TODO:

* ~~Subclass StatechartCellCycleModel creating ElegDevStatechartCellCycleMode~~
* ~~Condense StatechartCellCycleModel into one hpp file.~~
* ~~Fix serialization of GlobalParameterStruct.~~
* Create LeaderCellMovementModel, with DTCMovementModel as a child class
* Create ElegDevLeaderCellBoundaryCondition, a child class of LeaderCellBoundaryCondition, and move all stretching code into the Eleg specific class.

LOW PRIORITY

* Rename RandomCellKillerByType, making it clearly Elegans specific
* Find a way to output division events to file without altering the Cell class.

**TestElegansGermline.hpp**

This is the main model file that defines a *C. elegans* germ line simulation. It has a single method *TestLarvalDevelopment*, which is called when **TestElegansGermlineRunner** is executed.

Briefly, the main steps carried out in *TestLarvalDevelopment* are:

1) Read in a parameter file

2) Set up a collection of Nodes, positioned at the starting location of each cell

3) Turn the Nodes into a NodeBasedCellPopulation, where each cell has a cell cycle model based on some choice of statechart.

4) Specify the properties of each cell, and request ancestor tracking.

5) Make a Simulation object out of the cell population and set its

properties (time step, end time etc.).

6) Specify the force law between cells

7) Set up the leader cell boundary condition

8) If using contact inhibition in your cell cycle model, add a volume

tracker to measure cell compression

9) Add cell removal by apoptosis and fertilisation

10) Add data output

11) Solve, and save the final state

12) Delete the node collection

To write a model of your own building on our code, you will need to produce a similar **TestMyModel.hpp** file in the same directory as this file, perhaps using TestElegansGermline.hpp as a template.

**DTCMovementModel.hpp(cpp)**

This file specifies the movement of the DTC, and maintains a collection of equally spaced points on the DTC’s path.

**LeaderCellBoundaryCondition.hpp(cpp)**

This file enforces a tube shaped boundary condition, based on the collection of points on the leader cell path it receives from **DTCMovementModel**.

**Fertilisation.hpp(cpp)**

Removes oocytes from the simulation once they reach the proximal end, together with one sperm.

**RandomCellKillerByType.hpp(cpp)**

Kills oocyte fated cells that have yet to reach the proximal arm, with some probability.

**GlobalParameterStruct.hpp(cpp)**

Provides a collection of parameter values that can be set by reading in a text file, and is available to be read by any of the other files in the project.

**CellTrackingOutput.hpp(cpp)**

Outputs the positions of certain tracked cells over time. The constructor takes two arguments: the number of time steps between measurements (int), and the subset of cells to be tracked (int; a value of 5 results in every 5th cell being tracked).

**GonadArmDataOutput.hpp(cpp)**

Outputs a collection of data relevant to C. elegans germ line simulations. Takes one parameter: the number of time steps between measurements (int).

**RepulsionForceSizeCorrected.hpp(cpp)**

Defines a force law between cells that is based on Hooke’s law, but with a slight modification to increase the stability of multicellular simulations. Cell drag is also corrected for cell size by scaling according to the radius.

**AbstractStatechartCellCycleModel.hpp**

Defines two methods that every StatechartCellCycleModel should expose: *SetCellCyclePhase,* which allows a statechart to set the cell cycle phase, and *SetReadyToDivide*, which allows a statechart to call for division.

**StatechartCellCycleModel.hpp**

A cell cycle model wrapper around a statechart, designed to make the statechart “look” like a normal cell cycle model to the rest of Chaste. It defines the length of the G1, S, G2 and M phases, and how they update over time. This class also deals with archiving itself and its statechart when a simulation is saved/loaded, and handles creating new cell cycle models when a cell divides. StatechartCellCycleModels hold a pointer to a statechart, which takes over the jobs of updating the cell cycle phase and determining when a cell is ready to undergo division.

**StatechartInterface.hpp**

Defines a number of useful functions that may be needed in a variety of statechart models. These include things like setting the cell radius and calling for division.

**FateUpcoupledFromCycle.hpp(cpp)**

An example of a statechart model of cell behaviour. This class makes use of the Boost Statecharts library by Anderas Huber (<http://www.boost.org/doc/libs/1_57_0/libs/statechart/doc/index.html>).

To be compatible with the rest of Chaste, a Statechart model must define certain methods and member variables that allow it to control cell cycle progression, and be saved and loaded during archiving. These required methods/members are:

* *CellPtr pCell*  
  Pointer to the cell being controlled by the statechart.
* *double TimeInPhase*  
  Used to count the time a cell has spent in its current cell cycle phase.
* *boost::shared\_ptr<CLASSNAME>Copy(boost::shared\_ptr<CLASSNAME> myNewStatechart)*

Used to copy the state of the parent chart into a daughter chart during division

* *std::bitset<32> GetState()*

Encodes the current state as a bitset.

* *std::vector<double> GetVariables()*

Packages all variables associated with the chart in a vector for archiving.

* *void SetState(std::bitset<32> state)*

Sets the state of a statechart from a bitset, for loading from an archive.

* *void SetVariables(std::vector<double> variableValues)*Sets the values of chart-associated variables from a stored vector, again for loading from an archive.

A statechart will most likely also contain state that respond to the following events:

* *EvCheckCellData*Fired by StatechartCellCycleModel each time step to request that the statechart updates
* *EvGoToCellCycle\_Mitosis\_M*
* *EvGoToCellCycle\_Mitosis\_S*
* *EvGoToCellCycle\_Mitosis\_G2*
* *EvGoToCellCycle\_Mitosis\_G1*Events fired by StatechartCellCycleModel at the start of a new simulation to force cells into unsynchronised initial cell cycle phases.

**Failure to implement these required methods/variables/responses in your own statechart models will result in a template substitution error.**

The

**FateDecisionCoupledToCycle.hpp(cpp)**

A second example of a statechart model of cell behaviour. The primary difference is that in this model the sperm/oocyte fate decision occurs together with entry into meiosis, which is coupled to the cell cycle in the sense that it can only occur in G1. From a code point of view, this class

is very similar to **FateUpcoupledFromCycle.hpp(cpp)**.