Modelling The Regression Of Henson's Node Using A Modified Cellular Potts Model.

Nigel C Harrison

Department of Mathematical Sciences,
University of Liverpool
(MSc Mathematics Main Dissertation)

Supervisor:
Dr Bakhtier Vasiev

(2008)

Contents

1 Abstract.

Extension of the vertebrate body axis during the neurulation stage in embryogenesis, is marked by the regression of Henson's node along the primitive streak. As the node regresses the neural and presomitic stem zones produce cells for the forming central nervous, somites and some ingress through the primitive streak as mesenchymal cells. At the time of writing there is vast amounts of biological experimental data available on these processes, however little in the way of understanding has been achieved so far.

In an attempt to further understanding in this area, we developed a mathematical model based on the well established cellular potts model in an attempt to simulate this process in a simplified way. The simplification was to consider the region of Henson's node as motile homotypic domain of stem cells that proliferates and differentiates to form a homotypic trail of non-motile body cells. We assumed motility was the result of a chemoattractant and that stem cells are sensitive to this chemical while body cells are not.

Central to our investigation was the mechanisms that regulates the size of the stem cell domain and the progressive formation of the trail. To illustrate how this is possible we proposed two hypotheses: firstly mitotic division of stem cells in the stem zone produce two daughter cells, one for the stem zone and one for the forming body, in this way proliferation and differentiation combined with chemotaxis would, in theory, satisfy our requirements. Secondly that the interaction of morphogens, specifically FGF-8 and RA result in a co-moving travelling wave with the node, with it highest concentration in the anterior of the stem domain, where differentiation of stem cells would occur.

We will demonstrate that our first hypothesis failed to produce acceptable results and that methods -albeit unjustified- to correct this failure could be found. In addition we will demonstrate that our second hypothesis was correct, and there is a co-moving wave of a morphogenic chemical that can maintain the size of the stem cell domain and progressively form the trail of body cells.

2 Introduction

During the earliest stages of life, living organisms pass through a remarkable set of processes that lead to the formation of a complete body plan; the features that identify an organism as belonging to a particular species. What constitutes the "set of processes" is entirely dependant on the organism we are investigating -be it plant or animal- but in general Developmental Biologists refer to these processes as Embryology. This is the study of an organism from fertilization to birth, hatching or in plants, before germination occurs. It is within this phase of life - embryology - that we will attempt to model a specific feature in the regression of Hensons'node in a stage known as *neurulation*.

To model our problems we will use the well known Cellular Potts Model- CPM. This model, developed by Glazier and Graner [8], is used to model inter/intracellular interactions by discretizing the cells onto a 2D array and applying principles from statistical physics and random sampling methods - Monte Carlo methods - to simulate the kinetic and morphogenetic features of cells. However, the CPM is a generic tool for modelling cells, be they biological or non-biological; for example soap bubbles, metallic grains or epithelia - skin- cells. This implies we must develop methods to model features specific to our investigation on the CPM. What these features are will become clear in subsequent sections.

Our investigation then, is composed of two problems: Understanding the biological mechanisms involved under reasonable hypothesis, and developing computational techniques for the model of choice - the Cellular Potts Model - to model these mechanisms. Further, we make the restriction that any further biological discussion, will be in the context of the chick embryo. This restriction is not arbitrary, and is made on the basis that a developmental biologist and colleague of the supervisor of this paper[1], is investigating these phenomena in the chick embryo, and provides, hopefully, a biological correlation between our computational model and the biological reality.

2.1 The Biological Context

All animal life begins with the fertilization of an ovum to produce the *zygote*- a fertilized egg. After which several stages punctuate the development of the embryo: *Cleavage*, *Gastrulation*, *Neurulation* and *Organogenesis*, collectively they making up the stages of embryogenesis.

These stages, in general, are the same for all vertebrate embryos, but there are notable morphological differences and similarities within the different stages. For example, shortly after cleavage, the chick embryo is a tiny disc of cells -called the blastodisc- on top of a large spherical yolk. Whereas the mouse embryo, at an equivalent embryonic stage, is attached to the uterine wall as a small cup-shaped epithelial formation. In contrast to this is the so-called *phylotipic* stage, at which

[1]Dr Ruth Diez del Corral: University of Madrid

most vertebrates are very similar, and the overall body plan is laid down and basic structures such brain and spinal regions are recognizable.

From this point on, we will only be considering the stages of embryogenesis in the context of the chick embryo. We will not be giving an exhaustive account of this genesis, but a superficial overview, to give the reader the big biological picture so to speak.

Cleavage is when the zygote undergoes rapid *mitotic* cell divisions, with no discernable increase in mass. These divisions are a two stage process by which *eukaryotic* cells - those that have a genetic nucleus- literally divide into two new cells. The first stage, *mitosis*, is where the chromosomes are duplicated to produce two identical daughter nuclei, and followed immediately by *cytokinesis*, whereby the cell membrane cleaves/divides in two, with each half possessing one of the nuclei -see Fig. 1. We will consider this further when we come to talk about *proliferation*.

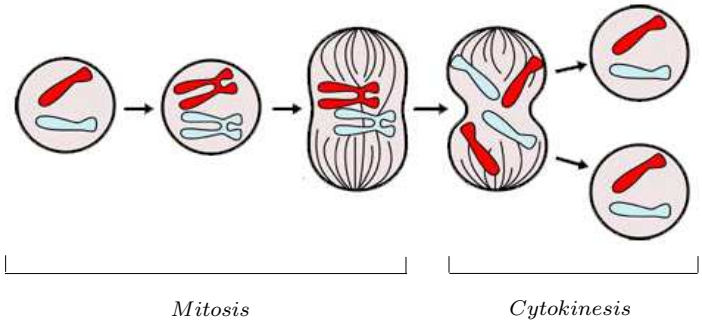
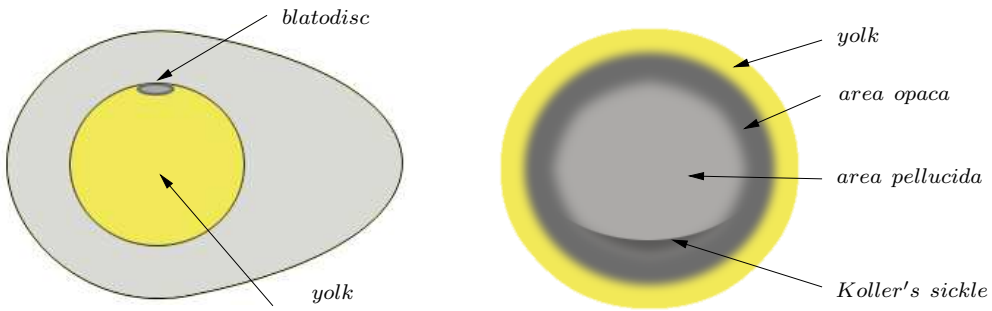


Figure 1: Mitotic Cell Division.

In the chick embryo, cleavage only takes place in a small circular region on top of the yolk called the blastodisc (Fig. 2 a), which will later form the embryo proper. This kind of cleavage, most commonly found in birds, fish and reptiles, is referred to as discoidal meroblastic cleavage. That is to say, the cleavages take place in a small region on a large yolk - Meroblastic - and does not penetrate the yolk and remains on top - discoidal. This is contrast to the more popular vision of cleavage as seen in humans embryos, *holoblastic* cleavage, where cytokinesis cleaves the entire zygote in half continually, doubling the number cells successively. The process of cleavage continues for approximately 20 hours after fertilization, until the chick egg is layed, at which time the blastodisc consists of approximately 60,000 cells. At this point a cavity has formed in the blastodisc from a space that developed blow the disc called *subgerminal space*. Above this space, and in the center of the blastodisc, the cell are translucent and given the name *area pellucida*, surrounded by a darker region of cells referred to as *area opaca*. More importantly, a sickle shaped region of cells has formed at the posterior of the blastodisc between the area opaca and are pellucida known as Koller's Sickle (Fig. 2 b). It is at this point the forerunner to the main body axis will begin, the *primitive streak*, and will signal the beginning of the next



a) Blastodisc in chick egg. *b) Dorsal view of blastodisc.*

Figure 2: Dorsal view of blastodisc with Koller's sickle signalling start of gastrulation.

stage of embryogenesis, Gastrulation.

Gastrulation induces a marked change in the morphology of the embryo where the 3 principal *germ layers*: *ectoderm*, *endoderm* and *mesoderm* migrate into positions that forms the primitive body plan for organism. In the chick embryo, this begins with the formation of the primitive streak. The streak appears at Koller's sickle and extends anteriorly about halfway across the area pellucida (Fig. 3).

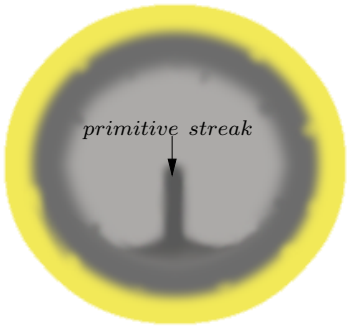
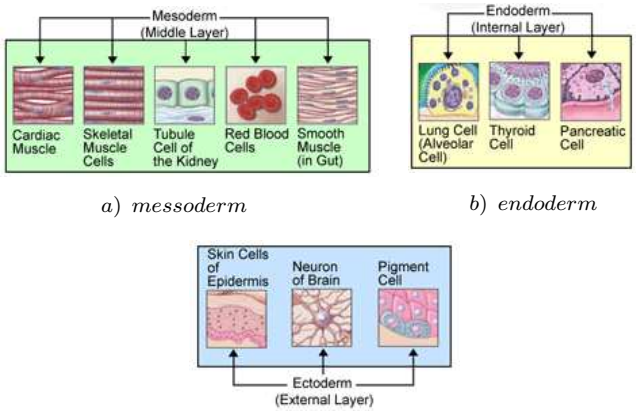


Figure 3: Onset of gastrulation signalled by primitive streak.

The primitive streak acts as a migratory point for the cells on the dorsal surface - top most. Some of the cells arriving at the streak, will enter into the subgerminal space inside the blastodisc as *mesenchyme* - loosely connected -cells, by a process know as *ingression*. These cells will become the endoderm and mesoderm layers, whereas the remaining cells on the surface will become the ectoderm.

The significance of the germ layers is that they will differentiate -or specialize- to give rise to the numerous cells throughout the chick. For example, the endoderm (Fig. 4b), gives rise to the digestive tract and liver, the mesoderm (Fig. 4a) to the

circulatory system and muscles and the ectoderm (Fig. 4c) to eyes central nervous system.



a) messoderm *b) endoderm*

c) ectoderm

Figure 4: Germ layers give rise to various body cells.

At the point when the primitive streak is fully extended, a condensation of cells know as *Henson's node* appears at the anterior most point of the primitive streak, as an indentation of inwardly moving cells. This is immediately surround by a thickening of ectoderm cells called the neural plate, on which the first formation of the brain and central nervous system will develop, and signifies the beginning of the neurulation stage.

Neurulation as its name implies, is concerned with the formation of the early brain and central nervous, and is often considered a phase in early organogenesis. In the beginning, cells are recruited from the ectoderm to form the neural plate, that moves centrally towards the midline. As they do, the plate begins to fold around this midline, forming a central cylindrical cavity known as the neural tube and is the pre-cursor to the central nervous system and brain. During this period, Henson's node and the primitive streak regress, extending the body axis along the central midline in a anterior to posterior motion.

In this setting, Henson's node is seen as the organizing center, orchestrating the formation of the neural tube and production of other cells in this region. Quite how this is accomplished is not fully understood, but in general it involves the production of chemicals in and around Henson's node, and in the newly formed zone left behind as it regresses. These chemicals FGF-8, FGF-8, mRNA and RA, are at the core of this paper and we will be considering this further in subsequent section. At this point, it is only important to understand this stage in the overall biological context of embryogenesis.

At the end of neurulation, the main structures of the vertebrate embryo are recognizable, brain, spinal area and beginnings of gastrointestinal tract. This stage is known as the phylotypic stage, as most vertebrate embryos at this stage look very

similar to one another and leads to the final stage of embryogenesis.

Organogenesis is where the internal organs of the body develop from the germ layers laid down during gastrulation. Major organs develop including the musclularskeletal , endocrine and circulatory systems, to name a few. The author would like to discuss this stage further, but in general the author believes readers understand this phase without further explanation. Suffice to say, at the end of this stage, the chick is fully formed and hatching quickly ensues.

It is clear this introduction represents only the most superficial treatment of this fascinating area of developmental biology, but hopefully the reader has gained a broad picture of the early phases of life. Armed with this information, we next consider the computational tool, the CPM.

2.2 The Cellular Potts Model (CPM)

The Cellular Potts Model, developed by Graner and Glazier [8], is a lattice based computational model for the simulation of cellular based phenomena. These cellular phenomena can be biological or non-biological, such as cells in organic tissue or foam structures such as those found in soap bubbles. Thus the model is characterized by the generalized idea of a cell and modeling the cellular interactions.

The way the CPM models these interactions can be broken down into 4 main areas: The Lattice, the Boltzmann Probability distribution and a Hamiltonian energy function and a random sampling algorithm.

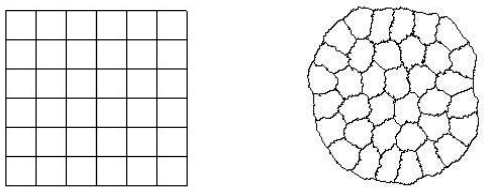
Broadly speaking, the CPM employs the random sampling algorithm to iterate over the sites of the lattice - which we can consider as a elements of a 2D matrix. For each of these sites, the algorithm randomly selects a site around it as potential swap site. This is usually on of the eight neighbours around the site, again we can consider this as one of the eight elements surrounding a matrix entry.

The basis of a swap is that each site has a characteristic energy value based on its neighbours, calculated with Hamiltonian energy function. For the swap to succeed, there must be a reduction in energy after the swap, and this is based on the Boltzmann probability distribution.

More specifically we calculate the energy of the current site before and after the potential swap, and if the difference between these two energies causes a reduction in energy -according to the Boltzmann distribution- then swap is accepted. If we consider the case of cellular structures on the lattice, and we restrict the swaps to occur only on the boundary of neighbouring cells, then ultimately this will lead to a minimized energy state (Fig. 5). In practice this leads to cells exploring their boundaries in an attempt to minimize the overall surface energy. As one would expect, this would lead to the cells having approximately circular boundaries.

To provide a little more depth to this model we will next consider the 4 main areas of this model in a little more detail, namely the Lattice, Hamiltonian, Boltzmann and the random sampling algorithm[2]

[2]Readers Wishing a more detailed review of the CPM are directed towards the bibliography.



a) - Initial b) - Minimized

Figure 5: Illustration of energy minimization from initial state.

The Lattice is the core data structure of the CPM computational model. The lattice named for historical reasons, as it was originally created as a model of ferromagnetism in ferrous crystalline -lattice- materials such as iron. It is this feature of ferrous materials, ferromagnetism, that was the topic of Ernst Isings' 1925 PhD thesis and lead him to develop the model which is commonly referred to as the Ising Model. Latter Renfrey Potts - in his 1952 PhD thesis - attempted to generalize this model further and it is from this model - the Clock/Potts model- that the CPM originated.

The lattice then, $\mathcal{L}$, can be visualized as 2D matrix -or 2D array/grid- of entries, each of which we refer to as a cell-site. We can denote a particular cell-site on the lattice, that is, a grid point as σ_{ij}, were i and j represent the coordinate axis identifying a grid point/cell-site. Further we consider that each σ_{ij} belongs to a particular cell on the lattice, that is there are several σ_{ij} all have the same state, $\sigma_{ij} = k$ were $k \in \{1, 2, 3, \ldots, n\}$ are the possible states or number of cells on the lattice. To simplify this notation, we suggest the following to refer to a cell-site, belonging to the k^{th} cell, σ_{ij}^k. This can be read as the grid point at position (i, j) belongs to cell k. In practice this means all cell-sites belonging to a particular cell, share an unique numeric identifier with which we can distinguish which σ_{ij}^k belongs to which σ^k, $\sigma^k = \{\sigma_{ij} : \sigma_{ij} = k\}$. If we consider Fig. 6, then there are six cells: $\sigma^1, \sigma^2, \ldots, \sigma^6$, each of which constituting several σ_{ij}^k. In addition there is a further σ^k, σ^0. This is not technically a cell, but the substrate medium surrounding the domain of cells and represents an outer border for the domain; one can think of this as a culture medium. We shall discuss this a little further when we come to talk about the Hamiltonian. What is important to note, is that the lattice is a discretization of real cellular phenomena, represented as homogeneous groups of numbers. The boundaries of cells are considered to be simple numeric differences, that can be another cell, or the substrate medium. With basic understanding we next consider how to compute energy on $\mathcal{L}$, that is allow cells to explore their borders.

The Hamiltonian Energy Function is the method by which we calculate energies on $\mathcal{L}$. As indicated in the introduction, σ^k explore their boundaries in an

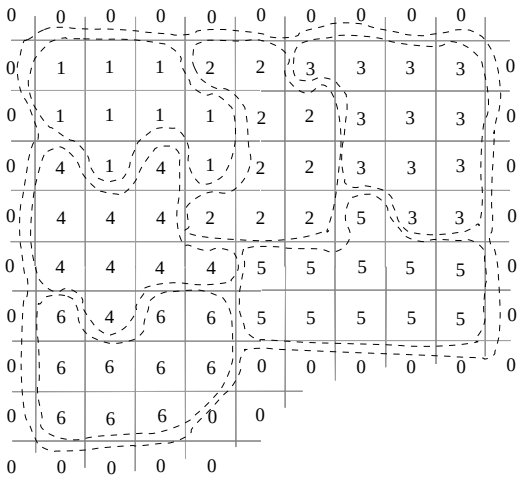


Figure 6: Discretization of cells on the lattice.

attempt to minimize their overall surface energy. This entails σ^k gaining or losing σ_{ij}^k on their boundaries, by calculating the energy differences such gains or losses might produce. If we denote the boundary of the σ^k as Ω_k, then the approach taken is consider the 8 nearest neighbours - for a $\sigma_{ij}^k \in \Omega_k$, from which we select a σ_{ij}^r with random probability such that $\sigma_{ij}^r \notin \{\sigma^k \cup \Omega_k\}$; σ_{ij}^r must belong to another cell. If this gain or loss is to succeed, then it must satisfy the following relation,

$$\mathcal{H}(\sigma_{ij}^r) - \mathcal{H}(\sigma_{ij}^k) > P,$$

where $\mathcal{H}$ is the Hamiltonian, and P is some random probability based on the Boltzmann function. Put simply, the swap will succeed if the calculated difference in energy before and after the swap is acceptable.

To make this a little clearer, consider Fig. 7. Clearly we have 3 cells: σ^1, σ^2 and σ^3, with the dotted lines highlighting their borders. The encircled $\sigma_{ij}^k \in \sigma^3$ denotes a $\sigma_{ij}^k \in \Omega_3$ that is exploring its eight neighbours- the squares. Of these eight neighbours, only those highlighted by hexagons are considered as a swap site. That is, as a potential gain or loss for $\sigma_{ij}^k \in \Omega_3$. Thus, determining whether a swap succeed or fails, can be summarized as by

$$\mathcal{H}(hexagon) - \mathcal{H}(circle) > P,$$

as suggested above. Now that we have an understanding of how energy calculations determine the updates/swaps on $\mathcal{L}$, we shall now introduce the standard definition

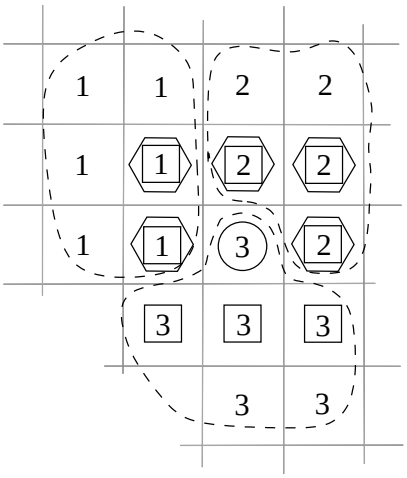


Figure 7: Boundary exploration for cells.

of the hamiltonian, $\mathcal{H}$:

$$\mathcal{H}(\sigma_{ij}^k) = \sum_{<i',j'>} J(\tau(\sigma_{ij}^k), \tau(\sigma_{i`j`}))\delta_m(\sigma_{ij}^k, \sigma_{i`j`}) + \lambda \sum_{types\ \sigma} (a(\sigma^k) - A_{\tau(\sigma^k)})^2\theta(\tau). \quad (1)$$

Rather than discuss this in detail, we will just point out the main features and how they calculate the energy.

Considering the first sum in eq (1), the $< i`, j` >$ represent the sum over the nearest neighbour indices around σ_{ij}^k, that is, the eight squares in Fig. 7. For each of the eight neighbours, $\sigma_{i`j`}$, we compute the bonding energy, J, between σ_{ij}^k and $\sigma_{i`j`}$ as a function of cell types τ. In the language of the CPM, this is the differential adhesivity between different σ^k on $\mathcal{L}$. In simple terms, we can consider this to be how energetically cells of differing types interact; higher J results in increased gains and losses on the boundaries between types, and conversely for low J. Next it ensure cells do not interact energetically with themselves. To prevent this, a modified kronecker function is used, $\delta_m(\sigma_{ij}^k, \sigma_{i`j`})$, with the definition: $\{0 : \sigma_{ij}^k = \sigma_{i`j`}; 1 : \sigma_{ij}^k \neq \sigma_{i`j`}\}$, which essentially results in a zero sum if $\sigma_{ij}^k \in \sigma^k$.

The second sum in eq (1), is concerned with imposing an area constraint. We have seen how cells explore their boundaries by attempting to gain or lose boundary sites, which may lead to a situation where cells on the lattice are consumed by surrounding cells as boundary swaps occur. Clearly the converse is also true, in that cells may grow unchecked as they consume the boundaries of their surrounding cells. This is solved by imposing a reward/penalty for a cell whose area ,$a(\sigma^k)$, deviates from its type dependant target area, $A_{\tau(\sigma^k)}$. That is, the contribution to the cells overall

energy will increase/decrease quadratically, if cells area decreases/increases. How this reward/penalty contributes to the overall energy is by the pre-factor, λ, that specifies the strength of the area constraint. Finally we recall there is a special cell type, σ^0, that models the substrate medium on which the cells reside. Clearly this substrate should not have an area constraint, and thus this sum should contribute for nothing σ^0. This is imposed by $\theta(\tau)$), with the definition: $\{0 : \sigma_{ij}^k \in \sigma^0; 1 : \sigma_{ij}^k \notin \sigma^0\}$.

Boltzmann Probability Function is named after the 19^{th} century Austrian physicist Ludwig Eduard Boltzmann, and provides us with a mechanism to calculate the probability of a particular energy state occurring on the lattice. This is achieved by use of the partition function from statistical physics,

$$Z = \sum_i e^{-\mathcal{H}(\sigma_{ij}^k)/k_B T}, \tag{2}$$

which encodes statistical information of the system -such as energy- as a sum over all possible states; In our case this is clearly the energy states $\forall \sigma_{ij}^k \in \mathcal{L}$. Then for a particular energy state to occur, that is, to calculate the probability for a energy state to occur, we need only take the ratio of the state with the partition function,

$$P(\sigma_{ij}^k) = \frac{e^{-\mathcal{H}(\sigma_{ij}^k)/k_B T}}{Z}. \tag{3}$$

If we now recall from the previous section the relation

$$\mathcal{H}(\sigma_{ij}^r) - \mathcal{H}(\sigma_{ij}^k) > P,$$

which crudely stated the condition for boundary swaps to occur, then in terms of the Boltzmann function we have

$$\frac{P(\sigma_{ij}^r)}{P(\sigma_{ij}^k)} = e^{\frac{-(\mathcal{H}(\sigma_{ij}^k)+\mathcal{H}(\sigma_{ij}^r))}{k_B T}}, \tag{4}$$

which gives us the probability of swap occurring on the lattice.

With our newly acquired knowledge of the embryogenesis and the cellular potts model, we next consider the problem at the heart of this paper, Regression of Henson's Node.

2.3 Regression Of Henson's Node

The regression of Henson's node marks the beginning of the formation of the central nervous system (CNS) during the phase of embryogenesis known as neurulation. During this period, the central body axis extends posteriorly, fueled by newly formed tissue emerging from the region of the node. Broadly speaking, there are two distinct

activities that are taking place: the formation of the neural tube, encompassing spinal chord and CNS, and formation of somitic cells, that will later differentiate to form skeletal muscle, dermis of the posterior and vertebrae (Fig. 8)

The neural cells that form the CNS are produced from the neural stem zone, recruited from the ectoderm of the surrounding neural plate. While the somites are fueled from mesoderm cells emerging from the primitive streak, in a zone called the paraxial mesoderm. This paraxial mesoderm is divided into two broad zones: the presomitic mesoderm, and the somitic stem zone. Together these two activities, production of neural and somitic cells combined with the regression of Henson's node, produce the main dorsal features of the chick.

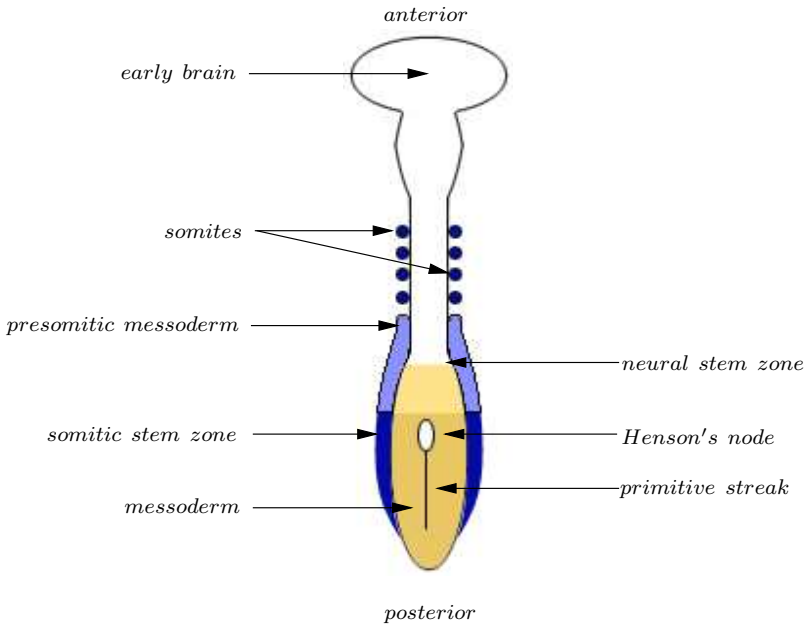


Figure 8: Illustration of early forming central nervous system, during regression of Henson's node.

The formation of the pattern -or patterning- in Fig. 8 that leads to the creation of the central nervous system and somites has been considered by others [2], and we will not be considering it here. We will however, be considering regression of the node and the proliferation of cells in the stem zones, that give rise to an extending body axis, modelled as a simple extending domain of nueral/somitic cells. More specifically, we will simplify our model by only considering a single motile homotypic domain of cells, that proliferates and differentiates to give rise to the progressive formation of a body domain while maintaining the stem zone. To model these features, we will employ the cellular potts model developed by Glazier and Graner [8],with modifications to accommodate the biological features we require.

2.4 Aims Of This Paper

We seek to investigate and model the regression of Henson's node, and in particular the formation of the extending body axis and the method by which the node maintains a consistent size. More specifically, we seek to generalize the region of the node as a motile zone of stem cells, we will refer to as a stem zone, that models proliferation and differentiation to produce cells for the extending body and the stem zone. In modelling these features, we aim to demonstrate how the stem zone size is maintained during extension of the body given biologically realistic hypothesis. These biological hypotheses are derived from observation, were in particular we know that the size of Henson's node does not exhibit any detectable change in size during regression, an maintains a size of approximately 1000 cells.

For our generalized model, we will assume chemotaxis to be the cause of motility and two types of cells: stem cells that remain in the stem zone, and body cell that will form the extending body. We consider stem cells to proliferate thereby growing the stem zone, and balanced by differentiation were stem cells differentiate into cells for the extending body, body cells. We will consider two hypotheses, one based solely on the principles of proliferation and differentiation, and the other based on chemical morphogens, FGF-8 messenger RNA (mRNA), FGF-8 and Retinoid Acid (RA).

Thus the main aim of this paper is to investigate the interplay between movement and differentiation of cells in the embryo, and to develop a mathematical model that could help to uncover mechanisms of chemical regulations during the regression of Henson's node and in particular the formation of the spinal cord.

2.5 Experimental Results.

To try to understand the regulation of the size of Henson's node and the extending body axis, we propose two hypotheses that could account for this in a mathematical model. To investigate this model we derived an implementation of the cellular potts model upon which we produced our simulations. As a first problem we developed and implementation of chemotaxis, that was the base assumption for modelling motility for our hypotheses.

Our first hypothesis, was that a proliferation/differentation alone could maintain the stem zone size and produce the extending body. We considered there to be a stem zone populated by stem cells that proliferated and differentiated into two daughter cells: one for the extending body, and one for the stem zone that would maintain a constant number of stem cells in the stem zone while contributing to the the extending body. While this technique satisfied our basic requirements for maintaining a consistent stem zone size, it produced undesirable results in the form of dissociation in the stem zone. Techniques were considered to remedy result which proved effective however it was ultimately abandoned in favour of a more biologically realistic approach.

Our second hypothesis involved the dynamics of 3 chemicals: a fibroblast growth

factor messenger RNA (FGF-8 mRNA), the fibroblast growth factor protein (FGF-8) and Retinoid Acid (RA). We considered cells in the stem zone to synthesize FGF-8 mRNA and translate and secrete this into the surrounding environment as FGF-8 protein. As before the stem zone was motile, and we observed an posterior to anterior decreasing gradient of FGF-8 with its highest concentration in the anterior of the stem zone as predicted. It was in this region of highest concentration that stem cells would be signalled to differentiate into cells that contribute to the extending body. Our results show that, as predicted, there is a travelling wave co-moving with the stem zone that regulates the size of the stem zone by differentiation. In particular we demonstrated that the dynamics of FGF-8 was the major controlling factor in this regulation, whereas mRNA played a lesser role. Further we showed that varying mRNA produces fragmentation in the anterior of the stem zone, and that our assumption concerning the role RA where, at least in the mathematical model, justified.

3 Modelling Regression On The CPM

The first investigation concerns the motility of cells on the lattice, in particular,the regression of Henson's node. Biologically speaking, this is the behavioural response of an organism - in our case cells- to some directional stimuli in its environment. This stimuli results in a response by the organism to initiate motility, depending on the type of stimuli such as: barotaxis (pressure), thermotaxis (temperature), rheotaxis (fluid flow) or chemotaxis (chemicals). However, it is not clear what stimuli causes the node to regress, that is, exhibit motility in a posterior fashion.

The approach taken, is to assume the presence of some chemical stimuli (chemotaxis)that stimulates motility in some direction. Clearly, from previous discussions, this cannot be random motility. That is, the node is axially constrained, and moves linearly away from the anterior head section (see Fig. 8). However chemotaxis defines a chemical gradient field that may vary in concentration, leading to cells travelling with brownian motion in an attempt to find the highest concentration, which is assumed to be at the source of production. Therefore we consider the concentration of the chemical to produce a kind of *chemostatic* attraction/repulsion field. That is to say, cells will travel directly towards the *"target"*, that is the source of production.

Thus we shall implement chemotaxis as point/target with the following definition:

$$\mathcal{E}_{ch,k} = \beta_k(\vec{x}\nabla(u)), \tag{5}$$

where β_k is a constant describing the chemotactic response of the k^{th} cell to the chemotactic agent, u, and $\vec{x}$ is a vector representing the local displacement of the cell's boundary; Note that, $\beta_k > 0$ corresponds to chemorepulision, whereas $\beta_k < 0$, corresponds to a chemoattractant.

3.1 Implementation Of Chemotaxis On The Lattice ($\mathcal{L}$).

It was shown in the introduction how cells, σ^k, explore their boundaries, Ω_k,in an attempt to minimize there overall surface energy, by attempting to gain or lose boundary cells based on an energy computation. This computation essentially determines whether a gain or loss will occur, depending on whether the energy before or after a gain or loss satisfies the Boltzmann probability function. This can be simplified by saying that if the gain or loss is below some energy threshold, say γ, then it is accepted regardless of any other factor. This leads to a simple mechanism by which we can allow cells to become motile on the lattice, $\mathcal{L}$.

Recalling from Fig. 7, that a cell only considers a potential gain or loss on it's boundary with the boundaries of other cells on the lattice; recall this may include the substrate medium, σ^0. Considering Fig. 9, there is a potential boundary gain/loss between two cells, σ^1 and σ^2, highlighted by the hexagon. In addition, there is a chemotaxis target, n, which is assumed to be some distant point on the substrate; $n = \sigma^k_{ij} \in \sigma^0$.

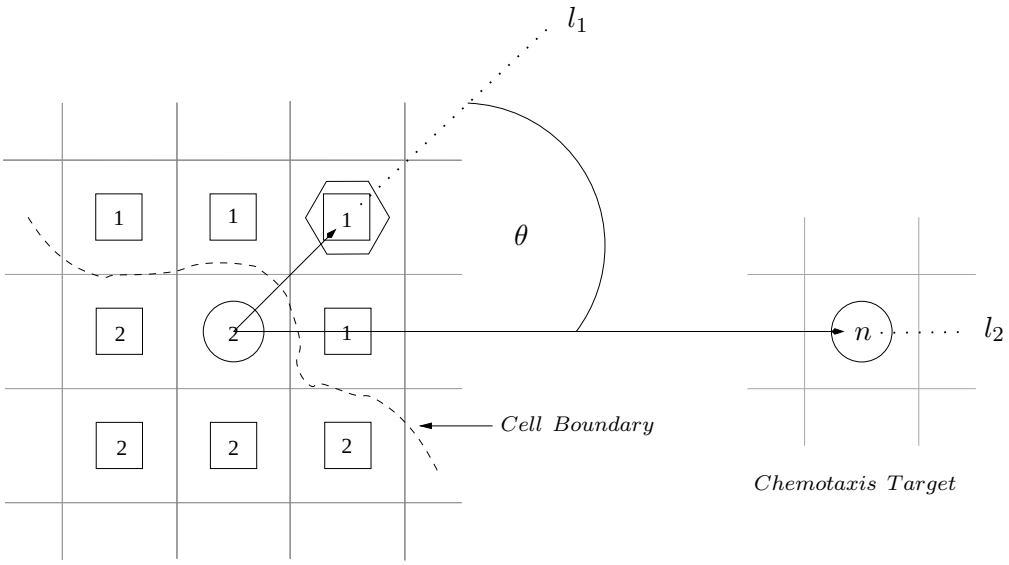


Figure 9: *Dot product with swap site and chemo target, provides chemotaxis based motility.*

Considering the line segments, l_1 and l_2 then, clearly $|l_1||l_2|cos(\theta) = l_1 \cdot l_2$, but more importantly $cos(\theta) = l_1 \cdot l_2$, for l_1 and l_2 having unit length. Since $cos(\theta)$ attains its maximum for $\theta = 0$, then clearly this maximum will be attained for those swaps that have $\theta \to 0$, that is, boundary gains or loses that are in the direction of the

chemotarget. With this, eq (5) can be re-written to give the CPM based equivalent:

$$\mathcal{E}_{ch,k} = \beta_k(l_1 \cdot l_2) = \beta_k(cos(\theta)), \tag{6}$$

that states that the chemotactic energy exerted on the k^{th} cell is proportional to the angle between a swap site and the chemotarget, and the cells response to the chemical, β_k. The significance of this can now be understood in the sense of motility, for γ, eq (6), and eq (4):

$$e^{\frac{-(\mathcal{H}(\sigma^2_{ij})+\mathcal{H}(\sigma^1_{ij}))}{k_B T}} - \mathcal{E}_{ch,k} < \gamma, \tag{7}$$

that states that if $|\mathcal{E}_{ch,k}| << |e^{\frac{-(\mathcal{H}(\sigma^k_{ij}2)+\mathcal{H}(\sigma^k_{ij}1))}{k_B T}}|$, then boundary gains/losses will always succeed in the direction of the chemotactic stimulus, and thus motility will ensue in this direction. However, given this mechanism for motility, it is unclear how the cells on the lattice will respond to the chemotactic energy for varying, β_k, number and size of σ^k.

3.2 Result Of Chemotaxis Simulations On the Lattice.

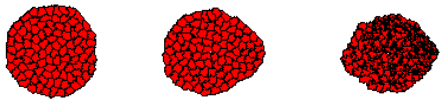
In the previous section a method was developed to enable cell motility on the lattice. In this section we discuss some results of this method in terms of varying: cellular response to chemotactic gradient, β_k and number of cells, to better understand the dynamics of this method. More specifically, we wish to investigate how the velocity of cells, v, changes under various conditions.

Unless otherwise stated, simulations will be conducted on $\mathcal{L}$ with $n \times n$ σ^k, each consisting of $7 \times 7 = 49$ σ^k_{ij}. Velocities were calculated by time-step derivative of an evenly distributed, continuous sampling of the position of the center of mass in the cell domain.

3.2.1 Varying Chemotaxis

Since our method of cell motility relies on - in a sense- interfering with natural energetic development of the a cellular structure on the lattice, it is important that the strength of this interference be managed so as not to damage the morphology of the cells (Fig. 10). This arises because the normal energetic interactions between cells, is overpowered by the energetic response to the chemotactic stimuli, which results in cells becoming distorted. Thus we only consider variances that result biologically realistic morphology. This was achieved by empirical means, running simulations and observing the morphology to determine a sensible upper bound for β_k.

