2.2 Ageing

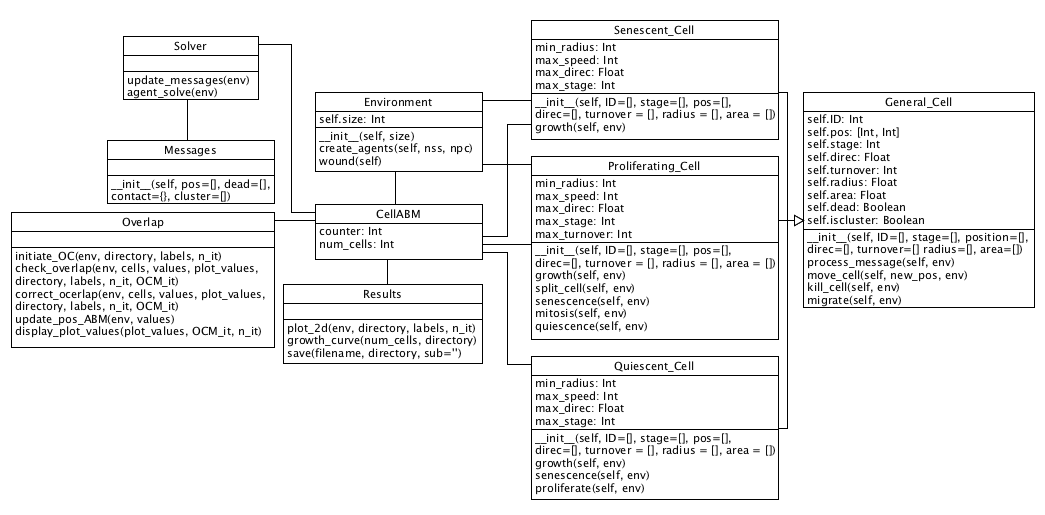
An important factor that contributes to pro-atherosclerotic changes to the endothelium is ageing [7]. The number of times an EC can divide is limited, and once reached the cell goes into growth arrest, known as senescence [8]. This is due to the shortening of the ECs telomeres (the end parts of DNA) by 50-200 base pairs each time the cell proliferates. Once these telomeres are shorter than a critical length, the cell becomes senescent. The number of times a cell can proliferate is known as the Hayflick Limit, and for normal ECs is around 50 [9].

Studies have shown that senescent cells accumulate in tissues with age [http://www.pnas.org/content/pnas/92/20/9363.full.pdf, https://onlinelibrary.wiley.com/doi/10.1111/j.1474-9726.2009.00481.x], and Cellular Senescence in Aging Primates [http://science.sciencemag.org.sheffield.idm.oclc.org/content/311/5765/1257.full] has shown that the number of senescent cells increases exponentially with age, with total cell count reaching >15% senescent in elderly cases. The limitations of this paper include: that the results are from baboons, not humans and so the lifespan is only from 5-30, and the cells were taken from the medial aspect of the arm rather than the endothelium layer. However, this paper is useful in the fact that baboon’s telomeres, like humans shorten with proliferation, and the baboon’s cells also undergo senescence.

1. **Design**As seen above, there are several ways of developing an ABM to implement the requirements. In this chapter, we will explore the underling language of the program and how it can be used to model an ABM, then discuss the class diagram and flow charts of how information will flow through the system, finally discussing what simulations will be run to answer the research question.
   1. An overview of Python and its Class System  
        
      Since the implementation will be driven using CellABM, Python is the language of choice for this project. Python is similar to other widely used languages such as Java and JavaScript [https://www.python.org/doc/essays/comparisons/] in that it is an interpreted and an Object Orientated Programming (OOP) language. However, Python has some significant differences that lead it to be syntactically easier to read than Java and it has better code reuse than JavaScript. A Python program is generally 3-5 times smaller than the same program written in Java, thus decreasing development time and reducing the chance of bugs.   
      In Python, data is encapsulated inside objects. These objects can change their own data or interact with other objects. This method of object orientation can be used to represent the different types of cells required in the program.

Python also uses inheritance. This means that instead of writing the same function for several classes, there can be one parent class with the function and other classes can inherited that function from them, reducing repeated code. In the case of CellABM, this means each cell type: Proliferative, Senescent, and Quiescent can all inherit the same apoptosis (cell death) function from an overall general\_cell class.

