using namespace PhysiCell;
73 // ...
74 // setup functions to help us along:
75 // ...
76 void create_cell_types( void );
77 void setup_tissue( void );
78 // ...
79 // set up the BioFVM microenvironment:
80 void setup_microenvironment( void );
81 // ...
82 // custom pathology coloring function:
83 // ...
84 std::vector<std::string> my_coloring_function( Cell* );
85 // ...
86 // custom functions can go here:
87 // ...
88 void phenotype_function( Cell* pCell, Phenotype& phenotype, double dt );
89 void custom_function( Cell* pCell, Phenotype& phenotype, double dt );
90 // ...
91 void contact_function( Cell* pMe, Phenotype& phenoMe, Cell* pOther, Phenotype&
  * phenoOther, double dt );
92 // ...
93
```

Project structure: main.cpp

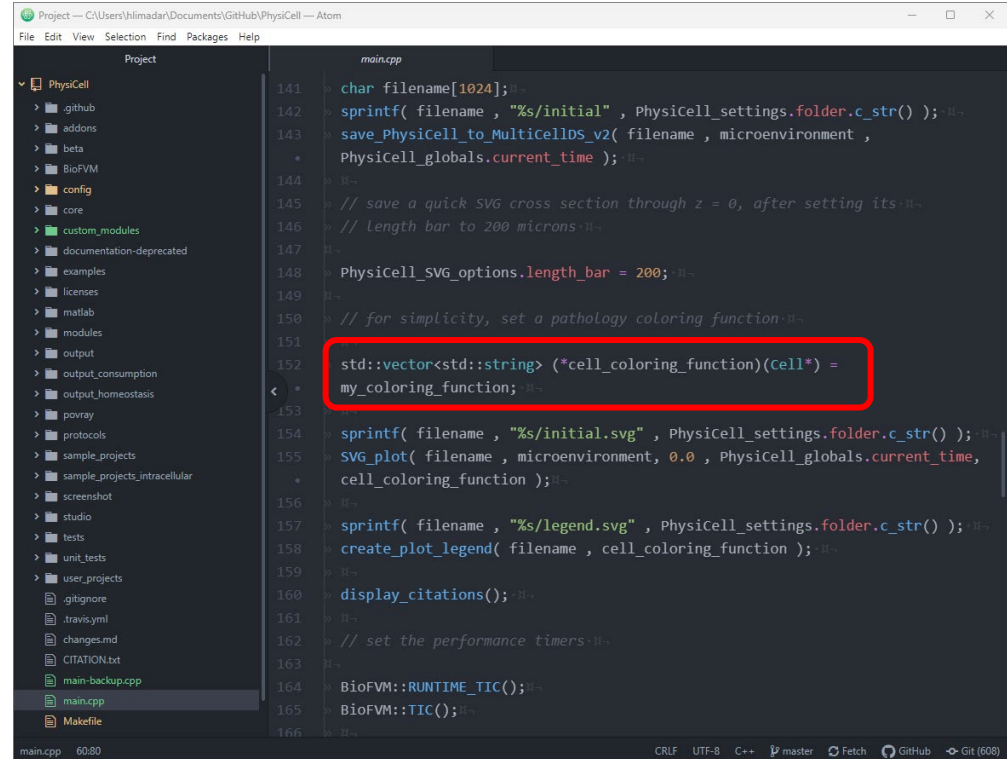
- `main.cpp`
 - (in the root directory)
 - Calls the setup functions



```
113 //
114 /* Microenvironment setup */
115
116 setup_microenvironment(); // modify this in the custom code
117
118 /* PhysiCell setup */
119
120 // set mechanics voxel size, and match the data structure to BioFVM
121 double mechanics_voxel_size = 30;
122 Cell_Container* cell_container = create_cell_container_for_microenvironment(
123     microenvironment, mechanics_voxel_size );
124
125 /* Users typically start modifying here. START USERMODS */
126
127 create_cell_types();
128
129 setup_tissue();
130
131 /* Users typically stop modifying here. END USERMODS */
132
133 // set MultiCellLDS save options
134
135 set_save_biofvm_mesh_as_matlab( true );
136 set_save_biofvm_data_as_matlab( true );
137 set_save_biofvm_cell_data( true );
138 set_save_biofvm_cell_data_as_custom_matlab( true );
139
140 // save a simulation snapshot
```

Project structure: main.cpp

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