If we consider Fig. 11, we see several simulations with the domain of β_k in $[1, 9]$ producing a range of v. Interestingly, we see that a linear increase in β_k, produces a



Initial *Just Right* *Too high*

Figure 10: Too much chemotaxis destroys cell morphology.

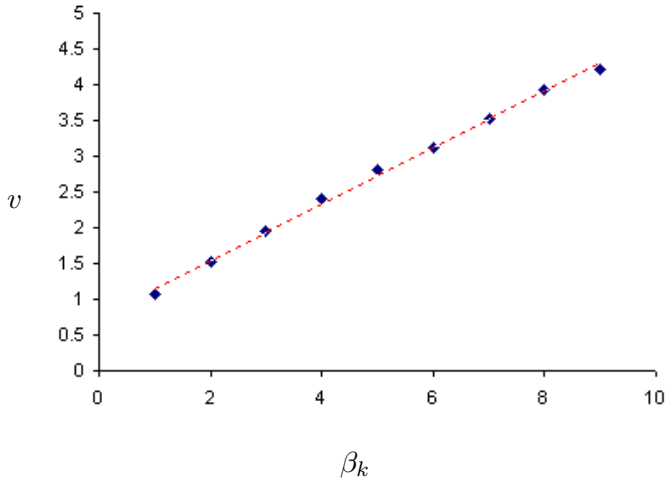


Figure 11: Varying chemotaxis, β_k, produce proportional increase in velocity, v.

linear increase in v, which clearly implies a linear relationship between v and β_k; a pleasing result that affords us a level of predictability for this parameter.

3.2.2 Varying Cell Count

For these simulations, we will only be varying the number of cells on the lattice, and holding β_k constant. As we've seen in the previous section, there is level of predictability in varying β_k, and so we shall, for sake of speed of simulation, set $\beta_k = 6$. With this value of β_k, several simulation were conducted with an $n \times n$ domain of cells, where $n = [1, 2, 3, ..., 8]$. One might consider this to be a small

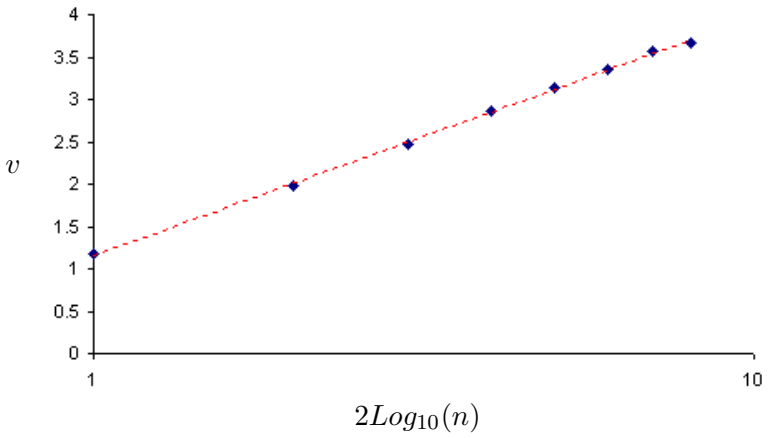


Figure 12: Varying cell counts, n, produce logarithmic increase in velocity, v.

range for n, considering that in the biological context we would expect thousands of cells, but there are two reasons we can expect this to be reasonable. Firstly, if we can infer some relationship for a exponentially growing domain of cells, then it is not unreasonable to expect this result $\forall n$. Secondly, and most importantly, we can expect an equivalent increase in computational complexity, $O(2^{49n})$ for an $n \times n$ lattice. Even for modest n, this can become prohibitively expensive for even the most modern of desktop computer[3]. Fig. 12 displays the result for our simulations, which clearly demonstrates a exponential response in v for varying n.

3.3 Conclusion

A method was developed to enable motility of cells on the lattice, as a first investigation to model regression of Henson's node. This was achieved by taking advantage of the geometric features of the lattice when performing energy computations on cell borders. To understand the dynamics of this method, we briefly investigated how cell

[3]As of writing, this would mean a highend workstation running dual quad core Intel/AMD processors

velocity varies under different conditions. These conditions, while not exhaustive, gave insight into the apparent regularity/predictability of the method, to which it is most likely to be exposed to during later simulations. In this context, the method proves to be acceptable for our requirements.

Lastly It is recognized that we cannot assume this method accurately models the biological reality, that is, the true underlying biological mechanisms by which the node regresses. However we believe the method sufficiently models the dynamics in the context of the mathematical model, and is based on realistic biological assumptions.

4 Proliferation/Differentiation To Control Size Of Stem Zone.

With a reliable model for motility on the lattice -chemotaxis- we next consider the problem of extending the body axis while maintaining the stem zone. Biologically, this means the production of tissue in the stem zone in the region of Henson's node, that progressively forms as the node regresses while at the same time ensuring the stem zone size remains constant (Fig. 13). However, the underlying processes that

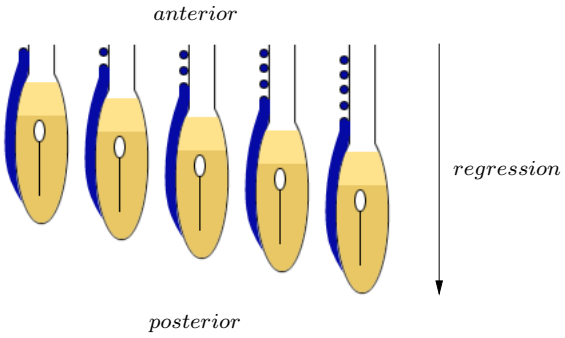


Figure 13: *Progressive extension of body axis.*

constitutes the progression are not clearly understood, and so we wish develop a model under reasonable hypotheses. In this section, the hypothesis is that cellular proliferation and differentiation are the fundamental processes that drive the extension of the body axis and can account for maintenance of the stem zone. That is, cells in the stem zone proliferate and differentiate to produce cells for the forming body, and for the regressing node. In particular, it is the differentiated cells that will contribute to the forming body and become immotile, while undifferentiated cells will remain motile in the stem zone (Fig. 14).

Clearly the justification for this approach, is derived from the fact that the number of cells in the stem zone must remain constant, or at least maintain a consistent size. The assumption being undifferentiated cells will maintain the stem zone size

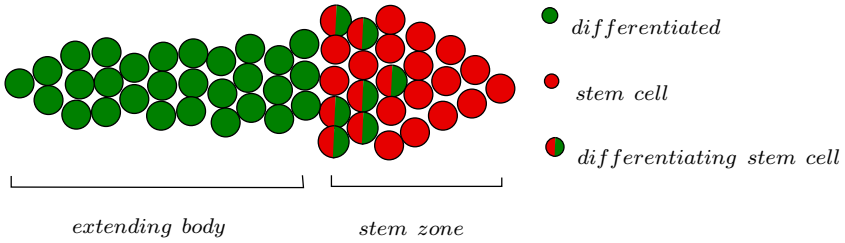


Figure 14: *Illustration of stem zone extending body axis.*

and differentiated will form the body. Therefore, our approach is based around modelling two biological features: proliferation and differentiation, in an attempt to extend the body axis while and maintaining the stem zone.

4.1 A Model Of Proliferation

A feature of all living organisms is growth. This growth takes place at various stages of the organisms life and for numerous different reasons and, in general, is divided into three main strategies: Cell Enlargement, Accretion and Proliferation. Enlargement, as its name suggests, is where cells increase in mass and fuse with other cells in their vicinity, as in the case of muscle cells. The second strategy, Accretion, is where cells secrete large quantities of extracellular matrix thereby enlarging their boundary as in the growth of bone and cartilage, etc. The last strategy, and indeed the most common is by Proliferation, where growth is by an increase in the number of cells characterized by the mitotic *cell-cycle*.

In eukaryotic cells - those possessing a genetic nucleus - the mitotic cell-cycle maps out the stages over which a cell divides into two identical daughter cells. Broadly speaking, this occurs in two stages: *Interphase* and M phase.

Interphase represents the majority of the lifespan -approximately 24 hours in humans- of a cell, during which it performs its intended functions: protein synthesis, gaining nutrients and replicating DNA. Over time the cells mass increases in preparation for cell division and once this preparation is complete the cells enters the M phase. The M phase consists of several stages over which the cell divides into two identical daughter cells, but in general there are two distinct activities: *Mitosis* and *Cytokinesis*. Mitosis is the process by which a cell duplicates the chromosomes in its nucleus into two identical nuclei, after which Cytokinesis divides the nuclei, cell contents - cytoplasm, organelles, mitochondria - and cell membrane into two equally sized daughter cells (Fig. 15). At the end of the cell-cycle the two daughter cells are identical to the parent cell, and each begins the mitotic cell-cycle all over again.

From the biological explanation, three main features are present that define the characteristics of proliferation: Increase in mass, mitosis and cytokinesis. From a CPM point of view, mitosis is clearly beyond the scope of the model and contributes little or no quantifiable information; our investigation is centered around morpho-

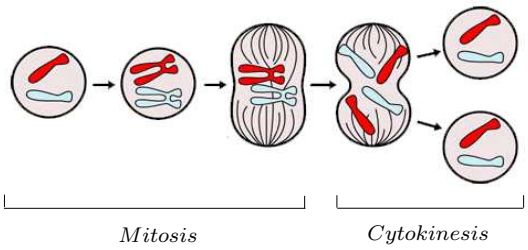


Figure 15: *M-Phase* Of Cell Division.

logical features. What remains is cytokinesis and increase in mass.

4.1.1 Modelling Cytokinesis

As stated, cytokinesis is the process by which a parent cell is divided into two daughter cells. Or said another way, it the physical process of cleavage that produces two daughter cells, each approximately half the mass of the parent. Achieving this with the CPM, requires us to consider a σ^k as having a center of mass (COM), and using this to partition the cell (Fig. 16). Finding the COM for a σ^k is a trivial matter and can be defined as

$$(\bar{i}_k, \bar{j}_k) = \left(\frac{\Sigma \sigma^k_{ij} i}{\Sigma \sigma^k_{ij}}, \frac{\Sigma \sigma^k_{ij} j}{\Sigma \sigma^k_{ij}} \right), \tag{8}$$

which is the standard form for describing the COM for 2D shape, were $\sigma^k_{ij} \in \sigma^k$ and i and j represent the indices for their respective axis. With this representation for COM, we can partitioning the $\sigma^k_{ij} \in \sigma^k$ with respect to the $\bar{i}_k$ or $\bar{j}_k$ axes(Fig. 16). Thus we can describe the two new daughter cells of a parent cell with the partitioning axis $\bar{i}_k$ as

$$\sigma^k \longmapsto \begin{cases} \sigma^{k,1} = \sum \sigma^k_{ij} \in \sigma^k, & j < \bar{j}_k \\ \sigma^{k,2} = \sum \sigma^k_{ij} \in \sigma^k, & j > \bar{j}_k \end{cases}, \tag{9}$$

and equaly with the partioning axis $\bar{j}_k$ as,

$$\sigma^k \longmapsto \begin{cases} \sigma^{k,1} = \sum \sigma^k_{ij} \in \sigma^k, & i < \bar{i}_k \\ \sigma^{k,2} = \sum \sigma^k_{ij} \in \sigma^k, & i > \bar{i}_k \end{cases}, \tag{10}$$

which we can interpret as the cell, σ^k, maps to two daughter cells $\sigma^{k,1}$ and $\sigma^{k,2}$, under mitotic cell division, with the partitioning axis, $\bar{j}_k$ or $\bar{i}_k$. It should be noted that while we have chosen to partition the cells in this manner, it is clear there are an infnitie number of partitions possible through the center of mass, all equally

valid, however we take an approach that is clearly simpler to implement. To simplify our lives further we shall adopt a new notation that is not quite so cumbersome, so for future prescriptions when referring to cell division, we shall use the shorthand notation for such a mapping:

$$\sigma^{k\mapsto\{1,2\}}, \tag{11}$$

were it is implied that $\bar{j}_k$ is the partitioning axis. Clearly this method may produce daughter cells that are of unequal size due to the concavity of a given dividing cell, but it is not unusual in the biological setting. In particular, biologists refer to this as asymmetric cell division, and can refer to either uneven mass or genetic material distribution in the daughter cells. Thus we have a method for simulating the division

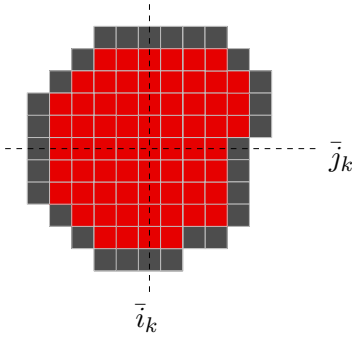


Figure 16: *Axial partitions through center of mass in preparation for cell division.*

of cells on the CPM, and more importantly we have a biologically realistic result. With this method of division, we next consider how we lead to this division by an increase in mass.

4.1.2 Increasing Cellular Mass

As previously suggested, proliferation is characterized by a domain of cells undergoing the mitotic cell-cycle to enable growth. An important feature of this growth, is that cells increase there mass over some period of time before undergoing cytokinesis or cell division. However proliferation is not a systematic process, and cells do not all follow the same growth clock. That is, there is an element of randomness to proliferation that we need to consider.

Clearly if we are to reflect the true biological system, then we need to consider introducing a random element to our growth algorithms. To do this we introduce a random variable, p, which we assume, for simplicity, follows a discrete uniform distribution, $p \in \{1, 2, \ldots, n\}$. That is, we consider a simple "roll of the dice" probability, where the dice in this case has n faces each with $\frac{1}{n}$ chance of occurring. Since we assume proliferation, or division of a cell, is dependant on a critical mass, and we assume an attempt to increase this occurs sequentially on the CPM, we can

vary the randomness of proliferation by simply allowing this increase if $p = 1$, since the $P(p) = \frac{1}{n}, \forall p \in \{1, 2, 3, \ldots, n\}$. Thus we can vary the randomness of increases in mass by varying n; number of faces on our dice if you will.

To implement growth on the CPM is a trivial matter, since it affords us this capability with little modification. If we consider again the standard Hamiltonian for the CPM, eq (1), and more specifically the second sum,

$$\lambda \sum_{types\ \sigma} (a(\sigma^k) - A(\tau(\sigma^k), t))^2 \theta(\tau),$$

then clearly we can manager the growth of cell by updating modification to its target area, $A(\sigma^k, t)$. To accomodate this modification we can modifiy this sum to reflect the temporal aspect of target area, that is, we wish to describe how target area varies over time. We know from the previous discussion this will take place every $\frac{1}{n}$ time steps, and thus we can describe the average change in a cells target area as:

$$A(\tau(\sigma^k), t) = A_0(\tau(\sigma^k), t)t + \frac{1}{n}(t - t^*)$$

where t^* represents the time from the last increase in mass, or proliferation step.

Thus we can model the increase in a cells mass by updating it target area with a basic random probability.

4.2 Modelling Differentiation.

Differentiation is a function of cells that allow them to produce different cell types. These types of cells are usually referred to as stem or progenitor cells, and are given names based on the range of different cells they can produce. For example *totipotent* cells are found in the zygote and have the ability to differentiate into any kind of cell, whereas *unipotent* cells are restricted to producing only a single cell type. Whatever the potency, the prominent feature of stem cells is that when they divide -undergo mitotic cell division, one of the daughter cells is differentiated into a new cell type, while the other remains a stem cell (Fig. 17). This allows stem cells to repeatedly

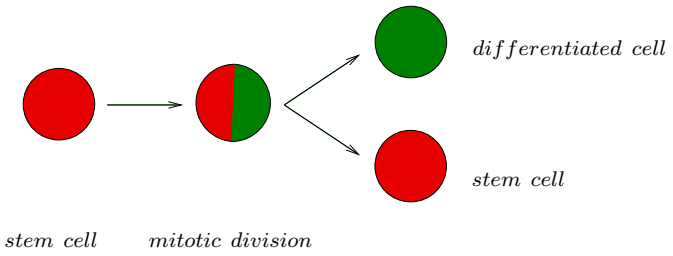


Figure 17: *Illustration of mitotic division of unipotent stem cell.*

renew themselves, while producing cells of varying types.

In the context of our model, stem cells are unipotent and make up the stem zone mass. When they divide, they produce two daughter cells, one of which is a stem cell that remains in the stem zone, and the other a differentiated cell that will contribute to the formation of the extending body, with the crucial difference that stem cells exhibit motility under the influence of chemotaxis, while the body cells do not.

With this stem cell model, we assume the stem zone mass will remain constant, a feature we require. For simplicity, we shall heronin refer to differentiated cells a body cells, $\sigma^{body} = \{\sigma^k : \tau(\sigma^k) = body\}$, and those that remain in the stem zone as stem cells, $\sigma^{stem} = \{\sigma^k : \tau(\sigma^k) = stem\}$. Recalling the discussion of proliferation, we can define the behavior of a σ^{stem} as,

$$\sigma^{stem \mapsto \{stem, body\}}$$

and for a σ^{body},

$$\sigma^{body \mapsto \{body, body\}},$$

and as before, we can interpret this as saying a σ^{stem} maps to two daughter cells under cell division: one daughter as σ^{stem} and the other a σ^{body}. And a σ^{body} maps to two daughter cells under cell division, both of which are σ^{body}. Implementing differentiation on the CPM requires extension of the types supported on $\mathcal{L}$, and so we introduce two new types giving, $\tau \in \{substrate, stem, body\}$, were we defined *substrate* previously to be σ^0, thus $substrate = \tau(\sigma^0)$

4.3 Considering Differential Adhesion.

Cells on the lattice are modelled as homogenous domains of cell sites where boundaries are differentiated by simple numerical values. Cells evolve and exhibit motility as a result of exploring their boundaries and determining gains or losses to the boundaries by calculating energy such changes might produce. This implies we can influence the strength of contact between cells by increasing or decreasing how energetically cells interact on their boundaries. This leads the concept of differential adhesion, where we can control how "sticky" the bonds are between cells. This is accounted for in the Hamiltonian function (eq (1)) as the pre-factor

$$J(\tau(\sigma_{ij}), \tau(\sigma_{i`j`})),$$

which gives an energy value as a function of cell types , τ, that are being considered in boundary gain or loss.

The practical implication of this, is that we can reduce the bonds between different cell types, and increase the bonds between same cell types. Considering the previous discussion, then we have two cell types of interest, σ^{body} and σ^{stem},

where we suggested σ^{stem} are motile cells while σ^{body} are not. Thus our aim is to facilitate the motility of σ^{stem} by reducing the bonding potential with σ^{body}

$$J(stem, body) << J(body, body)$$

and

$$J(stem, body) << J(stem, stem).$$

The assumption being that as proliferation and differentiation occurs, σ^{stem} will be uninhibited by σ^{body}, and in essence will allow the σ^{stem} to move freely among σ^{body} during motility. However the downfall of this approach is we have no biological premise for it. It is true, differential adhesion occurs biologically, but we have no evidence to suggest it occurs with the bounds of our model, and thus it was considered purely in the interest of completeness.

4.4 Model Results

4.4.1 Cytokinesis.

Recalling that cytokinesis is the process that physically divides a cell, we present the results of this on the CPM for an exaggerated cell. This was modelled on the CPM as a single cell on the lattice with a cell area of 50×50., where the simulation was performed twice to demonstrate $\bar{i}_k$ and $\bar{j}_k$ partitioning. From Fig. 18 we can see two cells, one demonstrating division horizontally, and the other vertically.

$\bar{i}_k$ *partitioning* $\bar{j}_k$ *partitioning*

Figure 18: *Exaggerated cell demonstrating line of cytokinesis.*

4.4.2 Cell Differentiation.

To demonstrate cellular differentiation we ran simulations of varying cell areas, but present an exaggerated cell of 50×50 area to demonstrate differentiation occurring at the point of cytokinesis. Considering Fig. 19, a single stem cell (red) has undergone cytokinesis to produce a daughter stem cell (red) and a differentiated daughter cell (green). Clearly in a realistic simulation there would be potentially hundreds of cells.

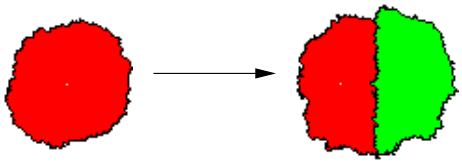


Figure 19: *Exaggerated cell demonstrating proliferation with differentiation.*

4.4.3 Proliferation.

For proliferation numerous simulations with varying cell counts and areas were ran successfully. To illustrate this success we show a general result, illustrated in Fig. 20 were, as intuition would suggest, proliferation follows a simple 2^n growth rate for a single cell.

Figure 20: *Example output demonstrating proliferative growth.*

4.4.4 Stem Cell Proliferation.

For these simulations we brought together the result found in Section (4.4.3) and Section (4.4.2) to demonstrate a non-motile proliferating, differentiating domain of cells. As with other results, various simulations were conducted for varying cell size and cell counts. Fig. 21 illustrates output of such a simulation for 5×5 cells each having 5×5 area, were green cells represent a differentiated daughter due to cell division and red are stem cells.

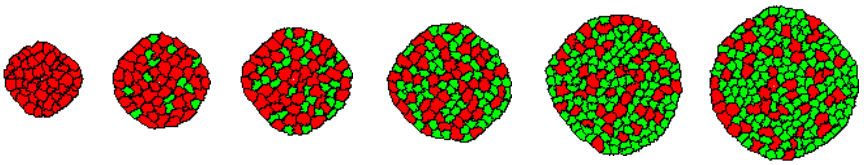


Figure 21: *Example output demonstrating stem cell proliferation/differentiation.*

4.4.5 Extending Body.

For this result our initial conditions was such that we had a 5×5 domain of cells each having 5×5 area, and that the cells where moving in response to chemotaxis, and proliferation and differentiation where both in effect. This result brings together all that has come before in an attempt to model the extending body axis, while maintaining the stem zone. An exhaustive number of simulations were conducted, but unfortunately we could not find an permutation that lead to a satisfactory result. As illustrated in Fig. 22, we can see the stem zone (red cells) has all but completely dissociated along the extending body (green cells).

Figure 22: *Output demonstrating failure due to cellular dissociation in the stem zone.*

4.4.6 Differential Adhesion.

This final result was to establish whether a result could be achieved by varying standard CPM parameters, in particular differential adhesion. However while an acceptable result was achieved (Fig. 23) the complexity in finding an acceptable configuration of the parameters was unjustified from a biological point of view, thus it served only as an exercise in the CPMs features.

Figure 23: *Example output demonstrating successful body extension while maintaining the stem zone mass using differential adhesion. In this case we considered varying bonding energies between cells, essentially reducing the friction between stem cells (red) and body cells (green).*

4.5 Conclusion

The model outlined in this section, provides a simplification to the problem of modelling the regression of Henson's node, by only considering two cell types: σ^{stem} and

σ^{body}. We assumed that there is stem zone mass that generalizes the region of Henson's node, and is sensitive to the chemotactic stimuli and is populated by σ^{stem}. Furthermore, that σ^{stem} divide and differentiate to give rise to σ^{body}, that are insensitive to chemotactic stimuli, and thus do not exhibit motility. It was the immotility of σ^{body}, that were assumed to produce the forming body and were, "left behind", as the stem zone regressed thereby achieving two things: progressive formation of the body axis and maintaining stem zone size. (Fig. 14).

In the first instance we demonstrated the success of the proliferation technique, in demonstrating biologically pleasing cell growth. We also demonstrated a successful method for modelling differentiation in stem cell division. We can say categorically that these methods were successful and their efficacy and indeed was born out in simulations. However, while we were able to maintain the stem cell population and to demonstrate an extending body axis, we were unable to maintain an homogenous stem zone, and in this regard we have not succeeded. Lastly, we cannot say this method will not produce satisfying results, we believe it to be only a question of whether the solution to the problem is biologically realistic. It is with this in mind we consider, a slight modification to this method, as an attempt find a result.

5 A Modified Proliferation/Differentiation Method.

Caveat lector: This section is concerned with developing a solution to the failure encountered in the previous section. There is no premise for what follows, biologically or otherwise, and can be considered as an attempt to salvage our previous failure.

With this in mind we will be considering a single solution/fix to the previous failure. We recall this failure was due to dissociation of cells in the stem zone caused by proliferation; the stem zone was literally dispersed by the production of σ^{body}. Given this, it is logical to assume that the solution needs to counteract this diffusion, and introduce a force that acts as an attractant, keeping σ^{stem} in the stem zone.

5.1 A Center Of Mass Attractor.

The principle behind this method, is that stem cells are attracted towards a center of mass (COM) in the stem zone, by a circularly symmetric inverse force. This implies that as cells disperse from the COM, an energy penalty is exacted driving them back. With this it is assumed that as σ^{stem} proliferate and differentiate, the daughter σ^{body} will be expelled from the stem zone by the forces driving σ^{stem} towards the COM.

From a CPM point of view, we need to implement two features: computing the COM for the stem zone, and implementing the inverse force. Of the two, COM has already been considered in Section (4.1.1), where we computed the COM for individual cells. It is therefore a simple matter of taking the COM of the COMs of the individual cells, to find the COM of the stem zone.

To implement the inverse force, we shall modify to the implementation of chemotaxis suggested in Section (3.1). We recall this method used the magnitude of the *cosine* between a potential swap site and chemotaxis target vectors, to produce a motile response in the direction of the chemotaxis target (Fig. 9). The modification

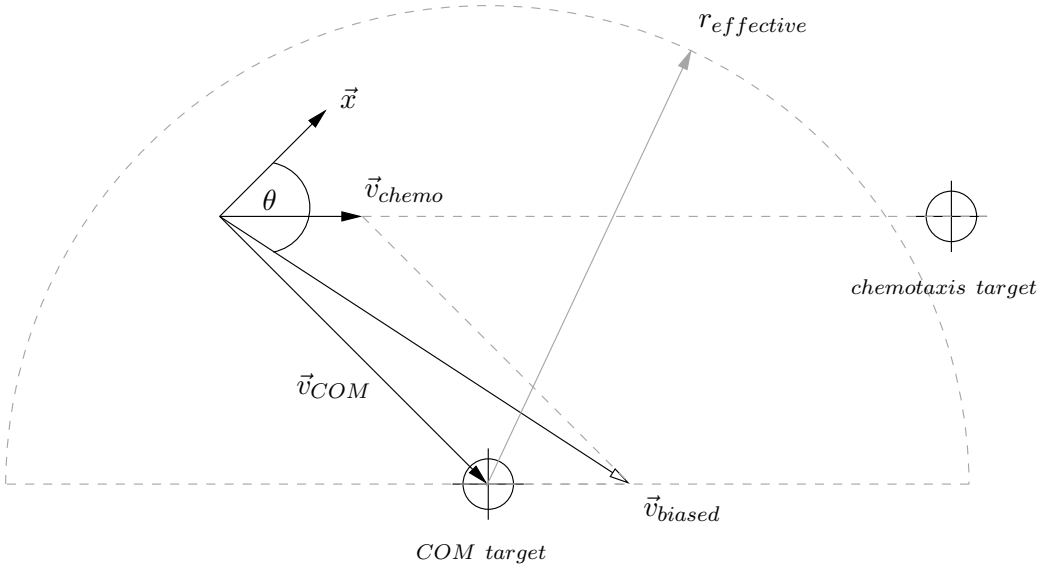


Figure 24: *Illustration of center of mass(COM) biased motility - lattice omitted for clarity.*

we shall employ, is to add a secondary target vector, $\vec{v}_{COM}$, representing the center of mass for the stem zone in addition to the chemotaxis vector, $\vec{v}_{chemo}$ (Fig. 24). This vector, $\vec{v}_{COM}$, will produce a directional bias towards the COM target. To compute this bias, we introduce the scalar quantity, $r_{effective}$, that defines an effective radius of the stem zone. This quantity is an initial condition for the stem zone, and is calculated at the start of a simulation before proliferation/ differentiation begins; this is acceptable because the mass of the stem zone is assumed to be constant in time. Thus we can define the vector that is biased towards the center of mass as

$$\vec{v}_{COM} = \alpha(\vec{r} - \vec{r}_{COM}),$$

were $\alpha = \frac{1}{r_{effective}}$, $\vec{r}$ is the boundary site we considering a swap for and $\vec{r}_{COM}$ is the center of mass of the stem zone. In plain language, the farther away from the COM we are, the greater the magnitude of $\vec{v}_{COM}$. With these quantities we can define the COM vector:

$$\vec{v}_{biased} = \vec{v}_{COM} + \vec{v}_{chemotaxis}, \tag{12}$$

with the inference that as $\|\vec{v}_{COM}\| \to 0$, $\vec{v}_{biased} \to \vec{v}_{chemo}$, and conversely, $\|\vec{v}_{COM}\| \to r_{effective}$, $\vec{v}_{biased} \to \vec{v}_{COM}$. Thus we can modify eq (6) to provide our new COM

biased motility:

$$\mathcal{E}_{ch,k} = \beta_k(\vec{x} \cdot \vec{v}_{biased}) = \beta_k(cos(\theta)). \tag{13}$$

where cells in the stem zone will be attracted to COM with inverse proportion to the effective radius and thereby ensuring motility towards the chemotaxis target while maintaining the stem zone mass.

5.2 Results.

On running simulations with varying cell counts and cell size, we found that this method successfully maintains the stem zone while progressively forming the extending body axis. Fig. 25 illustrates such a success where an initial domain of 5×5 cells having area 5×5 produced the extended body (green cells) from the stem zone (red cells).

Figure 25: *Output demonstrating successful center of mass extended body.*

5.3 Conclusion.

In an attempt to solve the a problem encountered in Section (4)- maintaining stem zone size, a method was developed that employed a center of mass bias vector to keep stem cells in the region of the stem zone. The assumption being, this vector would apply a inverse force, proportional to a cells distance form the center of mass. From the results it would appear this assumption was correct, and the desired result was achieved. However as indicated, it is not certain whether this method is biologically realistic, and as of writing there is no evidence to suggest the contrary[4]. Therefore it remains an interesting result, in so much as it is efficacious. In light of this moot success, we next turn to a more sophisticated method, that is more biologically realistic in its assumptions.

6 FGF-8/FGF mRNA Regulation Of Stem Zone Size.

Modeling the regression of Henson's node, and in particular the formation of an extending body axis while maintaining the stem zone are the main aims of this paper. It has been demonstrated that proliferation and differentiation, combined with

[4]Currently Dr Ruth Diez del Corral is conducting in-vivo experiments that may shed light on this ambiguity in the future.

chemotaxis is problematic at best on the CPM, and required the use of unprecedented methods to achieve a satisfactory result. This led to reconsidering what is possible with the CPM, and to consider an implementation that conforms to current biological thinking [2, 3, 6]. This thinking involves modeling several biological chemicals: fibroblast Growth Factor protein (FGF-8), FGF messenger RNA (mRNA) and Retinoid Acid (RA), that are assumed to play a part in the patterning and extension of the vertebrate body axis [2]. In general, this involves an FGF-8 gradient that, in its highest concentration in the anterior stem zone, induces differentiation of the cells for the production of the extending body axis, while allowing the stem zone to proliferate and maintain its size.

Rethinking the model in this way, requires developing sophisticated methods the canonical cellular potts model (CPM) does not offer. When considering terms such as chemicals and gradients, the implication is that diffusion must be considered to model the dynamics these terms imply. These dynamics are complicated further, by the necessity for interdependency in the kinetics of the chemicals; FGF mRNA is translated into FGF-8. Fortunately we are describing a situation that has a mathematical expression encapsulating these ideas, and is referred to as the *reaction-diffusion* equation, and in general takes the form:

$$\frac{\partial \vec{\psi}}{\partial t} = D\nabla^2(\vec{\psi}) + \vec{\kappa}\mathcal{K}(\vec{\psi}).$$

We will discuss this further when we consider its implementation on the CPM, but for now it is sufficient to remark that, $\vec{\psi}$ represents a vector of chemical concentrations - mRNA, FGF-8 and RA. D is a diagonal matrix of diffusion coefficients, $\nabla^2(\vec{\psi})$ is the laplacian representing diffusion and $\kappa\mathcal{K}(\vec{\psi})$ represent the reactions/kinetics of the chemicals. Thus we shall proceed by considering further the biology of our problem, then considering the form of the solution in terms of implementing reaction-diffusion equation on the CPM, and finally showing some results.

6.1 Chemical Dynamics of Body Extension.

Much of the chemical induced patterning that takes place during the extension of the body axis is not fully understood. The expression of *c-hariy 1* gene that sweeps from the posterior to the anterior of the presomitic mesoderm, takes approximately 90 minutes, which is the time taken for a somite to form, but the connection is not clear. However, what is know, is that this *segmentation clock* along with the fibroblast growth factor, FGF-8, determines the timing and positioning of the somites. Of these two features, it is the production of FGF-8 that is of interest, and is central to the model being developed.

FGF-8 is know to exist in and around the region of Henson's node, where its concentration is highest. FGF-8 degrades in an anterior to posterior fashion, producing a gradient that ends -approximately- around the presomitic mesoderm where

the last somite formed (Fig. 26). As the node regresses, along with the gradient of FGF-8, it is assumed that somites form in this region of low FGF-8, below some critical lower threshold. As the somites form they begin to produce and secrete Retinoid acid (RA), which is know to antagonize FGF, and is assumed to prevent the elongation of the presomitic region. The apparent gradient observed around

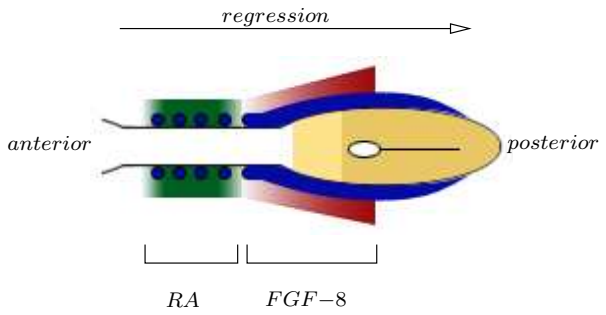


Figure 26: *Illustration of FGF-8 and RA Gradients in the stem zone.*

the node, is a result of the cells in the region synthesizing FGF messenger RNA (mRNA), and is known to degrade in the cells as they leave the region, and enter the presomitic mesoderm. This produces a gradient of intracellular mRNA, that in turn is translated and secreted from the cells, leading to a gradient of FGF-8 protein.

We can infer from this biological process, there is an interplay between three chemicals: RA, nRNA and FGF-8. The dynamics of this process are that the diffusion and kinetics of RA, are entirely dependant on the diffusion an kinetics of FGF-8, which in turn are dependant on the kinetics of mRNA. This last point is important, mRNA does exhibit diffuse behavior because it is synthesized entirely within the cell. More specifically, the synthesis of mRNA can be considered as switch within the producing cell, that when turned off, leads to its decay. That is, it is a chemical kinetics rather than a diffusion problem.

Thus, in terms of the CPM, there are several problems to model: The kinetics of mRNA of cells in the stem zone, the kinetics and diffusion of FGF-8 dependant on mRNA and the kinetics and diffusion of RA in the extending body axis, dependant on FGF-8.

6.2 CPM Implementation of Chemical Dynamics.

Before discussing the implementation of chemicals on the CPM, it is useful to consider the features that have been developed thus far, and how they can contribute to the problem at hand. One feature that is clearly required, is the ability to produce motility on the lattice through chemotaxis. However center of mass will not be considered, the assumption being chemical dynamics will negate it. Clearly the production of cells is important in producing the extending body which implies

differentiation, thus both proliferation and differentiation will be required. However it should be clear that, while differentiation, proliferation and chemotaxis will play an important role in the model, previous failures suggest new strategies in their use.

6.2.1 Outline Of Solution.

The approach will be to consider a motile domain of cells on the lattice, composed entirely of stem cells, σ^{stem}. This domain of cells will be referred to as the stem zone, $\mathcal{S}$, and will be proliferating. However in a change to the previous method, σ^{stem} will not exhibit differentiation under mitotic division, that is, $\sigma^{stem \mapsto \{stem,stem\}}$. The assumption being that dissociation cannot occur in $\mathcal{S}$, if differentiation is not present.

As implied by the biology, σ^{stem} are saturated with mRNA, which is translated and secreted as FGF-8. We assume that as the level of FGF-8 reaches a critical upper threshold, μ_{high}, σ^{stem} will differentiate into body cells, σ^{body}. It is important to note that differentiation of σ^{stem} into σ^{body}, does not involve mitotic division as it did previously; we can think of this as a spontaneous change in character from $\sigma^{stem} \rightarrow \sigma^{body}$ (Fig. 27). This change in character will cause σ^{body} to cease the production of mRNA thereby leading to a decrease in FGF-8 concentration. This is

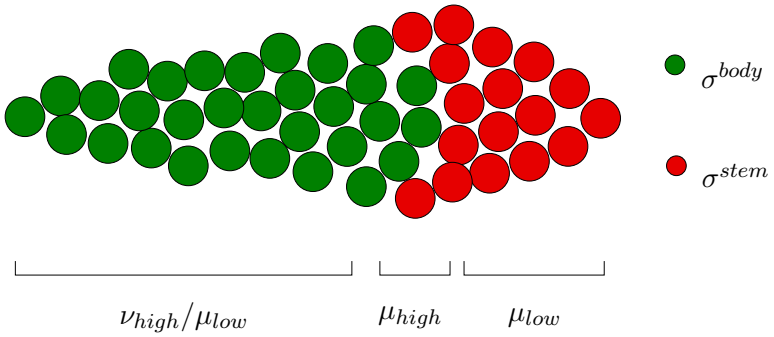


Figure 27: *Illustration of chemical thresholds: RA(ν). FGF(μ).*

the most important assumption for the model, in that the highest concentration of FGF-8 will be anterior of the $\mathcal{S}$, which defines the production region for σ^{body}. If this assumption is born out in simulations, we expect progressive formation of the extending body axis to occur ***only*** in the anterior of the $\mathcal{S}$. More importantly, we expect the stem zone to maintain a consistant size.

