PhysiCell Tutorial: A first dive into PhysiCell

Paul Macklin, Ph.D.

Intelligent Systems Engineering
Indiana University

March 6, 2023

Thank you to our sponsors!

- Jayne Koskinas Ted Giovanis Foundation for Health and Policy
- National Cancer Institute (U01CA232137)
- Administrative supplement to NCI U01CA232137 (Year 2)
- National Science Foundation (1720625, 1818187)
- NCI / DOE / Frederick National Lab for Cancer Research (21X126F)
- DOD / Defense Threat Reduction Agency (HDTRA12110015)
- NIH Common Fund (3OT2OD026671-01S4)

Goals

- Introduce key structures in PhysiCell
- Introduce three model workflows
  - Basic
  - Intermediate
  - Full
- Introduce PhysiCell studio

- ***Basic workflow:*** Run and visualize a built-in sample project
- ***Intermediate workflow:*** Build, run, and visualize a predator-prey model

Key Background

Sample and Template Projects

- Sample projects are pre-built projects that are bundled with PhysiCell

- They are accessed via **key makefile rules:**
  - make list-projects — get a list of bunded projects
  - make project_name — populate a sample project
  - make — compile the project
  - make data-cleanup — clean up date for another run
  - make reset — clear out the project to try another

- The **template** project is a good starting point for 2D and 3D projects.

New in PhysiCell 1.11.0-beta: user projects

- User projects can now readily be saved and retrieved in your PhysiCell download

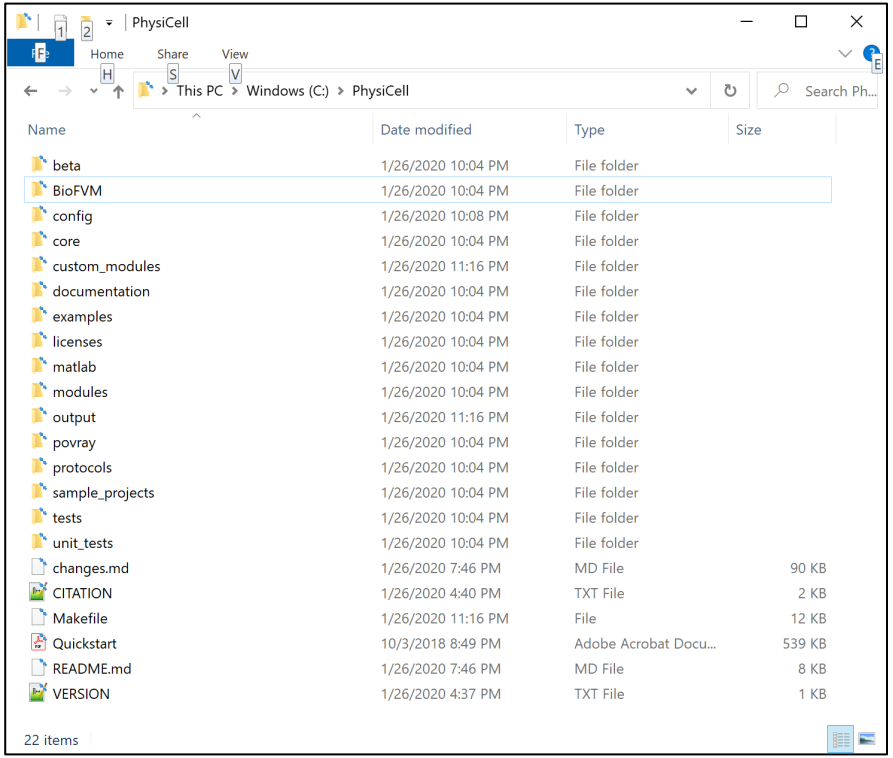
  - make save PROJ=[name]
    - save the following in ./user_projects/[name]:
      - ./config/*
      - main.cpp
      - Makefile
      - ./custom_modules/*

  - make load PROJ=[name]
    - load the following from ./user_projects/[name]:
      - ./config/*
      - main.cpp
      - Makefile
      - ./custom_modules/*

Project directory structure

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

Most of your work will be in the red directories



Cells (1)

- Cells are the key entity in PhysiCell.

- Each cell keeps track of:
  - Type
  - ID
  - Position and velocity
  - State
  - Phenotype
    - Intracellular model and data are included here.
  - Custom data

Cells (2)

- Cells have built-in techniques for:
  - Division
  - Death
  - Changing type
  - Accessing / sampling the microenvironment
  - Secretion
  - Finding nearby cells
  - Mechanics
  - Ingesting, damaging, and fusing with other cells
  - And more behaviors via phenotype

Key cell information

- Each cell agent is a member of the `Cell` class.
- Some key data:
  - `std::string type_name` // human-readable name of cell type
  - `int type` // machine-readable unique integer identifier for cell type
  - `int ID` // cell agent's unique integer identifier. (different for each cell)
  - `std::vector<double> position` // the cell's current position (**never write this!**)
  - `std::vector<double> velocity` // the cell's current velocity
  - `cell_state` // things like size, pressure, and cells in contact
  - `phenotype` // behavioral properties / state
  - `custom_data` // custom scalar and vector data
  - `functions` // list of key cell functions

Cell state

- Each **Cell** has an instance of **Cell_State** called **state:**
  - std::vector<Cell*> attached_cells:
    - Use `attach_cell` and `detach_cell` to add or remove cells to this list
    - Cell-cell contact functions are automatically evaluated for these cells
  - std::vector<Cell*> neighbors:
    - a vector of pointers to all (mechanically interacting) neighbor cells.
    - Automatically updated to include all cells within mechanical interaction distance
  - double simple_pressure:
    - a (normalized) measure of forces exerted by nearby adhered cells
    - in 3-D, fully confluent (packed) tissue, 12 neighbors, and simple_pressure = 1
    - in 2-D, fully confluent (packed) tissue, 6 neighbors, and simple_pressure = 0.5

Cell phenotype

- One of the most critical data elements in a PhysiCell Cell is ***phenotype***
- Hierarchically organize key behavioral elements:
  - Phenotype
    - **cycle**: advancement through a cell cycle model
    - **death**: one or more types of cell death
    - **volume:** cell's volume regulation
    - **geometry:** cell's radius and surface area
    - **mechanics:** adhesion and resistance to deformation ("repulsion")
    - **motility:** active motion (other than "passive" mechanics)
    - **secretion:** both release and uptake of chemical substrates. Interfaces with BioFVM
    - **molecular:** a place to store internalized substrates
    - **intracellular:** a place for intracellular models
    - **interactions:** cell-cell contact interactions & transformations

Phenotype-centric programming

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

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

Cell Definitions

- A **Cell Definition** is a convenient way to set the parameters and functions for a whole class of cells
  - Users can instantiate cells of a specific type via `create_cell( a_cell_defn )`
  - With no argument, new cells use the `cell_defaults` definition
    - For historical reasons, PhysiCell uses the first cell definition (with index 0) as its default

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

Modeling Workflows

PhysiCell Modeling Workflows

- There are three typical modeling workflows in PhysiCell
  - **Basic**
    - ♦ Build existing projects, change parameter values, and run
  - **Intermediate**
    - ♦ Build your own models based on the template project
    - ♦ All model setup in a GUI (no modification of C++)
  - **Full**
    - ♦ Enhance an intermediate model with custom C++ to implement cell hypotheses / rules

Basic modeling workflow

Basic modeling workflow

Suitable for running a built-in project with minor changes to parameters.