```
Project — C:\Users\hlimadar\Documents\GitHub\PhysiCell — Atom
File Edit View Selection Find Packages Help

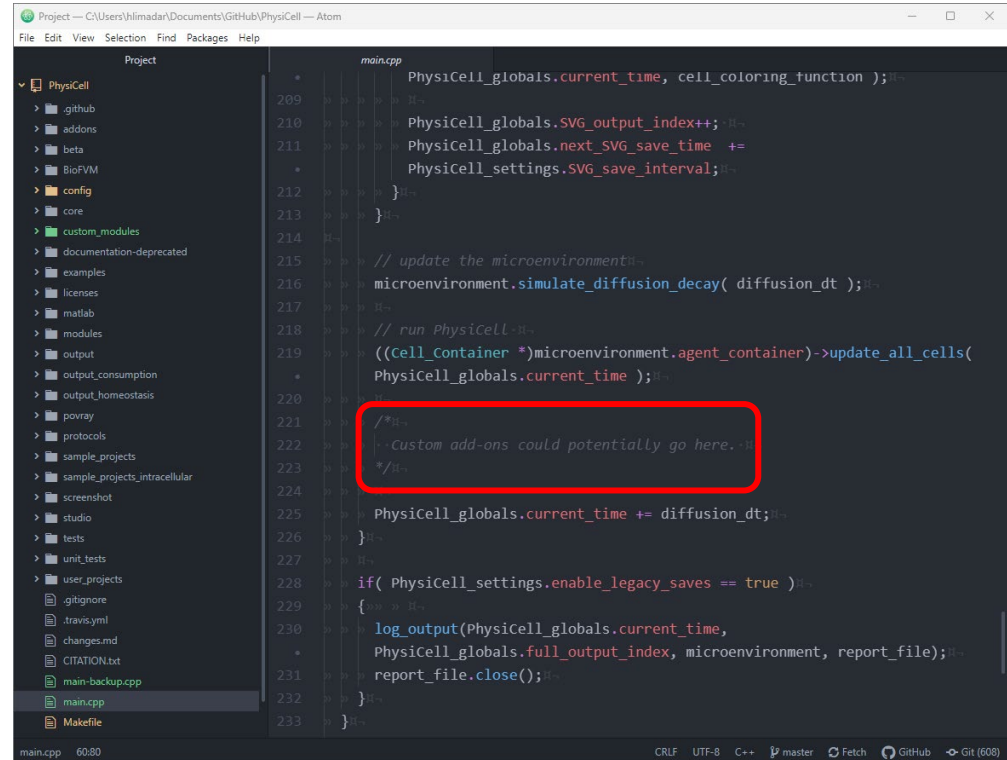
Project
  PhysiCell
    .github
    addons
    beta
    BioFVM
    config
    core
    custom_modules
    documentation-deprecated
    examples
    licenses
    matlab
    modules
    output
    output_consumption
    output_homeostasis
    povray
    protocols
    sample_projects
    sample_projects_intracellular
    screenshot
    studio
    tests
    unit_tests
    user_projects
    .gitignore
    .travis.yml
    changes.md
    CITATION.txt
    main-backup.cpp
    main.cpp
    Makefile