Finally as the level of FGF-8 falls to a lower critical threshold, μ_{low}, σ^{body} will start to secrete RA. As suggested in the biology, RA is assumed to prevent the $\mathcal{S}$ from extending. This is achieved by assuming an upper critical threshold for RA, ν_{high}, that above which σ^{stem} will differentiate into σ^{body}, thus preventing $\mathcal{S}$ extension.

What we have described is referred to as a traveling wave front. If for a moment we disregard mRNA and RA, then our solution assumes the presence of a stationary

(Turing) pattern of FGF-8. More specifically, the concentration curve exhibited by FGF-8 around the node as it regresses, is assumed to be constant. This typically arises in reaction-diffusion problems, where chemicals may inhibit or activate the production of each other, leading to pattern formations. In biology this could be responsible for certain morphological features, where reaction and diffusion of specific morphogens create patterns of cartilaginous growth, or it may be responsible for the periodic patterns of pigmentation observed in the animal kingdom. Whatever the result, they are encapsulated in the reaction-diffusion equation,

$$\frac{\partial \vec{\psi}}{\partial t} = D\nabla^2(\vec{\psi}) + \kappa\mathcal{K}(\vec{\psi}), \tag{14}$$

where $\vec{\psi}$ is a vector of chemical concentrations that are assumed to varying in time and space. D is a diagonal matrix of diffusion coefficients, $\nabla^2(\vec{\psi})$ is the laplacian representing diffusion of the chemicals and $\mathcal{K}(\vec{\psi})$ represents the chemical reactions or kinetics between the chemicals, varied by κ.

Clearly there are 3 partial differential equations, each modeling the behavior of a specific chemical. Each of the equations can be considered as having two components: diffusion and kinetics. More importantly, we can consider the components independent of each other, and thus can be treated separately. Specifically, the numerical/analytical solution of diffusion of a chemical, does not depend on the kinetics of that chemical, and vice versa. Therefore we will proceed under this assumption, and define and solve these problems independently, and aggregate the solutions to solve the entire problem. Thus in general, a solution to the following equation is sought

$$\frac{\partial \psi_c(x,y,t)}{\partial t} = D_c(\Delta\psi_c(x,y,t)) + \kappa_c\mathcal{K}_c(\psi_1(x,y,t), \psi_2(x,y,t), \psi_3(x,y,t)), \tag{15}$$

for each chemical denoted by the subscript c and $1, 2, 3$ subscripts represent the 3 chemicals in our problem, that is, the kinetics of each chemical depends on the concentrations of the other chemicals. Analytical solutions of the diffusion equation can be obtained by various methods with varying degrees of sophistication: fundamental solutions involving Green's functions and linear differential operators, or Fourier series solutions involving techniques such as separation of variables appealing to linearity. Our solution however, will take a more direct approach involving finite difference methods (FDM), which allow the derivation of a discrete form suitable for the lattice. We make the assumption that diffusion is the same for all chemicals, thus one specific solution fits all. However the second problem of kinetics defines the relationships between chemicals, and so it this we shall consider first.

6.2.2 Implementing Chemical Kinetics.

It should be clear from previous discussions, that there is level of dependency between production and decay-kinetics- of chemicals in the model. Succinctly: RA is

dependant of FGF-8, FGF-8 is dependant on mRNA. It is these dependencies we now define in terms of the CPM, and will allow to us model kinetics of chemicals.

The production of mRNA in $\mathcal{S}$, is the first chemical on which all others depend, in so much as FGF-8 has a direct dependency whereas RA an intransitive dependency. Thus it is logical to consider its definition first. From the biological description, mRNA is only produced in σ^{stem}. Further we assume that the concentration of mRNA is produced with a constant rate equal to 1 and that it decays equal to this and dependant on the current concentration.

With this description, we can formally describe the kinetics of mRNA as

$$\mathcal{K}_{mRNA}(\sigma_{ij}^k) = \begin{cases} 1 - \mathcal{C}_{mRNA}, & \tau(\sigma_{ij}^k) = stem \\ -\mathcal{C}_{mRNA}(\sigma_{ij}^k), & \tau(\sigma_{ij}^k) \neq stem \end{cases}, \tag{16}$$

were we note the kinetics are now in terms of lattice sites σ_{ij}^k and τ is the type system employed by the CPM on the lattice, $\mathcal{L}$. What we can infer from this definition, is that as σ^{stem} differentiate into σ^{body} as $\mathcal{S}$ regresses, we should observe a decreasing concentration of mRNA; what we require from the biological description.

Continuing the dependencies, we know FGF-8 is translated and secreted from cells containing mRNA. As previously suggested, this will be highest in σ^{stem}, but will also be present in recently differentiated cells, σ^{body}, as a decay gradient. Thus we can describe the concentration of FGF on $\mathcal{L}$, $\mathcal{C}_{FGF}(\sigma_{ij}^k)$, as a function of mRNA concentration, which leads the following definition:

$$\mathcal{K}_{FGF}(\sigma_{ij}^k) = \{ \ \mathcal{C}_{mRNA}(\sigma_{ij}^k) - \mathcal{C}_{FGF}(\sigma_{ij}^k), \quad \forall \sigma_{ij}^k \in \mathcal{L}. \tag{17}$$

Here we note the kinetics of FGF are calculated for $\forall \sigma_{ij}^k \in \mathcal{L}$. This is necessary as we cannot be certain which σ_{ij}^k currently have a concentration of FGF and mRNA. That is, FGF concentrations are a feature of the lattice as a secreted chemical, rather than cells.

Lastly we consider the kinetics of RA that is produced by σ^{body}. We assume that RA is produced in σ^{body} only when concentration of FGF-8 is below some threshold, μ_{low} and decays everywhere on $\mathcal{L}$ with a rate which is equal (or proportional) to its own concentration. Thus we can define the kinetics of RA in the following way:

$$\mathcal{K}_{RA}(\sigma_{ij}^k) = \begin{cases} 1 - \mathcal{C}_{RA}(\sigma_{ij}^k), & \tau(\sigma_{ij}^k) = body \\ -\mathcal{C}_{RA}(\sigma_{ij}^k), & \tau(\sigma_{ij}^k) \neq body. \end{cases} \tag{18}$$

This completes the definitions of kinetics we will need for implementation on the $\mathcal{L}$. Next we consider the implementation of diffusion.

6.2.3 Implementing Chemical Diffusion.

The second part of the chemical dynamics problem is chemical diffusion. We recall from the biological description, that only FGF and RA are secreted into the environment, $\mathcal{L}$, whereas mRNA is only produced within in the stem cells. Further we suggested diffusion is the same for all chemicals, which implies we will only be considering a single implementation which only varies parametrically for each chemical. Thus we wish to implement, and find a solution for, the canonical parabolic PDE,

$$\frac{\partial \psi(x,y,t)}{\partial t} = D\left(\frac{\partial^2 \psi(x,y,t)}{\partial x^2} + \frac{\partial^2 \psi(x,y,t)}{\partial y^2}\right), \tag{19}$$

which is the diffusion equation in 2 spatial dimensions, were D is the diffusion coefficient. Clearly this is a continuous spatio-temporal equation that will need be implemented on a discrete domain, $\mathcal{L}$. To do this we consider a method referred to as finite differencing, or a finite difference method (FDM), which is a standard approach for solving partial differential equations (PDE) on a discrete domain. However there are various ways to go about differencing depending on the type of PDE in question, elliptic, hyperbolic or parabolic, and the applicability of a method is highly dependant on the PDE. For example the FTCS scheme we will outline here is not applicable to hyperbolic PDEs but is to parabolic.

Our aim then, is to find an explicit representation of eq (19), as an initial value problem on the lattice. That is, we supply initial values to a function we assume will describe how the system evolves in time; the diffusion equation. This implies the function will be solved synchronously from one time step t to the next $t+1$, ultimately revealing the evolution of the system forward in time (FT). To derive this representation, we employ Taylor series to investigate how a function changes under small perturbations -*finite differences*-around a given point - centered space (CS). The arrangement and number of perturbations considered is referred to as the stencil for the method, and is illustrated in Fig. 28.

We start then, by considering a the Taylor series up to second order for a function that has been perturbed. That is, we wish to approximate the function at some displacement, Δx, from it current value at x_0, $f(x_0+\Delta x)$. Thus by Taylor series we have,

$$\begin{aligned} f(x_0+\Delta x) &\approx \frac{f(x_0)}{0!}(\Delta x)^0 + \frac{f'(x_0)}{1!}(\Delta x)^1 + \frac{f''(x_0)}{2!}(\Delta x)^2 + O((\Delta x)^3) \\ &\approx f(x_0) + f'(x_0)\Delta x + \frac{1}{2}f''(x_0)(\Delta x)^2 + O((\Delta x)^3). \end{aligned} \tag{20}$$

were $O((\Delta x)^3)$ is the higher order term signifying the truncation point for the series.

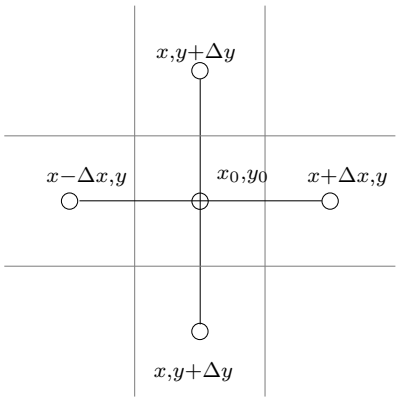


Figure 28: *Illustration of FTCS FDM stencil.*

If we consider the opposite perturbation, we can show an analogous result for $-\Delta x$

$$\begin{aligned} f(x_0 - \Delta x) &\approx \frac{f(x_0)}{0!}(\Delta x)^0 - \frac{f'(x_0)}{1!}(\Delta x)^1 + \frac{f''(x_0)}{2!}(\Delta x)^2 + O(\Delta x)^3 \\ &\approx f(x_0) - f'(x_0)\Delta x + \frac{1}{2}f''(x_0)(\Delta x)^2 - O(\Delta x)^3. \end{aligned} \tag{21}$$

Clearly, with eq (20) and eq (21) we can determine how a function varies about a given point, x_0, by finite differences around it, $\pm\Delta x$. However the significance of this is revealed when we take the sum of eq (20) and eq (21),

$$f(x_0 + \Delta x) + f(x_0 - \Delta x) \approx 2f(x_0) + f''(x_0)(\Delta x)^2$$

and rearranging to give,

$$f''(x_0) \approx \frac{f(x_0 + \Delta x) + f(x_0 - \Delta x) - 2f(x_0)}{(\Delta x)^2}, \tag{22}$$

which is a second order accurate representation of a second order derivative derived from Taylor series. More importantly, with this representation we can re-write the spatial components of eq (19) as

$$\frac{\partial^2\psi(x,y,t)}{\partial x^2} \approx \frac{\psi(x + \Delta x, y, t) + \psi(x - \Delta x, y, t) - 2\psi(x, y, t)}{(\Delta x)^2} \tag{23}$$

and

$$\frac{\partial^2\psi(x,y,t)}{\partial y^2} \approx \frac{\psi(x, y + \Delta y, t) + \psi(x, y - \Delta y, t) - 2\psi(x, y, t)}{(\Delta y)^2}, \tag{24}$$

which only depends on the sampling of spatial values around the given function. That is, we can compute a second order derivative by considering finite differences. To derive the temporal derivative, we can follow the same approach but only considering terms up to first order in the Taylor series,

$$f(t_0 + \Delta t) \approx f(t_0) + f'(t_0)\Delta t + O((\Delta t)^2)$$

thus,

$$f(t_0) \approx \frac{f(t_0 + \Delta t) - f'(t_0)}{\Delta t}$$

which implies,

$$\frac{\partial \psi(x, y, t)}{\partial t} \approx \frac{\psi(x, y, t + \Delta t) - \psi(x, y, t)}{(\Delta t)}. \tag{25}$$

With equations (23, 24, 25) we now present the discretized form of eq (19) suitable for $\mathcal{L}$, with the change of notation $\psi(x, y, t) \rightarrow \psi^t_{i,j}(\sigma_{ij})$ specifies the concentration of a particular chemical at cell-site σ_{ij} at time t, were it should be clear that there is a direct mapping from $(x, y) \rightarrow (i, j)$ (Fig. 29). eq (19) can now be written as

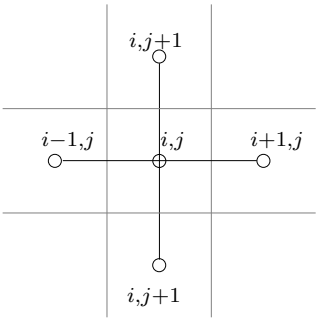


Figure 29: *Illustration of FCTS FDM stencil with lattice indices.*

$$\frac{\psi^{t+\Delta t}_{i,j} - \psi^t_{i,j}}{\Delta t} = D\left(\frac{\psi^t_{i+1,j} + \psi^t_{i-1,j} - 2\psi^t_{i,j}}{(\Delta i)^2} + \frac{\psi^t_{i,j+1} + \psi^t_{i,j-1} - 2\psi^t_{i,j}}{(\Delta j)^2}\right),$$

and if we consider $\mathcal{L}$ isotropic, $\Delta = \Delta y = \Delta x$, we can simplify this further to give

$$\frac{\psi^{t+\Delta t}_{i,j} - \psi^t_{i,j}}{\Delta t} = D\left(\frac{\psi^t_{i+1,j} + \psi^t_{i-1,j} + \psi^t_{i,j+1} + \psi^t_{i,j-1} - 4\psi^t_{i,j}}{\Delta^2} h\right). \tag{26}$$

Finally we can re-write eq (26) to give the diffusion for a lattice site, σ_{ij}, for the next simulation step at $t + \Delta t$, from the last at t

$$\psi_{i,j}^{t+\Delta t} = \mathcal{D}(\psi_{i,j}^{t}) = \left\{ \psi_{i,j}^{t} + D\frac{\Delta n}{\Delta^2}(\psi_{i+1,j}^{t} + \psi_{i-1,j}^{t} + \psi_{i,j+1}^{t} + \psi_{i,j-1}^{t} - 4\psi_{i,j}^{t}) \right\}, \quad (27)$$

which is commonly referred to as a forward time centered space for a finite difference method, FTCS FDM. With eq (27) we now have a method of implementing diffusion on the CPM. However there are issues of consistency and stability of the implementation, and of course specification of boundary conditions and initial values. Consistency is a statement of how well our implementation fits with the true PDE, which is clearly a feature of the truncation in the Taylor series approximation. That implies we should expect this truncation error to vanish as the spatial Δx, Δy and temporal Δt differences tend to zero. Stability on the hand is concerned with numerical errors that arise as the simulation evolves. There are several causes for this, but broadly they are classified into two main groups: imprecision of the computing hardware the simulations are running on, and those arising from differencing.

The first is caused by the lack of accuracy in numerical representation of real numbers, and also numerical truncations that occur during numerical computations. In general however, these are not so significant. However the latter is and is a feature of the differencing scheme we are using. In the continuous system, diffusion acts as a propagating wave of information with a maximum velocity v, which implies there is a temporal domain of dependency for each point. That is, if we consider computing $\sigma_{i,j}^{k,n}$ on $\mathcal{L}$ in a specific time slice Δt, then the amount of information contributing to $\sigma_{i,j}^{k,n}$ is restricted to the stencil we are using (Fig. 29) and thus if Δt is too large instabilities will occur as a result of lost information. Ultimately for our model, this implies specification of $D\frac{\Delta n}{\Delta^2}$ such that the instability is negated and fortunately we can do this with the so called Courant-Friedrichs- Lewy stability criterion,

$$D\frac{\Delta n}{\Delta^2} < \frac{1}{2}$$

or simply the Courant condition. Lastly boundary conditions and initial values are simply that $\mathcal{C}_{mRNA}(\sigma_{i,j}^{k,n}) = \mathcal{C}_{RA}(\sigma_{i,j}^{k,n}) = \mathcal{C}_{FGF-8}(\sigma_{i,j}^{k,n}) = 0 \ \forall \sigma_{i,j}^{k,n} \in \mathcal{L}$, and boundary conditions as Nuemann, $\frac{\partial \psi}{\partial n} = 0$. Finally we give the explicit representation for computing chemical dynamics of the lattice

$$\mathcal{C}_{chemical}(\psi_{n+1,i}^{k} j) = \mathcal{D}(\sigma_{i,j}^{k,n}) + \kappa \mathcal{K}_{chemical}(\sigma_{i,j}^{k,n}), \quad (28)$$

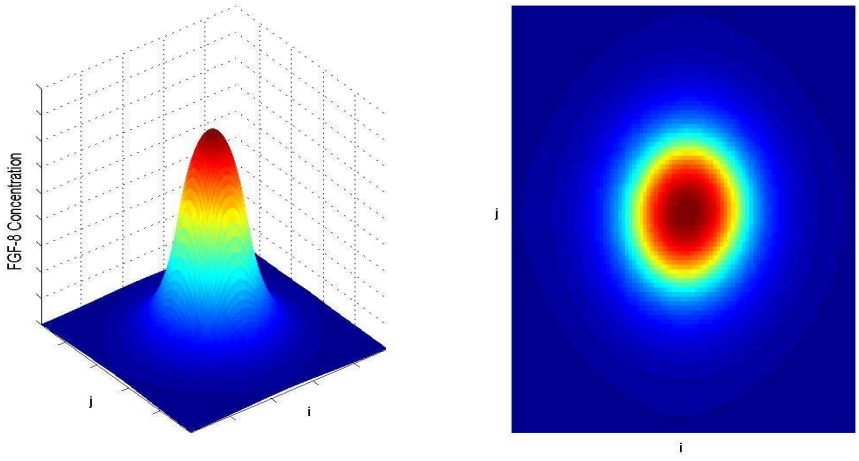
where κ represents the strength of the kinetics influence when computing concentrations.

6.3 Results

In this section we present results of simulations we performed with chemical dynamics. The simulation were are all performed with a 5×5 lattice of cells each with 5×5 volume. In terms of kinetics, production and decay values, ρ and δ, where set to 1 for all chemicals and only variances in the kinetic coefficient for chemicals, $\kappa_{chemical}$, where studied. In terms of diffusion, $D\frac{\Delta n}{\Delta^2} \approx \frac{1}{4}$, produced acceptable results. This was based on evolution of the diffusion equation being computed every 10 simulation steps, $\Delta n = 10$ and $\Delta^2 = 1$.

6.3.1 Basic Non-Motile FGF-8 Diffusion.

For this simulation, there was no chemotaxis or proliferation. In addition the kinetics used where: $\kappa_{FGF-8} = 0.003$. This result was produced as a demonstration of chemical dynamics in its simplest form, where Fig. 30a demonstrating concentration levels diffusing high to low on 3D surface, and Fig. 30b demonstrating same plot in 2D to better show diffusion. Dark red implies highest concentration, blue lowest.



a) 3D FGF−8 Concentration. *b) 2D FGF−8 Diffusion.*

Figure 30: *Maple plot of FGF-8 diffusion produced from CPM chemical dynamics.*

6.3.2 Varying FGF-8 mRNA Kinetics.

As we have seen the production of mRNA is the catalyst for our model, in that we assume its concentration level signals the differentiation of stem cells into body cells in the anterior of the stem zone. To investigate this we considered varying mRNA

kinetics in an attempt to understand the dynamics of this chemical and its effects in simulations. As with chemotaxis, we empirically found parametric ranges that constituted acceptable qualitative results. For mRNA kinetics this was found to be in the domain $\kappa_{mRNA} \in [0.001, 0.002, \ldots, 0.009]$, and FGF-8 $= 0.003$.

After running simulations for all values in the acceptable domain, we plotted the data for varying the κ_{mRNA} which suggested that there is growth in the number of cells in the stem zone, as intuition would suggest. That is, as FGF-8 concentrations are dependant on mRNA kinetics, we would expect varying the kinetics of mRNA to produce variances in stem zone mass. However interestingly, we did not observer a monotonic graph for these variances from which we can conclude, that we cannot infer a direct relationship between variances in mRNA and stem zone mass as illustrated in (Fig. 31).

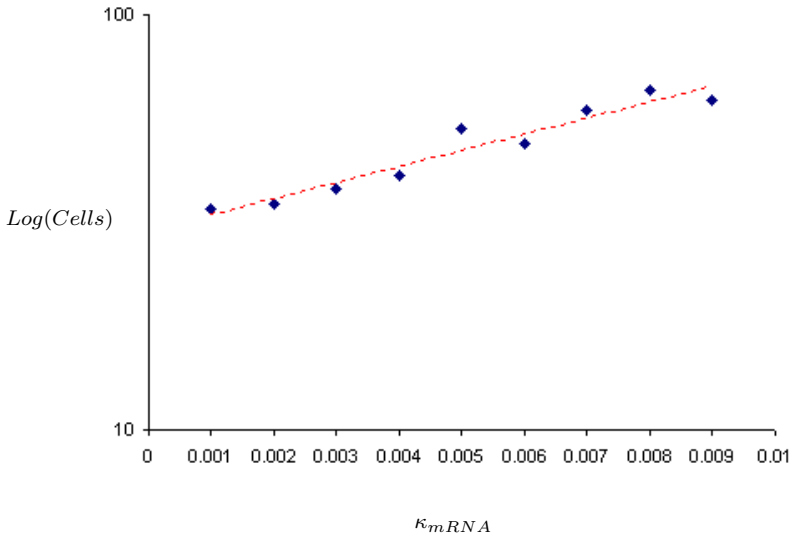


Figure 31: *Output illustrating non-monotonic increase in stem cells for varying mRNA, exhibiting an exponential rise in stem zone mass*

However upon inspecting the graphical results of the simulations, it is clear that the data is misleading, in that there is fragmentation occurring around the stem zone. More specifically, stem cells are becoming detached from the stem zone mass and being left behind as the zone regresses as illustrated in Fig. 32. Moreover, the problem seemed to occur for $\kappa_{mRNA} > 0.005$, and beyond this became progressively worse, leading to areas of proliferating stem cells in the extending body; the cause of misleading data. For any simulations with $\kappa_{mRNA} < 0.005$ we found simulations produced the desired result, and more importantly, corroborated our assumptions about an FGF-8 gradient enabling body extension as illustrated in Fig. 33.

The cause of the fragmented cells was due to two factors: the kinetics of FGF-8 and mRNA. Considering mRNA, we know it is fully saturates stem cells and that

Figure 32: *Output illustrating fragmentation of the stem zone FGF-8 mRNA kinetics,* $\kappa = 0.009$

Figure 33: *Output illustrating successful extending body axis for FGF-8 mRNA kinetics,* $\kappa = 0.001 \rightarrow 0.005$.

when differentiated into body cells, will simply decay. This decay rate, combined with motility, produces an mRNA gradient in cells. Thus an increase in the kinetics of mRNA, will result in an increase in the decay rate, thereby reducing the mRNA gradient as the stem zone regresses(Fig. 35left). Since the kinetics of FGF-8 are dependant on the concentration of mRNA, this leads to a reduction in its gradient. Ultimately this leads to a reduction in the area of highest concentration of FGF-8, which leads to stem cells bypassing the upper threshold region-μ_{high}- for differentiation (Fig. 35 right). Clearly this would imply we should reduce the kinetics of mRNA to maximize the differentiation threshold region, thereby solving the problem of fragmentation (Fig. 35). However the biological description suggests an alternate method, a method we have compensated for already in the model, Retinoid Acid (RA); we shall be discussing the further in a later section. To this end we enabled RA with arbitrary parameters: $\mu_{low} = 0.500$ and $\nu_{high} = 0.3000$ to solve this problem of and re-ran the simulations. Considering Fig. 36 we can see that for $\kappa_{mRNA} < 0.005$ the simulations coincide, and for $\kappa_{mRNA} > 0.005$ the simulations begin to diverge, as previously suggested. However as before, the problem of monotonicity remains and thus our previous statements hold; there is no predictable relationship for mRNA kinetics and stem zone mass, an interesting result.

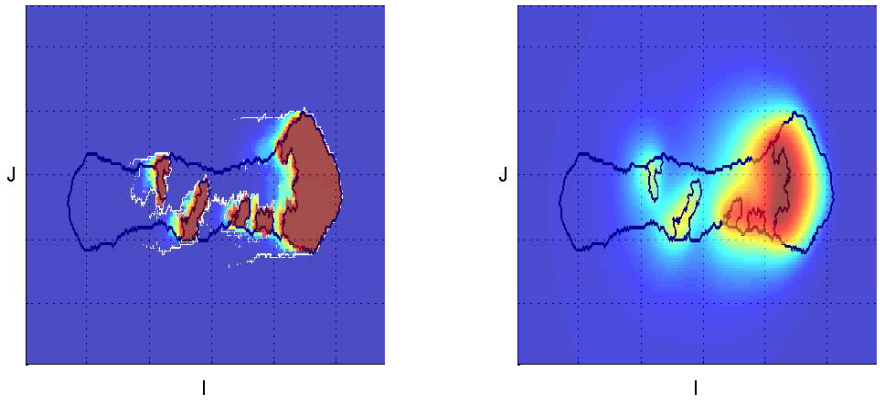


Figure 34: *Output illustrating mRNA(Left) and FGF-8 (Right) concentrations for* κ *= 0.009.*

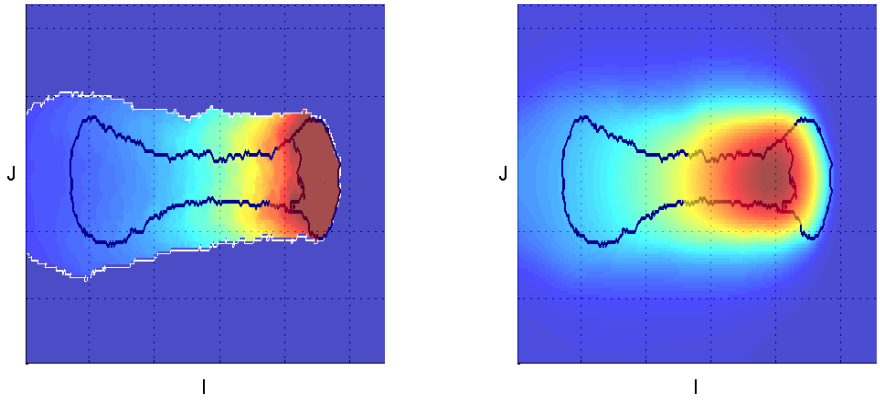


Figure 35: *Output illustrating mRNA(Left) and FGF-8 (Right) concentrations for* κ *= 0.001.*

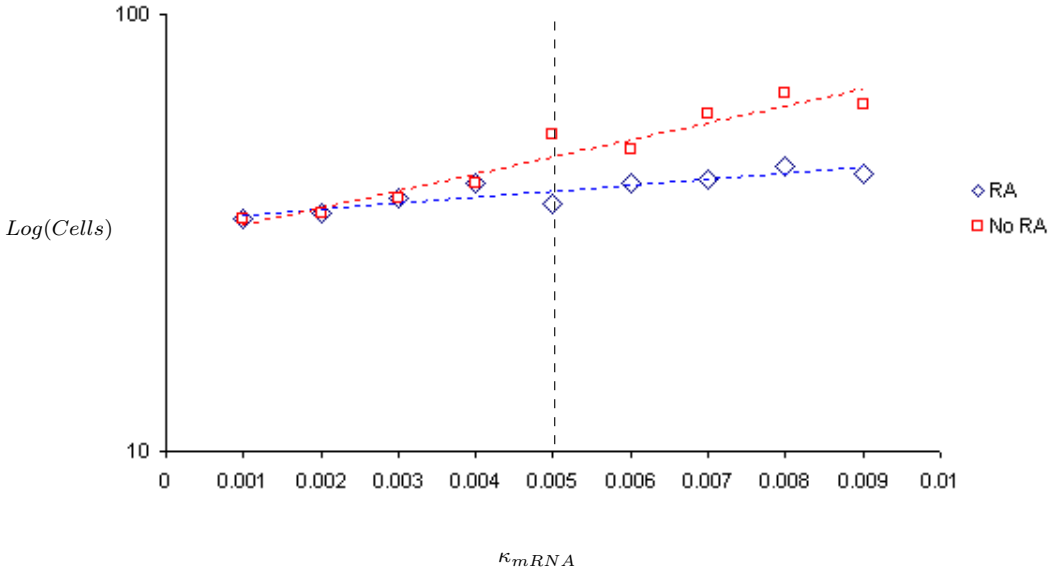


Figure 36: *Illustration of RA antagonist preventing stem zone fragmentation, with values reflecting true size of stem zone mass. The blue diamonds represents cell growth, with RA inhibitor show constant cell growth; red squares without RA inhibitor display exponential growth.*

6.3.3 Varying FGF-8 Kinetics.

The kinetics of FGF-8 are dependant on the production of mRNA, and produces a chemical gradient in and around the $\mathcal{S}$. We assumed motility of $\mathcal{S}$ would produce an area of high concentration of FGF-8 anterior to it, and in this region differentiation would occur; $\sigma^{stem} \to \sigma^{body}$. Clearly if this assumption is correct, we should expect the number σ^{stem} to decrease or increase -as differentiation occurs- depending on the diffusion and kinetics of FGF-8 in the anterior of $\mathcal{S}$. To investigate this, we considered an acceptable domain of FGF-8 kinetics $\kappa_{FGF_8} \in [0.001, 002, \ldots, 0, 007]$ and $\kappa_{mRNA} = 0.003$, that latter being chosen so as not to require RA in the simulation; recall from the previous section, that $\kappa_{mRNA} < 0.005$ would not produce fragmentation.

The results of running these simulations showed that there is indeed a reduction in the size of $\mathcal{S}$ proportional to the kinetics of FGF-8, κ_{FGF_8}, and that there is a saturation point at which the size of $\mathcal{S}$ becomes constant. This is illustrated in Fig. 37, where increase κ_{FGF_8} demonstrates a precipitous drop in the saturation points for $\mathcal{S}$. This drop is clearly a result of the kinetics becoming dominant over diffusion in the reaction-diffusion equation, leading to a much higher concentration of FGF-8 in the anterior stem zone, producing greater differentiation of stem cells. More importantly, we have demonstrated clearly that our assumptions where correct, and

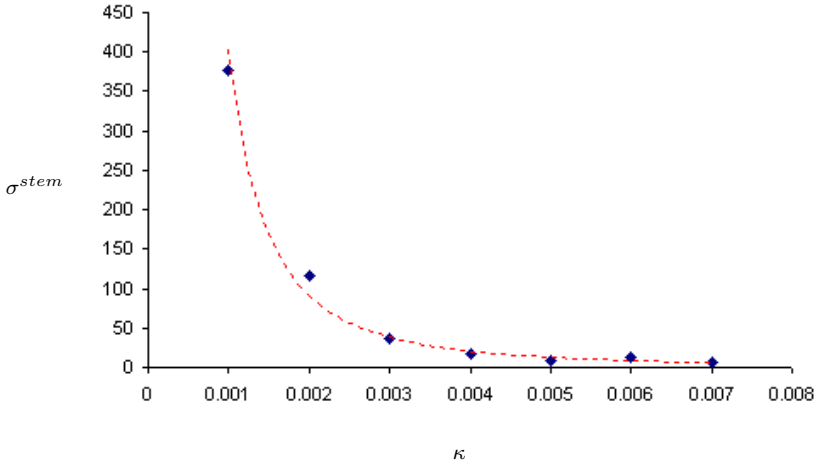


Figure 37: *Increase in FGF-8 kinetics, κ, produces precipitous decrease stem cells, σ^{stem}, in stem zone.*

that a high concentration of FGF-8 would form in the anterior of the stem zone leading to the production of an extending body axis. The graphical results of the simulations are presented with low, medium and high kinetics rates.

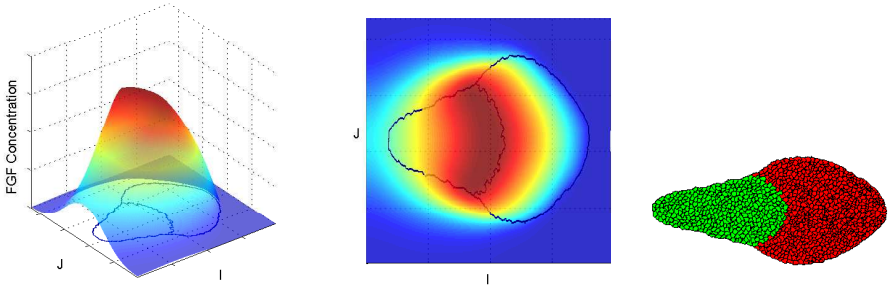


Figure 38: *FGF-8 Concentration Profiles for* $\kappa_{FGF_8} = 0.001$.

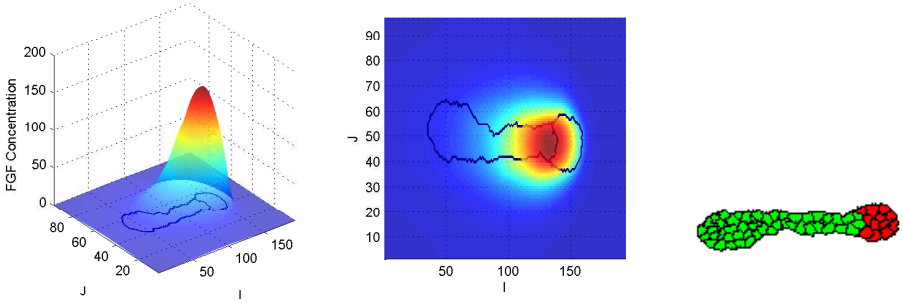


Figure 39: *FGF-8 Concentration Profiles for* $\kappa_{FGF_8} = 0.004$.

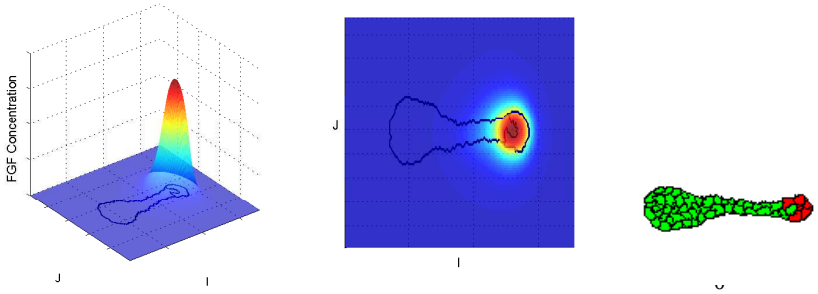


Figure 40: *FGF-8 Concentration Profiles for* $\kappa_{FGF_8} = 0.007$.

6.3.4 Varying FGF-8 Threshold (μ_{high}).

The FGF-8 threshold, μ_{high} is a parameter that in essence describes how sensitive to differentiation stem cells are in regions of FGF-8. From previous discussions we

know this to be in the anterior of the stem zone, and clearly it is in this region μ_{high} is most effective. Thus we wish to investigate the effects of varying μ_{high}, to see how it effects the morphology of the stem zone in terms of mass. As with mRNA we consider $\mathcal{C}_{FGF-8}(\sigma_{i,j}^{k,n}) \in [0.0, 1.0]$. That is, $\mathcal{C}_{FGF-8}(\sigma_{i,j}^{k,n}) = 0.0$ implies a zero concentration, whereas $\mathcal{C}_{FGF-8}(\sigma_{i,j}^{k,n}) = 1.0$ implies fully saturated. As in previous simulations, parameters where chosen to reflect a broad range of behavior. To simplify the simulations we set $\kappa_{mRNA} = 0.003$, to negate the need for RA, and set $\kappa_{FGF-8} = 0.003$ and with these setting, simulations where ran for $\mu_{high} \in [0.40, 0.45, \ldots, 0.75]$.

Intuitively we would expect the size of the stem zone to increase as the as μ_{high} increases, and conversely for μ_{high} decreasing. Indeed this is what we found, for $\mu_{high} < 0.40$ we found that the stem zone disappeared leaving a proliferating domain of body cells, and for increasing $\mu_{high} > 0.40$ the size of the stem zone increased exponentially (Fig. 41). In addition we note there is an increase in fragmentation

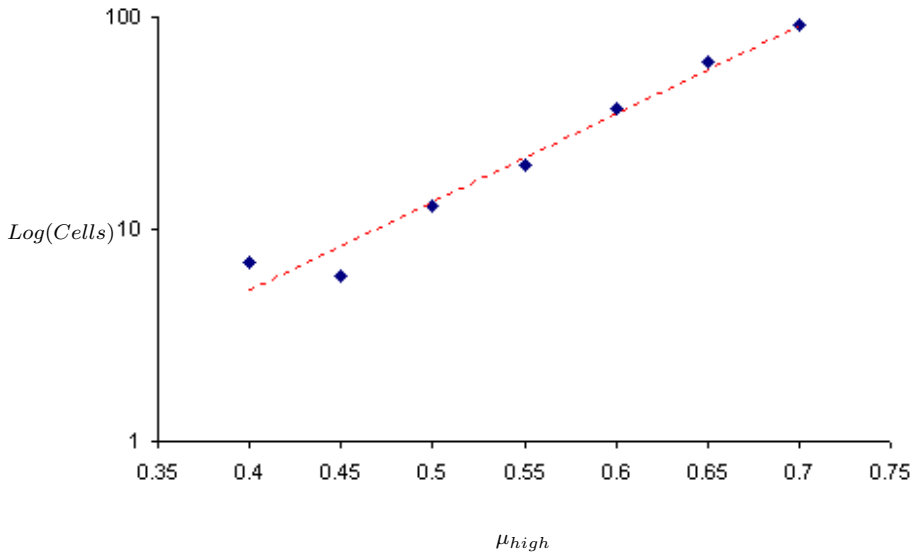


Figure 41: *Illustration of an exponential increase in stem zone mass for increasing* μ_{high}.

occurring specifically for $\mu_{high} > 0.70$, and as with previous simulation, we can account for this either variances in the kinetics of FGF-8 or by introducing RA. We now present some graphical results demonstrating the chemical concentration of mRNA, FGF-8 and the cell morphology.