- Populate and build a project
- Edit settings
- Run
- View results

Choose, populate, and build a project

- Get list of sample projects:
  - `make list-projects`
- Populate the heterogeneity sample:
  - `make heterogeneity-sample`
- Compile the project
  - `make`

Edit settings

- Open the settings file:
  - **./config/PhysiCell_settings.xml**

- Let's change:
  - Change domain to [-500,500] × [-500,500]
  - Reduce max simulation time to 2160 minutes
  - Save full data ever 360 minutes
  - Set oncoprotein standard deviation to 3 (increase heterogeneity)
  - Set the max oncoprotein value to 10 (mean + 3 standard deviations)

Edit settings: XML

- Open ./config/PhysiCell_settings.xml

- Major sections:
  - **domain** -- how big of a region to simulate
  - **overall** -- how long to simulate, time step sizes
  - **parallel** -- OpenMP settings
  - **save** -- how often to save SVG images and full data
  - **microenvironment** -- settings on diffusing substrates
  - **user_parameters** -- model-specific settings
  - **cell_definitions** -- set baseline cell properties

Edit settings: Domain size

- Open `./config/PhysiCell-settings.xml`
- Let's set the domain size in the **domain** block
  - Switch to [-500,500] x [-500,500] x [-10,10] to speed it up

```
<PhysiCell_settings version="devel-version">
        <domain>
                <x_min>-500</x_min>
                <x_max>500</x_max>
                <y_min>-500</y_min>
                <y_max>500</y_max>
                <z_min>-10</z_min>
                <z_max>10</z_max>
                <dx>20</dx>
                <dy>20</dy>
                <dz>20</dz>
                <use_2D>true</use_2D>
        </domain>
```

Edit settings: Save settings

- Let's look at the **overall** block
  - Set max time to 1.5 days = 1.5 x 24 x 60 = 2160 minutes

```
<overall>
        <max_time units="min">2160</max_time> <!-- 36 h * 60 min -->
        <time_units>min</time_units>
        <space_units>micron</space_units>
```

- Let's look at the **save** block
  - Set the full save interval to 6 hours = 360 minutes

```
<save>
        <folder>output</folder> <!-- use . for root -->
        <full_data>
                <interval units="min">360</interval>
                <enable>true</enable>
        </full_data>
```

Edit settings: User parameters

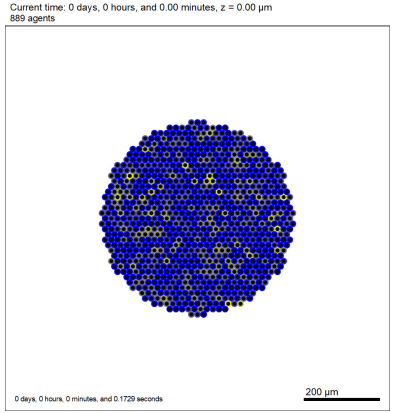
- Let's also look at the **user_parameters** block
  - Let's change the oncoprotein standard deviation (`oncoprotein_sd`) to 3 (more variation)
  - Let's change the max oncoprotein (`oncoprotein_max`) to mean + 3 sds = 1 + 9 = 10

```
<user_parameters>
  <tumor_radius type="double" units="micron">250.0</tumor_radius>
  <oncoprotein_mean type="double" units="dimensionless">
    1.0</oncoprotein_mean>
  <oncoprotein_sd type="double" units="dimensionless">3.0</oncoprotein_sd>
  <oncoprotein_min type="double" units="dimensionless">0.0</oncoprotein_min>
  <oncoprotein_max type="double" units="dimensionless">10</oncoprotein_max>
  <random_seed type="int" units="dimensionless">0</random_seed>
</user_parameters>
```

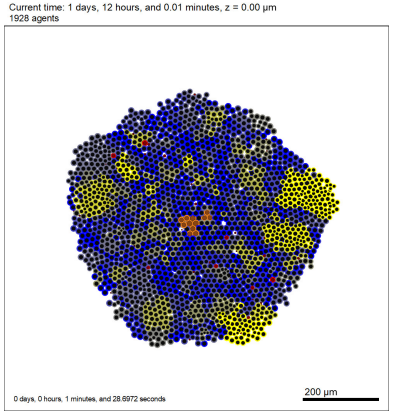
Run and View Results: Snapshots

- run:
  - **./heterogeneity** (MacOS or Linux)
  - **heterogeneity.exe** (Windows)

- Look in output:
  - Look at snapshot SVG files
  - Look at **legend.svg**
    - (Not much to see on this example)



initial.svg



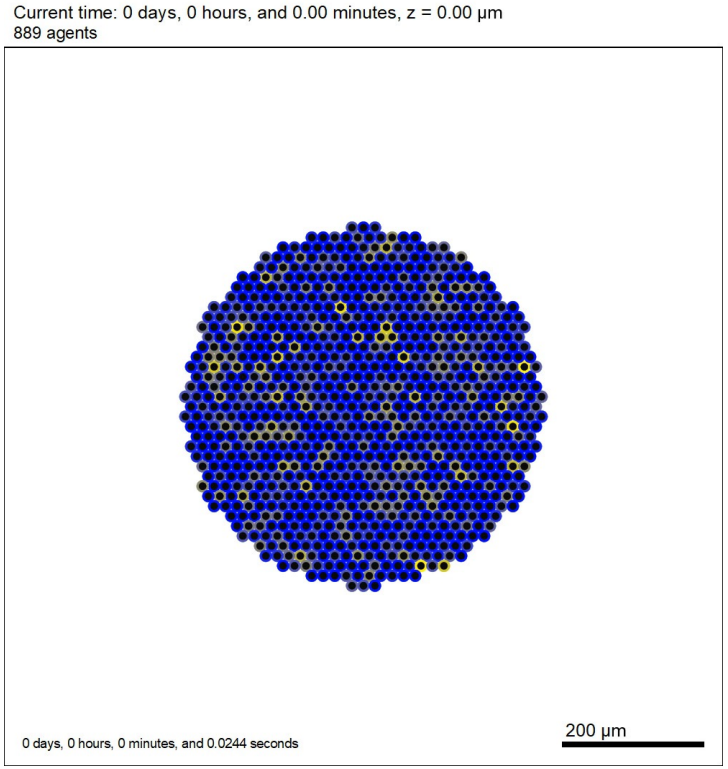
final.svg

legend.svg

- Convert snapshots to JPEG:
  - **`make jpeg`** (results: output/snapshot00000000.jpg ...)