main.cpp 60/80

main.cpp
141 char filename[1024];
142 sprintf( filename , "%s/initial" , PhysiCell_settings.folder.c_str() );
143 save_PhysiCell_to_MulticellDS_v2( filename , microenvironment ,
  * PhysiCell_globals.current_time );
144
145 // save a quick SVG cross section through z = 0, after setting its
146 // length bar to 200 microns
147
148 PhysiCell_SVG_options.length_bar = 200;
149
150 // for simplicity, set a pathology coloring function
151
152 std::vector<std::string> (*cell_coloring_function)(cell*) =
  < my_coloring_function;
153
154 sprintf( filename , "%s/initial.svg" , PhysiCell_settings.folder.c_str() );
155 SVG_plot( filename , microenvironment, 0.0 , PhysiCell_globals.current_time,
  * cell_coloring_function );
156
157 sprintf( filename , "%s/legend.svg" , PhysiCell_settings.folder.c_str() );
158 create_plot_legend( filename , cell_coloring_function );
159
160 display_citations();
161
162 // set the performance timers
163
164 BioFVM::RUNTIME_TIC();
165 BioFVM::TIC();
166
```

Project structure: main.cpp

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



The screenshot shows the Atom code editor interface. On the left, the 'Project' sidebar displays a file tree for a directory named 'PhysiCell'. The tree includes folders like '.github', 'addons', 'beta', 'BioFVM', 'config', 'core', 'custom_modules', 'documentation-deprecated', 'examples', 'licenses', 'matlab', 'modules', 'output', 'output_consumption', 'output_homeostasis', 'povray', 'protocols', 'sample_projects', 'sample_projects_intracellular', 'screenshot', 'studio', 'tests', 'unit_tests', 'user_projects', and files like '.gitignore', '.travis.yml', 'changes.md', 'CITATION.txt', 'main-backup.cpp', 'main.cpp', and 'Makefile'. The 'main.cpp' file is selected and highlighted. The main editor area shows the code for 'main.cpp'. The code includes a loop that updates the microenvironment and runs the PhysiCell simulation. A red rectangle highlights a comment in the code: `/* Custom add-ons could potentially go here. */`. The status bar at the bottom indicates the file is 'main.cpp', 6080 lines long, and the editor is using 'CRLF' line endings, 'UTF-8' encoding, and 'C++' language.