* 1. Class Diagrams  
       
     This class diagram is intended to show the information flow throughout the program and how the classes communicate with each other. An important feature to note is the general\_cell class acting as a parent class for the three cell types.

Figure 4.1: Class diagram of CellABM

* 1. EnvironmentAt the beginning of the program, the user will define several key parameters used to initialise the environment. Notably, the size (in micrometres), the number of starting Proliferating Cells and the number of starting Senescent Cells. This allows the user to define cell ratios for differing patient ages in accordance with the research question.

The Environment class creates the starting agents with a random set of parameters taken from a distributed range given, and appends them to a list of starting agents.

The environment will be modelled as a discrete space where agents cannot leave, to preserve computational runtime, and will provide the space for the agents to interact with each other. Cell positions can be mapped into this 2D space using a 2D array of equal size to the user’s definition and giving each cell an [x, y] coordinate.

Although in biology endothelial cells live in a 3D space, they tend not to over-lap one another, thus creating a 2D plane. For this reason, it is believed that little information is lost by modelling in 2D.

* 1. Theorised Program Flow  
       
     Below are the guides that will be followed during the development of the program. They provide the road map of how each class and function interacts with each other, leading to emergent behaviour of the cells. A quick overview of the cellular differentiation is given in Figure 4.5.1, showing how, generally, endothelial cells start out being normal Proliferating cells, then they can either move onto being Quiescent or Senescent. Quiescent cells can revert to Proliferating cells or turn Senescent if they persist long enough. As shown, Senescent cells act as a sink, trapping the cell in that state until the end of the simulation.

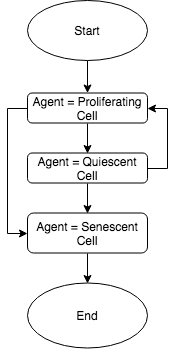
****

Figure 4.2: Cellular differentiation

* + 1. CellABM  
         
       This flow chart shows how the overall main class will run. It will start by taking the parameters from the user, initialising the environment with these parameters and ensuring the initial agents aren’t overlapping. When this is set up, the program will move into an iterative process of solving the agents (allowing to perform their programmed rules), ensuring they aren’t overlapping and then checking the number of quiescent cells in the environment. If the number of quiescent cells is larger than the threshold, the environment simulates the wound and the loop continues. At the end of each iteration, a graph will be plotted showing the location of each agent on the environment.

When the number of quiescent cells passes the threshold for a second time, the simulation is stopped as a confluence will have re-formed, this will also produce a growth curve of the agents over the iterations.

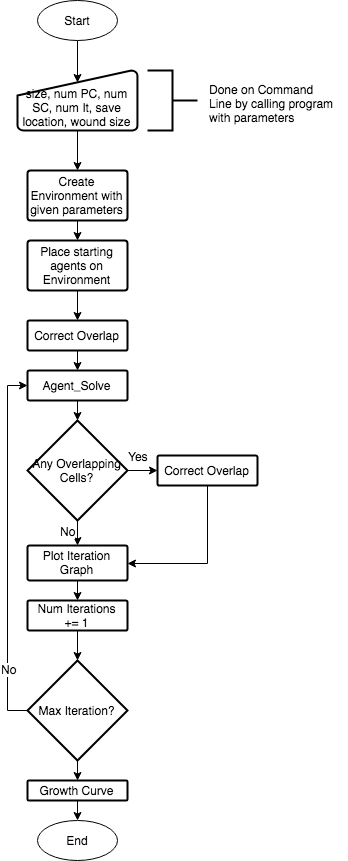


Figure 4.3: CellABM class overview

* + 1. Cell Differentiation

A more thorough plan of cell evolution is given below in figure 4.6.2.1. This shows the intended logic behind each of the cell stages, and how the cells will differentiate with the simulation.

The Proliferative cells have both a turnover value and stage value (not shown here). The turnover is the Hayflick Limit mentioned in the Literature review, and once reached, the proliferative cell will differentiate into a senescent cell. Cell stage however, will be used to track what stage in the cell cycle the cell is at and to decide whether the proliferative cell should undergo mitosis that iteration.