View results: GIF and movie

- Make and animated GIF:
  - **`make gif`** (result: output/out.gif)
- Make an mp4 movie
  - **`make movie`** (result: output/out.mp4)



Model builder GUI

Model Builder GUI

- Purpose:
  - Simple and robust editing of diffusing substrates and cell definitions
  - Easily change parameters, run, and visualize results.

- Requires:
  - Python (with Qt support)

- Recommended installation:
  - Source: https://github.com/PhysiCell-Tools/PhysiCell-model-builder
  - Unzip Python-Model-Builder alongside your PhysiCell root directory
    - some_directory
      - PhysiCell
      - PhysiCell-Model-Builder
  - **I'll include a copy in our Course GitHub whenever needed.**

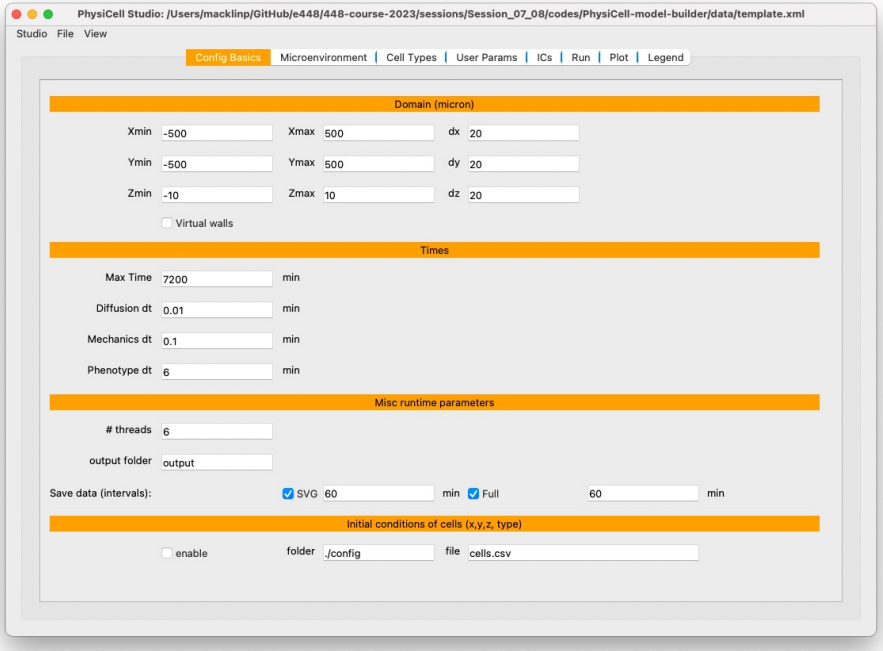
- To open
  - In terminal (in PhysiCell root directory)

```
python ../PhysiCell-model-builder/bin/pmb.py --studio &
```

PhysiCell model builder

A graphical user interface (GUI) application to make it easier to create and edit a PhysiCell (XML) model.

- **Config basics:**
  - Domain size, Simulation duration, Data output
- **Microenvironment:**
  - Define diffusing substrates and boundary conditions
- **Cell types:**
  - Define cell types, including their base phenotypes (behaviors)
- **User params:**
  - Model-specific parameters
- **Run:**
  - Use this to start executing the model
- **ICs:**
  - An optional place to draw initial cell positions
- **Plot:**
  - Plot cells and diffusible substrates
- **Legend:**
  - Define the cell colors

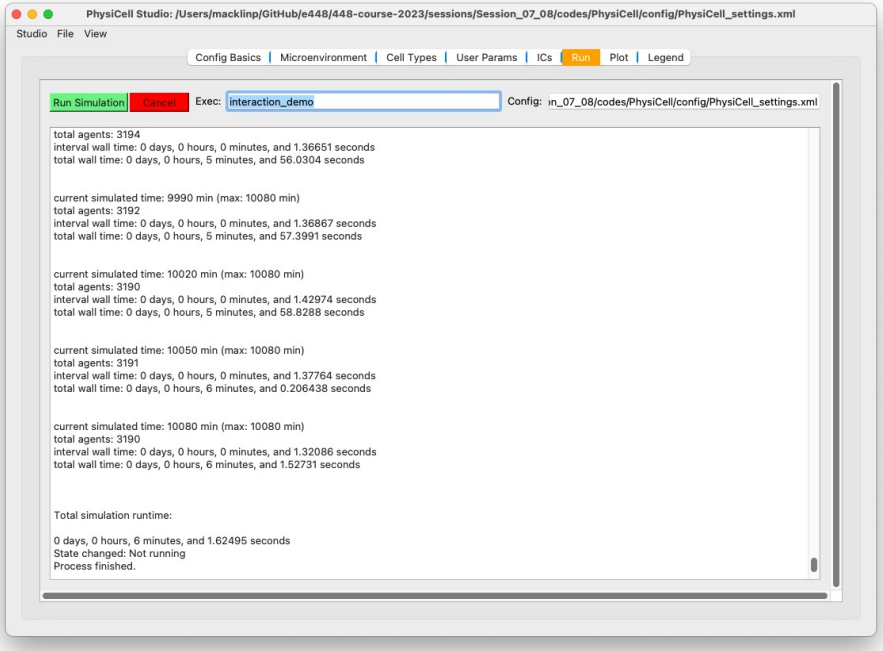


https://github.com/PhysiCell-Tools/PhysiCell-model-builder

Let's use the GUI for the *interaction* sample

- In a terminal:
  - **`make reset`** // get back to clean slate
  - **`make interaction-sample`** // populate project
  - **`make`** // compile
    - note: executable name is **`interaction_demo`**

- In the GUI:
  - Load the config file
    - File → Load → browse to **`PhysiCell_settings.xml`**
  - Set the executable name
    - Go to "Run" tab
    - In the "Exec" field replace with **`interaction_demo`**
  - Run!
    - Click the green **"Run Simulation"** button
  - Visualize
    - Go to the **"Plot"** tab and explore!