```
209 // PhysiCell_globals.current_time, cell_coloring_function );
210 //
211 PhysiCell_globals.SVG_output_index++;
212 PhysiCell_globals.next_SVG_save_time +=
213     PhysiCell_settings.SVG_save_interval;
214 }
215 // update the microenvironment
216 microenvironment.simulate_diffusion_decay( diffusion_dt );
217 //
218 // run PhysiCell
219 ((Cell_container *)microenvironment.agent_container)->update_all_cells(
220     PhysiCell_globals.current_time );
221 //
222 /* Custom add-ons could potentially go here. */
223 //
224 PhysiCell_globals.current_time += diffusion_dt;
225 }
226 //
227 if( PhysiCell_settings.enable_legacy_saves == true )
228 {
229     log_output(PhysiCell_globals.current_time,
230         PhysiCell_globals.full_output_index, microenvironment, report_file);
231     report_file.close();
232 }
233 }
```

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



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

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



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 [@PhysiCell](https://twitter.com/PhysiCell)

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 $N(\text{mean}, \text{standard_deviation})$
- `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, \dots, 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.**

Full Model Workflow: Example



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Scenario: simple antitumor immunity

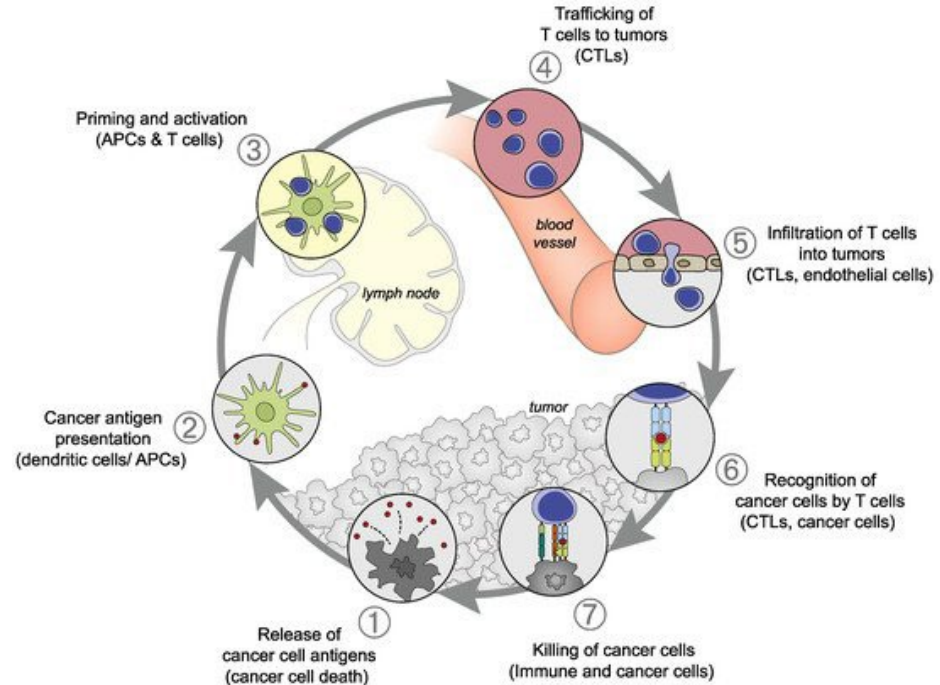
- Let's illustrate these with an example:

- 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)**



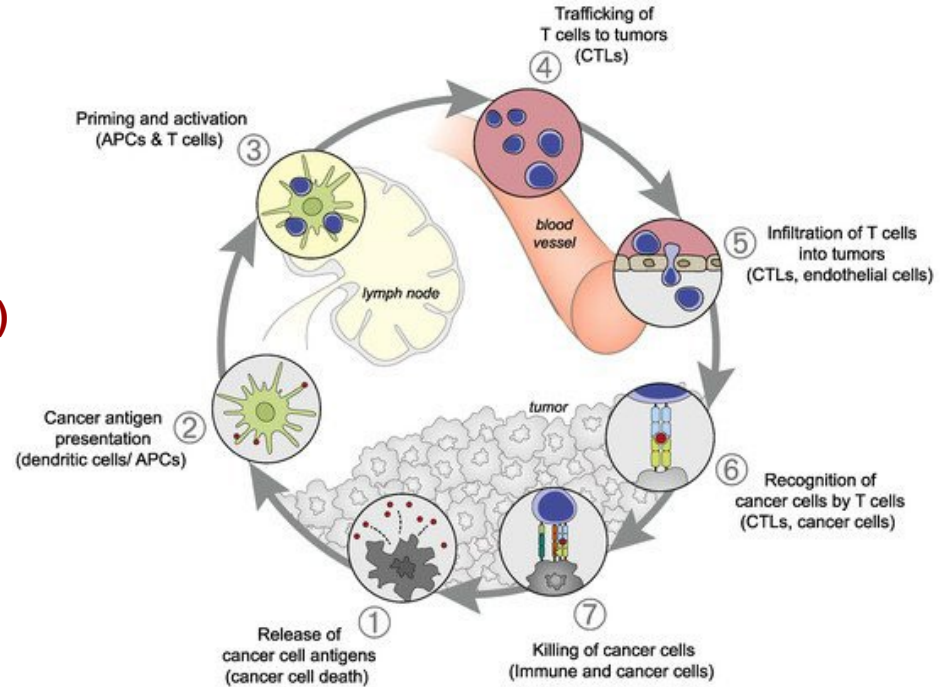
Kunimasa, Kei, and Taichiro Goto. "Immunosurveillance and immunoediting of lung cancer: current perspectives and challenges." *International Journal of Molecular Sciences* 21.2 (2020): 597.

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

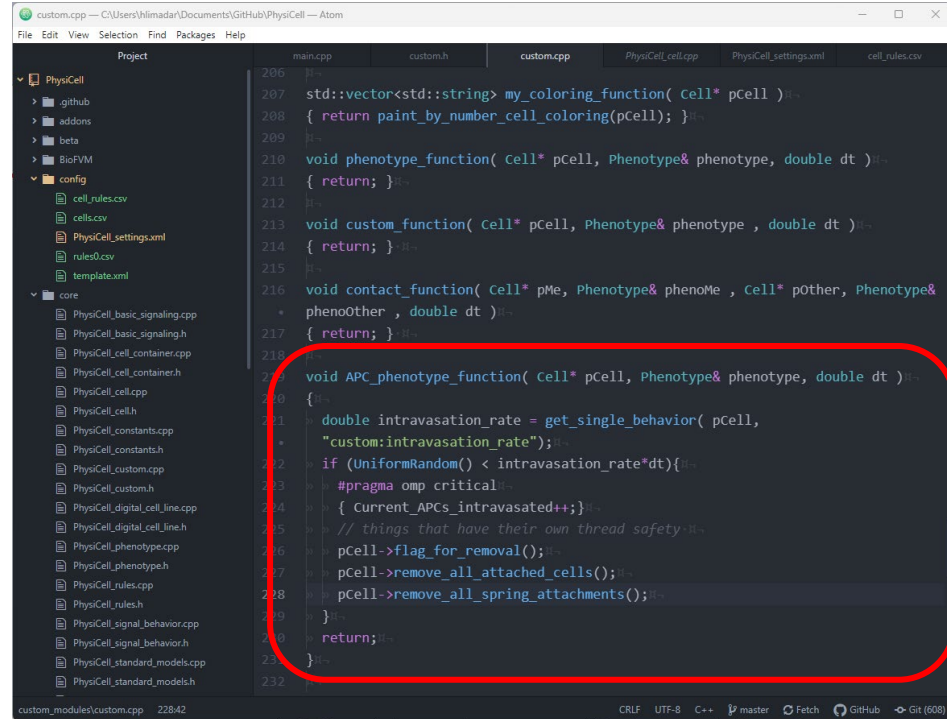
- Download ([click here](#)) and extract the project in `user_projects` folder
- Load and build
 - `make load PROJ=antiTumor_immunity`
 - `make`
- Open PhysiCell Studio
 - `python ../PhysiCell-model-builder/bin/ -c config/PhysiCell_settings.xml -e project.exe`
- Run and view results

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()`

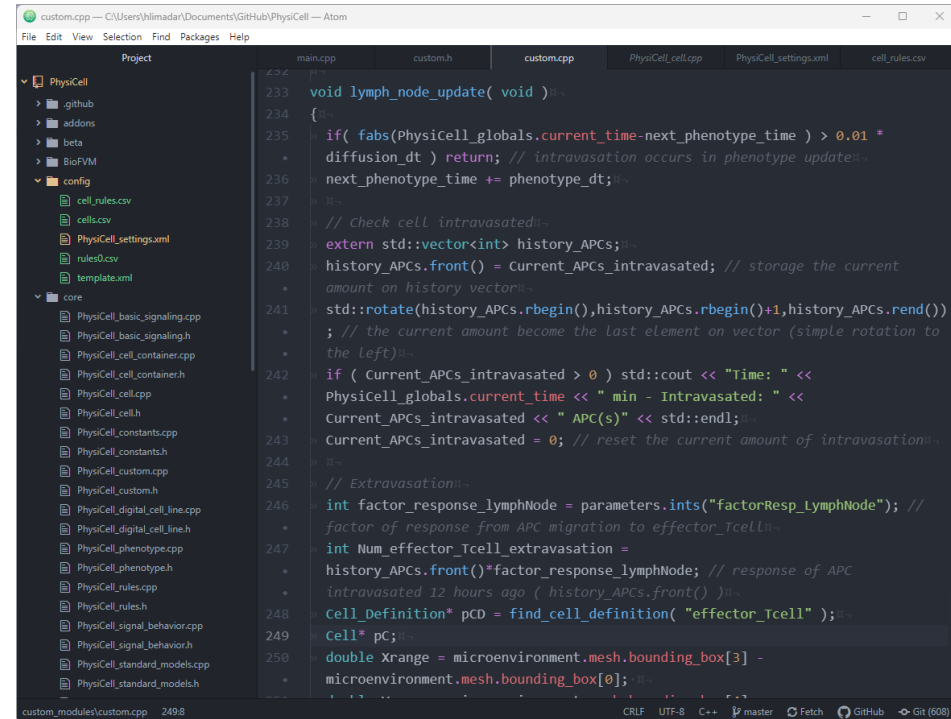
```
Cell_Definition* pCD = find_cell_definition( "APC" );  
pCD->functions.update_phenotype = APC_phenotype_function;
```



