Documentation

*#### Here's what each of the files do:*Run: Run this file in order to run the program as a whole. Everything is pretty malleable   
but there are a few lines of code that should remain to keep the model running;  
however, this isn't set in stone. Important lines are marked with "(base)". I encourage  
the addition, deletion, and editing of the functions other than "(base)". I have merely  
provided an example of functions I put together for my purposes.  
  
Input: This will open up the Setup\_files directory and parse through each of the  
.txt files. These files are written in a certain way so the they can be interpreted  
easily by the model. Though, if you need to change them, directions will be in   
Input. The files contain important parameters on how the model will be set up for  
each simulation run.  
   
Simulation: This class is really what is being run when each simulation is run.  
Each instance of this class corresponds to a setup .txt file. Most of the methods  
in the class are used to update the cells individually via Cell class method. There  
are a few functions that act on the cells collectively and run solely in the  
Simulation class. The instance variables of the class consist of holders and  
thresholds. The two main holders are for the Cell objects and the Gradient objects.  
The thresholds will be for division and differentiation checks.  
   
Gradient: The model consists of three different types of objects: Simulation (used  
to hold things as the simulation runs), Cell (used to represent each cell), and  
Gradient (used to apply a gradient of molecular concentrations to the grid). So  
what is the grid? The grid is space that the cell objects occupy. This can be  
either 2D or 3D. Each Gradient object will create an array with the dimensions of  
grid. To determine the location of a cell on the grid, we round the location to  
the nearest whole number corresponding to an index of the grid.  
  
Cell: The Cell class is housed here. Each cell in the simulation is representative  
of single cell in an experiment. All of the cell objects are held by an array as  
an instance in the Simulation class. You'll see that the cells will have instance   
variables that correspond to values such as radius, mass, state, and many others.  
The class also holds methods for updating the Cell instance variables. You may ask  
why some methods for the Cell class weren't integrated into the Simulation class;  
however, it's for simplicity and easy writing.  
  
Parallel: While the model will run at a reasonable speed when everything is run  
on the CPU, there is the benefit of GPU parallel processing when certain functions  
of the model are parallelized. Updating the grid takes significant time as a 3D  
representation involves a triple for-loop. In addition, checking for cell  
neighbors is taxing because there is a double for-loop iterating over thousands  
of cell objects which can take some time. Thankfully, an NVIDIA GPU combined with  
the CUDA toolkit and Numba library allows for significant decreases in run time.  
  
Output: All outgoing data from the model will be processed in this file. For each  
time step, an image and a .csv file will be produced. The image provides a visual  
representation of the cells in the simulation. The .csv is a way of transporting  
data from the model to other forms of statistical analysis. The .csv will contain  
information for each cell such as location and whether it is differentiated  
or pluripotent.

*#### Customize initial parameters:*Look for the "Setup\_files" directory. This should contain "Example.txt", which is a template for  
the initial parameters of each simulation. You can put multiple templates in the "Setup\_files" to  
run series of simulations one after another.  
  
  
  
  
*##  
   
#### Libraries:*I strongly recommend using Python 3.7 via Anaconda. This supplies most of the  
libraries needed for running the model. The Numba library will need to installed  
to run the GPU functions, but if your CPU is strong enough you may be able to get  
away with running things on the CPU only.  
   
*##  
  
#### Common Errors:*Along the lines of...  
   
"IndexError: index 1008 is out of bounds for axis 0 with size 1000"  
 - Cells are traveling outside the grid space and when the model tries to   
 coordinate the location with a spot on the grid it cannot as that grid  
 index does not exist  
   
"CudaAPIError: [1] Call to cuLaunchKernel results in CUDA\_ERROR\_INVALID\_VALUE"  
 - Check to see how the threads per block are managed in Parallel. You may  
 may have to decrease the number of threads per block.  
   
"TypingError: Failed in nopython mode pipeline (step: nopython frontend)"  
- Adjust the way the CUDA functions are performing operations. They deal in  
 in arrays so you may be adding an array to a float or something similar  
   
"MemoryError"  
- The size of the grid is too big for the model to handle. Resize by a factor  
 of 10 and retry.  
   
"CudaAPIError: [2] Call to cuMemAlloc results in CUDA\_ERROR\_OUT\_OF\_MEMORY"  
 - This occurs when the grid size is too large for the GPU to handle. A GPU  
 with more memory is recommended.  
   
Anything else you may come across should be easily diagnosable via Numpy support  
as the model heavily uses Numpy. Consult Numba Documentation for issues with   
the CUDA functions.  
   
   
*#### Other problems, please read the following quote.*"That's just how the peaches roll."  
 - Garret Fritz

Here begins the explanation and references of the new collision model.

1. <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006273>