Intermediate modeling workflow

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

- **Plan the model**
- Populate the template project
- **Edit configuration with the PhysiCell Model Builder GUI**
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
- **Edit initial cell placement**
- Build
- Run
- View results

Plan the model

Predator-Prey

- Prey:
  - secrete ”prey” signal (will act as a quorum factor)
  - Proliferate
  - No death
  - Chemotaxis away from predators
  - Chemotaxis towards prey to aggregate

- Predators:
  - Secrete “predator” signal
  - Proliferate (100x slower), no death
  - Chemotaxis towards prey (double speed)
  - Phagocytose prey (rate 0.01)

Start with template project

Prepare a clean slate

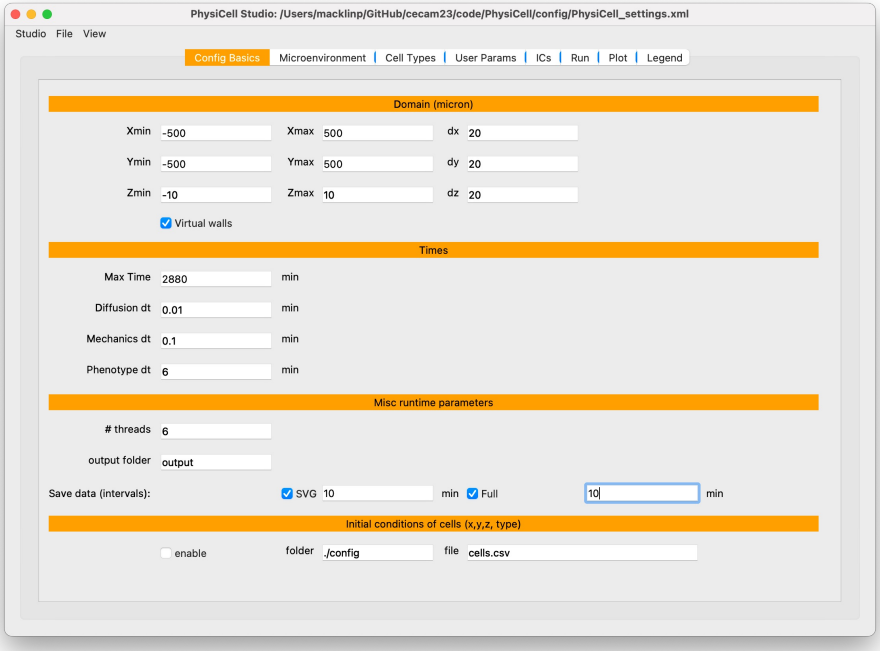
- In terminal window:
  - make data-cleanup // cleanup date a
  - make reset // return to blank slate
  - make template // populate the template project
  - make // compile
    - note: our executable name is project

- In GUI:
  - Load the PhysiCell_settings.xml as before

Set up time and other parameters

- Set the max simulation time to 2 days (2880 minutes)
  - Click the “config basics” tab
  - Set the “max time” to 2880

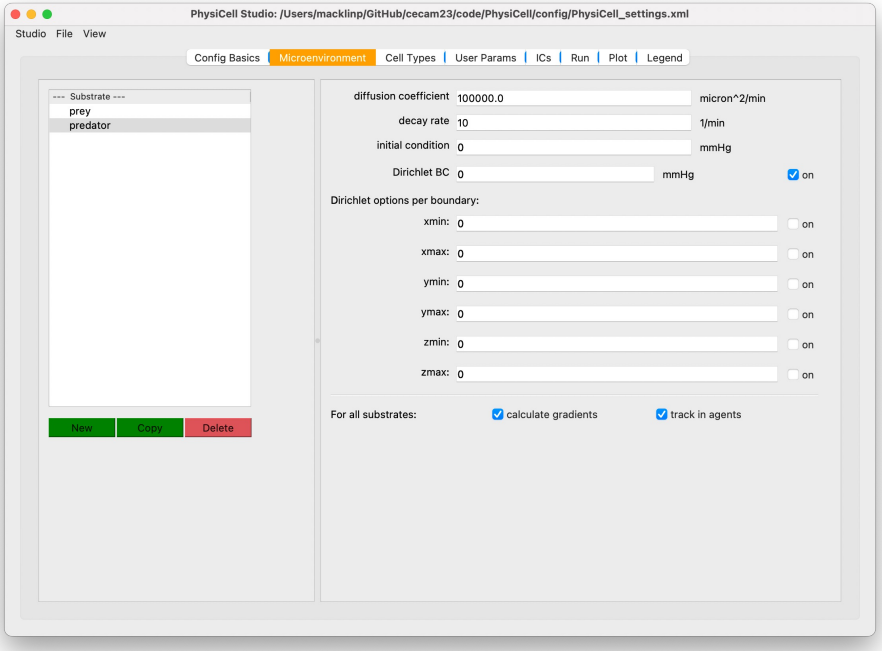
- Set the simulation outputs to every 10 minutes
  - Set SVG output interval to 10 min
    - This is how often the cell positions save
  - Set Full output interval to 10 min
    - This is how often the substrate data is saved



Prepare diffusing substrates

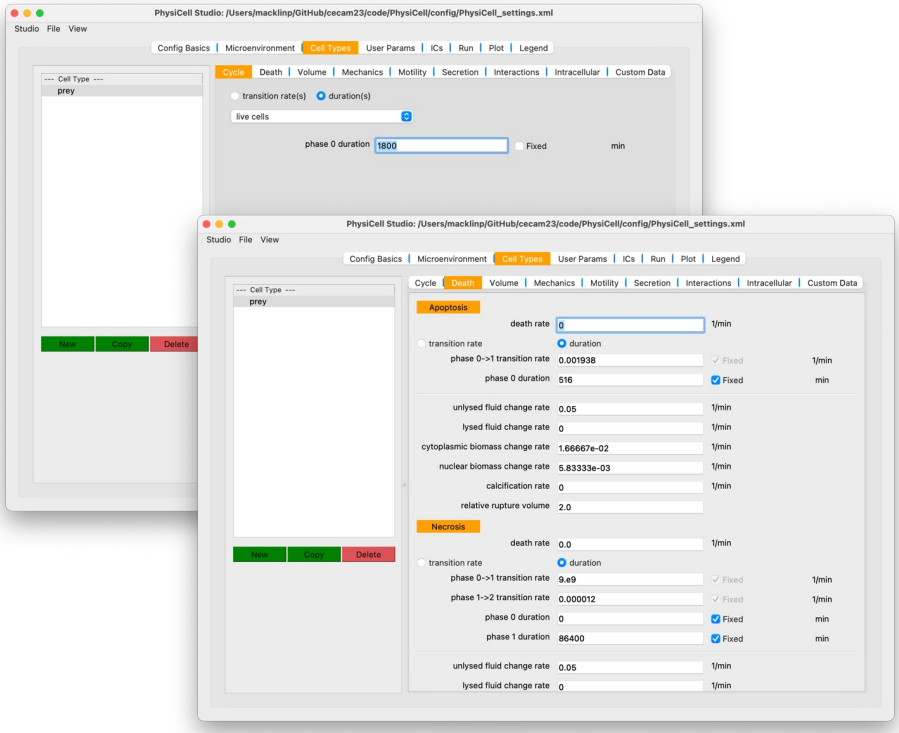
- Make a diffusing “prey” signal
  - click on environment tab
  - double-click “substrate” name, rename it to ”prey”
  - leave the parameters alone for now

- Make a diffusing “predator” signal
  - Click on “prey” and then “copy”
  - Double-click on the new substrate to rename it to “predator”
  - Leave the parameters alone