What should be clear, is that the morphology of the cells tells us that for low μ_{high}, motility is stagnated in that body extension is significantly reduced due to the

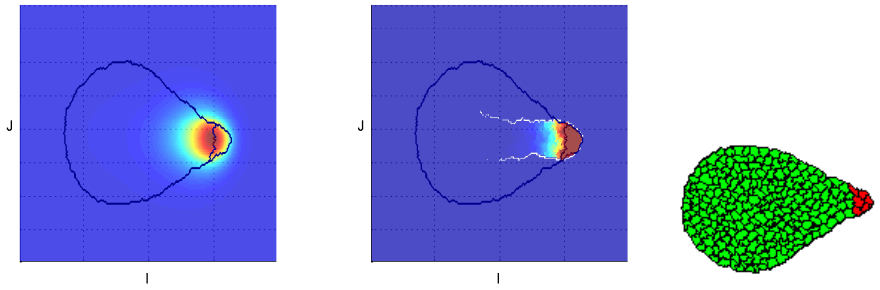


Figure 42: *FGF-8 Threshold Concentration profiles for* $\mu_{high} = 0.40$

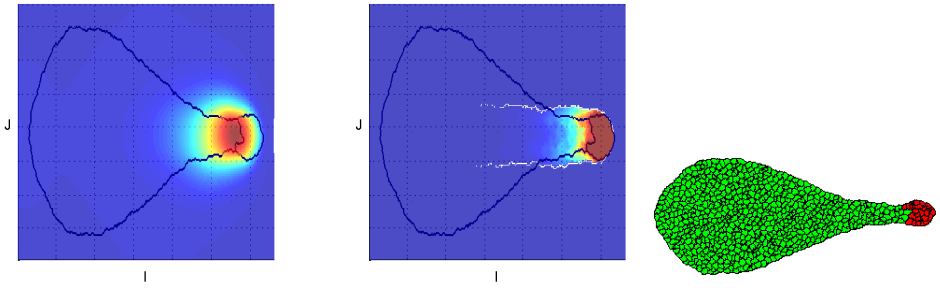


Figure 43: *FGF-8 Threshold Concentration profiles for* $\mu_{high} = 0.55$

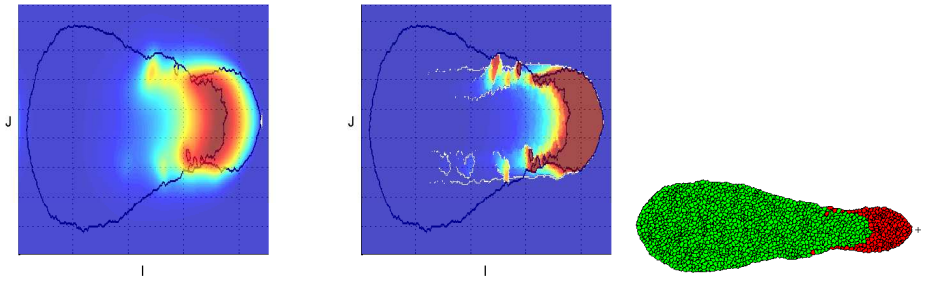


Figure 44: *FGF-8 Threshold Concentration profiles for* $\mu_{high} = 0.75$

low size of the stem zone. Clearly the converse is also true, in that the stem zone size increases which implies greater motility; from simulations it became clear that $\mu_{high} = 0.60$ is optimal.

6.3.5 Varying RA.

Unlike the other chemicals in the model, RA does not have a direct effect on the morphology of the cells. Its purpose is in general is to prevent the extension of the stem zone around Henson's node, and we have indeed seen this to be the case in both mRNA and FGF-8 simulations. However we also note that extension of the stem zone manifested as a small number of stem cells fragmenting in the anterior of the stem zone, and thus no significant changes to morphology where encountered. What this implies, is that varying the kinetics and diffusion of RA will only produce variances in chemical concentrations, a trivial matter compared to mRNA and FGF-8 simulations.

As we have seen RA production is signalled for low concentrations of FGF-8 or a lower FGF-8 threshold, μ_{low}, for differentiated cells that form the body. Furthermore, there is a signaling threshold for high concentrations of RA, ν_{high}, that signals stem cells in this region to differentiate into body cells. And finally, as with all previous chemicals, there is a kinetics term, κ_{RA}. Thus we investigated these three parameters and there effects. From previous simulations, we can infer some sensible values for the other chemicals: $\kappa_{mRNA} = 0.003$, $\kappa_{FGF-8} = 0.003$ and $\mu_{high} = 0.60$, and to be consistent we modeled a domain of 5×5 cells each having area 5×5.

Varying μ_{low} produced results that followed our intuition. That is, we know that FGF-8 forms a posterior to anterior decreasing gradient, and that it begins in the anterior of the stem zone. Thus what we observed in the graphical results is exactly what would expect, $\mu_{low} \to 0.0$ implies the boundary of RA production tends to the anterior, and conversely as $\mu_{low} \to 1.0$ - the highest concentration of FGF-8- the boundary tends to the posterior. Said another way, we can fully saturate the extending body with RA by setting $\mu_{low} \to 1.0$, or we can limit it to a small anterior regions by $\mu_{low} \to 0.0$. Graphically we can see this illustrated in Fig. 45, were $\kappa_{RA} = 0.003$ and $\nu_{high} = 0.6$ where chosen to reflect what was observed in FGF-8 and mRNA simulations.

Varying κ_{RA} can be considered coupled with the RA upper threshold parameter, ν_{high}, for signaling differentiation of orphaned stem cells in the extending body. If we consider only κ_{RA}, then it should be clear that for small κ_{RA} diffusion dominates and the concentration of RA is reduced over all. Conversely for large κ_{RA} we expect kinetics to dominate over diffusion and the concentration of RA to be approximately uniform over the cells, illustrated in Fig. 46. The significance of this, is that for high rates of diffusion, or low κ_{RA}, we can expect the concentration of RA to be reduced in the periphery of the cell domain. While this has no impact on the morphology of the cell domain it does have an effect on RA differentiation signaling, that is, the

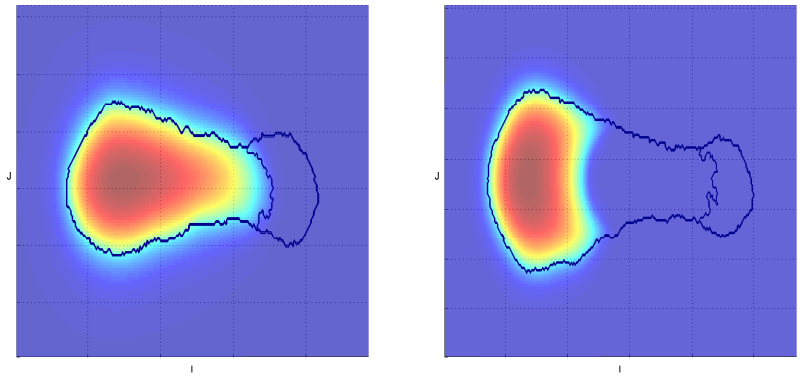


Figure 45: *Output demonstrating effect of varying RA production threshold* μ_{low}*, with* $\mu_{low} = 0.1$ *(left) and* $\mu_{low} = 1.0$ *(right).*

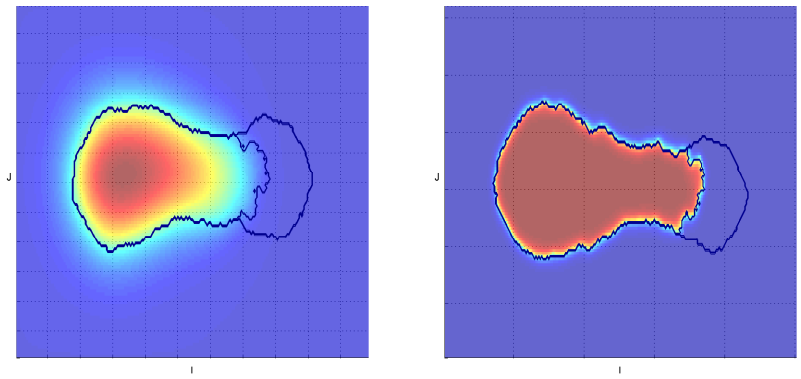


Figure 46: *Output demonstrating effect of varying RA kinetics,* κ_{RA}*, with low* κ_{RA} *(left) produces higher diffusion and low* κ_{RA} *(right) producing uniform concentrations of RA.*

upper threshold of RA ν_{high}.

Varying ν_{high} specifies the sensitivity of stem cells orphaned in the extending body to RA differentiation signaling. As previously suggested, ν_{high} is considered to be coupled with κ_{RA}, because κ_{RA} determines how much RA concentration levels vary over the extending body. More specifically, we know that for high rates of diffusion, or low κ_{RA}, concentrations will be lowest in the periphery of the extending body -as illustrated in Fig. 46 left-, which clearly implies that for RA signaling to be effective, ν_{high} must be proportional to the concentration in this region. Simply put, ν_{high} must be set to reflect the lowest concentrations on the extending body's outer boundary.

This would seem to complicate matters, and that it may seem logical to just set the κ_{RA} and μ_{low} very high and set ν_{high} very low, which would surely simplify the problem. However, we are attempting to model real biological phenomena, and so this simplification would detract from that. That is we know RA is a diffusing chemical in the environment, not merely a constant fully saturated chemical as the simplification would suggest.

6.4 Conclusion.

In the previous section we considered an approach to the problem of modeling the regression of Henson's node in extending the body axis. In a change from previous attempts, our motivation was consistent with current biological thinking, where formation of the extending body was due in part to the interplay between three primary chemicals: FGF-8 mRNA, FGF-8 and RA. We considered mRNA to be synthesized entirely with cells, which in turn was translated and secreted as FGF-8 into the environment. Further we assumed that as the stem zone regressed, a high concentration region of FGF-8 would occur in the posterior of the stem zone, in which differentiation from stem cell to body cell would occur, thereby leading to the progressive extension of the body axis. During simulations it became apparent that varying kinetics of various chemicals produced fragmentation of the stem zone, however appealing to the biology, we countered this with an RA gradient. Within the bounds of simulations conducted, we can categorically say these assumption were correct and matched the biological description. Further the role of RA in preventing stem zone extension, manifested itself in mRNA kinetics simulations, a result we find surprising at worst. Finally we acknowledge there is much room for further analysis, for example extensive testing involving broader ranges in cell counts and sizes, and kinetic and diffusion coefficients, however we're sufficiently confident that we have demonstrated success for the purposes of this paper.

7 Discussion.

The investigation over the last several sections considered how we could model the regression of Henson's node. We started by suggesting a suitable framework for the model - CPM - and the biological context in which our problem is posed. It is apparent that by and large our investigation is complicated by the lack knowledge in this area, and that for the most part, many assumptions were made to compensate for this. From a biological point of view, it is not at all clear how these processes actually take place in the embryo, but there indicators that lead to reasonable hypotheses.

Regarding the Cellular Potts Model, it is clear this is the model of choice when considering modelling cellular phenomena. This choice is supported by the numerous applications of the CPM from modelling ferromagnetism, foam/froth, crystalline formations, biofilms,... to name but a few. Further, from our point of view, it is its ability to represent biologically realistic cells that is the key for our requirements. More importantly, it demonstrates morphology that biologists find acceptable, and clearly this is an important feature for a biological mathematical model.

However there is the question of validating our implementation, after all there are implementations currently in existence that we could have used, most notably CompuCell3D maintained by Indiana University and Biocomplexity Institute, the latter being led by the co-creator of the CPM James Glazier. However, at the time little was understood about this tool, and even less about how we develop new ideas with it, and evidently time is a factor for this paper. Thus the validity of our implementation was is in the results we obtained, however we do recognize that this is an outstanding issue that could be considered further in future work.

Regarding our Hypotheses, we essentially considered two main areas: proliferation/differentiation and chemical dynamics to maintain the stem zone while progressively extending the body axis.

Clearly before we could investigate these hypothesis, we needed to consider motility. We know that there are varying mechanisms involved, but we cannot be certain what is actually causing this in the embryo. Our approach was to consider chemotaxis as the cause and more specifically, a chemoattractant for the node was specified. However there is the possibility, given that we know cell structures can be considered as incompressible fluids, that chemorepulsion at the posterior of the node causes extraneous tissue to be repelled thereby forcing the node to occupy the void it created; motion through repulsion. Others have suggested that it is the presence of a travelling wave that is the cause [4, 5]. Fundamentally however, we disagree with this in favour of chemotaxis, and we shall talk more about this a little later. Ultimately we demonstrated successfully that the chemotaxis based motility could be implemented on the CPM and illustrated this with various results.

With a successful model of chemotaxis, we turned to our first hypothesis: proliferation and differentiation to maintain stem zone size. The basis of this was that the behavior of stem cells could account for the extension with a biologically realistic

mechanism for growth known as proliferation. With this mechanism, we assumed our domain of cells to be populated entirely by stem cells that grew and divided, and with each division produced two daughter cells: a stem cell and a body cell. Our assumption then, was that if we made the body cells insensitive to the chemotaxis stimuli, then would expect three things: the stem zone would continue to be motile and its size would remain constant as would be continually replenished with stem cells, and that the body cells would be left behind to form the extending body.

However, simulations demonstrated that this was not the case and that dissociation occurred rapidly in the stem zone. To compensate for this we investigated two methods in an attempt to solve the problem including employing a center of mass attraction force, and employing differential adhesivity, and with these demonstrated success. Ultimately however, this approach was becoming more unjustified in its assumptions. Clearly we cannot say with any degree of accuracy that these solutions are biologically realistic, not least of which in terms of the morphology they display. This led to our next hypothesis.

Our attempts so far had been rooted in what we could achieve with the CPM given biological precedents, that is, most of the features in some respect had been shown to be possible on the CPM by others and that our assumptions where, at the very most, biologically realistic. However contrary to this there is no such precedents for our next hypothesis in terms of the CPM. However it has a well established biological precedent, and is based on chemical dynamics. Indeed, much of the latest materials researched [2, 3, 4, 5, 6] for this hypothesis, all agree fundamentally that chemicals are the main factor in patterning and extension of the body axis.

Our next hypothesis was to consider a motile domain of stem cells as before, however unlike before, differentiation would not occur during cell division. That is, stem cells would divide to produce two daughter stem cells. This motile domain of stem cells would synthesize mRNA and translate and secrete this as FGF-8 into the environment. As the stem zone regressed, we assumed that a posterior to anterior decreasing gradient of FGF-8 would form producing a high concentration in the anterior of the stem zone. Further we assumed that stem cells in this region would differentiate in to body cells, and become insensitive to the chemotactic stimuli. If these assumptions were valid, they would have two important consequences: Differentiation would only occur in a small anterior region of the stem zone, producing cells forg the extending body and more importantly, this would regulate the size of the proliferating stem zone.

Thus our hypothesis is to regulate the size of the stem zone with chemical dynamics, while producing the extending body. Clearly these assumption were correct and through simulations we have shown this categorically to be true. In particular, we demonstrated how regulation of the stem zone size could be achieved by varying the kinetics FGF-8, and how varying of the FGF-8 threshold produced large variances in the morphology of the extending body. In addition we found how varying the kinetics of mRNA led to fragmentation of the stem zone and how this could be regulated through production of RA in body cells.

Lastly we note there is some debate as to the nature of the FGF-8 gradient in terms of a travelling wave, and works of others [3, 4] tend to suggest that it is the presence of this wave that induces the motility of the node and subsequently the patterning that results. However our results contradict this assumption, in that it is the presence of motility in the node that induces a wave of FGF-8 to form which subsequently induces patterning, and indeed this is our primary result.

For the future there several areas worthy of further investigation, and in general they relate to development of a developmental biology simulation software. Clearly there is a group working in the United States of America on developing such software - CompuCell3D- but this implementation, as of writing, tends to aimed at a completely general framework and seems to required researches to have software development skills to use it effectively. Thus our aim could be to develop a highly specialized simulator for developmental biologists, that afforded a more user friendly "drag 'n' drop" environment, and in this respect, we have laid some of the foundation for this in this project.

For future work we could investigate more fully the mathematical tools we have employed such as reaction-diffusion equations, we referred to as chemical dynamics. Clearly our implementation produced acceptable results, but little in the way of validation of the method beyond its graphical results was investigated. There are several techniques available for such tools and a comparison of these would be required for future work.

In terms of future research, there are several areas of interest that our model could be developed for. As previously suggested there is room for a developmental biology simulator, and in this respect we could extend the current model to simulate features such the ingression of mesenchyme cells through the primitive streak, or the formation of somites and even the formation of the neural tube on the neural plate. Thus it is a logical next step to extend our model into 3D to better capture and model these phenomena.

To conclude, we have developed a method from biological assumptions to model the regression of Henson's node. During this development we considered methods that while having a biological basis, required justification in their assumptions we could not support, but nevertheless had merit in their efficacy. Despite this rather moot success, we developed a model that not only reflects current biological thinking, but also demonstrated validity for it. Further our model demonstrates a hypothesis that supports the concept of travelling waves in the extension and patterning of the embryo, but also raises issues concerning the nature of its involvement. In addition there is no known published work that can corroborate the issues raised, and so it is for future research to determine the efficacy of our results.

8 Appendices

A Guide To The Implementation.

The implementation of the cellular potts model and the requisite mathematics that form the solution to the problem outlined in this paper, were implemented using a the C++ compiler in Microsoft Visual Studio Professional 2005 Version 8.0.50727.762 Service Pack 1, with Microsoft .NET Framework Version 2.0.50727. The implementation was an extension to works done in the 552 Preliminary Dissertation: *"An Investigation Of Epithelial Cell Alignment Using A Modified Cellular Potts Model" - 2008, Supervisor Dr Bhaktier Vasiev.*. However, significant work and re-writing was required for more sophisticated modelling requirements of the current paper.

For example unlike the previous implementation we require features such as chemotaxis, which requires the size of the lattice to vary for simulations to accommodate this movement. In addition there several chemicals each having several parameters, that are user configurable to analyze their effect during simulations. Broadly speaking then, the implementation is divided into 3 main categories with associated classes:

1. Graphical User Interface (GUI).
    - class CPottsWnd
    - class CDynamics
    - class CChemicals
2. Cellular Potts Model (CPM).
    - class CTissue
    - class CCell
3. Utilities.
    - class INI
    - class LOG

that were developed to create an analysis tool for modelling the regression of Henson's node on the CPM. Clearly not all of the details of the implementation require explanation, and indeed we shall only be presenting an cursory overview of the main features, to give the reader an understanding of the implementation and how it is used.

As noted above, the implementation consists of several generic c++ classes and also classes derived from the Microsoft Foundation Classes (MFC) library. MFC can be thought of as a toolkit for developing Windows based applications, and indeed it is a collection of wrapper classes for all the functionality of the windows platform.

The generic classes are those that represent functionality no common to the windows platform and are specific to the model being developed. Thus we shall consider each of the main areas outlined and highlight the most notable features.

A.1 CPottsWnd Window

Central to the implementation of the CPM is the windows class $CPottsWnd$, derived from the MFC base window class CWnd. This class can be thought of as hosting the implementation CPM and manages all of the ancillary tasks such as: loading and saving of simulations, specifying simulation parameters, etc.. In essence it can be thought of as the software user interface into the implementation of the CPM. That is, users wishing to run simulations in their own applications need only include this class and provide GUI elements to modify its parameters and start/stop the simulations.

Thus we shall consider this class form a user point of view, and highlight the main features it exposes to users of its interface. Readers wanting a more in depth understanding are directed to the implementation in Section (B.1.3) and Section (B.2.3).

A.1.1 Instantiation of $CPottsWnd$

Instantiation is initiated via the one public constructor that requires a pointer to the host application window, and the dimensions of the drawing area it is allowed to use. To do this one would include this line in the initialization section of the application:

$$CPottsWnd * m\_pPottsWnd \; = \; new \; CPottsWnd(pWnd, \; \&WndRect);$$

were $pWnd$ is a pointer to a $CWnd$ object of the host application that represents the drawing area, and $WndRect$ is rectangle structure defining the hosts drawing window dimensions. This newly instantiated object, $m\_pPottsWnd$, has now initialized everything that is required to run CPM simulations, that is, a default simulation has beed loaded and is ready to be started. This is best illustrated by calling the simulation function of the $CPottsWnd$ window:

$$m\_pPottsWnd -> Simulate(TRUE, FALSE, FALSE).$$

This call is the central function and in this particular case, we are asking $CPottsWnd$ to draw itself to the parent window $pWnd$ passed during instantiation. The result is illustrated in Fig. 47, that shows what would appear on the host applications drawing area. With this it should be clear what little impact on the host application the CPM produces, that is, users are only required to execute two lines of code in the host application to run simulations.

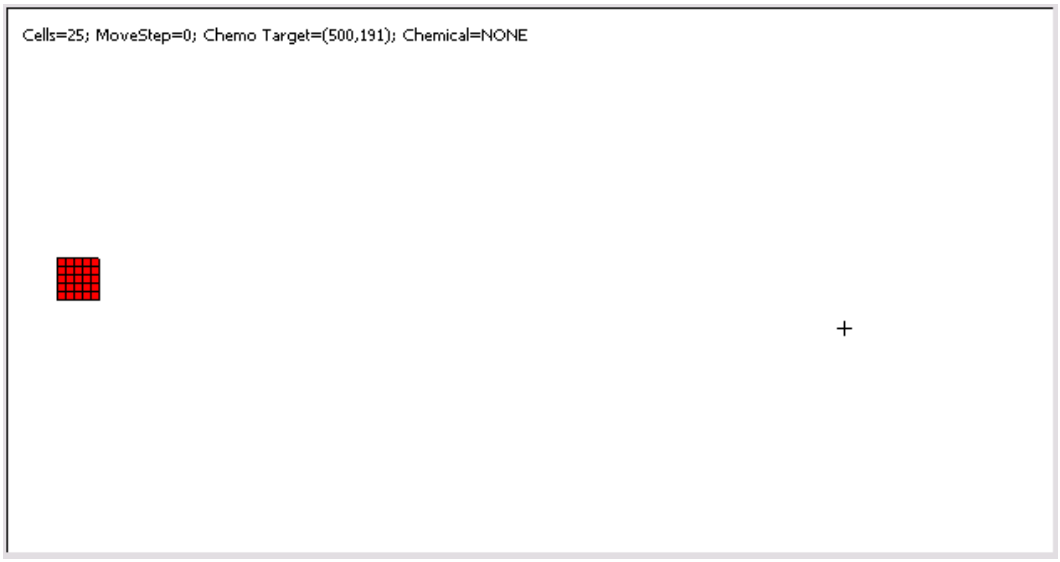


Figure 47: *Output of the default CPM graphical view in a host application, illustrating chemotaxis target cross and* 5×5 *cells ready for simulating.*

A.1.2 Starting/Stopping Simulations

With an successfully instantiated $CPottsWnd$ object $m\_pPottsWnd$ we consider further how to properly run simulations. We suggested in the last section this is handled via the $Simulate(\ldots)$ function and for the most part this is correct. However things are little more complicated when we look more closely at the arguments to this function:

$$Simulate(BOOL\ bDraw,\ BOOL\ bInvalidate,\ BOOL\ bStep).$$

What we can infer is that the simulation function performs two operations: drawing $bDraw$ and progressing the simulations $bSetp$. This implies that $Simulate$ is intended to be called iteratively and that when called, the caller specifies whether to simply draw the current state of the simulation with the boolean argument $bDraw$, or to progress the simulation, $bStep$ or both.

Ultimately this means that a host application can vary the speed of the simulation by the rate at which $Simulate$ is called. For our implementation of the host, we specified a dual approach where a standard timer event could be used or a more advanced multi-threaded approach designed to optimize the simulation for speed. That is, we allow an external thread to call $Simulate$ at maximum speed afforded by the host computer processor, or we can simply observer a the simulation at a desired speed with a timer. The author expects the reader to understand now why we may not require drawing in each call to $Simulate$.

A.1.3 Specifying Simulation Parameters

Besides the implementation of the CPM, this is probably the most sophisticated part of $CPottsWnd$ in that there are several ways achieve the same goal, and the number of parameters available. In general there though, they are divided into two main ways: modification via graphical user interface elements, or by modification of the initialization file $Tissue.ini$, see Section (C); we will consider the latter in a later section (Section (A.2)).

To enable user interaction with the cells, such as colour, type and viewing chemicals, a mouse activated context menu is exposed as illustrated in Fig. 48 from $CPottsWnd$ class.

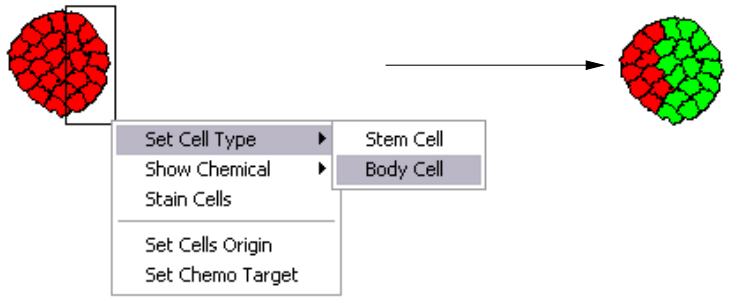


Figure 48: *Output illustrating changing cell type from stem cells (red) to body cells (green) via CPottsWnd context menu.*

With this menu we can create more sophisticated initial conditions by created patterns of stem cells and body cell to observer what dynamics occur with these conditions. In a similar fashion can stain particluar groups of cells as in Fig. 49.

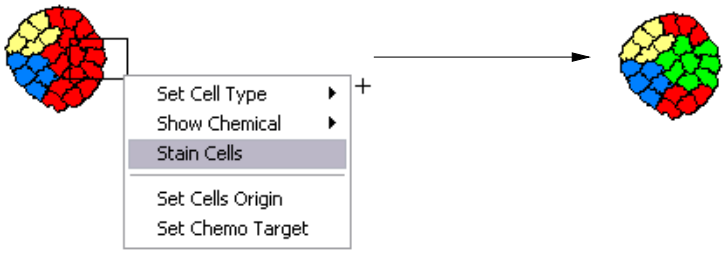


Figure 49: *Output illustrating staining cells for simulations.*

To second method focuses on modifying parameters of the CPM and the diffusion and kinetics of the chemicals. In particular we can vary the rate of proliferation and chemotaxis, vary differential adhesivity matrices to name but a few. To present these

parameters to the user two additional classes were created that abstract the tabs on a properties dialog *CChemical* and *CDynamic*.

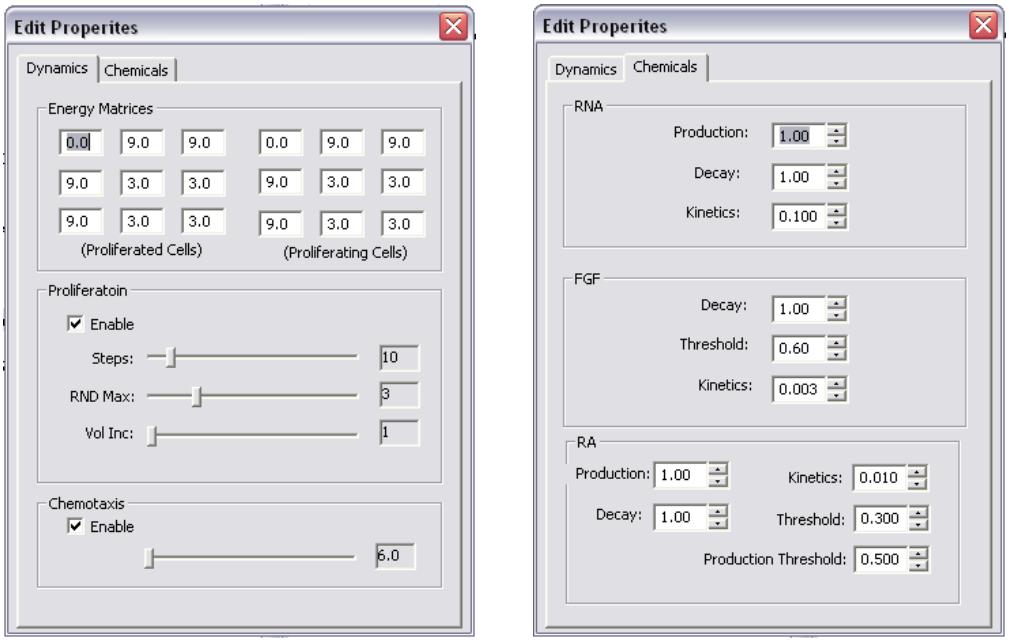


Figure 50: *Output illustrating properties dialog for changing chemical and CPM parameters.*

CChemical as its name suggested is solely for specifying chemical parameters for the three chemicals in our model: FGF-8, FGF-8 mRNA and RA, and CDynamic is for parameters for the basic mentioned previously. Fig. 50 illustrates the graphical interface for these classes hosted on a generic dialog box.

A.1.4 Loading and Saving Simulations

Because of the complexity in simulations, and more importantly how long simulations take and the specification of initial conditions, it becomes necessary to be able to capture the state of a simulation and store it. That is, we require the ability to arbitrarily save and load simulation. This was achieved in a rudimentary way by creating a binary file and storing the state of all essential parameter and structures to it. We gave this file has the extension *.tse. Readers interested in the specifics of the load/save process are directed to look at the *Load*(...) and *Save*(...) functions in Section (B.2.1)

A.2 Logging And Initialization

In Section (A.1.1) we showed how a default group of cells ready for simulation could be achieved in only 2 lines of code for a host application, then in Section (A.1.3) we illustrated the numerous parameters that are available for modification. Clearly then, there should be a method to specify parameters for this default simulation. In addition, we required the ability to export data from our application and log details of specific simulations and furthermore these features need to be available to all classes in the implementation.

This clearly presents a problem; how do we make a single open file globally accessible. A naive approach, would be to declare a single global variable in the host application, say $gLogFile$, and place $extern\ gLogFile$ in every implementation file that required access to it. However this bad programming practice, especially in an object oriented language such as c++, not least because it can become unclear over time whom has responsibility for the variable.

The solution was to consider a standard design pattern known as a singleton. In essence, a singleton is a static class that does not require instantiation and exposes static member functions to access member variable, such as a file, whose construction and destruction is managed solely by the singleton. In this way the singleton ensures that only a *single* instance of such a variable is ever in existence. Thus we can include this class in any file it is required, confident that the singleton will take responsibility for all aspects of the singular variable. At this point we accept that this can be counter intuitive, and we suggest looking at the implementation in Section (B.2.6) and in particular at $GetInst(\ldots)$, to see how this works.

Therefore we created to classes: LOG and INI that are based on the singleton pattern. INI is a representation of standard Windows initialization file, and its contents can be viewed in Section (C).

A.3 $CTissue$ and $CCell$

The class $CTissue$ represent the core logic for computing our simulations, and is instantiated as a member of $CPottsWnd$. $CCell$ is less significant and merely represents an abstraction of a lattice grid point or cell-site; interested readers are directed to its implementation in Section (B.2.1) and Section (B.1.2). However, $CTissue$ is large class with dozens of member variables of dozens of member functions most of which we will not be discussing here, and interested readers can see the complete implementation in Section (B.2.1) and Section (B.1.1). The important features of $CTissue$ are those that represent the features we have discussed in the preceding sections, such as chemicals, proliferation/differentiation and chemotaxis. However, these features are meaningless without and instantiated object.

A.3.1 Instantiation of $CTissue$

As previously stated $CTissue$ as large complicated class with numerous members, with several ways in which it can be instantiated. The reason for this is because we needed to be able run simulations for various different initial conditions. The type of initial conditions we wish use, defines the type of instantiation we require. That is, changing certain parameters require a re-instantiation of $CTissue$, such as when number or volume of cells is altered. Others can be altered at will, and do not require re-instantiation, such as chemical dynamics, and proliferation and differentiation. Put simply, changes that fundamentally alter the underlying structure of the cells and the lattice, requires us to delete and re-instantiation $CTissue$.

In general the instantiation is broken into two distinct phases. Firstly users create an instance of the class with one of three constructors:

1) $CTissue \quad (VOID);$

2) $CTissue \quad (char * filename, CDC * pDC)$

3) $CTissue \quad (INT\ nCellsWide,\ INT\ nCellsHigh,$
$INT\ nVolume, FLOAT\ fElasticiy,$
$CDC *\ pDC,\ INT\ nIBorder,\ INT\ nJBorder);$

and then initialize all of the members of the class with a call to the $Init(VOID)$ member function. The three constructors represent the 3 ways in which the lattice can be created: the first is seldom used and merely creates the object, the second is how we create the lattice from a previously saved file, and the third is the more common method were features of the lattice are specified during construction.

Calling the $Init(VOID)$ function is vital as numerous variables and structures are initialized, such as a call the member function $Boltzmann()$ that creates an array representing the Boltzmann probability distribution, a 2D array that represents the lattice as well dozens of others.

With an instantiated object, users are now free to modify the parameters of the lattice at will through the numerous get/set methods. It should be noted that this last comment is the only by which a user can modify the parameters of $CTissue$, in that most of the member variables are private to the implementation and therefore are not accessible directly.

A.3.2 The Structure of $CTissue$ simulation

With an instantiated and initialized $CTissue$ object we now give the basic structure of simulation on the lattice. Apart from those aspects that are a part of the initialization process, all of the processing required for a simulation are orchestrated through the $CTissue :: MoveStep()$ member function. In terms of what we have said in Section (A.1.1), concerning $CPottsWnd$ instantiation, $CTissue ::$

$MoveStep()$ is the member that is called as a result of a call to $CPottsWnd :: Simulate(BOOL, BOOL, BOOL)$, along with rendering of the simulation with a call to the $CTissue :: Draw()$ member function. In simple terms, we can consider the a simulation as consisting of 1) a call to $CTissue :: MoveStep()$ to progress the state of the simulation and 2) a call to $CTissue :: Draw()$ to render the current state of the lattice.

Thus we can give a pseudo code representation of the $CTissue :: MoveStep()$ to illustrate the structure of the processing required for a simulation.

Algorithm A.1 *CTissue::MoveStep()*

1. *Update Cell Volumes.*
2. *Update Chemicals.*
3. *Iterate over all lattice sites (i,j).*
   - *select random neighbour, r, around current site, c.*
   - *compute energy of potential swap, eps, of r with c.*
   - *if chemotaxis enabled, adjust eps as a bias towards chemotarget.*
   - *if eps satisfies Boltzmann criteria allow swap.*
4. *End lattice site iteration.*
5. *Update center of mass for all cells.*
6. *Attempt cell differentiation if enabled*
7. *Attempt cell proliferations if enables.*

Readers interested in a finer level of detail are directed towards Section (B.2.1) and Section (B.1.1). In particular, the lines in the pseudo code map to functions on the $CTissue$ class in the following way:

Update Cell Volumes	$\rightarrow CTissue :: Volume()$
Update Chemicals	$\rightarrow CTissue :: CalcChemicals()$
Chemotaxis	$\rightarrow CTissue :: CalcChemotaxis()$
Proliferations	$\rightarrow CTissue :: Proliferation()$
Differentiation	$\rightarrow CTissue :: Differentiation()$

This is my no means an exhaustive look at the processing of simulations or the $CTissue$, but it should be enough of a road map to understanding the structure of $CTissue$.