```
206  
207 std::vector<std::string> my_coloring_function( Cell* pCell )  
208 { return paint_by_number_cell_coloring(pCell); }  
209  
210 void phenotype_function( Cell* pCell, Phenotype& phenotype, double dt )  
211 { return; }  
212  
213 void custom_function( Cell* pCell, Phenotype& phenotype, double dt )  
214 { return; }  
215  
216 void contact_function( Cell* pMe, Phenotype& phenoMe, Cell* pOther, Phenotype&  
217 * phenoOther, double dt )  
218 {  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Digging into the code (2)

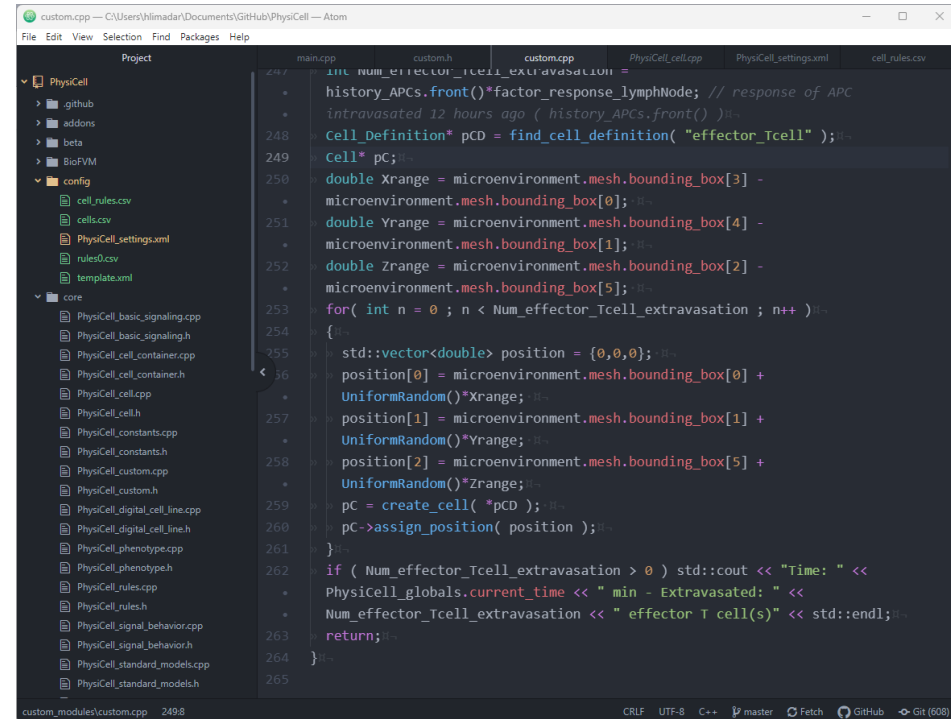
- Lymph node update:
 - declared the function in `custom.h`
 - write the function in `custom.cpp`
 - ♦ update according to `phenotype_dt`
 - ♦ saving number of intravasated cells in history vector
 - ♦ check number of intravasated cells 12 hours ago (vector length * `phenotype_dt`)
 - ♦ multiply by the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



```
233 void lymph_node_update( void ) {
234 {
235     if( fabs(PhysiCell_globals.current_time-next_phenotype_time ) > 0.01 *
        * diffusion_dt ) return; // intravasation occurs in phenotype update
236     next_phenotype_time += phenotype_dt;
237 }
238 // Check cell intravasated
239 extern std::vector<int> history_APCs;
240 history_APCs.front() = Current_APCs_intravasated; // storage the current
        * amount on history vector
241 std::rotate(history_APCs.rbegin(),history_APCs.rbegin()+1,history_APCs.rend())
        * ; // the current amount become the last element on vector (simple rotation to
        * the left)
242 if ( Current_APCs_intravasated > 0 ) std::cout << "Time: " <<
        * PhysiCell_globals.current_time << " min - Intravasated: " <<
        * Current_APCs_intravasated << " APC(s)" << std::endl;
243 Current_APCs_intravasated = 0; // reset the current amount of intravasation
244 }
245 // Extravasation
246 int factor_response_lymphNode = parameters.ints("factorResp_LymphNode"); //
        * factor of response from APC migration to effector_Tcells
247 int Num_effector_Tcell_extravasation =
        * history_APCs.front()*factor_response_lymphNode; // response of APC
        * intravasated 12 hours ago ( history_APCs.front() )
248 CellDefinition* pCD = find_cell_definition( "effector_Tcell" );
249 Cell* pC;
250 double Xrange = microenvironment.mesh.bounding_box[3] -
        * microenvironment.mesh.bounding_box[0];
```