Create a prey cell type: birth and death

- Go to the “cell types” tab
- Double-click on “default” and rename it to “prey”
- Set its cycling rate
  - Click on the “cycle” sub-tab
  - On the drop-down, select the “live cells” cycle model
  - For duration, choose 1800 min
- Set apoptotic death to zero
  - Choose the “death” sub-tab
  - Under “Apoptosis”, set the death rate to 0

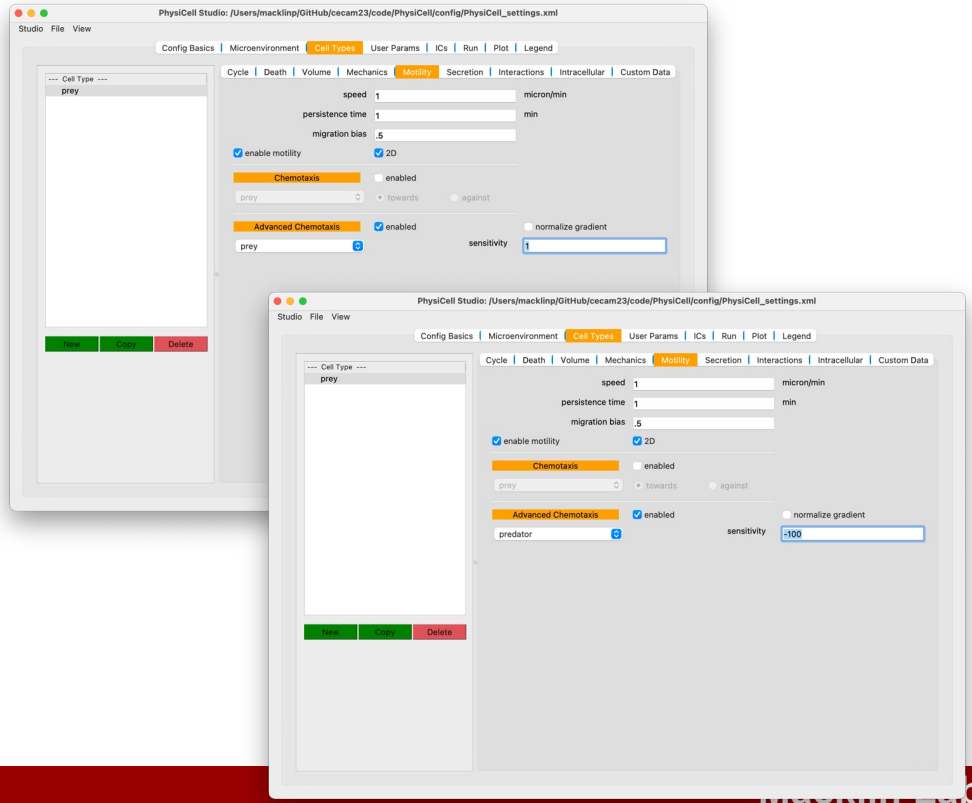


Create a prey cell type: motility

- Continue working on the “prey” type

- Set it to be motile
  - Click on the “motility” sub-tab
  - Choose “enable motility”
  - Leave the parameters alone for now

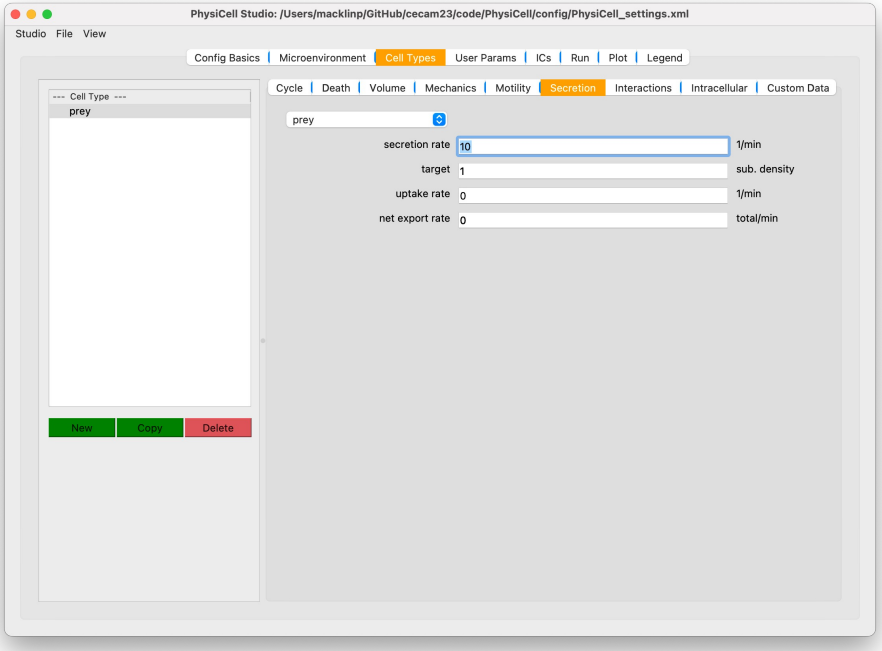
- Set up advanced chemotaxis
  - Choose “enabled” next to advanced chemotaxis
  - Under “prey”, set sensitivity to 1.0
    - This makes prey cells chemotax towards (positive weight) prey signal
  - Under “predator”, set sensitivity to -100
    - This makes prey cells chemotax away (negative weight) from predators, with a much higher priority (larger weight value)



Create a prey cell type: secretion

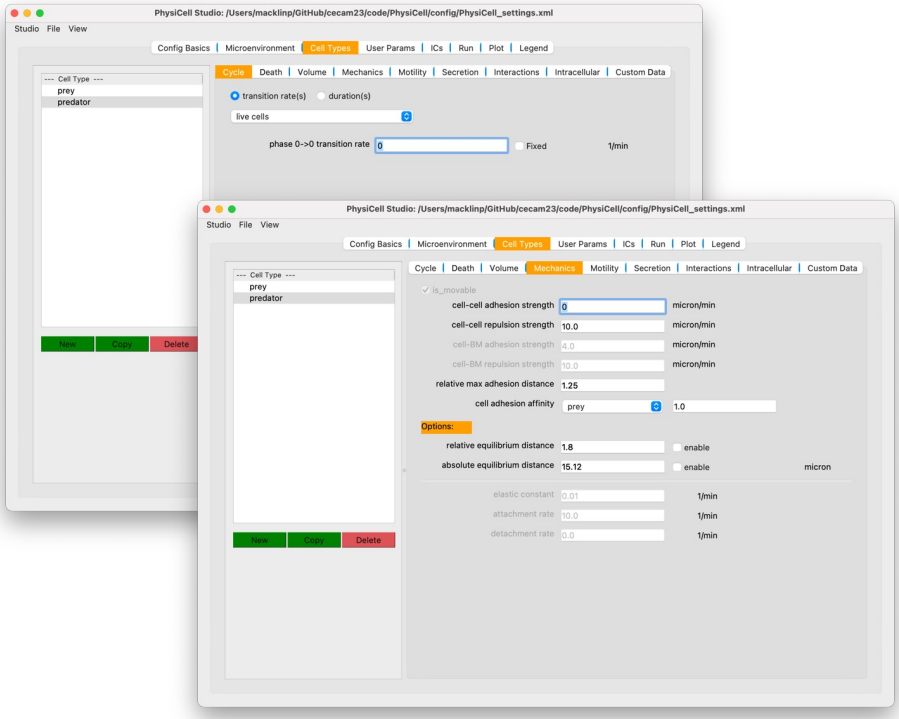
- Continue working on the “prey” type

- Set it to secrete the “prey” signal
  - Click on the “secretion” sub-tab
  - Choose “prey” signal from the drop-down
  - Set the secretion rate to 10
  - Set the secretion target to 1