B C++ Implementation

B.1 C++ Header File

B.1.1 CTissue.h

```
//|---------------------------------------------------------------------
//| Name:
//|   class CTissue
//| Effect:
//|   An abstract class a lattice based tissue, complete with
//|   surrounding substrate.
//|---------------------------------------------------------------------
class CTissue
{
public:
    CTissue(void);
    CTissue(char* filename, CDC*pDC);
    CTissue(INT nCellsWide, INT nCellsHigh, INT nVolume,
    FLOAT fElasticiy, CDC* pDC,INT nIBorder=100, INT nJBorder=100);
    virtual ~CTissue(void);
public:
    enum Border { //The border the user is interested in.
        I=0, //Default cell type.
        J,
    };
    enum Chemical{ //The chemical the user wishes to see.
        RNA=0,
        FGF=1,
        RA=2,
        Cells=3,
        None=4,
        All=5,
    };
private: //Chemicals and such
    BOOL m_bChemotaxis;
    FLOAT m_fChemoEnergy;
    INT m_nChemoAttract;
    FLOAT m_fRADecay;
    FLOAT m_fRNADecay;
    FLOAT m_fFGFDecay;
    FLOAT m_fRAProduction;
    FLOAT m_fRNAProduction;
    FLOAT m_fRAKinetics;
```

```
    FLOAT m_fRNAKinetics;
    FLOAT m_fFGFKinetics;
    FLOAT m_fFGFThreshold;
    FLOAT m_fRAThreshold;
    FLOAT m_fRAProductionThreshold;
private:
    BOOL m_bProliferation;
    INT m_nProlifSteps;
    INT m_nProlifRNDMax;
    INT m_nProlifVolIncrease;
    BOOL m_bInitProliferation;
    INT m_nMoveStep;
    CPoint m_ptLastCoM;
    CPoint m_ptCenterOfMass;
    CPoint m_ptChemoTarget;
    FLOAT m_fElasticity;
    INT m_nRndMin;
    INT m_nRndMax;
    INT m_nMaxCells;
    INT m_nMaxSubCells;
    INT m_nNumCells;
    INT m_nNumSubCells;
    INT m_nCellsWide;
    INT m_nCellsHigh;
    INT m_nCellsMinI;
    INT m_nCellsMinJ;
    INT m_nCellsMaxI;
    INT m_nCellsMaxJ;
    INT m_nCellInitVolume;
    INT m_nTargetVolume;
    INT m_nMaxVolume;
    CBitmap m_bmpCOMDiffusion;
private: //Arrays of attributes for cells.
    CCell*     m_pCells;
    CCell*     m_phCells;
    PFLOAT m_pRNA;
    PFLOAT m_pFGF;
    PFLOAT m_phFGF;
    PFLOAT m_pRA;
    PFLOAT m_phRA;
    PUSHORT m_pCellVolume;
    PUSHORT m_pCellTargetVolume;
    PFLOAT m_pCellVolumeDecrease;
```

```
    PFLOAT m_pCellVolumeIncrease;
    CPoint* m_pCellCoM;
PBOOL m_pCellProliferate;
    PDWORD m_pCellStain;
    INT m_nBoltzman[1000];
public:
    FLOAT       m_fTBind[3][3];
    FLOAT       m_fBBind[3][3];
    FLOAT       m_fHBind[3][3];
private: //Drawing stuff...
    CDC* m_pDC;
    CPoint m_ptViewportOrg;
    CPoint m_ptCellsOrigin;
    CRect m_rcSubstrate;
    COLORREF m_clrHighLight;
    INT m_nCellToHighLight;
    CRect m_Rect;
    BOOL m_bStain;
    CFont m_Font;
    CFont* m_pOldFont;
    BOOL m_bCentering;
    INT m_nSubstrateJBorder;
    INT m_nSubstrateIBorder;
    INT m_nSubstrateWidth;
    INT m_nSubstrateHeight;
    Chemical m_chemShowChemical;
public: //Accessor functions...
    CRect GetCellRect();
    CRect GetSubstrateRect();
    CPoint  GetViewportOrg();
    CDC* GetOutputDC() {return m_pDC;}
    INT GetCellsWide(){ return m_nCellsWide;}
    INT GetCellsHigh(){ return m_nCellsHigh;}
    INT GetCellVolume(){return m_nCellInitVolume;}
    INT GetMoveStep(){ return m_nMoveStep;}
    FLOAT GetCellElasticity(){return m_fElasticity;}
    BOOL GetChemotaxis() {return m_bChemotaxis;}
    FLOAT GetChemoEnergy() {return m_fChemoEnergy;}
    INT GetChemoAttraction() {return m_nChemoAttract;}
    BOOL GetProliferation(){return m_bProliferation;}
    INT GetProlifSteps(){return m_nProlifSteps;}
    INT GetProlifRNDMax(){return m_nProlifRNDMax;}
    INT GetProlifVolInc(){return m_nProlifVolIncrease;}
```

```
    INT GetSubstrateBorder(Border B){if (B==CTissue::I){return
            m_nSubstrateIBorder;}else{ return m_nSubstrateJBorder;}}
    CPoint* GetChemoTarget(){return &m_ptChemoTarget;}
    CPoint* GetCellsOrigin(){return &m_ptCellsOrigin;}
    BOOL  GetCellsCentering(){return m_bCentering;}
    FLOAT   GetRADecay(){return m_fRADecay;}
    FLOAT   GetRNADecay(){return m_fRNADecay;}
    FLOAT   GetFGFDecay(){return m_fFGFDecay;}
    FLOAT   GetRAProduction(){return m_fRAProduction;}
    FLOAT   GetRAThreshold(){return m_fRAThreshold;}
    FLOAT   GetRAProductionThreshold()
                {return m_fRAProductionThreshold;}
    FLOAT   GetRNAProduction(){return m_fRNAProduction;}
    FLOAT   GetFGFThreshold() {return m_fFGFThreshold;}
    FLOAT   GetRAKinetics(){return m_fRAKinetics;}
    FLOAT   GetRNAKinetics(){return m_fRNAKinetics;}
    FLOAT   GetFGFKinetics(){return m_fFGFKinetics;}
    VOID    SetViewportOrg(CPoint* ptValue){m_ptViewportOrg=*ptValue;
                          m_pDC->SetViewportOrg(*ptValue);}
    VOID SetCellsWide(INT nValue){ m_nCellsWide=nValue;}
    VOID SetCellsHigh(INT nValue){ m_nCellsHigh=nValue;}
    VOID SetCellVolume(INT nValue){m_nCellInitVolume=nValue;}
    VOID SetCellElasticity(FLOAT fValue){m_fElasticity=fValue;}
    VOID SetChemotaxis(BOOL bValue) {m_bChemotaxis=bValue;}
    VOID SetChemoEnergy(FLOAT fValue) {m_fChemoEnergy=fValue;}
    VOID    SetChemoAttraction(INT nValue) {m_nChemoAttract=nValue;}
    VOID SetProliferation(BOOL bValue){m_bProliferation = bValue;}
    VOID SetProlifSteps(INT nValue){m_nProlifSteps = nValue;}
    VOID SetProlifRNDMax(INT nValue){m_nProlifRNDMax = nValue;}
    VOID SetProlifVolInc(INT nValue){m_nProlifVolIncrease = nValue;}
    VOID SetSubstrateBorder(Border B, INT nValue){
            if(B==CTissue::I){m_nSubstrateIBorder=nValue;}
            else{m_nSubstrateJBorder=nValue;}};
VOID SetOutputDC(CDC* pDC) {pDC->AssertValid(); m_pDC=pDC;}
    VOID    SetCellStain(INT nID, COLORREF clr)
                        {m_pCellStain[nID]=clr;}
    VOID SetChemoTarget(CPoint* pptValue)
                          {m_ptChemoTarget = *pptValue;}
    VOID  SetCellsOrigin(CPoint* pptValue)
                          {m_ptCellsOrigin = *pptValue;}
    VOID  SetCellsCentering(BOOL bValue){m_bCentering=bValue;}
    VOID    SetRADecay(FLOAT fValue){m_fRADecay = fValue;}
```

```
    VOID    SetRNADecay(FLOAT fValue){m_fRNADecay = fValue;}
    VOID    SetFGFDecay(FLOAT fValue){m_fFGFDecay = fValue;}
    VOID    SetRAProduction(FLOAT fValue)
                        {m_fRAProduction = fValue;}
    VOID    SetRNAProduction(FLOAT fValue)
                        {m_fRNAProduction = fValue;}
    VOID    SetFGFThreshold(FLOAT fValue)
                        {m_fFGFThreshold = fValue;}
    VOID    SetRAThreshold(FLOAT fValue){m_fRAThreshold = fValue;}
    VOID    SetRAProductionThreshold(FLOAT fValue)
                            {m_fRAProductionThreshold=fValue;}
    VOID    SetRAKinetics(FLOAT fValue){m_fRAKinetics = fValue;}
    VOID    SetRNAKinetics(FLOAT fValue){m_fRNAKinetics = fValue;}
    VOID    SetFGFKinetics(FLOAT fValue){m_fFGFKinetics = fValue;}
public:
    VOID  Init();
    VOID  MoveStep();
    VOID  Draw(BOOL bInvalidate=FALSE);
    CTissue* Clone();
    VOID  CalcChemicals();
    FLOAT  CalcChemotaxis(CCell* pOrigin, CCell* pNeighbour);
    BOOL  Save(PCHAR filename);
    BOOL  Load(PCHAR filename);
private: //Methods.
    VOID    Proliferate();
    VOID  Differentiate();
    VOID  CenterOfMass();
    VOID  ShowChemicals(Chemical chemValue);
    VOID  Center();
    VOID  Volume();
    VOID  Boltzen();
public: //DEBUG STUFF...
    CString m_strBuffer;
    CStdioFile  m_hFile;
    BOOL m_bStopSimulating;
public:
    friend class CPottsWnd; //grant private access to the window.
};
```

B.1.2 CCell.h

```
//|---------------------------------------------------------------------
//| Name:
//|   class CCell : public CPoint
//| Effect:
//|   An abstract class representation of a lattice site.
//|---------------------------------------------------------------------
class CCell : public CPoint
{
public:
    CCell(void);
    virtual ~CCell(void);
public:
    enum Type {
        Substrate=0, //Default cell type.
        Pellucida,
        Proliferated,
        Opaca,
        Sickle,
        StreakTip,
    };
public:
    USHORT m_nID;
    USHORT m_nhID;
    Type m_Type;
    Type m_hType;
public:
    friend class CTissue;
};
```

B.1.3 CPottsWnd.h

```
#pragma once
#define POTTSWND_ID 0x0000FFFF
#include "Tissue.h"
#include "../IO/Static.h"
#include <atlimage.h>
#include <Gdiplusimaging.h>
#include <Algorithm>


// CPottsWnd
class CPottsWnd : public CWnd
```

```
{
    DECLARE_DYNAMIC(CPottsWnd)
private:
    CPottsWnd();
public:
    CPottsWnd::CPottsWnd(CWnd* pParent, CRect* pWndRect);
    virtual ~CPottsWnd();
public:
    enum Matrix {
        Top=0,
        Bottom,
        Horizontal
    };
public:
    HCURSOR m_hCursor;
    HCURSOR m_hOldCursor;
    BOOL m_bSetCenter;
    BOOL m_bSetChemoTarget;
    CTissue* m_pTissue;
    CRect m_rcWndRect;
    CSize m_csWndSize;
    USHORT m_nWidth;
    USHORT m_nHeight;
    USHORT m_nVolume;
    FLOAT m_fElasticity;
public: //Window data
    CDC    m_ZoomDC;
    CBitmap    m_bmpCells;
    CBitmap    m_bmpVectors;
    BOOL    m_bVecField;
    CBitmap    m_Bmp;
    CBitmap*    m_pOldBmp;
    CFont    m_fntTahoma;
    CFont*    m_pOldFont;
    INT    m_nRefreshRate;
    FLOAT    m_fZoomFactor;
    CDC*    m_pDC;
    CMenu    m_mnuPopup;
    CMenu    m_mnuCellType;
    CMenu    m_mnuChems;
    CPoint    m_ptStart;
    CPoint    m_ptEnd;
    CRect    m_rcSelected;
```

```
    BOOL    m_bLeftBtnDown;
    CString    m_strWndName;
    vector<INT>    m_SelectedCells;
public:
    VOID Staining(BOOL bStain);
    VOID DCtoBMPFile(LPCSTR pFile);
    VOID WriteSurfaceData(LPCSTR pFile, CTissue::Chemical);
    FLOAT GetMatrixEntry(DWORD BitField, USHORT i, USHORT j);
    VOID SetMatrixEntry(DWORD BitField, USHORT i, USHORT j,
                FLOAT fValue);
    VOID Reset(INT nWidth, INT nHeight, INT nVolume,
                      FLOAT fElasticity,
                      INT nIBorder=100, INT nJBorder=100 );
    CCell* CellHitTest(CPoint*p);
    VOID SetCellsType(CCell::Type T);
    VOID ShowRNA();
    VOID ShowFGF();
    VOID ShowRA();
    VOID ShowNone();
protected:
    virtual INT OnInit(VOID);
    VOID SaveSettings();
    VOID LoadSettings();
    VOID Simulate(BOOL bDraw=TRUE,
                         BOOL bInvalidate=FALSE,
                         BOOL bStep=TRUE);
private:
    friend UINT PottsSimulate( LPVOID pParam );

/////////////////////////////////////////////////////////
//Window Based stuff only...
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg int  OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnPaint();
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
    afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest,
```

```
                                UINT message);
    afx_msg void SetChemoTarget();
    afx_msg void CPottsWnd::OnSetCenter();
    afx_msg void OnStainCell();
    afx_msg void OnSetCellTypePellucida();
    afx_msg void OnSetCellTypeProliferated();
};
```

B.1.4 CChemicals.h

```
#pragma once
#include "PottsWnd.h"
#include "../resource.h"

//|---------------------------------------------------------------------
//| Name:
//|   class CChemicals : public CPropertyPage
//| Effect:
//|   An abstract class representing a tab on the application
//|   properties dialog. Specifically on this tab users can modify
//|   the kinetics and diffusion equation parameters.
//|---------------------------------------------------------------------
class CChemicals : public CPropertyPage
{
DECLARE_DYNAMIC(CChemicals)

public:
    CChemicals();
    CChemicals(CPottsWnd* pPottsWnd);
    virtual ~CChemicals();
    enum { IDD = IDD_CHEMICALS };
public:
    CPottsWnd* m_pPottsWnd;
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    DECLARE_MESSAGE_MAP()
public:
    VOID  SetUIState(bool bStart);
VOID  SetDlgFloat(INT nID, FLOAT fValue, INT nDecPoints=1)
                    {CString txtValue;
                     CString temp;
                     temp.Format(".%df",nDecPoints);
```

```
                    txtValue.Format("%"+temp, fValue);
                    SetDlgItemText(nID, txtValue); }
VOID  SetDlgInt(INT nID, INT nValue)
                   {CString txtValue;
                    txtValue.Format("%d", nValue);
                    SetDlgItemText(nID, txtValue); }
    VOID  SetDlgCheck(INT nID, BOOL bValue)
                   {CButton* btn = (CButton*)GetDlgItem(nID);
                   btn->SetCheck(bValue);}
    INT  GetDlgInt(INT nID)
                   {CString T;
                   GetDlgItemText(nID, T);
                   return atoi(T);}
    FLOAT  GetDlgFloat(INT nID)
                   {CString T;
                   GetDlgItemText(nID, T);
                   return (FLOAT)atof(T);}
    BOOL  GetDlgCheck(INT nID)
                {return ((CButton*)GetDlgItem(nID))->GetCheck();}
    CEdit*  GetDlgEdit(INT nID)
                {return (CEdit*)GetDlgItem(nID);}
    CSliderCtrl* GetDlgSlider(INT nID)
                {return (CSliderCtrl*)GetDlgItem(nID);}
    CSpinButtonCtrl* GetDlgSpin(INT nID)
                {return (CSpinButtonCtrl*)GetDlgItem(nID);}
    virtual BOOL OnInitDialog();
    afx_msg void OnSpinChange(NMHDR *pNMHDR, LRESULT *pResult);
    afx_msg void OnChangeRAProduction();
    afx_msg void OnChangeRNAProduction();
    afx_msg void OnChangeRADecay();
    afx_msg void OnChangeRNADecay();
    afx_msg void OnChangeRAKinetics();
    afx_msg void OnChangeRNAKinetics();
    afx_msg void OnChangeFGFKinetics();
    afx_msg void OnChangeFGFDecay();
    afx_msg void OnChangeFGFThreshold();
    afx_msg void OnChangeRAThreshold();
    afx_msg void OnChangeRAProductionThreshold();
};
```

B.1.5 CDynamics.h

```
#pragma once
#include "PottsWnd.h"
#include "../resource.h"
//|---------------------------------------------------------------------
//| Name:
//|   class CDyanmics : public CPropertyPage
//| Effect:
//|   An abstract class representing a tab on the application
//|   properties dialog. Specifically on this tab users can modify
//|   the CPM model parameters.
//|---------------------------------------------------------------------
class CDynamics : public CPropertyPage
{
DECLARE_DYNAMIC(CDynamics)
public:
    CDynamics();
    CDynamics(CPottsWnd* pPottsWnd);
    virtual ~CDynamics();
    enum { IDD = IDD_DYNAMICS };
public:
    CPottsWnd* m_pPottsWnd;
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    DECLARE_MESSAGE_MAP()
public:
    virtual BOOL OnInitDialog();
public:
    VOID  SetUIState(bool bStart);
VOID  SetDlgFloat(INT nID, FLOAT fValue, INT nDecPoints=1)
                    {CString txtValue;
                     CString temp;
                     temp.Format(".%df",nDecPoints);
                     txtValue.Format("%"+temp, fValue);
                     SetDlgItemText(nID, txtValue); }
VOID  SetDlgInt(INT nID, INT nValue)
                    {CString txtValue;
                     txtValue.Format("%d", nValue);
                     SetDlgItemText(nID, txtValue); }
    VOID  SetDlgCheck(INT nID, BOOL bValue)
                    {CButton* btn = (CButton*)GetDlgItem(nID);
```

```
                btn->SetCheck(bValue);}
    INT  GetDlgInt(INT nID)
                {CString T;
                GetDlgItemText(nID, T);
                return atoi(T);}
    FLOAT  GetDlgFloat(INT nID)
                {CString T;
                GetDlgItemText(nID, T);
                return (FLOAT)atof(T);}
    BOOL  GetDlgCheck(INT nID)
                {return ((CButton*)GetDlgItem(nID))->GetCheck();}
    CEdit*  GetDlgEdit(INT nID)
                {return (CEdit*)GetDlgItem(nID);}
    CSliderCtrl* GetDlgSlider(INT nID)
                {return (CSliderCtrl*)GetDlgItem(nID);}

public:
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos,
                           CScrollBar* pScrollBar);
    afx_msg void OnEnergyChangeVTL();
    afx_msg void OnEnergyChangeVTM();
    afx_msg void OnEnergyChangeVTR();
    afx_msg void OnEnergyChangeVML();
    afx_msg void OnEnergyChangeVMM();
    afx_msg void OnEnergyChangeVMR();
    afx_msg void OnEnergyChangeVBL();
    afx_msg void OnEnergyChangeVBM();
    afx_msg void OnEnergyChangeVBR();

    afx_msg void OnEnergyChangeHTL();
    afx_msg void OnEnergyChangeHTM();
    afx_msg void OnEnergyChangeHTR();
    afx_msg void OnEnergyChangeHML();
    afx_msg void OnEnergyChangeHMM();
    afx_msg void OnEnergyChangeHMR();
    afx_msg void OnEnergyChangeHBL();
    afx_msg void OnEnergyChangeHBM();
    afx_msg void OnEnergyChangeHBR();
    afx_msg void OnChemoClick();
    afx_msg void OnProlifClick();
};
```

B.1.6 Static.h

```
#pragma once
#include <stdio.h>
#include <stdarg.h>

#define TISSUE_INI "./Tissue.ini"
#define APP_SECTION "application"
#define APP_LOG     "applog"
#define POTTS_LOG   "pottslog"

//|---------------------------------------------------------------------
//| Name: class LOG
//| Usage:
//|   This class is based on a singleton design pattern that
//|   encapsulates the idea of a log file. This file can be included
//|   in any file that requires the used of either the application
//|   log of Potts log. No instantiation is required and all use is
//|   via scope resolution operater '::', ie, LOG::Format(...).
//|---------------------------------------------------------------------
class LOG
{
public:
    static enum Log {
        Potts=0, //Default cell type.
        Application,
    };
private:
    LOG();
protected:
    LOG(LOG::Log log); //No public constructions...
public:
    virtual ~LOG(void);
private:
    CStdioFile  m_hLog;
    static LOG m_pAppInst; // The Application Log.
    static LOG m_pPottsInst;   // The Potts log.
public:
    static LOG* GetInst(Log log);
public:
    static void WriteLine(LOG::Log, CString* pMsg);
```

```
    static void WriteLine(LOG::Log, LPCSTR pMsg);
    static void Format(LOG::Log, LPCSTR fmt, ...);
};


//|-------------------------------------------------------------------
//| Name: class INI
//| Usage:
//|   This class is based on a singleton design pattern that
//|   encapsulates the idea of a ini file. This file can be included
//|   in any file that requires the used the one application ini
//|   file. No instantiation is required and all use is via scope
//|   resolution operater '::', ie, INI::Format(...).
//|-------------------------------------------------------------------
class INI
{
private:
    INI(void){}
public:
    ~INI(void);
public:
    static int      GetIntValue(LPCSTR pSection, LPCSTR pKey);
    static float    GetFloatValue(LPCSTR pSection, LPCSTR pKey);
    static CString  GetStringValue(LPCSTR pSection, LPCSTR pKey);
    static void     SetIntValue(LPCSTR pSection, LPCSTR pKey,
                                INT nValue);
    static void     SetFloatValue(LPCSTR pSection, LPCSTR pKey,
                                FLOAT fValue);
};
```