The quiescent and senescent cells only have a stage value associated with them. As these cells do not undergo mitosis, there is no need to track what stage of the cell cycle these cells are in and is therefore used as the Hayflick representation.

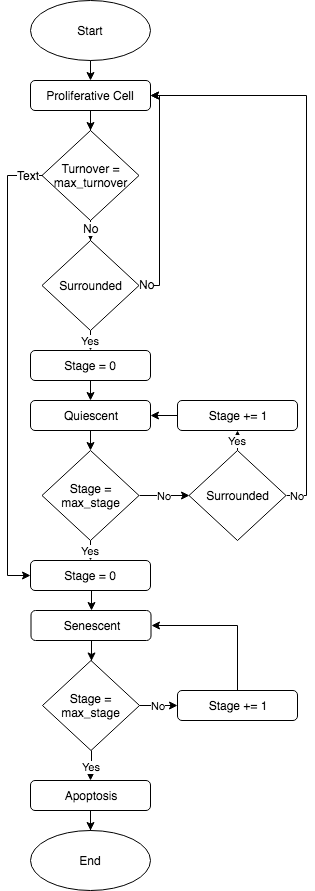
****

Figure 4.4: Cell Differentiation Steps

* + 1. Agent\_Solve  
         
       This flow chart has been created by looking at the current underlying logic for the agent\_solve class in CellABM and including the extra steps required to allow for the new rules and cells the question requires. These steps will be run on every cell in the model.

For Proliferative and Quiescent Cells, it is important to test whether they will become Senescent first as if this is true it shows the cells have passed the Hayflick limit, as seen in chapter 2.2, and their telomere ends have passed their critical length, so the cell must turn Senescent.

Senescent cells are unable to differentiate back to a PC or QC, thus ever iteration they only test to see whether they will under apoptosis.

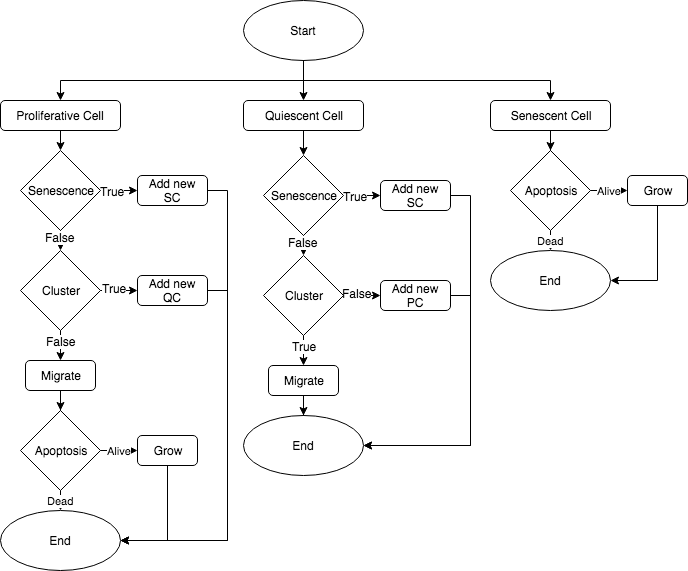


Figure4.5: Overview of agent\_solve class flow

* 1. Simulations to Run

As the main objective of this project is to determine the different times taken for a wound to heal whilst varying the person’s age, several simulations will be run with varying percentages of senescent cells in accordance with the primate paper in chapter 2.2, with the time taken being plotted.

ABMs are generally stochastic, and CellABM is no different. The initial placement of cells onto the environment is random, so too is their starting size and stage in the cell cycle. Due to these random variables, several simulations with the same starting parameters must be run to achieve adequate analysis of the model.

Results of the simulations will be compared to an in vitro study of human umbilical vein endothelial cells which have been wounded with p20 pipette (around 400μm) on an area of 1mm by 1mm [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5154238/pdf/kcam-08-05-969641.pdf].

To show that the implemented rules are working as expected, micro simulations will be run with set parameters and cell stages to ensure the behaviours will work correctly on the macro scale. The simulations will involve a low number of cells, around 1 to 10, and will be simulated for the least amount of time required to observe the desired behaviour.