Create a predator cell type: birth & mechanics

- Go to the “cell types” tab
- Select the “prey” cell type, and choose “copy”
- Double-click the new cell type, and rename it to “predator”
- Set its cycling rate to 0
  - Click on the “cycle” sub-tab
  - Switch from “”duration” to “transition rates”
  - Set the Phase 0->Phase 0 transition rate to 0
- Since we copied the prey type, it already has zero apoptotic death
- Go to the “mechanics” sub-tab
  - Set cell-cell adhesion strength to 0

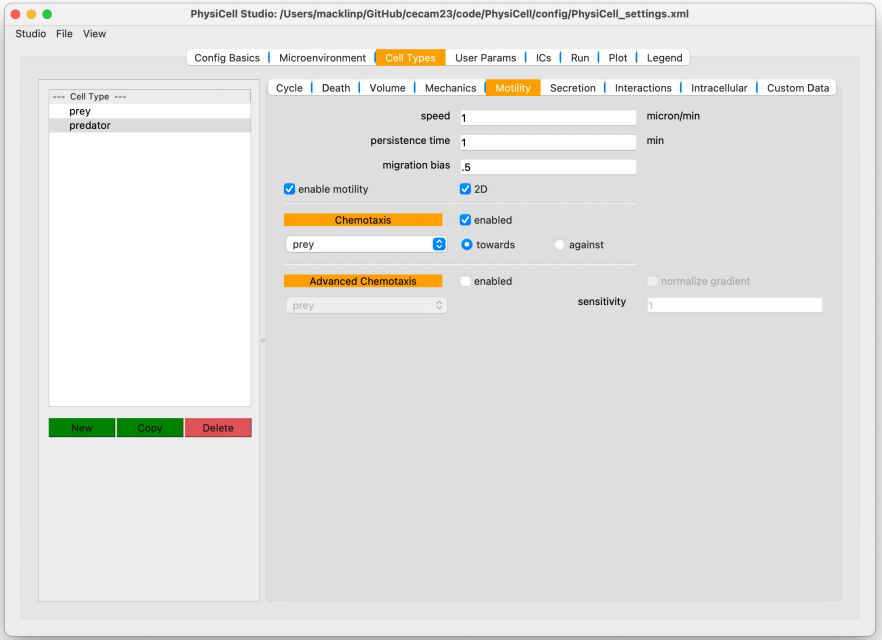


Create a predator cell type: motility

- Continue working on the “predator” type

- Set it to be motile
  - Click on the “motility” sub-tab
  - Choose “enable motility”
    - Should already be checked from the “copy” operation
  - Make them faster: set speed to 1.5

- Set up regular chemotaxis
  - Choose “enabled” next to chemotaxis
    - Advanced chemotaxis should now uncheck.
  - Select “prey” as the substrate
  - Choose “towards” as the direction

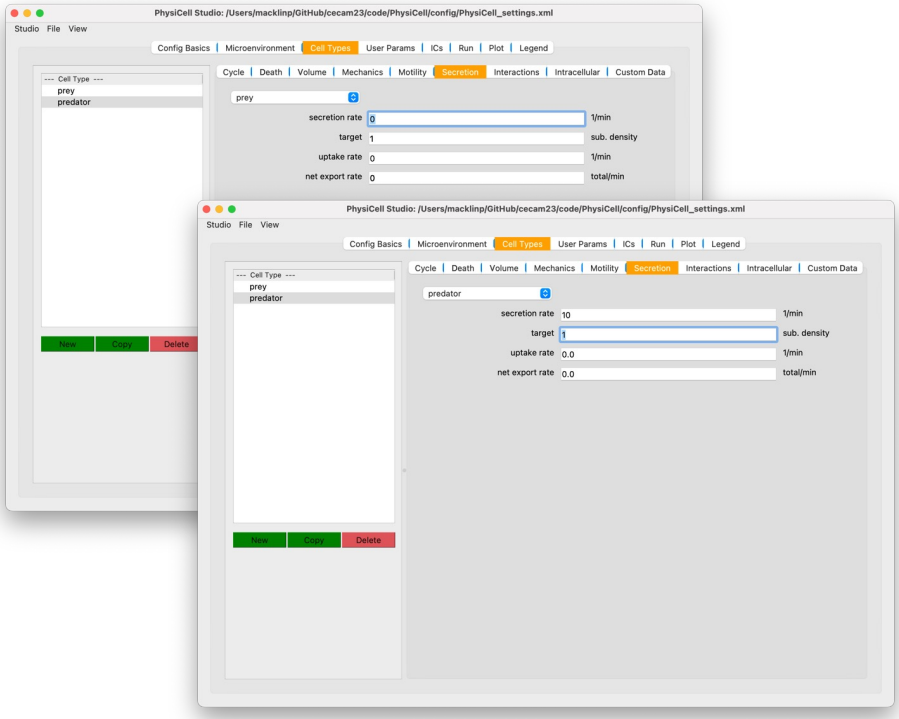


Create a predator cell type: secretion

- Continue working on “predator” cells

- Set it to not secrete the “prey” signal
  - Click on the “secretion” sub-tab
  - Choose “prey” signal from the drop-down
  - Set the secretion rate to 0
  - Set the secretion target to 1