B.2 C++ Source Files

B.2.1 CTissue.cpp

```
#include "StdAfx.h"
#include "Tissue.h"

////////////////////////////////////////////////////////////////////
// CTissue Implementation   //
////////////////////////////////////////////////////////////////////
```

```
//|-------------------------------------------------------------------
//| Name: CTissue::CTissue(void)
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Standard default destructor; destroy all allocated
//|   memory, files....
//|-------------------------------------------------------------------
CTissue::CTissue(void)
{

}


//|-------------------------------------------------------------------
//| Name: CTissue::~CTissue(void)
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Standard default destructor; destroy all allocated
//|   memory, files....
//|-------------------------------------------------------------------
CTissue::~CTissue(void)
{
    delete [] m_pCellProliferate;
    delete [] m_pCellCoM;
    delete [] m_pCells;
    delete [] m_phCells;
    delete [] m_pCellVolume;
    delete [] m_pCellVolumeIncrease;
    delete [] m_pCellVolumeDecrease;
    delete [] m_pCellTargetVolume;
    delete [] m_pCellStain;
    delete [] m_pFGF;
    delete [] m_phFGF;
    delete [] m_pRA;
    delete [] m_phRA;
    delete [] m_pRNA;
```

```
}

//|-------------------------------------------------------------------
//| Name: CTissue(char* filename, CDC*pDC)
//| Arguments:
//|    char*  - The fully qualified filename to load the tissue.
//|    CDC*   - The windows device context for drawing the tissue.
//| Output:
//|    N/A
//| Effect:
//|   To create a tissue from a given filename. Typically used when
//|   opening a file.
//|-------------------------------------------------------------------
CTissue::CTissue(char* filename, CDC*pDC)
{
    Load(filename);
    m_pDC = pDC;

    //Get the window dimensions into we will
    // draw the cells/substrate.
    CRect pR;
    CWnd* pWnd = m_pDC->GetWindow();
    pWnd->GetWindowRect(&pR);

    //Set the start point for drawing...
    m_ptViewportOrg.SetPoint((pR.Width()-m_nSubstrateWidth)/2,
                             (pR.Height()-m_nSubstrateHeight)/2);
    m_pDC->SetViewportOrg(m_ptViewportOrg);
    m_rcSubstrate.SetRect(0, 0, m_nSubstrateWidth,
                          m_nSubstrateHeight);
}



//|-------------------------------------------------------------------
//| Name: CTissue(INT nCellsWide, INT nCellsHigh, INT nVolume,
//|               FLOAT fElasticiy, CDC* pDC,
//|               INT nIBorder, INT nJBorder)
//| Arguments:
//|    INT   - The number of cells wide the medium will be.
//|    INT   - The number of cells high the medium will be.
//|    INT   - The square root for the volume for each cell.
```

```
//|    FLOAT - The strength of the boundaries of the cells.
//|    CDC*  - The windows device context for drawing the cells.
//|    INT   - The width of the border on the I-axis.
//|    INT   - The width of the border on the J-axis.
//| Output:
//|    N/A
//| Effect:
//|    Constructor for creating lattice based tissues.
//|-------------------------------------------------------------------
CTissue::CTissue(INT nCellsWide, INT nCellsHigh, INT nVolume,
                 FLOAT fElasticiy, CDC* pDC,
                 INT nIBorder, INT nJBorder)
{
    //Initialise any usefull variables.
    SetCellElasticity(fElasticiy);
    SetSubstrateBorder(CTissue::I, nIBorder);
    SetSubstrateBorder(CTissue::J, nJBorder);
    SetCellsWide(nCellsWide);
    SetCellsHigh(nCellsHigh);
    SetCellVolume(nVolume);
    SetChemotaxis(FALSE);
    SetChemoEnergy(10.f);
    SetProliferation(FALSE);
    SetProlifSteps(10);
    SetProlifRNDMax(3);
    SetProlifVolInc(1);
    SetOutputDC(pDC);
}

//|-------------------------------------------------------------------
//| Name: CTissue::Clone()
//| Arguments:
//|     N/A
//| Output:
//|    CTissue* - A copy of this CTissue object.
//| Effect:
//|   To create a new cell that is a copy of this one but
//|    not initialized.
//|-------------------------------------------------------------------
CTissue* CTissue::Clone()
{
    CTissue* pT = new CTissue();
```

```
    pT->SetCellElasticity(GetCellElasticity());
    pT->SetSubstrateBorder(CTissue::I,
                           GetSubstrateBorder(CTissue::I));
    pT->SetSubstrateBorder(CTissue::J,
                           GetSubstrateBorder(CTissue::J));
    pT->SetCellsWide(GetCellsWide());
    pT->SetCellsHigh(GetCellsHigh());
    pT->SetCellsOrigin(GetCellsOrigin());
    pT->SetCellVolume(GetCellVolume());
    pT->SetChemotaxis(GetChemotaxis());
    pT->SetChemoEnergy(GetChemoEnergy());
    pT->SetChemoTarget(GetChemoTarget());
    pT->SetProliferation(GetProliferation());
    pT->SetProlifSteps(GetProlifSteps());
    pT->SetProlifRNDMax(GetProlifRNDMax());
    pT->SetProlifVolInc(GetProlifVolInc());
    pT->SetOutputDC(GetOutputDC());
    pT->SetCellsCentering(GetCellsCentering());
    pT->SetRAProduction(GetRAProduction());
    pT->SetRNAProduction(GetRNAProduction());
    pT->SetRADecay(GetRADecay());
    pT->SetRNADecay(GetRNADecay());
    pT->SetFGFDecay(GetFGFDecay());
    pT->SetRAKinetics(GetRAKinetics());
    pT->SetRNAKinetics(GetRNAKinetics());
    pT->SetFGFKinetics(GetFGFKinetics());
    pT->SetFGFThreshold(GetFGFThreshold());
    pT->SetRAThreshold(GetRAThreshold());
    pT->SetRAProductionThreshold(GetRAProductionThreshold());

    //Bulk copy the matrices
    memcpy(pT->m_fBBind, m_fBBind, sizeof(FLOAT)*9);
    memcpy(pT->m_fHBind, m_fHBind, sizeof(FLOAT)*9);
    memcpy(pT->m_fTBind, m_fTBind, sizeof(FLOAT)*9);

    LOG::WriteLine(LOG::Potts, "CTissue::Clone()");
    return pT;
}

//|---------------------------------------------------------------------
//| Name: CTissue::Init()
```

```
//| Arguments:
//|     N/A
//| Output:
//|    void
//| Effect:
//|   To initialize the tissue after setting essential variables
//|    via the property get/set s.
//|-------------------------------------------------------------------
void CTissue::Init()
{
    //Initialise any usefull variables.
    m_bStain  = FALSE;
    CCell* pCell  = 0x0;
    m_bInitProliferation = TRUE;
    m_nMoveStep  = 0;
    m_nCellToHighLight  = 1;//HIGHLIGHT_NONE;
    m_clrHighLight  = RGB(0,0,0xFF);
    m_nSubstrateWidth  = m_nCellsWide*m_nCellInitVolume
                         + 2*m_nSubstrateIBorder;
    m_nSubstrateHeight  = m_nCellsHigh*m_nCellInitVolume
                         + 2*m_nSubstrateJBorder;
    m_nCellsMinI  = m_nSubstrateIBorder + m_ptCellsOrigin.x;
    m_nCellsMinJ  = m_nSubstrateJBorder + m_ptCellsOrigin.y;
    m_nCellsMaxI  = m_nSubstrateWidth  - m_nSubstrateIBorder
                         + m_ptCellsOrigin.x;
    m_nCellsMaxJ  = m_nSubstrateHeight - m_nSubstrateJBorder
                         + m_ptCellsOrigin.y;
    m_nNumCells  = m_nCellsWide * m_nCellsHigh+1;
    m_nNumSubCells  = m_nSubstrateWidth * m_nSubstrateHeight;
    m_nMaxCells  = 200 * m_nNumCells;
    m_nMaxSubCells  = m_nNumSubCells;
    m_nTargetVolume  = 2 * m_nCellInitVolume * m_nCellInitVolume;
    m_nMaxVolume  = m_nTargetVolume;
    m_nChemoAttract  = INI::GetIntValue("chemotaxis", "attraction");
    m_ptLastCoM.SetPoint(0,0);
    m_ptCenterOfMass.SetPoint(0,0);

    //Create the array of cells Dynamically.
    m_pCells = new CCell[m_nMaxSubCells];
    m_phCells = new CCell[m_nMaxSubCells];
    m_pRA = new FLOAT[m_nMaxSubCells];
    m_phRA = new FLOAT[m_nMaxSubCells];
    m_pFGF = new FLOAT[m_nMaxSubCells];
```

```
m_phFGF = new FLOAT[m_nMaxSubCells];
m_pRNA = new FLOAT[m_nMaxSubCells];

//Create arrays for describing cell properties.
// By this we mean a Cell as a collection of
// sub-cells.
m_pCellProliferate = new BOOL[m_nMaxCells];
m_pCellCoM = new CPoint[m_nMaxCells];
m_pCellVolume     = new USHORT[m_nMaxCells];
m_pCellVolumeDecrease = new FLOAT[m_nMaxCells];
m_pCellVolumeIncrease = new FLOAT[m_nMaxCells];
m_pCellTargetVolume = new USHORT[m_nMaxCells];
m_pCellStain = new DWORD[m_nMaxCells];

//Get the window dimensions into which we will
// draw the cells/substrate.
CRect R;
CWnd*pWnd = m_pDC->GetWindow();
pWnd->GetWindowRect(&R);

//Set the start point for drawing...
//m_ptViewportOrg.SetPoint((R.Width()-m_nSubstrateWidth)/2,
                         (R.Height()-m_nSubstrateHeight)/2);
m_ptViewportOrg.SetPoint(INI::GetIntValue("substrate","izero"),
                       INI::GetIntValue("substrate","jzero"));
m_pDC->SetViewportOrg(m_ptViewportOrg);
m_rcSubstrate.SetRect(0, 0, m_nSubstrateWidth,
                            m_nSubstrateHeight);

//Initialise the cells.
//memset((void*)m_pCellProliferate, 0, m_nMaxCells*sizeof(BOOL));
memset((void*)m_pRA,0, m_nMaxSubCells*sizeof(BOOL));
memset((void*)m_phRA,0, m_nMaxSubCells*sizeof(BOOL));
memset((void*)m_pRNA,0, m_nMaxSubCells*sizeof(BOOL));
memset((void*)m_pFGF,0, m_nMaxSubCells*sizeof(BOOL));
memset((void*)m_phFGF,0, m_nMaxSubCells*sizeof(BOOL));

for (int i=0; i < m_nMaxCells; i++)
{
    m_pCellProliferate[i] = TRUE;
    m_pCellVolume[i] = 0;
    m_pCellVolumeIncrease[i] = 0;
```

```
        m_pCellVolumeDecrease[i] = 0;
        m_pCellTargetVolume[i] = m_nTargetVolume/2;
        m_pCellStain[i] = PELLUCIDA_CELL;
        m_pCellCoM[i].SetPoint(0, 0);
    }
    m_pCellStain[CCell::Substrate]   = SUBSTRATE_CELL;
    m_pCellTargetVolume[CCell::Substrate] = INT_MAX;

    //Set the coordindates for each cell on the
    // lattice.
    for (UINT J=0; J<m_nSubstrateHeight; J++)
        for (UINT I=0; I<m_nSubstrateWidth; I++)
        {
            m_pCells[J*m_nSubstrateWidth + I].x = I;
            m_pCells[J*m_nSubstrateWidth + I].y = J;
        }

    //Iterate over the cells assigning a cellular ID
    // and cell type; a sequence of numbers {1,2,...,m_nNumCells-1}
    for (UINT J=m_nCellsMinJ; J<m_nCellsMaxJ; J++)
    {
        for (UINT I=m_nCellsMinI; I<m_nCellsMaxI; I++)
        {
            //Pick the cell we're setting ID for.
            pCell = &m_pCells[J*m_nSubstrateWidth + I];

            //Set the default cell type.
            pCell->m_Type = pCell->m_hType = CCell::Pellucida;

            //ID's start from 1 -> m_nNumCells
            pCell->m_nID = pCell->m_nhID = ((J-m_nCellsMinJ))/
                    m_nCellInitVolume * m_nCellsWide +
                    (I-m_nCellsMinI)/m_nCellInitVolume +1;
        }
    }
    Boltzen();
}

//|---------------------------------------------------------------------
//| Name: CTissue::CalcChemicals()
//| Arguments:
//|     N/A
//| Output:
```

```
//|     N/A
//| Effect:
//|     To evolve the current state of chemicals on the lattice,
//|     by the reaction-diffustion equation. Specifically: mRNA,
//|     FGF-8 and RA.
//|-------------------------------------------------------------------
VOID CTissue::CalcChemicals()
{
    float fFGFDiffusion     = 0.f,fFGFConcentration  = 0.f;
    float fRNAConcentration = 0.f;
    float fRADiffusion     = 0.f,  fRAConcentration   = 0.f;
    float fDiffCoeff = 25.1f;

    INT I=0, J=0, nIdx=0;
    CCell*pCell=0;

    for(J=m_nCellsMinJ; J<m_nCellsMaxJ; J++)
    {
        for(I=m_nCellsMinI; I<m_nCellsMaxI; I++)
        {
            nIdx  = J*m_nSubstrateWidth + I;
            pCell = &m_pCells[nIdx];

            //Set the current chemical concentrations
            fRNAConcentration = m_pRNA[nIdx];
            fRAConcentration = m_pRA[nIdx];
            fFGFConcentration = m_pFGF[nIdx];

            //Calculating third chemical.
            fFGFDiffusion=(4*
            (
                m_pFGF[(J+1)*m_nSubstrateWidth + (I-0)]+
                m_pFGF[(J-1)*m_nSubstrateWidth + (I-0)]+
                m_pFGF[(J-0)*m_nSubstrateWidth + (I+1)]+
                m_pFGF[(J-0)*m_nSubstrateWidth + (I-1)]
            )+
                m_pFGF[(J+1)*m_nSubstrateWidth + (I+1)]+
                m_pFGF[(J-1)*m_nSubstrateWidth + (I+1)]+
                m_pFGF[(J+1)*m_nSubstrateWidth + (I-1)]+
                m_pFGF[(J-1)*m_nSubstrateWidth + (I-1)]
                -20*fFGFConcentration
            );
            m_phFGF[nIdx]=(fFGFConcentration
```

```
            +(fFGFDiffusion/fDiffCoeff)
            + m_fFGFKinetics*(fRNAConcentration
            - m_fFGFDecay*fFGFConcentration));

//RNA Produced only by pellucida cells but without
// diffusion. That is, RNA is only produced inside
// cells,and is not secreted externally, however FGF is
// and is dependant on the level of RNA.
if(pCell->m_Type == CCell::Pellucida)
    m_pRNA[nIdx]= 1.f;//(fRNAConcentration
            + m_fRNAKinetics*(m_fRNAProduction-
            m_fRNADecay*fRNAConcentration));
else
    m_pRNA[nIdx]=(fRNAConcentration
            + m_fRNAKinetics*(0-
            m_fRNADecay*fRNAConcentration));

//Calculateing RA Production
fRADiffusion=(4*
(
    m_pRA[(J+1)*m_nSubstrateWidth + (I-0)]+
    m_pRA[(J-1)*m_nSubstrateWidth + (I-0)]+
    m_pRA[(J-0)*m_nSubstrateWidth + (I+1)]+
    m_pRA[(J-0)*m_nSubstrateWidth + (I-1)]
)+
    m_pRA[(J+1)*m_nSubstrateWidth + (I+1)]+
    m_pRA[(J-1)*m_nSubstrateWidth + (I+1)]+
    m_pRA[(J+1)*m_nSubstrateWidth + (I-1)]+
    m_pRA[(J-1)*m_nSubstrateWidth + (I-1)]-
    20*fRAConcentration
);

//RA Production is dependant on critical lower
// threshold of FGF.Below this threshold, RA
// is produced.
if(pCell->m_Type == CCell::Proliferated
        && m_pFGF[nIdx]
        < m_fRAProductionThreshold)
    m_phRA[nIdx]=(fRAConcentration
                +(fRADiffusion/fDiffCoeff)
                + m_fRAKinetics*(m_fRAProduction
                -m_fRADecay*fRAConcentration));
```

```
        else
            m_phRA[nIdx]= (fRAConcentration
                        + (fRADiffusion/fDiffCoeff)
                        + m_fRAKinetics*(0
                        - m_fRADecay*fRAConcentration));
    }
}

for(J=m_nCellsMinJ; J<m_nCellsMaxJ; J++)
    for(I=m_nCellsMinI; I<m_nCellsMaxI; I++)
    {
        m_pFGF[J*m_nSubstrateWidth+I]
            = m_phFGF[J*m_nSubstrateWidth+I];
        m_pRA[J*m_nSubstrateWidth+I]
            = m_phRA[J*m_nSubstrateWidth+I];
    }

//Impose Neumann boundary conditions for RNA and FGF.
for(J=m_nCellsMinJ; J<m_nCellsMaxJ; J++)
{
    m_pRNA[J*m_nSubstrateWidth+(m_nCellsMinI-1)]
        =m_pRNA[J*m_nSubstrateWidth+m_nCellsMinI];
    m_pRNA[J*m_nSubstrateWidth + m_nCellsMaxI]
        =m_pRNA[(J)*m_nSubstrateWidth+(m_nCellsMaxI-1)];
    m_pFGF[J*m_nSubstrateWidth+(m_nCellsMinI-1)]
        =m_pFGF[J*m_nSubstrateWidth + m_nCellsMinI];
    m_pFGF[J*m_nSubstrateWidth + m_nCellsMaxI]
        =m_pFGF[(J)*m_nSubstrateWidth + (m_nCellsMaxI-1)];
}

for(I=m_nCellsMinI; I<m_nCellsMaxI; I++)
{
    m_pRNA[(m_nCellsMinJ-1)*m_nSubstrateWidth + I]
        = m_pRNA[m_nCellsMinJ*m_nSubstrateWidth + I];
    m_pRNA[m_nCellsMaxJ*m_nSubstrateWidth + I]
        = m_pRNA[(m_nCellsMaxJ-1)*m_nSubstrateWidth + I];
    m_pFGF[(m_nCellsMinJ-1)*m_nSubstrateWidth + I]
        = m_pFGF[m_nCellsMinJ*m_nSubstrateWidth + I];
    m_pFGF[m_nCellsMaxJ*m_nSubstrateWidth + I]
        = m_pFGF[(m_nCellsMaxJ-1)*m_nSubstrateWidth + I];

}
```

```
}

//|-------------------------------------------------------------------
//| Name: CTissue::CalcChemotaxis(CCell* pOrigin, CCell* pNeighbour)
//| Arguments:
//|     CCell* - The cell site that represents origin.
//|     CCell* - The current cell that is the potential swap.
//| Output:
//|     FLOAT  - A pre-factor represent the energy bias for
//|              cell swaps.
//| Effect:
//| To calculate the energy bias for a potential swap on the
//|     lattice given the swaps angle the chemo target.
//|-------------------------------------------------------------------
FLOAT CTissue::CalcChemotaxis(CCell* pOrigin, CCell* pNeighbour)
{
    FLOAT  fNI = (FLOAT)pNeighbour->x - (FLOAT)pOrigin->x;
    FLOAT  fNJ = (FLOAT)pNeighbour->y - (FLOAT)pOrigin->y;
    FLOAT  fTI = (FLOAT)m_ptChemoTarget.x - (FLOAT)pOrigin->x;
    FLOAT  fTJ = (FLOAT)m_ptChemoTarget.y - (FLOAT)pOrigin->y;

    FLOAT fMag = sqrt(fTI*fTI + fTJ*fTJ); fTI/=fMag; fTJ/=fMag;
    fMag    = sqrt(fNI*fNI + fNJ*fNJ); fNI/=fMag; fNJ/=fMag;

    FLOAT fDot  = fNI*fTI + fNJ*fTJ;

    return m_nChemoAttract*fDot;
}

//|-------------------------------------------------------------------
//| Name: CTissue::MoveStep()
//| Arguments:
//|     N/A
//| Output:
//|    N/A
//| Effect:
//|   The main simulation point for evolving the state of the
//|   lattice; volumes, proliferation, differentation, ...
//|-------------------------------------------------------------------
void  CTissue::MoveStep()
{
    FLOAT  fDelta = 0.f;
```

```
FLOAT  fEnergy1 = 0.f;
FLOAT  fEnergy2 = 0.f;
FLOAT  fD = 0.f;
INT    nRND = 0x0;
INT    nMinI = m_nSubstrateWidth; //i3
INT    nMaxI = 0;   //i4
INT    nMinJ = m_nSubstrateHeight;  //j3
INT    nMaxJ = 0;
INT    nIdx = 0;
INT    nRNDIdx = 0;
CCell* pCell = 0x0;
CCell* pSelf = 0x0;
CCell* pRight = 0x0;
CCell* pRNDNeighbour = 0x0;
CCell* pLocalNeighbour = 0x0;
CCell* pRNDLocalNeighbour = 0x0;

//Update all of the volumes...
Volume();
CalcChemicals();

//Only Shift the cells every five steps...
if (!(m_nMoveStep%5) && m_bCentering)
    Center();

for (int J=m_nCellsMinJ,CMJ=m_nCellsMaxJ; J<CMJ; J++)
{
    for (int I=m_nCellsMinI,CMI=m_nCellsMaxI; I<CMI; I++)
    {
        nIdx = J*m_nSubstrateWidth + I;
        pSelf  = &m_pCells[nIdx];

        //Calculate any neccessary changes to the min/max's
        // for the cell medium.
        if(pSelf->m_Type != CCell::Substrate)
        {
            if(nMinI > I) nMinI = I; //Find minimum I
            if(nMaxI < I) nMaxI = I; //Find maximum I
            if(nMinJ > J) nMinJ = J; //Find minimum J
            if(nMaxJ < J) nMaxJ = J; //Find maximum J
        }

        //Pick a randon neighbour around the current cell
```

```
// for which we wish to compute a volume change between
// this cell - self - and the random neighbour - nRND.
nRND = rand()%8;
nRNDIdx = (J+gPosY[nRND])*m_nSubstrateWidth
                + (I+gPosX[nRND]);
pRNDNeighbour = &m_pCells[nRNDIdx];

//obviously we only compute the volume change
// for this cell and a neighbouring different cell.
if(pSelf->m_nID != pRNDNeighbour->m_nID)
{
    fDelta  = 0.f;
    fEnergy1 = 0.f;
    fEnergy2 = 0.f;

    //Calculate the total binding energy
    // for all neighbours of the current
    // cell - pSelf.
    for(int k=0; k<8; k++)
    {
        //Select one of eight local neighbours of pSelf.
        pLocalNeighbour = &m_pCells[(J+gPosY[k])
                        *(m_nSubstrateWidth)
                        +(I+gPosX[k])];

        //Sum the total binding energy of all local
        // neighbours of the current cell - self.
        if(pLocalNeighbour->m_nID != pSelf->m_nID)
            fEnergy1 +=m_fHBind[pSelf->m_Type]
                               [pLocalNeighbour->m_Type];

        //Sum the total binding energy of all neighbours
        if(pLocalNeighbour->m_nID !=
                           pRNDNeighbour->m_nID)
            fEnergy2 += m_fHBind[pRNDNeighbour->m_Type]
                        [pLocalNeighbour->m_Type];
    }

    //Compute the total change in energy of
    //  the cell-self.
    fDelta += m_pCellVolumeDecrease[pSelf->m_nID]
           + m_pCellVolumeIncrease[pRNDNeighbour->m_nID];
```

```
                fDelta += fEnergy2 - fEnergy1;

                //Enable cell motility via chemotaxis...
                if (m_bChemotaxis && pSelf->m_Type
                                  == CCell::Pellucida)
                    fDelta -= m_fChemoEnergy
                            *CalcChemotaxis(pSelf, pRNDNeighbour);

                //Update cell if probability is acceptable...
                if (fDelta<0.f || (fDelta<33.
                               && rand()
                               <  m_nBoltzman[(int)(30.*fDelta)]))
                {
                    pSelf->m_nhID  = pRNDNeighbour->m_nID;
                    pSelf->m_hType = pRNDNeighbour->m_Type;
                }
            }
        }
    }

    //Update all of the cell changes...
    for(int i=0; i<m_nNumSubCells; i++)
    {
        m_pCells[i].m_nID  = m_pCells[i].m_nhID;
        m_pCells[i].m_Type = m_pCells[i].m_hType;
    }

    //Adjust all the min/max for drawing.
    m_nCellsMinI = nMinI-35;
    m_nCellsMaxI = nMaxI+35;
    m_nCellsMinJ = nMinJ-35;
    m_nCellsMaxJ = nMaxJ+35;
    m_nMoveStep++;
    CenterOfMass();

    if(m_nMoveStep%m_nProlifSteps==0 && m_bProliferation)
    {
        Differentiate();
        Proliferate();
    }
}

//|-------------------------------------------------------------------
```

```
//| Name: CTissue::ShowChemicals(Chemical chemValue)
//| Arguments:
//|   Chemical - The chemical the user wants to show.
//| Output:
//|   N/A
//| Effect:
//|   To enable a user to view the state of the chemicals.
//|-----------------------------------------------------------------
void CTissue::ShowChemicals(Chemical chemValue)
{
    m_chemShowChemical = chemValue;
}

//|-----------------------------------------------------------------
//| Name: CTissue::Draw(CDC*pDC)
//| Arguments:
//|     BOOL - Do we have to clear the substrate fully before
//|            re-drawing the cells.
//| Output:
//|    N/A
//| Effect:
//|   To draw the current state of the substrate to the currently
//|   active device context.
//|-----------------------------------------------------------------
void  CTissue::Draw(BOOL bInvalidate)
{
    CCell* pCell = 0x0;
    COLORREF C;
    COLORREF outColor;
    COLORREF clrFGF, clrRNA, clrRA;

    //This signifies a windows msg 'WM_PAINT' initiated
    // this draw call so clear the whole substrate.
    if(bInvalidate)
        m_pDC->FillRect(&m_rcSubstrate, &CBrush(SUBSTRATE_CELL));

    //Draw a pretty border around the substrate.
    m_pDC->MoveTo(0,0);
    m_pDC->LineTo(m_rcSubstrate.Width(),0);
    m_pDC->LineTo(m_rcSubstrate.Width(), m_rcSubstrate.Height());
    m_pDC->LineTo(0, m_rcSubstrate.Height());
    m_pDC->LineTo(0,0);
```

```
//Draw the entire medium - Substrate/Cells.
for (INT J=m_nCellsMinJ-1; J < m_nCellsMaxJ+1; J++)
{
    for (INT I=m_nCellsMinI; I < m_nCellsMaxI+1; I++)
    {
        pCell = &m_pCells[J*m_nSubstrateWidth + I];

        //All Cell colours are managed through this array.
        C = m_pCellStain[pCell->m_nID];

        if (pCell->m_nID
            != m_pCells[(J+1)*m_nSubstrateWidth+I].m_nID
            || pCell->m_nID
            != m_pCells[J*m_nSubstrateWidth + (I-1)].m_nID)
            C = BLACK_CELL;

        //Now set the colours of the new cells.
        FLOAT fgf = m_pFGF[J*m_nSubstrateWidth+I]*0xff;
        FLOAT rna = m_pRNA[J*m_nSubstrateWidth+I]*0xff;
        FLOAT ra  = m_pRA[J*m_nSubstrateWidth+I]*0xff;
        clrFGF = RGB(0xff,     0xff-fgf, 0xff-fgf);
        clrRA  = RGB(0xff-ra,  0xff-ra,  0xff);
        clrRNA = RGB(0xff-rna, 0xff,     0xff-rna);

        if (m_chemShowChemical==RNA)
            outColor = clrRNA;
        else if(m_chemShowChemical==FGF)
            outColor = clrFGF;
        else if (m_chemShowChemical==RA)
            outColor = clrRA;
        else
            outColor = C;

        //m_pDC->SetPixelV(I, J, outColor);
        //m_pDC->SetPixelV(I, J+60, C);
        //m_pDC->SetPixelV(I, J-40, clrFGF);
        //m_pDC->SetPixelV(I, J+110, clrRA);
        //m_pDC->SetPixelV(I, J+240, clrRNA);
        m_pDC->SetPixelV(I, J, outColor);

    }
}
```

```
//Erase the stuff that gets left behind...
CBrush* pB = (CBrush*)GetStockObject(WHITE_BRUSH);
CRect r(m_nCellsMinI-1,
        m_nCellsMinJ-100,
        m_nCellsMinI,
        m_nCellsMaxJ+100);
m_pDC->FillRect(&r, pB);
//m_pDC->Rectangle(&r);

//Draw a cross to display the chemotaxis target...
m_pDC->MoveTo(m_ptChemoTarget.x-4, m_ptChemoTarget.y);
m_pDC->LineTo(m_ptChemoTarget.x+5, m_ptChemoTarget.y);
m_pDC->MoveTo(m_ptChemoTarget.x, m_ptChemoTarget.y-4);
m_pDC->LineTo(m_ptChemoTarget.x, m_ptChemoTarget.y+5);

//Draw the center of mass of the red cell mass...
//m_pDC->SetPixelV(m_ptCenterOfMass, RGB(0xff,0xff,0xff));


CString strChem;
if (m_chemShowChemical == RNA)
    strChem = "RNA";
else if (m_chemShowChemical == FGF)
    strChem = "FGF";
else if (m_chemShowChemical == RA)
    strChem = "RA";
else
    strChem = "NONE";

int space=4;
CString str;
str.Format("Cells=%d;
            MoveStep=%d; Chemo
            Target=(%d,%d);
            Chemical=%s",
            m_nNumCells-1,
            m_nMoveStep,
            m_ptChemoTarget.x,
            m_ptChemoTarget.y, strChem);
CSize sz = m_pDC->GetTextExtent(str);
m_pDC->TextOut(10,10, str, str.GetLength());
```

```
}

//|-------------------------------------------------------------------
//| Name: CRect& CTissue::CellRect()
//| Arguments:
//|     N/A
//| Output:
//|    N/A
//| Effect:
//|   Ensure the cells are kept in the center of the substrate.
//|-------------------------------------------------------------------
CRect CTissue::GetCellRect()
{
    return CRect(m_nCellsMinI,
                 m_nCellsMinJ,
                 m_nCellsMaxI,
                 m_nCellsMaxJ);
}


//|-------------------------------------------------------------------
//| Name: CRect& CTissue::SubstrateRect()
//| Arguments:
//|     N/A
//| Output:
//|    N/A
//| Effect:
//|   Ensure the cells are kept in the center of the substrate.
//|-------------------------------------------------------------------
CRect CTissue::GetSubstrateRect()
{
    return m_rcSubstrate;
}

//|-------------------------------------------------------------------
//| Name: CRect& CTissue::SubstrateRect()
//| Arguments:
//|     N/A
//| Output:
//|    N/A
//| Effect:
//|   Ensure the cells are kept in the center of the substrate.
//|-------------------------------------------------------------------
```

```
CPoint CTissue::GetViewportOrg()
{
    return m_ptViewportOrg;
}

//|-------------------------------------------------------------------
//| Name: CTissue::Center()
//| Arguments:
//|     N/A
//| Output:
//|    N/A
//| Effect:
//|   Ensure the cells are kept in the center of the substrate.
//|-------------------------------------------------------------------
void CTissue::Center()
{
    int nMinI = 0;
    int nMaxI = 0;
    int nMinJ = 0;
    int nMaxJ = 0;
    int Idx   = 0;
    int hIdx  = 0;

    //Initialize all cells to be substrate cells.
    for (int i=0,SC=m_nNumSubCells; i<SC; i++)
    {
        m_phCells[i].m_hType = m_phCells[i].m_Type
                             = CCell::Substrate;
        m_phCells[i].m_nID   = m_phCells[i].m_nhID
                             = 0;
    }

    //Calculate new min/max variables
    nMinI = m_nSubstrateIBorder + m_ptCellsOrigin.x;
    nMinJ = m_nSubstrateJBorder + m_ptCellsOrigin.y;
    nMaxI = nMinI + (m_nCellsMaxI - m_nCellsMinI);
    nMaxJ = nMinJ + (m_nCellsMaxJ - m_nCellsMinJ);

    //Draw the recentered cells into the temporary medium.
    for (INT I = m_nCellsMinI, hI=nMinI, CMI=m_nCellsMaxI;
            I < CMI; I++, hI++)
    {
        for (INT J=m_nCellsMinJ, hJ=nMinJ, CMJ=m_nCellsMaxJ;
```

```
            J < CMJ; J++, hJ++)
        {
            //Calculate the the indexes.
            Idx  = J  * m_nSubstrateWidth + I;
            hIdx = hJ * m_nSubstrateWidth + hI;

            m_phCells[hIdx].m_hType = m_pCells[Idx].m_hType;
            m_phCells[hIdx].m_Type  = m_pCells[Idx].m_Type;
            m_phCells[hIdx].m_nhID  = m_pCells[Idx].m_nhID;
            m_phCells[hIdx].m_nID   = m_pCells[Idx].m_nID;
            m_phCells[hIdx].x = hI;
            m_phCells[hIdx].y = hJ;

            //Shift the chemicals back to left
            m_phRA[hIdx] = m_pRA[Idx];
            m_pRNA[hIdx] = m_pRNA[Idx];
            m_phFGF[hIdx] = m_pFGF[Idx];
        }
    }

    //Update the main min/max variables.
    m_nCellsMinI = nMinI;
    m_nCellsMinJ = nMinJ;
    m_nCellsMaxI = nMaxI;
    m_nCellsMaxJ = nMaxJ;

    //Now copy back from the temporary medium
    // to the drawing medium
    memcpy(m_pCells, m_phCells, sizeof(CCell)*m_nNumSubCells);
}

//|---------------------------------------------------------------------
//| Name: CTissue::Volume()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Compute the changes in volumes for all cells.
//|---------------------------------------------------------------------
void CTissue::Volume()
{
```

```
    INT nVolumeDiff = 0;

    //Reset all volumes to 0
    for (int i=0,NC=m_nNumCells; i<NC; i++)
        m_pCellVolume[i] = NO_VOLUME;

    //Iterator over the medium updating cell volumes
    // form the occurences.
    for (int Cell=0, NSC=m_nNumSubCells; Cell<NSC; Cell++)
        m_pCellVolume[m_pCells[Cell].m_nID]++;

    // Calculate for each cell how much the volume should
    //  increase/decrease for each cell.
    for(int Cell=1, NC=m_nNumCells; Cell < NC; Cell++)
    {
        nVolumeDiff = m_pCellVolume[Cell]
                    - m_pCellTargetVolume[Cell];
        m_pCellVolumeIncrease[Cell] = m_fElasticity
                                    *(1+2*nVolumeDiff);
        m_pCellVolumeDecrease[Cell] = m_fElasticity
                                    *(1-2*nVolumeDiff);
    }

    m_pCellVolumeIncrease[0] = 0;
    m_pCellVolumeDecrease[0] = 0;
}

//|---------------------------------------------------------------------
//| Name: CTissue::Boltzen()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To create a probabillity distribution based on the Boltzman
//|---------------------------------------------------------------------
void CTissue::Boltzen()
{
    for(int i=0;  i<1000; i++)
        m_nBoltzman[i]=(int)(RAND_MAX*exp((double) -i/180.));
}
```

```
//|---------------------------------------------------------------------
//| Name:
//|    CTissue::CenterOfMass()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To calculate the center of mass of the cells on the lattice.
//|---------------------------------------------------------------------
void CTissue::CenterOfMass()
{
    int nCellID = 0;
    m_ptCenterOfMass.SetPoint(0,0);
    INT totalvolume = 0;

    for(nCellID=0; nCellID<m_nNumCells;  nCellID++)
    {
        m_pCellCoM[nCellID].x = m_pCellCoM[nCellID].y = 0;
        m_pCellVolume[nCellID] = 0;
    }

    CCell*  pCell = 0x0;
    CPoint* pCoM  = 0x0;
    for(int I=m_nCellsMinI; I<=m_nCellsMaxI; I++)
    {
        for(int J=m_nCellsMinJ; J<=m_nCellsMaxJ; J++)
        {
            pCell = &m_pCells[J*m_nSubstrateWidth + I];
            pCoM  = &m_pCellCoM[pCell->m_nID];

            if(pCell->m_Type != CCell::Substrate)
            {
                pCoM->x += I;
                pCoM->y += J;
                m_pCellVolume[pCell->m_nID]++;

                m_ptCenterOfMass.x += I;
                m_ptCenterOfMass.y += J;
                totalvolume++;
            }
        }
    }
```

```
    m_ptCenterOfMass.SetPoint(m_ptCenterOfMass.x/totalvolume,
                              m_ptCenterOfMass.y/totalvolume);

    for(nCellID=0; nCellID<m_nNumCells; nCellID++)
    {
        pCoM = &m_pCellCoM[nCellID];
        if(m_pCellVolume[nCellID] != 0)
            pCoM->SetPoint(pCoM->x/m_pCellVolume[nCellID],
                           pCoM->y/m_pCellVolume[nCellID]);
        else
            pCoM->x = 0;
    }
}

//|---------------------------------------------------------------------
//| Name: CTissue::Proliferate()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To enable differentiation of cells in regions of high
//|   FGF and RA.
//|---------------------------------------------------------------------
void CTissue::Differentiate()
{
    CCell*  pCell = 0x0;
    CPoint* pP;
    for (int J=m_nCellsMinJ,CMJ=m_nCellsMaxJ; J<CMJ; J++)
    {
        for (int I=m_nCellsMinI,CMI=m_nCellsMaxI; I<CMI; I++)
        {
            pCell = &m_pCells[J*m_nSubstrateWidth + I];
            pP   = &m_pCellCoM[pCell->m_nID];

            //In the highest concetration of FGF red cells switch
            // to green.
            if (m_pFGF[pP->y*m_nSubstrateWidth + pP->x]
                    > m_fFGFThreshold)
            {
             pCell->m_Type = pCell->m_hType
                           = CCell::Proliferated;
```

```
             m_pCellStain[pCell->m_nID] = PROLIFERATED_CELL;
            }

            //If RA level is high enough then any stray red
            // cells should be switched to green.
            if (m_pRA[pP->y*m_nSubstrateWidth + pP->x]
                        > m_fRAThreshold
                        && pCell->m_Type == CCell::Pellucida)
            {
                pCell->m_Type = pCell->m_hType
                              = CCell::Proliferated;
                m_pCellStain[pCell->m_nID] = PROLIFERATED_CELL;
            }
        }
    }
}

//|---------------------------------------------------------------------
//| Name: CTissue::Proliferate()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To model biological proliferation by simulating mitotic cell
//|   cell division.
//|---------------------------------------------------------------------
void CTissue::Proliferate()
{
    static BOOL once = TRUE;
    INT nCellID = 0x0;
    INT nNewCellID = 0x0;
    INT NCell1 = 0x0;
    CCell*  pCell = 0x0;
    CPoint  TempPoint = 0x0;
    CPoint* pNewPoint = 0x0;
    CPoint* pCurPoint = 0x0;
    BOOL bCoM = TRUE;
    bCoM = TRUE;
    INT rndnum = 0;

    //This is the first time this method has been called
    // so initialise all of the target volumes.
```

```
if(m_bInitProliferation)
{
    for (INT i=0; i<m_nMaxCells; i++)
    {
        rndnum = (INT)(m_nTargetVolume/3
               + ((double)rand()/(double)RAND_MAX)
               * ((2*m_nTargetVolume)/3));
        m_pCellTargetVolume[i] = rndnum;
    }
    m_bInitProliferation = FALSE;
}


nNewCellID = m_nNumCells;
//Iterate over the cells performing the proliferation...
for(nCellID=1; nCellID<m_nNumCells; nCellID++)
{

    //if (m_pCellStain[nCellID] == PROLIFERATED_CELL)
    // continue;

    //Increase the cells target volume with a random
    // probability - 1 in m_nProlifRNDMax.
    rndnum = (INT)((((double) rand()/(double)RAND_MAX)
           * m_nProlifRNDMax +1));
    if (rndnum==1)
        m_pCellTargetVolume[nCellID] += m_nProlifVolIncrease;

    //If the cells have grown greater than the
    // maximum volume then divide cells...
    if(m_pCellVolume[nCellID] >= m_nMaxVolume)
    {
        pCurPoint  = &m_pCellCoM[nCellID];
        TempPoint  = *pCurPoint;
        pNewPoint  = &m_pCellCoM[nNewCellID];

        //Old Cell to be divided...
        m_pCellVolume[nCellID] = 0;
        pCurPoint->SetPoint(0, 0);

        //New cell that will be half the division cell.
        m_pCellVolume[nNewCellID] = 0;
        pNewPoint->SetPoint(0, 0);
```

```
            //Iterate over the lattice calulating the
            // cell volumes for the old and new cells...
            for(int I=m_nCellsMinI; I<=m_nCellsMaxI; I++)
            {
                for(int J=m_nCellsMinJ; J<=m_nCellsMaxJ; J++)
                {
                    pCell = &m_pCells[J*m_nSubstrateWidth + I];

                    if(pCell->m_nID ==nCellID)
                    {
                        if(J > TempPoint.y)
                        {
                            pCell->m_nID= Cell->m_nhID=nNewCellID;
                            m_pCellVolume[nNewCellID]++;
                            pNewPoint->Offset(I,J);
                        }
                        else
                        {
                            pCell->m_nID=pCell->m_nhID = nCellID;
                            m_pCellVolume[nCellID]++;
                            pCurPoint->Offset(I, J);
                        }
                    }
                }
            }

            //Reset all of the cell volumes and center fo masses...
            pCurPoint->SetPoint(
                            pCurPoint->x/m_pCellVolume[nCellID],
                            pCurPoint->y/m_pCellVolume[nCellID]);
            pNewPoint->SetPoint(
                        pNewPoint->x/m_pCellVolume[nNewCellID],
                        pNewPoint->y/m_pCellVolume[nNewCellID]);
            m_pCellTargetVolume[nCellID]    = m_nTargetVolume/2;
            m_pCellTargetVolume[nNewCellID] = m_nTargetVolume/2;
            m_pCellStain[nNewCellID] = m_pCellStain[nCellID];
            nNewCellID++;
        }
    }
    m_nNumCells = nNewCellID;
}
```

B.2.2 CCell.cpp

```
//|---------------------------------------------------------------------
//| Name: CCell::CCell(void)
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Standard constructor.
//|---------------------------------------------------------------------
CCell::CCell(void)
{
    m_nID = 0;
    m_nhID = 0;
    m_Type = CCell::Substrate; //Default cell type.
    m_hType = CCell::Substrate; //Default cell type.
    x = y = 0;
}


//|---------------------------------------------------------------------
//| Name: CCell::~CCell(void)
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Standard default destructor; destroy all allocated
//|   memory, files....
//|---------------------------------------------------------------------
CCell::~CCell(void)
{
}
```