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`

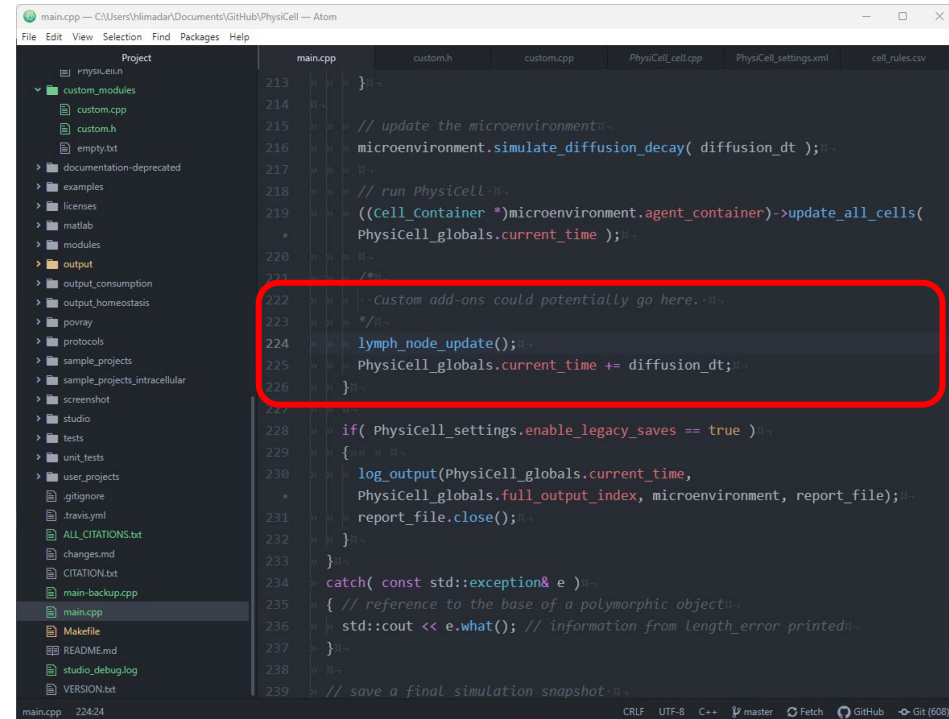


The screenshot shows the Atom code editor with the PhysiCell project open. The left sidebar displays the project structure, including folders like `github`, `addons`, `beta`, `BioFVM`, `config`, and `core`. The `core` folder is expanded, showing various source files. The main editor area displays the `custom.cpp` file, which contains C++ code for creating effector T cells. The code includes comments and function calls like `find_cell_definition`, `create_cell`, and `assign_position`. The status bar at the bottom indicates the file is `custom_modules/custom.cpp` at line 2498, with encoding set to UTF-8 and the C++ language mode selected.

```
2498 // response of APC
2499 intravasated 12 hours ago ( history_APCs.front() ):-
2500 cell_definition* pCD = find_cell_definition( "effector_Tcell" );
2501 cell* pC;
2502 double Xrange = microenvironment.mesh.bounding_box[3] -
2503               microenvironment.mesh.bounding_box[0];
2504 double Yrange = microenvironment.mesh.bounding_box[4] -
2505               microenvironment.mesh.bounding_box[1];
2506 double Zrange = microenvironment.mesh.bounding_box[2] -
2507               microenvironment.mesh.bounding_box[5];
2508 for( int n = 0 ; n < Num_effector_Tcell_extravasation ; n++ )
2509 {
2510     std::vector<double> position = {0,0,0};
2511     position[0] = microenvironment.mesh.bounding_box[0] +
2512                 UniformRandom()*Xrange;
2513     position[1] = microenvironment.mesh.bounding_box[1] +
2514                 UniformRandom()*Yrange;
2515     position[2] = microenvironment.mesh.bounding_box[5] +
2516                 UniformRandom()*Zrange;
2517     pC = create_cell( *pCD );
2518     pC->assign_position( position );
2519 }
2520 if ( Num_effector_Tcell_extravasation > 0 ) std::cout << "Time: " <<
2521     PhysiCell_globals.current_time << " min - Extravasated: " <<
2522     Num_effector_Tcell_extravasation << " effector T cell(s)" << std::endl;
2523 return;
2524 }
2525
```


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`



```
213 }
214
215 // update the microenvironment
216 microenvironment.simulate_diffusion_decay( diffusion_dt );
217
218 // run PhysiCell
219 ((Cell_Container *)microenvironment.agent_container)->update_all_cells(
220     PhysiCell_globals.current_time );
221
222 // Custom add-ons could potentially go here
223
224 lymph_node_update();
225 PhysiCell_globals.current_time += diffusion_dt;
226
227
228 if( PhysiCell_settings.enable_legacy_saves == true )
229 {
230     log_output(PhysiCell_globals.current_time,
231         PhysiCell_globals.full_output_index, microenvironment, report_file);
232     report_file.close();
233 }
234
235 catch( const std::exception& e )
236 { // reference to the base of a polymorphic object
237     std::cout << e.what(); // information from Length_error printed
238 }
239
240 // save a final simulation snapshot
```

Funding Acknowledgements



NATIONAL
CANCER
INSTITUTE



leidos



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)