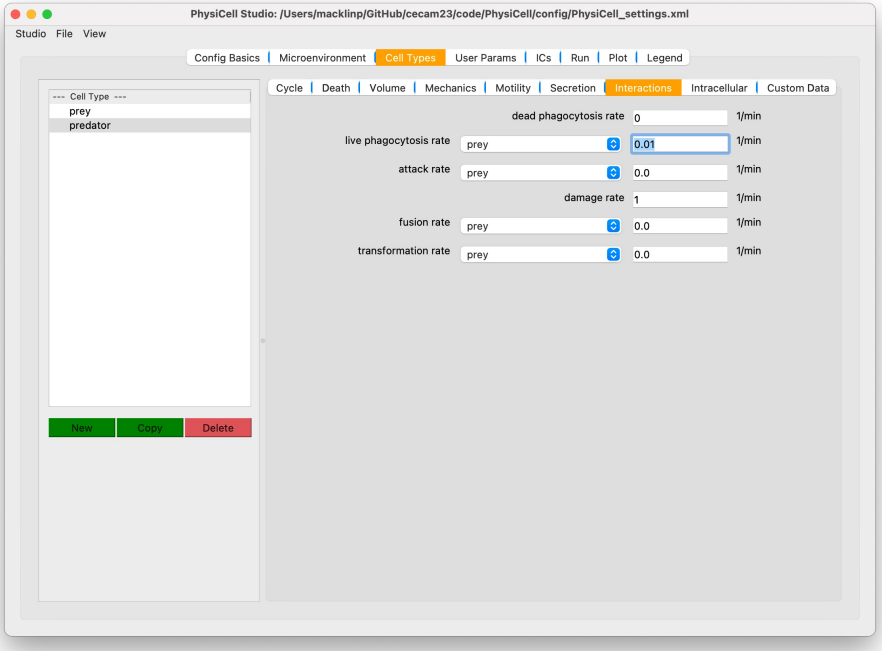
- Set it to secrete the “predator” signal
  - Click on the “secretion” sub-tab
  - Choose “prey” signal from the drop-down
  - Set the secretion rate to 10
  - **Set the secretion target to 1**



Create a predator cell type: interactions

- Continue working on “predator” cells

- Set it to phagocytose (eat) prey
  - Click on the “interactions” sub-tab
  - Choose “prey” signal from the drop-down next to “live phagocytosis rate”
  - Set this live phagocytosis rate to 0.01



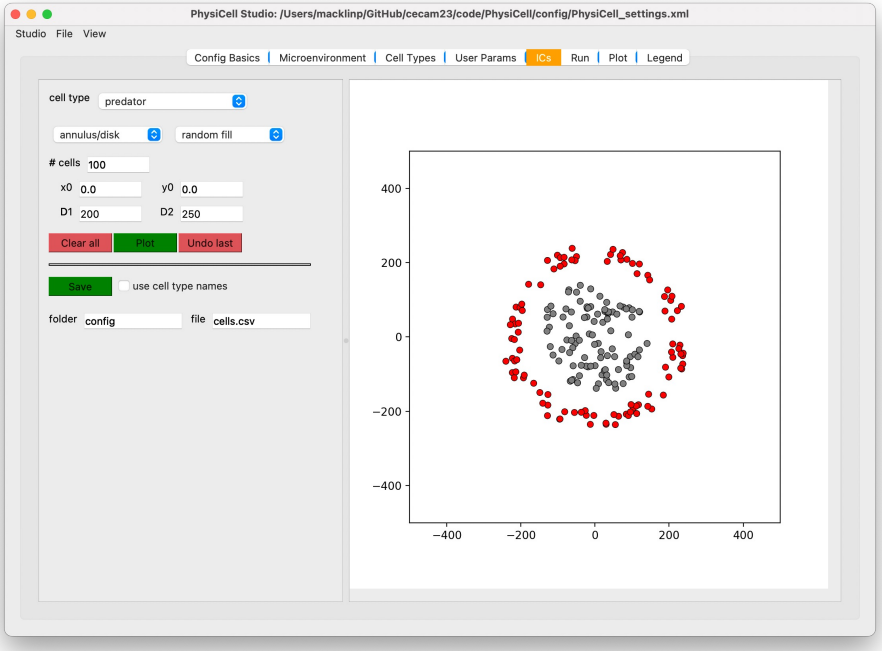
Place cells in the environment

- Go to the ”ICs” (initial conditions) tab

- Place 100 prey randomly in a 150 micron circle
  - Choose “prey” in the drop-down
  - Choose “annulus / disk”
  - Choose “random fill”
  - Set L1 (min radius) to 0, L2 (max radius) to 150
  - Click “plot”

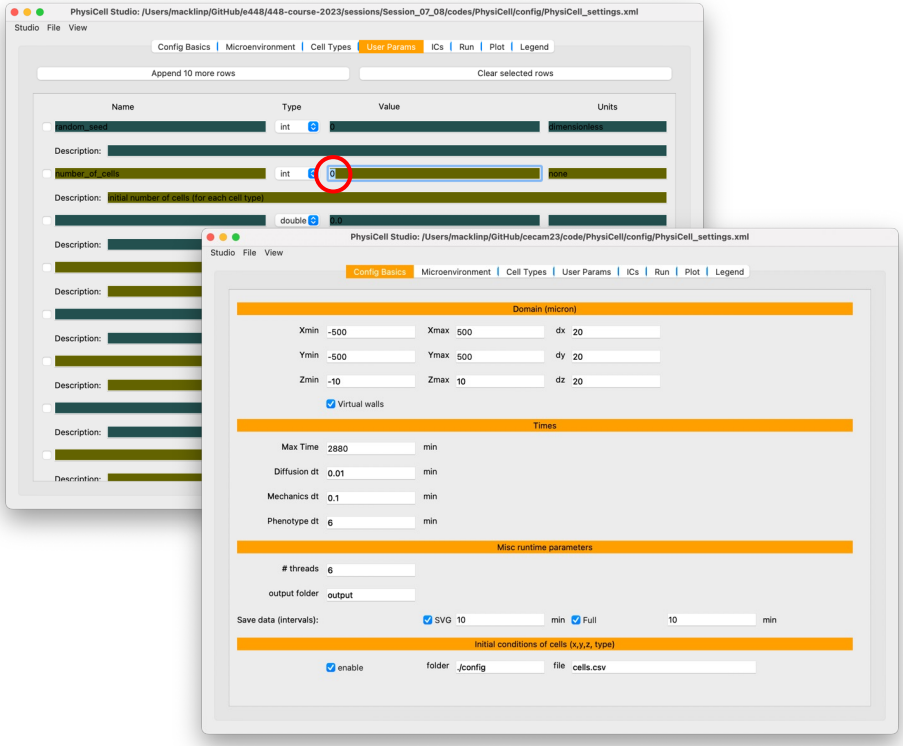
- Place 100 predators randomly in an annulus from 200 to 250 microns
  - Choose “predator” in the drop-down
  - Choose “annulus / disk”
  - Choose “random fill”
  - Set L1 (min radius) to 200, L2 (max radius) to 250
  - Click “plot”

- Export the cells
  - Click the "save" button to write this plot to a CSV file
    - Make sure it saves to "config" as "cells.csv"