B.2.3 CPottsWnd.cpp

```
// PottsWnd.cpp : implementation file
```

```
//

#include "stdafx.h"
#include "PottsWnd.h"
#include "../IO/Static.h"
#include "../Resource.h"

//|---------------------------------------------------------------------
//|  Usefull defines...
//|---------------------------------------------------------------------
#define ID_POPUP_STAINCELL                  WM_USER+1
#define ID_POPUP_ATTRIBUTES                 WM_USER+2
#define ID_POPUP_CELLTYPE_PELLUCIDA         WM_USER+3
#define ID_POPUP_CELLTYPE_PROLIFERATED      WM_USER+4
#define ID_POPUP_SETCHEMOTARGET             WM_USER+5
#define ID_POPUP_SETCELLSORIGIN WM_USER+6
#define ID_POPUP_SHOW_RNA WM_USER+7
#define ID_POPUP_SHOW_FGF WM_USER+8
#define ID_POPUP_SHOW_RA WM_USER+9
#define ID_POPUP_SHOW_NONE WM_USER+10

#define MNULABEL_SHOW_RNA "&RNA"
#define MNULABEL_SHOW_FGF "&FGF"
#define MNULABEL_SHOW_RA "&RA"
#define MNULABEL_SHOW_NONE "&None"

IMPLEMENT_DYNAMIC(CPottsWnd, CWnd)


//|---------------------------------------------------------------------
//| Name:
//|    CPottsWnd::CPottsWnd(CWnd* pParent, CRect* pWndRect)
//| Arguments:
//|    CWnd*  - The parent window that created this window.
//|    Crect* - The rectangle that represents the drawing area for
//|             this one.
//| Output:
//|
//| Effect:
//|
//|---------------------------------------------------------------------
CPottsWnd::CPottsWnd(CWnd* pParent, CRect* pWndRect)
{
```

```
    DWORD dwStdWndStyles = WS_CHILD | WS_VISIBLE
                                    | WS_CLIPCHILDREN
                                    | WS_CLIPSIBLINGS;
    if (!Create(NULL, _T("Potts Window"), dwStdWndStyles,
                *pWndRect,
                pParent, POTTSWND_ID))
        throw "Potts Window Creation Failed.";
}


//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::~CPottsWnd()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   Standard destructor to release any memory or windows objects
//|-------------------------------------------------------------------
CPottsWnd::~CPottsWnd()
{
    delete m_pTissue;

    //Restore default objects to device context.
    m_ZoomDC.SelectObject(m_pOldFont);
    m_ZoomDC.SelectObject(m_pOldBmp);

    //Free all allocated resources.
    m_fntTahoma.DeleteObject();
    m_bmpCells.DeleteObject();
    m_bmpVectors.DeleteObject();
    m_ZoomDC.DeleteDC();
    m_mnuPopup.DestroyMenu();
    m_mnuCellType.DestroyMenu();
    m_mnuChems.DestroyMenu();
    DestroyCursor(m_hCursor);
    m_hCursor =0x0;
    SetCursor(m_hOldCursor);
}


//|-------------------------------------------------------------------
```

```
//| Name:
//|  CPottsWnd::SaveSettings()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To save any current changes to settings to the ini file.
//|-------------------------------------------------------------------
VOID CPottsWnd::SaveSettings()
{
    INI::SetFloatValue("tenergy", "TL", m_pTissue->m_fTBind[0][0]);
    INI::SetFloatValue("tenergy", "TM", m_pTissue->m_fTBind[0][1]);
    INI::SetFloatValue("tenergy", "TR", m_pTissue->m_fTBind[0][2]);
    INI::SetFloatValue("tenergy", "ML", m_pTissue->m_fTBind[1][0]);
    INI::SetFloatValue("tenergy", "MM", m_pTissue->m_fTBind[1][1]);
    INI::SetFloatValue("tenergy", "MR", m_pTissue->m_fTBind[1][2]);
    INI::SetFloatValue("tenergy", "BL", m_pTissue->m_fTBind[2][0]);
    INI::SetFloatValue("tenergy", "BM", m_pTissue->m_fTBind[2][1]);
    INI::SetFloatValue("tenergy", "BR", m_pTissue->m_fTBind[2][2]);

    INI::SetFloatValue("benergy", "TL", m_pTissue->m_fBBind[0][0]);
    INI::SetFloatValue("benergy", "TM", m_pTissue->m_fBBind[0][1]);
    INI::SetFloatValue("benergy", "TR", m_pTissue->m_fBBind[0][2]);
    INI::SetFloatValue("benergy", "ML", m_pTissue->m_fBBind[1][0]);
    INI::SetFloatValue("benergy", "MM", m_pTissue->m_fBBind[1][1]);
    INI::SetFloatValue("benergy", "MR", m_pTissue->m_fBBind[1][2]);
    INI::SetFloatValue("benergy", "BL", m_pTissue->m_fBBind[2][0]);
    INI::SetFloatValue("benergy", "BM", m_pTissue->m_fBBind[2][1]);
    INI::SetFloatValue("benergy", "BR", m_pTissue->m_fBBind[2][2]);

    INI::SetFloatValue("henergy", "TL", m_pTissue->m_fHBind[0][0]);
    INI::SetFloatValue("henergy", "TM", m_pTissue->m_fHBind[0][1]);
    INI::SetFloatValue("henergy", "TR", m_pTissue->m_fHBind[0][2]);
    INI::SetFloatValue("henergy", "ML", m_pTissue->m_fHBind[1][0]);
    INI::SetFloatValue("henergy", "MM", m_pTissue->m_fHBind[1][1]);
    INI::SetFloatValue("henergy", "MR", m_pTissue->m_fHBind[1][2]);
    INI::SetFloatValue("henergy", "BL", m_pTissue->m_fHBind[2][0]);
    INI::SetFloatValue("henergy", "BM", m_pTissue->m_fHBind[2][1]);
    INI::SetFloatValue("henergy", "BR", m_pTissue->m_fHBind[2][2]);
```

```
    INI::SetIntValue("lattice", "cellswide",
                     m_pTissue->GetCellsHigh());
    INI::SetIntValue("lattice", "cellshigh",
                     m_pTissue->GetCellsHigh());
    INI::SetIntValue("lattice", "volume",
                     m_pTissue->GetCellVolume());
    INI::SetFloatValue("lattice", "elasticity",
                     m_pTissue->GetCellElasticity());
    INI::SetFloatValue("chemicals", "RAProduction",
                     m_pTissue->GetRAProduction());
    INI::SetFloatValue("chemicals", "RNAProduction",
                     m_pTissue->GetRNAProduction());
    INI::SetFloatValue("chemicals", "RADecay",
                     m_pTissue->GetRADecay());
    INI::SetFloatValue("chemicals", "RNADecay",
                     m_pTissue->GetRNADecay());
    INI::SetFloatValue("chemicals", "FGFDecay",
                     m_pTissue->GetFGFDecay());
    INI::SetFloatValue("chemicals", "RAKinetics",
                     m_pTissue->GetRAKinetics());
    INI::SetFloatValue("chemicals", "RNAKinetics",
                     m_pTissue->GetRNAKinetics());
    INI::SetFloatValue("chemicals", "FGFKinetics",
                    m_pTissue->GetFGFKinetics());
}


//|---------------------------------------------------------------------
//| Name:
//|    CPottsWnd::LoadSettings()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To load all the settings form the initialization files.
//|---------------------------------------------------------------------
VOID CPottsWnd::LoadSettings()
{
    m_pTissue->m_fTBind[0][0] = INI::GetFloatValue("tenergy", "TL");
    m_pTissue->m_fTBind[0][1] = INI::GetFloatValue("tenergy", "TM");
    m_pTissue->m_fTBind[0][2] = INI::GetFloatValue("tenergy", "TR");
```

```
m_pTissue->m_fTBind[1][0] = INI::GetFloatValue("tenergy", "ML");
m_pTissue->m_fTBind[1][1] = INI::GetFloatValue("tenergy", "MM");
m_pTissue->m_fTBind[1][2] = INI::GetFloatValue("tenergy", "MR");
m_pTissue->m_fTBind[2][0] = INI::GetFloatValue("tenergy", "BL");
m_pTissue->m_fTBind[2][1] = INI::GetFloatValue("tenergy", "BM");
m_pTissue->m_fTBind[2][2] = INI::GetFloatValue("tenergy", "BR");

m_pTissue->m_fBBind[0][0] = INI::GetFloatValue("benergy", "TL");
m_pTissue->m_fBBind[0][1] = INI::GetFloatValue("benergy", "TM");
m_pTissue->m_fBBind[0][2] = INI::GetFloatValue("benergy", "TR");
m_pTissue->m_fBBind[1][0] = INI::GetFloatValue("benergy", "ML");
m_pTissue->m_fBBind[1][1] = INI::GetFloatValue("benergy", "MM");
m_pTissue->m_fBBind[1][2] = INI::GetFloatValue("benergy", "MR");
m_pTissue->m_fBBind[2][0] = INI::GetFloatValue("benergy", "BL");
m_pTissue->m_fBBind[2][1] = INI::GetFloatValue("benergy", "BM");
m_pTissue->m_fBBind[2][2] = INI::GetFloatValue("benergy", "BR");

m_pTissue->m_fHBind[0][0] = INI::GetFloatValue("henergy", "TL");
m_pTissue->m_fHBind[0][1] = INI::GetFloatValue("henergy", "TM");
m_pTissue->m_fHBind[0][2] = INI::GetFloatValue("henergy", "TR");
m_pTissue->m_fHBind[1][0] = INI::GetFloatValue("henergy", "ML");
m_pTissue->m_fHBind[1][1] = INI::GetFloatValue("henergy", "MM");
m_pTissue->m_fHBind[1][2] = INI::GetFloatValue("henergy", "MR");
m_pTissue->m_fHBind[2][0] = INI::GetFloatValue("henergy", "BL");
m_pTissue->m_fHBind[2][1] = INI::GetFloatValue("henergy", "BM");
m_pTissue->m_fHBind[2][2] = INI::GetFloatValue("henergy", "BR");

m_pTissue->SetCellElasticity(INI::GetFloatValue("lattice",
                             "elasticity"));
m_pTissue->SetCellVolume(INI::GetIntValue("lattice",
                         "volume"));

m_pTissue->SetCellsHigh(INI::GetIntValue("lattice",
                        "cellshigh"));
m_pTissue->SetCellsWide(INI::GetIntValue("lattice",
                        "cellswide"));
m_pTissue->SetCellsCentering(INI::GetIntValue("lattice",
                        "centering"));
m_pTissue->SetCellsOrigin(&CPoint(INI::GetIntValue("lattice",
                        "icenter"),
                        INI::GetIntValue("lattice",
```

```
                                          "jcenter")));
m_pTissue->SetSubstrateBorder(CTissue::I,
                        INI::GetIntValue("substrate",
                                          "iborder"));
m_pTissue->SetSubstrateBorder(CTissue::J,
                        INI::GetIntValue("substrate",
                                          "jborder"));
m_pTissue->SetChemotaxis(INI::GetIntValue("chemotaxis",
                                           "enable"));
m_pTissue->SetChemoEnergy(INI::GetFloatValue("chemotaxis",
                                            "energy"));
m_pTissue->SetChemoAttraction(INI::GetIntValue("chemotaxis",
                                            "attraction"));
m_pTissue->SetChemoTarget(&CPoint(
 INI::GetIntValue("chemotaxis", "itarget"),
 INI::GetIntValue("chemotaxis", "jtarget")));


m_pTissue->SetProliferation(
            INI::GetIntValue("proliferation", "enable"));
m_pTissue->SetProlifSteps(
            INI::GetIntValue("proliferation", "steps"));
m_pTissue->SetProlifRNDMax(
            INI::GetIntValue("proliferation", "rndmax"));
m_pTissue->SetProlifVolInc(
            INI::GetIntValue("proliferation", "volinc"));
m_pTissue->SetRADecay(
            INI::GetFloatValue("chemicals", "RADecay"));
m_pTissue->SetRNADecay(
            INI::GetFloatValue("chemicals", "RNADecay"));
m_pTissue->SetFGFDecay(
            INI::GetFloatValue("chemicals", "FGFDecay"));
m_pTissue->SetRAKinetics(
            INI::GetFloatValue("chemicals", "RAKinetics"));
m_pTissue->SetRNAKinetics(
            INI::GetFloatValue("chemicals", "RNAKinetics"));
m_pTissue->SetFGFKinetics(
            INI::GetFloatValue("chemicals", "FGFKinetics"));
m_pTissue->SetRAProduction(
            INI::GetFloatValue("chemicals", "RAProduction"));
m_pTissue->SetRNAProduction(
```

```
                INI::GetFloatValue("chemicals", "RNAProduction"));
    m_pTissue->SetFGFThreshold(
                INI::GetFloatValue("chemicals", "FGFThreshold"));
    m_pTissue->SetRAThreshold(
                INI::GetFloatValue("chemicals", "RAThreshold"));
    m_pTissue->SetRAProductionThreshold(
                INI::GetFloatValue("chemicals",
                                   "RAProductionThreshold"));
}

//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::DCtoBMPFile(LPCSTR pFile)
//| Arguments:
//|    LPCSTR - The file to save the device context to.
//| Output:
//|    N/A
//| Effect:
//|   To write the current device context to a *.BMP file. More
//|   specifically, the substrate and domain of cells.
//|-------------------------------------------------------------------
VOID CPottsWnd::DCtoBMPFile(LPCSTR pFile)
{
    CImage image;
    CBitmap bmp;
    CDC*    pDC = m_pTissue->GetOutputDC();
    CRect R = m_pTissue->GetCellRect();
    //CRect R = m_pTissue->GetSubstrateRect();
    bmp.CreateCompatibleBitmap(pDC, R.Width(), R.Height());
    image.Attach(bmp);

    for (int I=R.top, i=0 ; I< R.bottom; I++, i++)
        for (int J=R.left, j=0; J<R.right; J++, j++)
            image.SetPixel(j, i, pDC->GetPixel(J, I));

    image.Save(pFile, Gdiplus::ImageFormatBMP);
    image.Detach();
    bmp.DeleteObject();
}

BEGIN_MESSAGE_MAP(CPottsWnd, CWnd)
    ON_WM_CREATE()
```

```
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_PAINT()
    ON_WM_LBUTTONUP()
    ON_WM_KEYUP()
    ON_WM_RBUTTONUP()
    ON_WM_SETCURSOR()
    ON_COMMAND(ID_POPUP_SETCELLSORIGIN,
                &CPottsWnd::OnSetCenter)
    ON_COMMAND(ID_POPUP_STAINCELL,
                &CPottsWnd::OnStainCell)
    ON_COMMAND(ID_POPUP_SETCHEMOTARGET,
                &CPottsWnd::SetChemoTarget)
    ON_COMMAND(ID_POPUP_STAINCELL,
                &CPottsWnd::OnStainCell)
    ON_COMMAND(ID_POPUP_CELLTYPE_PELLUCIDA,
                &CPottsWnd::OnSetCellTypePellucida)
    ON_COMMAND(ID_POPUP_CELLTYPE_PROLIFERATED,
                &CPottsWnd::OnSetCellTypeProliferated)
    ON_COMMAND(ID_POPUP_SHOW_FGF, &CPottsWnd::ShowFGF)
    ON_COMMAND(ID_POPUP_SHOW_RNA, &CPottsWnd::ShowRNA)
    ON_COMMAND(ID_POPUP_SHOW_RA, &CPottsWnd::ShowRA)
    ON_COMMAND(ID_POPUP_SHOW_NONE, &CPottsWnd::ShowNone)
END_MESSAGE_MAP()
// CPottsWnd message handlers

//|---------------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
//| Arguments:
//|   LPCREATESTRUCT - Structure to be modified or left alone before
//|                    the window is fully created.
//| Output:
//|
//| Effect:
//|
//|---------------------------------------------------------------------
int CPottsWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    //Setup some variables.
    m_rcWndRect.SetRect(0,0, lpCreateStruct->cx+20,
```

```
                    lpCreateStruct->cy+20);
m_csWndSize.SetSize(lpCreateStruct->cx,
                    lpCreateStruct->cy);
m_strWndName = lpCreateStruct->lpszName;
m_bSetCenter = FALSE;
m_bSetChemoTarget = FALSE;
m_hOldCursor = GetCursor();
m_hCursor = LoadCursor(NULL, IDC_CROSS);

//Create all context menus for the window...
VERIFY(m_mnuCellType.CreateMenu());
m_mnuCellType.InsertMenu(0,MF_BYPOSITION,
                            ID_POPUP_CELLTYPE_PELLUCIDA,
                            "&Stem Cell");
m_mnuCellType.InsertMenu(1,MF_BYPOSITION,
                            ID_POPUP_CELLTYPE_PROLIFERATED,
                            "&Body Cell");
//Create chemical sub menu...
VERIFY(m_mnuChems.CreateMenu());
m_mnuChems.AppendMenu(MF_BYPOSITION, ID_POPUP_SHOW_RNA,
                        MNULABEL_SHOW_RNA);
m_mnuChems.AppendMenu(MF_BYPOSITION, ID_POPUP_SHOW_FGF,
                        MNULABEL_SHOW_FGF);
m_mnuChems.AppendMenu(MF_BYPOSITION, ID_POPUP_SHOW_RA,
                           MNULABEL_SHOW_RA);
m_mnuChems.AppendMenu(MF_BYPOSITION, ID_POPUP_SHOW_NONE,
                           MNULABEL_SHOW_NONE);



VERIFY(m_mnuPopup.CreatePopupMenu());
m_mnuPopup.AppendMenu(MF_POPUP, (UINT)m_mnuCellType.m_hMenu,
                        "&Set Cell Type");
m_mnuPopup.AppendMenu(MF_POPUP, (UINT)m_mnuChems.m_hMenu,
                        "&Show Chemical");
m_mnuPopup.AppendMenu(MF_BYPOSITION, ID_POPUP_STAINCELL,
                        "&Stain Cells");
m_mnuPopup.AppendMenu(MF_SEPARATOR, NULL,
                        "&Set Cell Type");
m_mnuPopup.AppendMenu(MF_BYPOSITION, ID_POPUP_SETCELLSORIGIN,
                        "&Set Cells Origin");
m_mnuPopup.AppendMenu(MF_BYPOSITION, ID_POPUP_SETCHEMOTARGET,
```

```
                            "&Set Chemo Target");
    MENUITEMINFO MII;
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);
    //Now perform the tissue specific initialization...
    if (!OnInit())
        return FALSE;
    return TRUE;
}


//|---------------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnInit(VOID)
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   Initialization function for this subclassed window, were we
//|   will create structures and fonts necessary for the simulation.
//|---------------------------------------------------------------------
INT CPottsWnd::OnInit(VOID)
{
    m_bLeftBtnDown  = FALSE;
    m_pDC = this->GetDC();
    m_nRefreshRate = INI::GetIntValue("pottswindow",
                                      "refreshrate");
    m_fZoomFactor = INI::GetFloatValue("pottswindow",
                                       "zoomfactor");
    m_bVecField = INI::GetIntValue("vectorfieldwindow",
                                      "enable");

    //Create a nice font for showing stats
    // in the window.
    VERIFY(m_fntTahoma.CreateFont(
        12,                         // nHeight
        0,                          // nWidth
        0,                          // nEscapement
        0,                          // nOrientation
        FW_NORMAL,                  // nWeight
```

```
        FALSE,                      // bItalic
        FALSE,                      // bUnderline
        0,                          // cStrikeOut
        ANSI_CHARSET,               // nCharSet
        OUT_DEFAULT_PRECIS,         // nOutPrecision
        CLIP_DEFAULT_PRECIS,        // nClipPrecision
        ANTIALIASED_QUALITY,        // nQuality
        DEFAULT_PITCH | FF_SWISS,   // nPitchAndFamily
        "Tahoma"));                 // lpszFacename


    m_pTissue = new CTissue();
    m_pTissue->SetOutputDC(m_pDC);
    LoadSettings();
    m_pTissue->Init();

    //Create a device context for allowing zooming of
    // potts windows.
    m_ZoomDC.CreateCompatibleDC(m_pDC);
    //m_Bmp.CreateCompatibleBitmap(m_pDC, m_rcWndRect.Width(),
                    m_rcWndRect.Height());
    m_bmpCells.CreateCompatibleBitmap(
     m_pDC,
     m_pTissue->GetSubstrateRect().Width(),
     m_pTissue->GetSubstrateRect().Height());
    m_bmpVectors.CreateCompatibleBitmap(
     m_pDC,
     m_pTissue->GetSubstrateRect().Width(),
     m_pTissue->GetSubstrateRect().Height());

    m_pOldBmp  = m_ZoomDC.SelectObject(&m_bmpCells);
    m_pOldFont = m_ZoomDC.SelectObject(&m_fntTahoma);

    m_ZoomDC.SelectObject(&m_bmpVectors);
    m_ZoomDC.Rectangle(m_pTissue->m_rcSubstrate);
    m_ZoomDC.SelectObject(&m_bmpCells);

    m_pTissue->SetOutputDC(&m_ZoomDC);
    return TRUE;
}
```

```
//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::Reset(INT nWidth, INT nHeight, INT nVolume,
//|    FLOAT fElasticity, INT nIBorder, INT nJBorder)
//| Arguments:
//|   INT   - The width of the new cells.
//|   INT   - The height of the new cells.
//|   INT   - The volume of the cells on the lattice.
//|   FLOAT - The strength of the boundaries of the cells.
//|   INT   - The widht of the I border of the cells.
//|   INT   - The height of the J border of the cells.
//| Output:
//|    N/A
//| Effect:
//|   The user chose to resize of change a fundamental parameter
//|   for the tissue and so we must recreate/reset it.
//|-------------------------------------------------------------------
VOID CPottsWnd::Reset(INT nWidth, INT nHeight,
                      INT nVolume, FLOAT fElasticity,
                      INT nIBorder, INT nJBorder)
{
    CTissue* pTemp = m_pTissue->Clone();
    delete m_pTissue;
    m_pTissue = pTemp;
    m_pTissue->SetCellsWide(nWidth);
    m_pTissue->SetCellsHigh(nHeight);
    m_pTissue->SetCellVolume(nVolume);
    m_pTissue->SetCellElasticity(fElasticity);
    m_pTissue->Init();

    m_ZoomDC.SelectObject(&m_bmpVectors);
    m_ZoomDC.Rectangle(m_pTissue->m_rcSubstrate);
    m_ZoomDC.SelectObject(&m_bmpCells);

    MENUITEMINFO MII;
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_pTissue->ShowChemicals(CTissue::None);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);

    MII.fState = MFS_UNCHECKED;
```

```
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RNA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_FGF, &MII, FALSE);
    Invalidate();
}


//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::GetMatrixEntry(DWORD dwMatrix, USHORT i, USHORT j)
//| Arguments:
//|    DWORD  - The matrix to set an entry for.
//|    USHORT - I index into the matrix.
//|    USHORT - J index into the matrix.
//| Output:
//|    The value to set in the binding matrix at the (i,j) position.
//| Effect:
//|    To set the value of one of the binding matrix elements.
//|-------------------------------------------------------------------
VOID CPottsWnd::SetMatrixEntry(DWORD BitField, USHORT i,
                               USHORT j, FLOAT fValue)
{
    if (!((i>=0&& i<=2) &&(j>=0&& j<=2)))
        throw "Indices out of bounds! - SetMatrixEntry(...)";

        if (!(BitField & MATRIX_TOP || BitField & MATRIX_BOTTOM
                       || BitField & MATRIX_HORIZONTAL))
            throw "Invalid Matrix Type! - SetMatrixEntry(...)";

    if (BitField & MATRIX_TOP)
        m_pTissue->m_fTBind[i][j] = fValue;
    if (BitField & MATRIX_BOTTOM)
        m_pTissue->m_fBBind[i][j] = fValue;
    if (BitField & MATRIX_HORIZONTAL)
        m_pTissue->m_fHBind[i][j] = fValue;
}


//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::GetMatrixEntry(DWORD dwMatrix, USHORT i, USHORT j)
//| Arguments:
```

```
//|    DWORD  - The matrix to return an entry for.
//|    USHORT - I index into the matrix.
//|    USHORT - J index into the matrix.
//| Output:
//|    The value in the binding matrix at the (i,j) position.
//| Effect:
//|    To get the value of one of the binding matrix elements.
//|-----------------------------------------------------------------
FLOAT CPottsWnd::GetMatrixEntry(DWORD dwMatrix, USHORT i, USHORT j)
{
    if (!((i>=0&& i<=2) &&(j>=0&& j<=2)))
        throw "Indices out of bounds! - GetMatrixEntry(...)";

    if (dwMatrix & MATRIX_TOP)
        return m_pTissue->m_fTBind[i][j];
    else if (dwMatrix & MATRIX_BOTTOM)
        return m_pTissue->m_fBBind[i][j];
    else if (dwMatrix & MATRIX_HORIZONTAL)
        return m_pTissue->m_fHBind[i][j];
    else
        throw "Invalid Matrix Type! - GetMatrixEntry(...)";
}



//|-----------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|-----------------------------------------------------------------
void CPottsWnd::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_bLeftBtnDown  = FALSE;
    if (nFlags & MK_CONTROL || nFlags & MK_SHIFT)
    {
        CCell* pCell = 0x0;
        CTissue* pT = m_pTissue;
        CPoint p = point - m_pTissue->GetViewportOrg();
```

```
        //Draw the entire medium - Substrate/Cells.
        for (INT J=pT->m_nCellsMinJ-1; J < pT->m_nCellsMaxJ+1; J++)
        {
            for (INT I=pT->m_nCellsMinI; I < pT->m_nCellsMaxI+1; I++)
            {
                pCell = &pT->m_pCells[J*pT->m_nSubstrateWidth + I];

                if (p == ((*(CPoint*)pCell)))
                {
                 if (nFlags&MK_CONTROL)
                        pT->m_pCellStain[pCell->m_nID]
                            = RGB(0x00,0x00,0xFF);
                 else if(nFlags&MK_SHIFT)
                        pT->m_pCellStain[pCell->m_nID]
                            = RGB(0xFF,0x00,0x00);

                 }
                }
        }
    }
    else
    {
        m_bLeftBtnDown  = TRUE;
        m_ptStart = point;
    }

    CWnd::OnLButtonDown(nFlags, point);

}


//|---------------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|---------------------------------------------------------------------
void CPottsWnd::OnMouseMove(UINT nFlags, CPoint point)
{
```

```
    // TODO: Add your message handler code here and/or call default
    if(m_bLeftBtnDown)
    {
        Invalidate();
        m_ptEnd = point;
    }

    CWnd::OnMouseMove(nFlags, point);
}


//|-------------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnPaint()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   Windows message handler signalling that the window needs to
//|   be re-drawn, but without simulation!
//|-------------------------------------------------------------------
void CPottsWnd::OnPaint()
{
    CPaintDC DC(this); // device context for painting

    DC.SetViewportOrg(m_pTissue->GetViewportOrg());
    Simulate(TRUE, TRUE, FALSE);

    if (m_bLeftBtnDown)
    {
        CPoint vpo = m_pTissue->GetViewportOrg();
        CPoint s = m_ptStart-vpo, e = m_ptEnd-vpo;

        m_rcSelected.SetRect(s.x,  s.y, e.x, e.y);
        m_rcSelected.NormalizeRect();
        DC.MoveTo(s);
        DC.LineTo(e.x, s.y);
        DC.LineTo(e.x, e.y);
        DC.LineTo(s.x, e.y);
        DC.LineTo(s);
    }
    // Do not call CWnd::OnPaint() for painting messages
```

```
}



//|---------------------------------------------------------------------
//| Name:
//|    CPottsWnd::Simulate(BOOL bDraw, BOOL bInvalidate, BOOL bStep)
//| Arguments:
//|    BOOL - Are we drawing the cells and simulating?
//|    BOOL - Do we need to clear the screen BEFORE simulating?
//|    BOOL - Draw only and not simulate?
//| Output:
//|    N/A
//| Effect:
//|    The function that keeps drawing and evolving the cells
//|    on the lattice.
//|---------------------------------------------------------------------
VOID CPottsWnd::Simulate(BOOL bDraw, BOOL bInvalidate, BOOL bStep)
{
    if(bDraw)
    {
        //Draw the Cells Window...
        m_ZoomDC.AssertValid();
        m_pTissue->SetOutputDC(&m_ZoomDC);
        m_pTissue->Draw(bInvalidate);

        m_pDC->StretchBlt(
        0,
        0,
        m_pTissue->m_rcSubstrate.Width()*m_fZoomFactor,
        m_pTissue->m_rcSubstrate.Height()*m_fZoomFactor,
        &m_ZoomDC,
        0,
        0,
        m_pTissue->m_rcSubstrate.Width()+1,
        m_pTissue->m_rcSubstrate.Height(),
        SRCCOPY);


        if (m_bVecField)
        {
            //Now draw the vector window.
```

```
            m_ZoomDC.SelectObject(&m_bmpVectors);


            //m_ZoomDC.TextOutA(50,50,"vector field window", 19);
            m_pDC->StretchBlt(
                            0,
                            200,
            m_pTissue->m_rcSubstrate.Width()*m_fZoomFactor,
            m_pTissue->m_rcSubstrate.Height()*m_fZoomFactor,
                            &m_ZoomDC,
                            0,
                            0,
            m_pTissue->m_rcSubstrate.Width(),
            m_pTissue->m_rcSubstrate.Height(),
            SRCCOPY);

            m_ZoomDC.SelectObject(&m_bmpCells);
        }
        m_pTissue->SetOutputDC(m_pDC);
    }

    if(bStep)
    m_pTissue->MoveStep();


}


//|---------------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnLButtonUp(UINT nFlags, CPoint point)
//| Effect:
//|   Various effects including selecting cells
//|---------------------------------------------------------------------
void CPottsWnd::OnLButtonUp(UINT nFlags, CPoint point)
{
    m_bLeftBtnDown = FALSE;
    CTissue* T = m_pTissue;
    CCell* pCell = 0x0;
    vector<INT>::iterator result;

    CPoint p = point - T->GetViewportOrg();
    //p.x /= m_fZoomFactor; p.y /= m_fZoomFactor;
```

```
//Set a point in the window towards which
// the cells will travel...
if (m_bSetChemoTarget)
{
    m_bSetChemoTarget = FALSE;
    T->SetChemoTarget(&p);
    SetCursor(m_hOldCursor);
}

//Set the top left position for drawing the
// potts window...
if (m_bSetCenter)
{
    m_bSetCenter = FALSE;
    m_pTissue->SetViewportOrg(&point);
}

//Create a vector of cells that where within the
// bounding rectangle drawn by the user...
m_SelectedCells.clear();
if (!m_rcSelected.IsRectEmpty() && !(nFlags&MK_CONTROL
                                || nFlags&MK_SHIFT))
{
    //Draw the entire medium - Substrate/Cells...
    for (INT J=T->m_nCellsMinJ-1; J < T->m_nCellsMaxJ+1; J++)
    {
        for (INT I=T->m_nCellsMinI; I < T->m_nCellsMaxI+1; I++)
        {
         pCell = &T->m_pCells[J*T->m_nSubstrateWidth + I];

         if (m_rcSelected.PtInRect((*(CPoint*)pCell))
                && pCell->m_Type != CCell::Substrate)
         {
                //Only add the cell if we haven't added it yet
                result = find( m_SelectedCells.begin( ),
                       m_SelectedCells.end( ), pCell->m_nID);
                if (result == m_SelectedCells.end())
                    m_SelectedCells.push_back(pCell->m_nID);
         }
        }
    }
```

```
    }

    //If Some cells were selected above then show
    // a popup menu for the user to select what action
    // take with this selection...
    if (m_SelectedCells.size())
    {
     CRect R; ((CWnd*)this)->GetParent()->GetWindowRect(&R);
     CPoint p1 = R.TopLeft()+point;
     m_mnuPopup.TrackPopupMenu(TPM_LEFTALIGN
                    |TPM_RIGHTBUTTON, p1.x, p1.y,  (CWnd*)this);
    }

    //Now force a redraw of the window...
    GetParent()->Invalidate();
    CWnd::OnLButtonUp(nFlags, point);
}

//|-------------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|-------------------------------------------------------------------
VOID CPottsWnd::Staining(BOOL bStain)
{
    m_pTissue->m_bStain = bStain;
}


//|-------------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|-------------------------------------------------------------------
void CPottsWnd::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
```

```
{
    CWnd::OnKeyUp(nChar, nRepCnt, nFlags);
}


//|-------------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|-------------------------------------------------------------------
void CPottsWnd::OnRButtonUp(UINT nFlags, CPoint point)
{
    CWnd::OnRButtonUp(nFlags, point);

    //There's the potential for some kind of popup
    // menu here show get some variables ready just
    // in case.
    CRect R;
    ((CWnd*)this)->GetParent()->GetWindowRect(&R);
    CPoint p = R.TopLeft() +point;

    CCell* pCell=CellHitTest(&point);
    if (pCell) // Hit a Cell.
        m_mnuPopup.TrackPopupMenu(TPM_LEFTALIGN |TPM_RIGHTBUTTON,
                                  p.x, p.y,  (CWnd*)this);
    else //Standard Substrate hit
        m_mnuPopup.TrackPopupMenu(TPM_LEFTALIGN |TPM_RIGHTBUTTON,
                                  p.x, p.y,  (CWnd*)this);
}



//|-------------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
//|-------------------------------------------------------------------
BOOL CPottsWnd::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
```

```
    if (m_bSetCenter || m_bSetChemoTarget)
    {
        SetCursor(m_hCursor);
        return TRUE;
    }
    else
        SetCursor(m_hOldCursor);

    return CWnd::OnSetCursor(pWnd, nHitTest, message);
}



//|---------------------------------------------------------------------
//| Name:
//|    CPottsWnd::CellHitTest(CPoint*p)
//| Arguments:
//|    CPoint* - The point in the window that was clicked.
//| Output:
//|    N/A
//| Effect:
//|    The user wishes get the lattice based cell that they
//|    clicked on.
//|---------------------------------------------------------------------
CCell* CPottsWnd::CellHitTest(CPoint*p)
{
    CRect r = m_pTissue->GetCellRect();
    CPoint pt = (*p)-m_pTissue->GetViewportOrg();

    if (r.PtInRect(pt))
    {
        CTissue* pT = m_pTissue;

        //Draw the entire medium - Substrate/Cells.
        CCell* pCell=NULL;
        for (INT J=pT->m_nCellsMinJ-1;J<pT->m_nCellsMaxJ+1;J++)
        {
            for (INT I=pT->m_nCellsMinI;I<pT->m_nCellsMaxI+1;I++)
            {
                pCell = &pT->m_pCells[J*pT->m_nSubstrateWidth+I];

                if (pt == ((*(CPoint*)pCell)))
                 return pCell;
```

```
            }
        }
    }

    return NULL;
}

//|---------------------------------------------------------------------
//| Name:
//|   CPottsWnd::SetChemoTarget()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   To reposition the chemotaxis target on the substrate.
//|---------------------------------------------------------------------
void CPottsWnd::SetChemoTarget()
{
    m_bSetChemoTarget = TRUE;
    SetCursor(m_hCursor);
}


//|---------------------------------------------------------------------
//| Name:
//| Arguments:
//|
//| Output:
//|
//| Effect:
//|
//|---------------------------------------------------------------------
void CPottsWnd::OnSetCenter()
{
    m_bSetCenter = TRUE;
    SetCursor(m_hCursor);
}



//|---------------------------------------------------------------------
//| Name:
```

```
//|   CPottsWnd::OnStainCell()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   To change the colour of the currently selected cells
//|-----------------------------------------------------------------------
void CPottsWnd::OnStainCell()
{
    CColorDialog dlgColor;
    if(dlgColor.DoModal())
    {
        COLORREF clr = dlgColor.GetColor();

        // TODO: Add your command handler code here
        CCell* pCell =0x0;
        INT nID =0;
        vector<INT>::iterator itr;
        vector<INT>::iterator result;
        CTissue* pT = m_pTissue;

        for (itr=m_SelectedCells.begin();
             itr<m_SelectedCells.end(); itr++)
        {
            //Draw the entire medium - Substrate/Cells.
            for (INT J=pT->m_nCellsMinJ-1;J<pT->m_nCellsMaxJ+1;J++)
            {
                for (INT I=pT->m_nCellsMinI;I<pT->m_nCellsMaxI+1;I++)
                {
                 pCell = &pT->m_pCells[J*pT->m_nSubstrateWidth+I];
                 if (*itr == pCell->m_nID)
                 pT->m_pCellStain[*itr] = clr;
                }
            }
        }
    }

    GetParent()->Invalidate();
}

//|-----------------------------------------------------------------------
//| Name:
```

```
//|   CPottsWnd::OnSetCellTypePellucida()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   User Interface selection to set the selected cells to pellucida
//|-----------------------------------------------------------------
void CPottsWnd::OnSetCellTypePellucida()
{
    SetCellsType(CCell::Pellucida);
}


//|-----------------------------------------------------------------
//| Name:
//|   CPottsWnd::OnSetCellTypeProliferated()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   User Interface selection to set selected cells to proliferated.
//|-----------------------------------------------------------------
void CPottsWnd::OnSetCellTypeProliferated()
{
    SetCellsType(CCell::Proliferated);
}


//|-----------------------------------------------------------------
//| Name:
//|   CPottsWnd::SetCellsType(CCell::Type T)
//| Arguments:
//|   Type  - The type of cells to set the selection to.
//| Output:
//|    N/A
//| Effect:
//|   To change the selection of cells to Type T.
//|-----------------------------------------------------------------
VOID CPottsWnd::SetCellsType(CCell::Type T)
{
```

```
    // TODO: Add your command handler code here
    CCell* pCell =0x0;
    INT nID =0;
    vector<INT>::iterator itr;
    vector<INT>::iterator result;
    CTissue* pT = m_pTissue;

    for (itr=m_SelectedCells.begin();itr<m_SelectedCells.end();itr++)
    {
        //Draw the entire medium - Substrate/Cells.
        for (INT J=pT->m_nCellsMinJ-1; J < pT->m_nCellsMaxJ+1; J++)
        {
            for (INT I=pT->m_nCellsMinI; I < pT->m_nCellsMaxI+1; I++)
            {
                pCell = &pT->m_pCells[J*pT->m_nSubstrateWidth + I];
                if (*itr == pCell->m_nID)
                {
                    if (T==CCell::Pellucida)
                    {
                        pT->m_pCellProliferate[*itr] = 1;
                        pT->m_pCellStain[*itr] = PELLUCIDA_CELL;
                    }
                    else
                    {
                        pT->m_pCellProliferate[*itr] = 0;
                        pT->m_pCellStain[*itr] = PROLIFERATED_CELL;
                    }

                    pCell->m_hType = pCell->m_Type = T;
                }
            }
        }
    }
}


//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::ShowRNA()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
```

```
//| Effect:
//|   Switch rendering to only show RNA chemical.
//|-----------------------------------------------------------------
VOID CPottsWnd::ShowRNA()
{
    MENUITEMINFO MII;
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_pTissue->ShowChemicals(CTissue::RNA);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RNA, &MII, FALSE);

    MII.fState = MFS_UNCHECKED;
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_FGF, &MII, FALSE);
    Invalidate();
}


//|-----------------------------------------------------------------
//| Name:
//|   CPottsWnd::ShowFGF()
//| Arguments:
//|   N/A
//| Output:
//|   N/A
//| Effect:
//|   Switch rendering to only show FGF- protien.
//|-----------------------------------------------------------------
VOID CPottsWnd::ShowFGF()
{
    MENUITEMINFO MII;
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_pTissue->ShowChemicals(CTissue::FGF);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_FGF, &MII, FALSE);

    MII.fState = MFS_UNCHECKED;
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RNA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);
```

```
    Invalidate();
}


//|-------------------------------------------------------------------
//| Name:
//|   CPottsWnd::ShowRA()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Switch rendereing to only show the retinoid acide chemical.
//|-------------------------------------------------------------------
VOID CPottsWnd::ShowRA()
{
    MENUITEMINFO MII;
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_pTissue->ShowChemicals(CTissue::RA);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RA, &MII, FALSE);

    MII.fState = MFS_UNCHECKED;
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RNA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_FGF, &MII, FALSE);
    Invalidate();
}

//|-------------------------------------------------------------------
//| Name:
//|    CPottsWnd::ShowNone()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   To switch off chemical rendering and just show the cells.
//|-------------------------------------------------------------------
VOID CPottsWnd::ShowNone()
{
    MENUITEMINFO MII;
```

```
    MII.cbSize = sizeof(MENUITEMINFO);
    MII.fMask = MIIM_STATE;
    MII.fState = MFS_CHECKED;
    m_pTissue->ShowChemicals(CTissue::None);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_NONE, &MII, FALSE);

    MII.fState = MFS_UNCHECKED;
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_RNA, &MII, FALSE);
    m_mnuPopup.SetMenuItemInfoA(ID_POPUP_SHOW_FGF, &MII, FALSE);
    Invalidate();
}


//|-------------------------------------------------------------------
//| Name:
//|   CPottsWnd::WriteSurfaceData(LPCSTR pFile,
                                  CTissue::Chemical Chem)
//| Arguments:
//|   LPCSTR   - The path to write the surface data to.
//|   Chemical - The chemical data to write to the file(s).
//| Output:
//|    N/A
//| Effect:
//|   To write on or more surface chemical data to files.
//|-------------------------------------------------------------------
VOID CPottsWnd::WriteSurfaceData(LPCSTR pFile,
                                 CTissue::Chemical Chem)
{
    CStdioFile f, fRA, fRNA, fFGF, fCells;
    CFileException ex;
    CTissue* T = m_pTissue;


    //Does the user wish to save all chemical surface data.
    if (Chem == CTissue::All)
    {
        CString str(pFile), path;
        int suffix = str.Find(".");
        path = str.Left(suffix);
        fRA.Open(path+"_ra.txt", CFile::modeCreate
                    | CFile::modeWrite, &ex);
        fRNA.Open(path+"_rna.txt", CFile::modeCreate
```

```
                | CFile::modeWrite, &ex);
    fFGF.Open(path+"_fgf.txt", CFile::modeCreate
                | CFile::modeWrite, &ex);
    fCells.Open(path+"_cells.txt", CFile::modeCreate
                | CFile::modeWrite, &ex);
}
else
    f.Open(pFile, CFile::modeCreate | CFile::modeWrite, &ex);

//Write all of the surface data to the respective files.
void* fData = 0x0;
int W = m_pTissue->m_nSubstrateWidth;
CString str;
for (INT J=m_pTissue->m_nCellsMinJ; J <
        m_pTissue->m_nCellsMaxJ; J++)
{
    for (INT I=m_pTissue->m_nCellsMinI; I <
             m_pTissue->m_nCellsMaxI; I++)
    {
        if (Chem == CTissue::All)
        {
            str.Format("%f,", m_pTissue->m_pRA[J*W+I]*0xFF);
            fRA.WriteString(str);
            str.Format("%f,", m_pTissue->m_pRNA[J*W+I]*0xFF);
            fRNA.WriteString(str);
            str.Format("%f,", m_pTissue->m_pFGF[J*W+I]*0xFF);
            fFGF.WriteString(str);
            str.Format("%d,", m_pTissue->m_pCells[J*W+I].m_Type);
            fCells.WriteString(str);
        }
        else
        {
            if (Chem == CTissue::RA)
             str.Format("%f,",
                            m_pTissue->m_pRA[J*W+I]*0xFF);
            else if(Chem == CTissue::RNA)
             str.Format("%f,",
                            m_pTissue->m_pRNA[J*W+I]*0xFF);
            else if (Chem == CTissue::FGF)
             str.Format("%f,",
                            m_pTissue->m_pFGF[J*W+I]*0xFF);
            else if (Chem == CTissue::Cells)
```

```
                str.Format("%f,",
                                m_pTissue->m_pCells[J*W+I].m_Type);

                f.WriteString(str);
            }
        }
        //Write the current surface data to the respective files.
        if (Chem==CTissue::All)
        {
            fRA.WriteString("\n");
            fRNA.WriteString("\n");
            fFGF.WriteString("\n");
            fCells.WriteString("\n");
        }
        else
            f.WriteString("\n");
    }

    //Close all open files.
    if (Chem==CTissue::All)
    {
        fRA.Close();
        fRNA.Close();
        fFGF.Close();
        fCells.Close();
    }
    else
        f.Close();

}
```

B.2.4 CChemicals.cpp

```
// Chemicals.cpp : implementation file
#include "stdafx.h"
#include "Chemicals.h"
IMPLEMENT_DYNAMIC(CChemicals, CPropertyPage)

//|---------------------------------------------------------------------
//| Name: CChemicals()
//| Arguments:
```

```
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|   Default constructor for property page/ !!not used!!
//|-----------------------------------------------------------------
CChemicals::CChemicals():CPropertyPage(CChemicals::IDD)
{

}

//|-----------------------------------------------------------------
//| Name: CChemicals(CPottsWnd* pPottsWnd)
//| Arguments:
//|    CPottsWnd* - Pointer to the parent window that contains the
//|                 the CPM
//| Output:
//|    N/A
//| Effect:
//|    Typical constructor for setting the parent window that has
//|    a reference to the implementation of the CPM.
//|-----------------------------------------------------------------
CChemicals::CChemicals(CPottsWnd* pPottsWnd)
                      : CPropertyPage(CChemicals::IDD)
{
    m_pPottsWnd = pPottsWnd;
}

//|-----------------------------------------------------------------
//| Name: ~CChemicals()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Default destructor.
//|-----------------------------------------------------------------
CChemicals::~CChemicals()
{
}
```

```
//|-------------------------------------------------------------------
//| Name: DoDataExchange(CDataExchange* pDX)
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    N/A
//| Effect:
//|    CDialog Overide: Initiated when dialog data changes so that
//|    dialog variables can be updated from control values.
//|-------------------------------------------------------------------
void CChemicals::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
}



//|-------------------------------------------------------------------
//| Windows message handler associations to class member functions
//|-------------------------------------------------------------------
BEGIN_MESSAGE_MAP(CChemicals, CPropertyPage)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RAPRODUCTIONSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RNAPRODUCTIONSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RADECAYSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RNADECAYSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RAKINETICSSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RNAKINETICSSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_FGFDECAYSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_FGFTHRESHOLDSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_FGFKINETICSSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
            IDC_RATHRESHOLDSPIN, &CChemicals::OnSpinChange)
    ON_NOTIFY(UDN_DELTAPOS,
```

```
    IDC_RAPRODUCTIONTHRESHOLDSPIN, &CChemicals::OnSpinChange)
    ON_EN_CHANGE(IDC_RAPRODUCTIONTXT,
            &CChemicals::OnChangeRAProduction)
    ON_EN_CHANGE(IDC_RNAPRODUCTIONTXT,
            &CChemicals::OnChangeRNAProduction)
    ON_EN_CHANGE(IDC_RADECAYTXT,
            &CChemicals::OnChangeRADecay)
    ON_EN_CHANGE(IDC_RNADECAYTXT,
            &CChemicals::OnChangeRNADecay)
    ON_EN_CHANGE(IDC_RAKINETICSTXT,
            &CChemicals::OnChangeRAKinetics)
    ON_EN_CHANGE(IDC_RNAKINETICSTXT,
            &CChemicals::OnChangeRNAKinetics)
    ON_EN_CHANGE(IDC_FGFKINETICSTXT,
            &CChemicals::OnChangeFGFKinetics)
    ON_EN_CHANGE(IDC_FGFDECAYTXT,
            &CChemicals::OnChangeFGFDecay)
    ON_EN_CHANGE(IDC_FGFTHRESHOLDTXT,
            &CChemicals::OnChangeFGFThreshold)
    ON_EN_CHANGE(IDC_RATHRESHOLDTXT,
            &CChemicals::OnChangeRAThreshold)
    ON_EN_CHANGE(IDC_RAPRODUCTIONTHRESHOLDTXT,
            &CChemicals::OnChangeRAProductionThreshold)
END_MESSAGE_MAP()


//|-------------------------------------------------------------------
//| Name: DoDataExchange(CDataExchange* pDX)
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    N/A
//| Effect:
//|    CDialog Overide: Called after instantiation and before the
//|    dialog is shown to allow constrols to be initialized along
//|    with other required variables.
//|-------------------------------------------------------------------
BOOL CChemicals::OnInitDialog()
{
    //Must call this first!...
    CPropertyPage::OnInitDialog();
```

```
CEdit* pE = 0x0;
CTissue* pT = m_pPottsWnd->m_pTissue;


//Setup RNA chemicals
pE = GetDlgEdit(IDC_RNAPRODUCTIONTXT);
GetDlgSpin(IDC_RNAPRODUCTIONSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RNAPRODUCTIONTXT, pT->GetRNAProduction(), 2);
pE = GetDlgEdit(IDC_RNADECAYTXT);
GetDlgSpin(IDC_RNADECAYSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RNADECAYTXT, pT->GetRNADecay(), 2);
pE = GetDlgEdit(IDC_RNAKINETICSTXT);
GetDlgSpin(IDC_RNAKINETICSSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RNAKINETICSTXT, pT->GetRNAKinetics(), 3);

//Setup FGF chemicals
pE = GetDlgEdit(IDC_FGFTHRESHOLDTXT);
GetDlgSpin(IDC_FGFTHRESHOLDSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_FGFTHRESHOLDTXT, pT->GetFGFThreshold(), 2);
pE = GetDlgEdit(IDC_FGFDECAYTXT);
GetDlgSpin(IDC_FGFDECAYSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_FGFDECAYTXT, pT->GetFGFDecay(), 2);
pE = GetDlgEdit(IDC_FGFKINETICSTXT);
GetDlgSpin(IDC_FGFKINETICSSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_FGFKINETICSTXT, pT->GetFGFKinetics(), 3);

//Setup RA chemicals
pE = GetDlgEdit(IDC_RAPRODUCTIONTXT);
GetDlgSpin(IDC_RAPRODUCTIONSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RAPRODUCTIONTXT, pT->GetRAProduction(), 2);
pE = GetDlgEdit(IDC_RADECAYTXT);
GetDlgSpin(IDC_RADECAYSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RADECAYTXT, pT->GetRADecay(), 2);
pE = GetDlgEdit(IDC_RAKINETICSTXT);
GetDlgSpin(IDC_RAKINETICSSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RAKINETICSTXT, pT->GetRAKinetics(), 3);
pE = GetDlgEdit(IDC_RATHRESHOLDTXT);
GetDlgSpin(IDC_RATHRESHOLDSPIN)->SetBuddy(pE);
SetDlgFloat(IDC_RATHRESHOLDTXT, pT->GetRAThreshold(), 3);
pE = GetDlgEdit(IDC_RAPRODUCTIONTHRESHOLDTXT);
GetDlgSpin(IDC_RAPRODUCTIONTHRESHOLDSPIN)->SetBuddy(pE);
```

```
    SetDlgFloat(IDC_RAPRODUCTIONTHRESHOLDTXT,
                pT->GetRAProductionThreshold(), 3);
    // return TRUE unless you set the focus to a control
    return TRUE;
}


//|-------------------------------------------------------------------
//| Name: OnSpinChange(NMHDR *pNMHDR, LRESULT *pResult)
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    NMHDR*   - Pointer to structure representing the spin control
//|               that changed.
//|    LRESULT* - How we wish to respond to the change; not used.
//| Effect:
//|    Control Message Handler:
//|    Global message handler for all spin controls on the dialog;
//|    update CPM implementation with parameter changes.
//|-------------------------------------------------------------------
void CChemicals::OnSpinChange(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
    CTissue* pT = m_pPottsWnd->m_pTissue;

    FLOAT fValue = 0.f;

    switch(pNMUpDown->hdr.idFrom)
    {
    case IDC_RNAPRODUCTIONSPIN:
        fValue = GetDlgFloat(IDC_RNAPRODUCTIONTXT)
               + (-pNMUpDown->iDelta*0.01);
        pT->SetRNAProduction(fValue);
        SetDlgFloat(IDC_RNAPRODUCTIONTXT, fValue, 2);
        break;
    case IDC_RAPRODUCTIONSPIN:
        fValue = GetDlgFloat(IDC_RAPRODUCTIONTXT)
               + (-pNMUpDown->iDelta*0.01);
        pT->SetRAProduction(fValue);
        SetDlgFloat(IDC_RAPRODUCTIONTXT, fValue, 2);
        break;
    case IDC_RNADECAYSPIN:
```

```
    fValue = GetDlgFloat(IDC_RNADECAYTXT)
           + (-pNMUpDown->iDelta*0.01);
    pT->SetRNADecay(fValue);
    SetDlgFloat(IDC_RNADECAYTXT, fValue, 2);
    break;
case IDC_RADECAYSPIN:
    fValue = GetDlgFloat(IDC_RADECAYTXT)
           + (-pNMUpDown->iDelta*0.01);
    pT->SetRADecay(fValue);
    SetDlgFloat(IDC_RADECAYTXT, fValue, 2);
    break;
case IDC_RNAKINETICSSPIN:
    fValue = GetDlgFloat(IDC_RNAKINETICSTXT)
           + (-pNMUpDown->iDelta*0.001);
    pT->SetRNAKinetics(fValue);
    SetDlgFloat(IDC_RNAKINETICSTXT, fValue, 3);
    break;
case IDC_RAKINETICSSPIN:
    fValue = GetDlgFloat(IDC_RAKINETICSTXT)
           + (-pNMUpDown->iDelta*0.001);
    pT->SetRAKinetics(fValue);
    SetDlgFloat(IDC_RAKINETICSTXT, fValue, 3);
    break;
case IDC_FGFKINETICSSPIN:
    fValue = GetDlgFloat(IDC_FGFKINETICSTXT)
           + (-pNMUpDown->iDelta*0.001);
    pT->SetFGFKinetics(fValue);
    SetDlgFloat(IDC_FGFKINETICSTXT, fValue, 3);
    break;
case IDC_FGFDECAYSPIN:
    fValue = GetDlgFloat(IDC_FGFDECAYTXT)
           + (-pNMUpDown->iDelta*0.01);
    pT->SetFGFDecay(fValue);
    SetDlgFloat(IDC_FGFDECAYTXT, fValue, 2);
    break;
case IDC_FGFTHRESHOLDSPIN:
    fValue = GetDlgFloat(IDC_FGFTHRESHOLDTXT)
           + (-pNMUpDown->iDelta*0.01);
    pT->SetFGFThreshold(fValue);
    SetDlgFloat(IDC_FGFTHRESHOLDTXT, fValue, 2);
    break;
case IDC_RATHRESHOLDSPIN:
```

```
        fValue = GetDlgFloat(IDC_RATHRESHOLDTXT)
               + (-pNMUpDown->iDelta*0.01);
        pT->SetRAThreshold(fValue);
        SetDlgFloat(IDC_RATHRESHOLDTXT, fValue, 2);
        break;
    case IDC_RAPRODUCTIONTHRESHOLDSPIN:
        fValue = GetDlgFloat(IDC_RAPRODUCTIONTHRESHOLDTXT)
               + (-pNMUpDown->iDelta*0.01);
        pT->SetRAProductionThreshold(fValue);
        SetDlgFloat(IDC_RAPRODUCTIONTHRESHOLDTXT, fValue, 2);
        break;
    }
    *pResult = 0;
}


//|---------------------------------------------------------------------
//| Name: OnChangeRNAProduction()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RNA production textbox changed value so update CPM.
//|---------------------------------------------------------------------
void CChemicals::OnChangeRNAProduction()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRNAProduction(GetDlgFloat(IDC_RNAPRODUCTIONTXT));
}


//|---------------------------------------------------------------------
//| Name: OnChangeRAProduction()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RA production textbox changed value so update CPM.
```

```
//|-------------------------------------------------------------------
void CChemicals::OnChangeRAProduction()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRAProduction(GetDlgFloat(IDC_RAPRODUCTIONTXT));
}

//|-------------------------------------------------------------------
//| Name: OnChangeRNADecay()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RNA Decay textbox changed value so update CPM.
//|-------------------------------------------------------------------
void CChemicals::OnChangeRNADecay()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRNADecay(GetDlgFloat(IDC_RNADECAYTXT));
}

//|-------------------------------------------------------------------
//| Name: OnChangeRADecay()
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RA Decay textbox changed value so update CPM.
//|-------------------------------------------------------------------
void CChemicals::OnChangeRADecay()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRADecay(GetDlgFloat(IDC_RADECAYTXT));
}


//|-------------------------------------------------------------------
//| Name: OnChangeFGFDecay()
```

```
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    FGF Decay textbox changed value so update CPM.
//|-----------------------------------------------------------------
void CChemicals::OnChangeFGFDecay()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetFGFDecay(GetDlgFloat(IDC_FGFDECAYTXT));
}


//|-----------------------------------------------------------------
//| Name: OnChangeRNAKinetics
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RNA Kinetics textbox changed value so update CPM.
//|-----------------------------------------------------------------
void CChemicals::OnChangeRNAKinetics()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRNAKinetics(GetDlgFloat(IDC_RNAKINETICSTXT));
}

//|-----------------------------------------------------------------
//| Name: OnChangeRAKinetics()
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RA kinetics textbox changed value so update CPM.
//|-----------------------------------------------------------------
void CChemicals::OnChangeRAKinetics()
```

```
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRAKinetics(GetDlgFloat(IDC_RAKINETICSTXT));
}


//|---------------------------------------------------------------------
//| Name: OnChangeFGFKinetics()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    FGF kinetics textbox changed value so update CPM.
//|---------------------------------------------------------------------
void CChemicals::OnChangeFGFKinetics()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetFGFKinetics(GetDlgFloat(IDC_FGFKINETICSTXT));
}

//|---------------------------------------------------------------------
//| Name: OnChangeFGFThreshold()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    FGF threshold textbox changed value so update CPM.
//|---------------------------------------------------------------------
void CChemicals::OnChangeFGFThreshold()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetFGFThreshold(GetDlgFloat(IDC_FGFTHRESHOLDTXT));
}

//|---------------------------------------------------------------------
//| Name: OnChangeRAThreshold()
//| Arguments:
//|    N/A
```

```
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RA threshold textbox changed value so update CPM.
//|-----------------------------------------------------------------
void CChemicals::OnChangeRAThreshold()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRAThreshold(GetDlgFloat(IDC_RATHRESHOLDTXT));
}

//|-----------------------------------------------------------------
//| Name: OnChangeRAProductionThreshold()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Control Message Handler:
//|    RA production threshold textbox changed value so update CPM.
//|-----------------------------------------------------------------
void CChemicals::OnChangeRAProductionThreshold()
{
CTissue* pT = m_pPottsWnd->m_pTissue;
pT->SetRAProductionThreshold(
                    GetDlgFloat(IDC_RAPRODUCTIONTHRESHOLDTXT));
}
```

B.2.5 CDynamics.cpp

```
// Dynamics.cpp : implementation file
#include "stdafx.h"
#include "Dynamics.h"
IMPLEMENT_DYNAMIC(CDynamics, CPropertyPage)
//|-----------------------------------------------------------------
//| Name: CDynamics()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
```

```
//|   Default constructor for property page/ !!not used!!
//|-----------------------------------------------------------------
CDynamics::CDynamics()
: CPropertyPage(CDynamics::IDD)
{

}


//|-----------------------------------------------------------------
//| Name: CDynamics(CPottsWnd* pPottsWnd)
//| Arguments:
//|    CPottsWnd* - Pointer to the parent window that contains the
//|                 the CPM
//| Output:
//|    N/A
//| Effect:
//|    Typical constructor for setting the parent window that has
//|    a reference to the implementation of the CPM.
//|-----------------------------------------------------------------
CDynamics::CDynamics(CPottsWnd* pPottsWnd)
                : CPropertyPage(CDynamics::IDD)
{
    m_pPottsWnd = pPottsWnd;
}

//|-----------------------------------------------------------------
//| Name: ~CDynamics()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    Default destructor.
//|-----------------------------------------------------------------
CDynamics::~CDynamics()
{
}


//|-----------------------------------------------------------------
//| Name: DoDataExchange(CDataExchange* pDX)
```

```
//| Arguments:
//|    CDataExchange* - Point for exchangin dialog data variables.
//| Output:
//|    N/A
//| Effect:
//|    CDialog Overide: Initiated when dialog data changes so that
//|    dialog variables can be updated from control values.
//|-----------------------------------------------------------------
void CDynamics::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
}


//|-----------------------------------------------------------------
//| Windows message handler associations to class member functions
//|-----------------------------------------------------------------
BEGIN_MESSAGE_MAP(CDynamics, CPropertyPage)
    ON_WM_HSCROLL()
    ON_EN_CHANGE(IDC_VTL, &CDynamics::OnEnergyChangeVTL)
    ON_EN_CHANGE(IDC_VTM, &CDynamics::OnEnergyChangeVTM)
    ON_EN_CHANGE(IDC_VTR, &CDynamics::OnEnergyChangeVTR)
    ON_EN_CHANGE(IDC_VML, &CDynamics::OnEnergyChangeVML)
    ON_EN_CHANGE(IDC_VMM, &CDynamics::OnEnergyChangeVMM)
    ON_EN_CHANGE(IDC_VMR, &CDynamics::OnEnergyChangeVMR)
    ON_EN_CHANGE(IDC_VBL, &CDynamics::OnEnergyChangeVBL)
    ON_EN_CHANGE(IDC_VBM, &CDynamics::OnEnergyChangeVBM)
    ON_EN_CHANGE(IDC_VBR, &CDynamics::OnEnergyChangeVBR)
    //Horizontal energies.
    ON_EN_CHANGE(IDC_HTL, &CDynamics::OnEnergyChangeHTL)
    ON_EN_CHANGE(IDC_HTM, &CDynamics::OnEnergyChangeHTM)
    ON_EN_CHANGE(IDC_HTR, &CDynamics::OnEnergyChangeHTR)
    ON_EN_CHANGE(IDC_HML, &CDynamics::OnEnergyChangeHML)
    ON_EN_CHANGE(IDC_HMM, &CDynamics::OnEnergyChangeHMM)
    ON_EN_CHANGE(IDC_HMR, &CDynamics::OnEnergyChangeHMR)
    ON_EN_CHANGE(IDC_HBL, &CDynamics::OnEnergyChangeHBL)
    ON_EN_CHANGE(IDC_HBM, &CDynamics::OnEnergyChangeHBM)
    ON_EN_CHANGE(IDC_HBR, &CDynamics::OnEnergyChangeHBR)
    ON_BN_CLICKED(IDC_CHEMO, &CDynamics::OnChemoClick)
    ON_BN_CLICKED(IDC_PROLIF, &CDynamics::OnProlifClick)
END_MESSAGE_MAP()
```

```
//|---------------------------------------------------------------------
//| Name: OnInitDialog()
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|    CDialog Overide: Called after instantiation and before the
//|    dialog is shown to allow constrols to be initialized along
//|    with other required variables.
//|---------------------------------------------------------------------
BOOL CDynamics::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    CTissue* pT = m_pPottsWnd->m_pTissue;

    SetDlgFloat(IDC_VTL,  pT->m_fTBind[0][0]);
    SetDlgFloat(IDC_VTM,  pT->m_fTBind[0][1]);
    SetDlgFloat(IDC_VTR,  pT->m_fTBind[0][2]);
    SetDlgFloat(IDC_VML,  pT->m_fTBind[1][0]);
    SetDlgFloat(IDC_VMM,  pT->m_fTBind[1][1]);
    SetDlgFloat(IDC_VMR,  pT->m_fTBind[1][2]);
    SetDlgFloat(IDC_VBL,  pT->m_fTBind[2][0]);
    SetDlgFloat(IDC_VBM,  pT->m_fTBind[2][1]);
    SetDlgFloat(IDC_VBR,  pT->m_fTBind[2][2]);

    SetDlgFloat(IDC_HTL,  pT->m_fHBind[0][0]);
    SetDlgFloat(IDC_HTM,  pT->m_fHBind[0][1]);
    SetDlgFloat(IDC_HTR,  pT->m_fHBind[0][2]);
    SetDlgFloat(IDC_HML,  pT->m_fHBind[1][0]);
    SetDlgFloat(IDC_HMM,  pT->m_fHBind[1][1]);
    SetDlgFloat(IDC_HMR,  pT->m_fHBind[1][2]);
    SetDlgFloat(IDC_HBL,  pT->m_fHBind[2][0]);
    SetDlgFloat(IDC_HBM,  pT->m_fHBind[2][1]);
    SetDlgFloat(IDC_HBR,  pT->m_fHBind[2][2]);

    ////Setup the slider controls.
    GetDlgSlider(IDC_CHEMOENERGY)->SetRange(0, 5000, 1);
    GetDlgSlider(IDC_PROLIFSTEPS)->SetRange(1, 100,   1);
```

```
    GetDlgSlider(IDC_CHEMOENERGY)->SetPos((INT)pT->GetChemoEnergy());
    GetDlgSlider(IDC_PROLIFSTEPS)->SetPos(pT->GetProlifSteps());

    GetDlgSlider(IDC_PROLIFRNDMAX)->SetRange(1, 10, 1);
    GetDlgSlider(IDC_PROLIFVOLINC)->SetRange(1, 100, 1);
    GetDlgSlider(IDC_PROLIFRNDMAX)->SetPos(pT->GetProlifRNDMax());
    GetDlgSlider(IDC_PROLIFVOLINC)->SetPos(pT->GetProlifVolInc());

    SetDlgInt(IDC_PROLIFRNDMAXTXT, pT->GetProlifRNDMax());
    SetDlgInt(IDC_PROLIFVOLINCTXT, pT->GetProlifVolInc());
    SetDlgFloat(IDC_CHEMOTXT, pT->GetChemoEnergy());
    SetDlgInt(IDC_PROLIFTXT, pT->GetProlifSteps());
    SetDlgCheck(IDC_PROLIF, pT->GetProliferation());
    SetDlgCheck(IDC_CHEMO, pT->GetChemotaxis());

    // return TRUE unless you set the focus to a control
    return TRUE;
    // EXCEPTION: OCX Property Pages should return FALSE
}

//|---------------------------------------------------------------------
//| Name: OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
//| Arguments:
//|    UINT  - The nature of the scroll change.
//| Output:
//|    N/A
//| Effect:
//|    CDialog Overide:
//|    Global dialog handler for all scroll bar changes.
//|---------------------------------------------------------------------
void CDynamics::OnHScroll(UINT nSBCode, UINT nPos,
                          CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default
    CTissue* pT  = m_pPottsWnd->m_pTissue;
    CSliderCtrl* pSC =
              reinterpret_cast<CSliderCtrl*>( pScrollBar );
    CSliderCtrl* pC = GetDlgSlider(IDC_CHEMOENERGY);
    CSliderCtrl* pP = GetDlgSlider(IDC_PROLIFSTEPS);
    CSliderCtrl* pRND = GetDlgSlider(IDC_PROLIFRNDMAX);
    CSliderCtrl* pVInc = GetDlgSlider(IDC_PROLIFVOLINC);
```

```
    int pos = pSC->GetPos();
    if (pSC == pC)
    {
        SetDlgFloat(IDC_CHEMOTXT, ((float)pos)/100, 2);
        pT->SetChemoEnergy(((float)pos)/100);
    }
    else if (pSC == pP)
    {
        SetDlgInt(IDC_PROLIFTXT, pos);
        pT->SetProlifSteps(pos);
    }
    else if (pSC == pRND)
    {
        SetDlgInt(IDC_PROLIFRNDMAXTXT, pos);
        pT->SetProlifRNDMax(pos);
    }
    else if (pSC == pVInc)
    {
        SetDlgInt(IDC_PROLIFVOLINCTXT, pos);
        pT->SetProlifVolInc(pos);
    }

    CPropertyPage::OnHScroll(nSBCode, nPos, pScrollBar);
}



//|---------------------------------------------------------------------
//| Name: OnEnergyChangeVTL
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top left' element
//|    has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeVTL()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                0, 0, GetDlgFloat(IDC_VTL));
}
```

```
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeVTM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top middle' element
//|    has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeVTM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                0, 1, GetDlgFloat(IDC_VTM));
}

//|---------------------------------------------------------------------
//| Name: OnEnergyChangeVTR
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top right' element
//|    has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeVTR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                0, 2, GetDlgFloat(IDC_VTR));
}

//|---------------------------------------------------------------------
//| Name: OnEnergyChangeVML
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'middle left'element
//|    has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeVML()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
```

```
                                1, 0, GetDlgFloat(IDC_VML));
}

//|-------------------------------------------------------------------
//| Name: OnEnergyChangeVMM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'middle middle'
//|    element has changed so update the CPM.
//|-------------------------------------------------------------------
void CDynamics::OnEnergyChangeVMM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                1, 1, GetDlgFloat(IDC_VMM));
}

//|-------------------------------------------------------------------
//| Name: OnEnergyChangeVMR
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'middle right'
//|    element has changed so update the CPM.
//|-------------------------------------------------------------------
void CDynamics::OnEnergyChangeVMR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                1, 2, GetDlgFloat(IDC_VMR));
}

//|-------------------------------------------------------------------
//| Name: OnEnergyChangeVBL
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom left'
//|    element has changed so update the CPM.
//|-------------------------------------------------------------------
```

```
void CDynamics::OnEnergyChangeVBL()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                2, 0, GetDlgFloat(IDC_VBL));
}

//|---------------------------------------------------------------
//| Name: OnEnergyChangeVBM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom middle'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------
void CDynamics::OnEnergyChangeVBM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                2, 1, GetDlgFloat(IDC_VBM));
}

//|---------------------------------------------------------------
//| Name: OnEnergyChangeVBR
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom right'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------
void CDynamics::OnEnergyChangeVBR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_TOP|MATRIX_BOTTOM,
                                2, 2, GetDlgFloat(IDC_VBR));
}

////////////////////////////////////////////////////
// HORIZONTAL TEXT BOXES
////////////////////////////////////////////////////

//|---------------------------------------------------------------
//| Name: OnEnergyChangeHTL
```

```
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top left'
//|    element has changed so update the CPM.
//|-----------------------------------------------------------------
void CDynamics::OnEnergyChangeHTL()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                0, 0, GetDlgFloat(IDC_HTL));
}

//|-----------------------------------------------------------------
//| Name: OnEnergyChangeHTM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top middle'
//|    element has changed so update the CPM.
//|-----------------------------------------------------------------
void CDynamics::OnEnergyChangeHTM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                0, 1, GetDlgFloat(IDC_HTM));
}

//|-----------------------------------------------------------------
//| Name: OnEnergyChangeHTR
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'top right'
//|    element has changed so update the CPM.
//|-----------------------------------------------------------------
void CDynamics::OnEnergyChangeHTR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                0, 2, GetDlgFloat(IDC_HTR));
}
```

```
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeHML
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'middle left'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeHML()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                1, 0, GetDlgFloat(IDC_HML));
}
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeHMM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'middle middle'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeHMM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                1, 1, GetDlgFloat(IDC_HMM));
}

void CDynamics::OnEnergyChangeHMR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                1, 2, GetDlgFloat(IDC_HMR));
}
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeHBL
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom left'
//|    element has changed so update the CPM.
```

```
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeHBL()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                2, 0, GetDlgFloat(IDC_HBL));
}
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeHBM
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom middle'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeHBM()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                2, 1, GetDlgFloat(IDC_HBM));
}
//|---------------------------------------------------------------------
//| Name: OnEnergyChangeHBR
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Textbox Control message handler:
//|    the textbox representation of the matrix 'bottom right'
//|    element has changed so update the CPM.
//|---------------------------------------------------------------------
void CDynamics::OnEnergyChangeHBR()
{
    m_pPottsWnd->SetMatrixEntry(MATRIX_HORIZONTAL,
                                2, 2, GetDlgFloat(IDC_HBR));
}
//|---------------------------------------------------------------------
//| Name: OnChemoClick
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Checkbox Control message handler:
//|    The user wishes to enable/disable chemotaxis.
//|---------------------------------------------------------------------
```

```
void CDynamics::OnChemoClick()
{
m_pPottsWnd->m_pTissue->SetChemotaxis(GetDlgCheck(IDC_CHEMO));
}

//|---------------------------------------------------------------------
//| Name: SetProliferation
//| Arguments:  N/A
//| Output:  N/A
//| Effect:
//|    Windows Checkbox Control message handler:
//|    The user wishes to enable/disable proliferation.
//|---------------------------------------------------------------------
void CDynamics::OnProlifClick()
{
m_pPottsWnd->m_pTissue->SetProliferation(
                                GetDlgCheck(IDC_PROLIF));
}
```

B.2.6 Static.cpp

```
#include "StdAfx.h"
#include "Static.h"

LOG LOG::m_pAppInst(LOG::Application);
LOG LOG::m_pPottsInst(LOG::Potts);

//|---------------------------------------------------------------------
//| Name: LOG::LOG(LOG::Log log)
//| Arguments:
//|    LPCSTR - The Log file we wish to log data to.
//| Output:
//|    void
//| Effect:
//|   Only protected constructor used internally to create LOG
//|   instances.
//|---------------------------------------------------------------------
LOG::LOG(LOG::Log log)
{
    CString strLog;

    if (log ==LOG::Application)
```

```
        strLog = INI::GetStringValue(APP_SECTION, APP_LOG);
    else if (log == LOG::Potts)
        strLog = INI::GetStringValue(APP_SECTION, POTTS_LOG);
    else
        throw "LOG::LOG(LOG::Log) :-  No such log file!";


    CFileException ex;
    if (m_hLog.Open(strLog, CFile::modeCreate
                          | CFile::modeNoTruncate
                          | CFile::modeWrite, &ex))
    {
        // We are appending to the log file so place a time
        //stamp in the log file to identify this session.
        m_hLog.SeekToEnd();
        //Initialize the time zone from the OS.
        char tmpbuf[128]; _tzset();
        CString str;
        str = "\n\n\n--------------------
                ---------------------------------\n";
        _strtime_s( tmpbuf, 128 );
        str.AppendFormat("// LOG SESSION BEGIN -- ( %s,  ", tmpbuf );
        _strdate_s( tmpbuf, 128 );
        str.AppendFormat("%s ) -- //\n\n", tmpbuf );
        m_hLog.WriteString(str);
    }
    else
    {
        TCHAR szError[1024];
        ex.GetErrorMessage(szError, 1024);
        throw szError;
    }
}


LOG::~LOG(void)
{
    // We are appending to the log file so place a time stamp in
    // the log file to identify this session.
    m_hLog.SeekToEnd();
    //Initialize the time zone from the OS.
    CString str;
```

```
    str = "\n\n\n//----------------
          (LOG SESSION  END)---------------//\n";
    m_hLog.WriteString(str);

    m_hLog.Close();
}


//|-------------------------------------------------------------------
//| Name: void LOG::WriteLine(LOG::Log, LPCSTR* pMsg)
//| Arguments:
//|    Log - The Log file we wish to log data to.
//|    LPCSTR - The string that represents the data to log
//| Output:
//|    void
//| Effect:
//|   To Log a CString to logfile suggested by pLog.
//|-------------------------------------------------------------------
void LOG::WriteLine(LOG::Log log, CString* pMsg)
{
    CString str; str+= *pMsg; str+="\n";
    GetInst(log)->m_hLog.WriteString(str);
}


//|-------------------------------------------------------------------
//| Name: void LOG::Log(LOG::Log log, LPCSTR* pMsg)
//| Arguments:
//|    Log - The Log file we wish to log data to.
//|    LPCSTR - The string that represents the data to log
//| Output:
//|    void
//| Effect:
//|   To Log a string to logfile suggested by pLog.
//|-------------------------------------------------------------------
void LOG::WriteLine(LOG::Log log, LPCSTR pMsg)
{
    CString str; str+= pMsg; str+="\n";
    GetInst(log)->m_hLog.WriteString(str);
}
```

```
//|-------------------------------------------------------------------
//| Name: void LOG::Format(LOG::Log, LPCSTR fmt, ...)
//| Arguments:
//|    LPCSTR - The Log file we wish to log data to.
//|    LPCSTR - The string that represents the data to log
//|    ...   - A list of variable arguments for the string 'fmt'
//| Output:
//|    void
//| Effect:
//|   To Log a formatted string from a variable list of arguemtns.
//|-------------------------------------------------------------------
void LOG::Format(LOG::Log log, LPCSTR fmt, ...)
{
   // The list of variable arguments to the function.
   va_list marker
   // Get the list of variable arguments.
   va_start( marker, fmt );
   // The string to write to the log file.
   CString str;
   //Compensates for the %x pairs in fmt.
   BOOL bLastWasMarker = FALSE;


    // Step through the list.
    for(INT i = 0; fmt[i] != '\0'; ++i )
    {
        if (fmt[i] == '%')
        {
            switch(fmt[i+1])
            {
                case 'd': // Integer.
                    str.AppendFormat("%d",
                                    (INT)va_arg(marker, INT));
                break;
                case 'c': // Single Character.
                    str.AppendFormat("%c",
                                    (CHAR)va_arg(marker, CHAR));
                break;
                case 's': // String.
                    str.AppendFormat("%s",
                                    (PCHAR)va_arg(marker, PCHAR));
```

```
                break;
                case 'f': // floating point value.
                    str.AppendFormat("%f", va_arg(marker, DOUBLE));
                break;
            }
            bLastWasMarker=TRUE;
            continue;
        }
        else
        {
            //If the last character was '%' then the current
            // character in the string will be a format specifier
            // so don't add it to the output string for the log.
            if (bLastWasMarker)
            {
                bLastWasMarker=FALSE;
                continue;
            }
            str += fmt[i];
        }

    }

    //Write the output to the log.
    GetInst(log)->m_hLog.WriteString(str);
    va_end( marker );
}



//|-------------------------------------------------------------------
//| Name: LOG* LOG::GetInst(LOG::Log log)
//| Arguments:
//|    LPCSTR - The Log file we wish to get the instance for.
//| Output:
//|    LOG* - The log file associated with pLog.
//| Effect:
//|   To return the instance of the LOG the caller wishes to
//|   log too. This is a private function only called by
//|   static members.
//|-------------------------------------------------------------------
LOG* LOG::GetInst(LOG::Log log)
{
```

```
    LOG* _Instance = 0x0;
    if (log == LOG::Application)
        _Instance = &m_pAppInst;
    else if(log == LOG::Potts)
        _Instance = &m_pPottsInst;
    else
        throw "LOG::GetInst() - No Such Log File!";

    return _Instance;
}




////////////////////////////////////////////////////////////////////
//    //
// INI Implementation    //
//    //
////////////////////////////////////////////////////////////////////


//|-----------------------------------------------------------------
//| Name:
//| Arguments:
//|    N/A
//| Output:
//|    N/A
//| Effect:
//|
//|-----------------------------------------------------------------
INI::~INI(void){}


//|-----------------------------------------------------------------
//| Name:
//|    INI::GetFloatValue(LPCSTR pSection, LPCSTR pKey)
//| Arguments:
//|    LPCSTR - The section in the ini files to read the float.
//|    LPCSTR - The value of the key in the section.
//| Output:
//|    FLOAT - The FLOAT value in the ini file to return the
//|            value for.
//| Effect:
```

```
//|   To extract a value from the current ini files from the pKey
//|   in the pSection.
//|------------------------------------------------------------------
float INI::GetFloatValue(LPCSTR pSection, LPCSTR pKey)
{
    char buffer[1024];
    GetPrivateProfileString(pSection,
                            pKey,
                            NULL,
                            buffer ,
                            1024,
                            TISSUE_INI);
    float f = atof(buffer);
    return f;
}


//|------------------------------------------------------------------
//| Name:
//|    INI::GetStringValue(LPCSTR pSection, LPCSTR pKey)
//| Arguments:
//|    LPCSTR - The section in the ini files to read the int.
//|    LPCSTR - The value of the key in the section.
//| Output:
//|    INT - The int value in the ini file to return the
//|              string value for.
//| Effect:
//|   To extract a value from the current ini files from the pKey
//|   in the pSection.
//|------------------------------------------------------------------
int INI::GetIntValue(LPCSTR pSection, LPCSTR pKey)
{
    return (int)GetPrivateProfileInt(pSection,
                                     pKey,
                                     0xFF,
                                     TISSUE_INI);
}


//|------------------------------------------------------------------
//| Name:
//|    INI::SetIntValue(LPCSTR pSection, LPCSTR pKey, INT nValue)
//| Arguments:
```

```
//|    LPCSTR - The section in the ini files to write the int to.
//|    LPCSTR - The value of the key in the section.
//|    INT    - The int value to write to the ini file.
//| Output:
//|    N/A
//| Effect:
//|   To write a int value to the current ini files at the pKey
//|   in the pSection.
//|-----------------------------------------------------------------
void  INI::SetIntValue(LPCSTR pSection, LPCSTR pKey, INT nValue)
{
    CString str;
    str.Format("%d", nValue);
    WritePrivateProfileString(pSection, pKey, str, TISSUE_INI);
}


//|-----------------------------------------------------------------
//| Name:
//|    INI::SetFloatValue(LPCSTR pSection, LPCSTR pKey, FLOAT fValue)
//| Arguments:
//|    LPCSTR - The section in the ini files to write the string to.
//|    LPCSTR - The value of the key in the section.
//|    FLOAT  - The float value to write to the ini file.
//| Output:
//|    N/A
//| Effect:
//|   To write a float value to the current ini files at the pKey
//|   in the pSection.
//|-----------------------------------------------------------------
void  INI::SetFloatValue(LPCSTR pSection, LPCSTR pKey, FLOAT fValue)
{
    CString str;
    str.Format("%0.1f", fValue);
    WritePrivateProfileString(pSection, pKey, str, TISSUE_INI);
}

//|-----------------------------------------------------------------
//| Name:
//|    INI::GetStringValue(LPCSTR pSection, LPCSTR pKey)
//| Arguments:
//|    LPCSTR - The section in the ini files to read the string to.
```

```
//|    LPCSTR - The value of the key in the section.
//| Output:
//|    CString - The string value in the ini file to return the
//|              string value for.
//| Effect:
//|   To extract a value from the current ini files from the pKey
//|   in the pSection.
//|-----------------------------------------------------------------
CString INI::GetStringValue(LPCSTR pSection, LPCSTR pKey)
{
    char buffer[1024];
    GetPrivateProfileString(pSection, pKey,
                            NULL, buffer ,1024, TISSUE_INI);
    return CString(buffer);
}
```

C Default Initialization file Tissue.ini

```
[application]
applog="./applog.txt"
pottslog="./pottslog.txt"
statslog="./statslog.txt"

[simulation]
usetimer=0  ;0-Windows Timer, 1-Independent Thread.
timerinterval=10 ;How often the windows timer event fires.

[pottswindow]
refreshrate=1
zoomfactor=1.0

[chemicals]
RAKinetics=0.01
RNAKinetics=0.003
FGFKinetics=0.003
RAProduction=1.0
RNAProduction=1.0
RADecay=1.0
RNADecay=1.0
```

```
FGFDecay=1.0
FGFThreshold=0.6
RAThreshold=0.3
RAProductionThreshold=0.0

[lattice]
cellswide=5    ;how many cells wide?
cellshigh=5    ;how many cells high?
volume=5       ;volume of each cell
elasticity=0.5 ;elasticiy of the cell boundary
icenter=-270   ;The left for drawing cells.
jcenter=0      ;The top for drawing cells.
centering=0    ;Should cells stay in one place

[substrate]
izero=10     ;The left for drawing the substrate.
jzero=35     ;The top for drawing the substrate.
iborder=300  ;The space left and right of the cells in the substrate
jborder=150  ;The space top and bottom of the the in the substrate.

[chemotaxis]
enable=1      ;
attraction=-1 ;-1 repulse, 1 attract.
energy=6.f    ;strength of chemo force.
itarget=500   ;x coord of chemo target.
jtarget=191   ;y coord of chemo target.

[proliferation]
enable=1
steps=10   ;proliferate every steps simulations
rndmax=3   ;1/rndmax probability of volume increase
volinc=1   ;increase volume by volinc cell-sites.

[colours]
pellucida=0xFF000000
proliferated=0x00FF00
substrate=0xFFFFFFFF

[tenergy]
TL=0.0
TM=9.0
TR=9.0
ML=9.0
```

```
MM=3.0
MR=3.0
BL=9.0
BM=3.0
BR=3.0

[benergy]
TL=0.0
TM=9.0
TR=9.0
ML=9.0
MM=3.0
MR=3.0
BL=9.0
BM=3.0
BR=3.0

[henergy]
TL=0.0
TM=9.0
TR=9.0
ML=9.0
MM=3.0
MR=3.0
BL=9.0
BM=3.0
BR=3.0
```

References

[1] B. Vasiev, M. Chaplin, C. J. Weijer (2007), "Mathematical modelling of the formation of the primitive streak in the chick embryo" *Dept of Mathematics, University of Dundee*

[2] R. D. del Corral, K. G. Storey (2004), "Opposing FGF and retinoid pathways: a signalling switch that controls differntiation and patterning onset in the extending vertebrate body axis" *BioEssays 26.8, Wily Periodicals : 26:857-869*

[3] I. Roszko, P. Faure, L. Mathis, (2007), Stem cell growth becomes predominant while neural plate progenitor pool decreases during spinal cord elongation. *Developmental Biology 304: pp232-245.*

[4] R. E. Baker, P.K. Maini (2007), Travelling gradients in interacting morphogen systems. *Oxford University, Mathematical Biosciences.*

[5] H. Meinhadt (2008), Models of biological pattern formation: from elementary steps to the organization of embryonic axes. *Max-Plank Institute.*

[6] L. Wolpert, T. Jessel, P. Lawrence, E. Meyerowitz, E. Robertson, J. Smith (2007), Principals Of Development. *Oxford University Press.*

[7] J. C. M. Mombach, J. A. Glazier, R. C. Raphael, M. Zajac, (1995), Quantitive Comparision between Differential Adhesion and Cell Sorting in the Presence and Absence of Fluctuations. *The American Physical Society: Physical Review Letters - Volume 75, Number 11.*

[8] F. Graner, J. A. Grazier (1992), "Simulation of Biological Cell Sorting Using a Two Dimensinal Extended Potts Model" *The American Physical Society: Physical Review Letters - Volume 69, Number 13*

[9] J. A. Grazier, F. Graner (1993), "Simulation of differentail adhesion driven rearrangement of biological cells" *The American Physical Society: Physical Review Letters - Volume 47, Number 3*

[10] R. M. H. Merks, J. A. Glazier (2005), "A cell-centered approach to developmental biology" *Department of Physics, Biocomplexity Institute, Indiana University.*

[11] B. A. Cipra(1987), "An introduction to the Ising Model" *The American Mathematical Monthly - Volume 94, No. 10*

[12] R. Peierls (1936), "Ising's Model Of Ferromagnetism" *Procedings of Cambridge Philosophical Socity, 32*

[13] H. A. Kramers, G. H. Wannier (1941), "Statistics of the two-dimensional magnet I and II" *Phycisal Review - Volume 60*

[14] L. Onsager (1944), "A Two-Dimensional Model With an Order-Disorder Transition." *Physical Review, Volume 65*

[15] G. S. Grest, M. P. Anderson, D. J. Srolovitz (1988), "Domain-growth kinetics for the Q-state Potts model two and three dimensions" *The American Phyical Review B: Volume 38, No. 7*