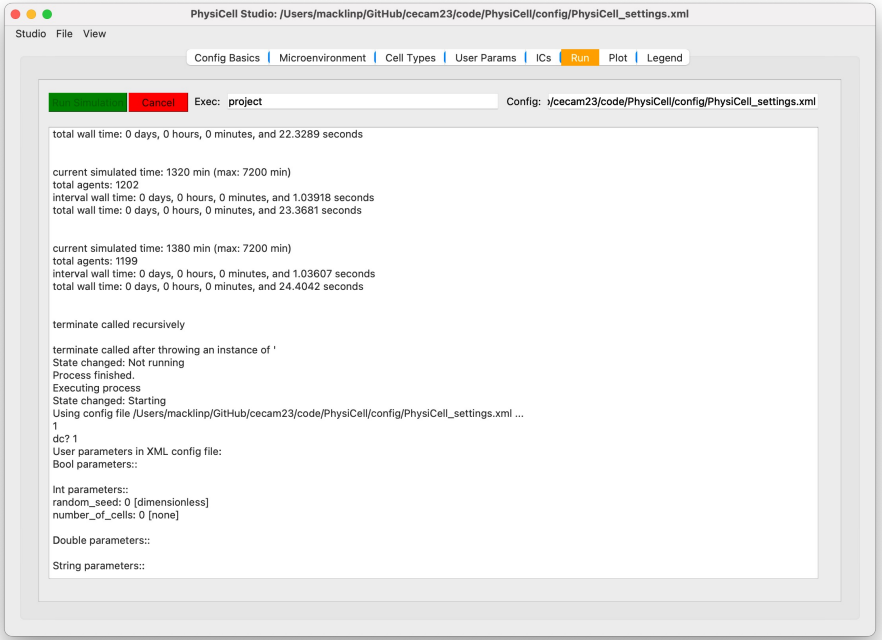
Use these cell positions

- Disable random placement of cells
  - Go to the ”User params” tab
  - Click on the “number_of_cells” variable
  - Set the value to 0
- Enable the code to read the initial cell positions in CSV
  - Go to the “Config basics” tab
  - In the bottom “initial conditions of cells” section, click “enable”



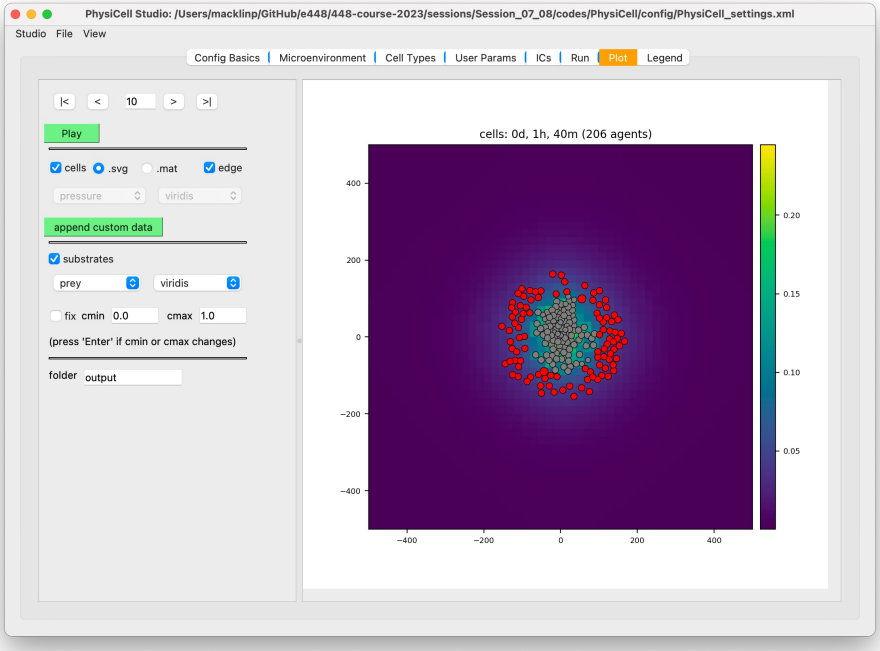
Run the simulation

- Go to the “Run” tab
- Make sure the executable name is correct (in this case, “project”)
- Click “run simulation”



Visualize results

- Go to the “Plot” tab
- Click the Play and similar buttons to animate the plots
- Use the “substrates” to choose different substrates to plot

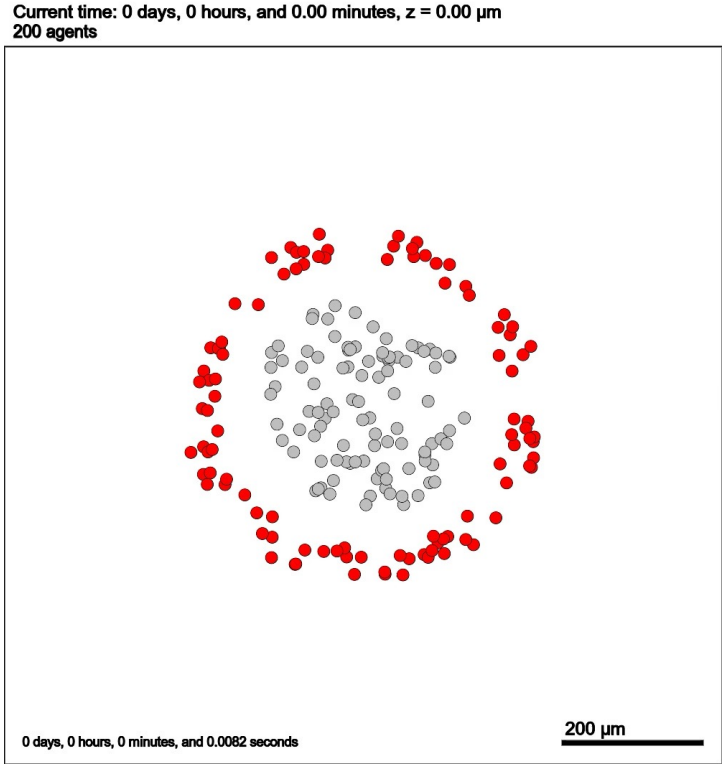


On your own:

- Change the prey proliferation rate so that the mean cycle time is 1000 minutes.
  - What happens?

- In addition, change the live phagocytosis rate to 0.001 min^{-1}.
  - What happens?

- Now, in addition, add some coordination in the predators:
  - Change from regular chemotaxis to advanced
    - Prey: sensitivity = 2
    - Predator: sensitivity = -1



Retrieving this project

- This project is saved as "demo1"
  - make load PROJ=demo1

Looking Forward: Full modeling workflow

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

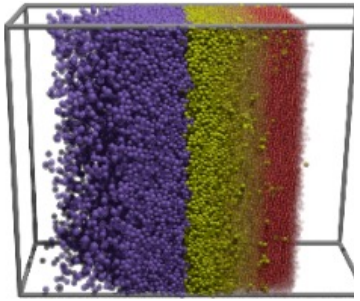
- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**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

Save the date!

2023 Virtual PhysiCell Workshop and Hackathon

July 23-29, 2023

- **Build and explore multicellular agent-based simulations of cancer and other systems**
- **Learn to share your models online**
- **Meet other modelers in the PhysiCell community**
- **Compete in an exclusive mentored hackathon**
- **PhysiCell swag available for accepted participants**



- **Application and full agenda coming soon at:** https://github.com/PhysiCell-Training/ws2023