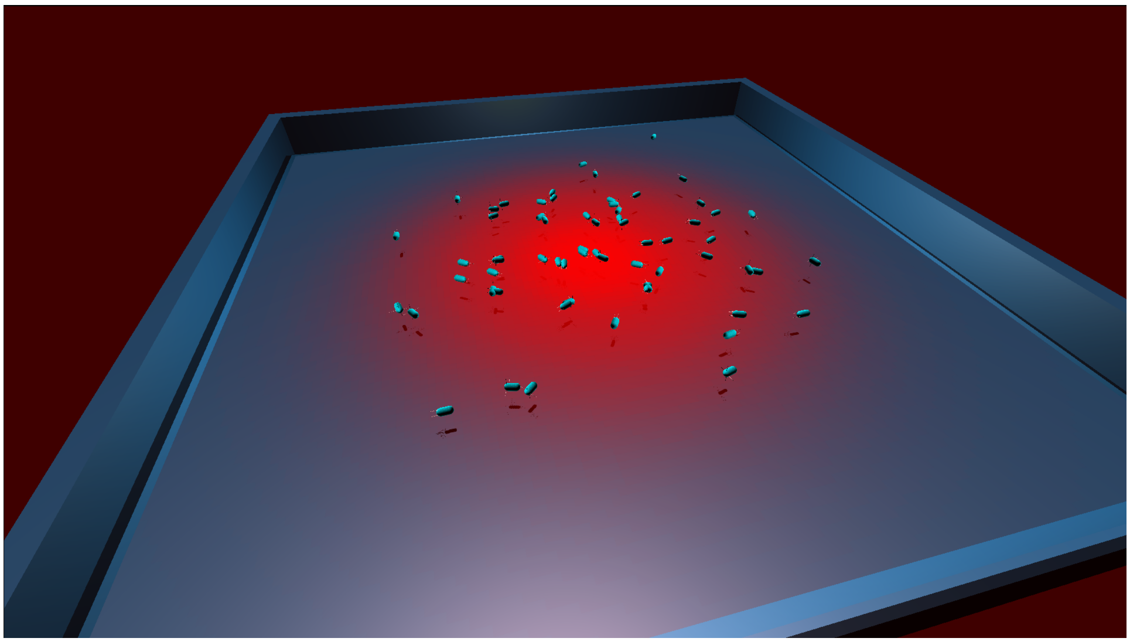
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Simulate Bacterial Movement through Chemotaxis

Bachelor's thesis in Computer science and engineering

Rasmus Durgé
Johan Ek
Jonny Fredriksson
Emil Logren
Mohamad Melhem
Rik Muijs

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Bachelor's thesis 2021

Simulate Bacterial Movement through Chemotaxis

Rasmus Durgé
Johan Ek
Jonny Fredriksson
Emil Logren
Mohamad Melhem
Rik Muijs

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Simulate Bacterial Movement through Chemotaxis

Rasmus Durgé Johan Ek Jonny Fredriksson Emil Logren Mohamad Melhem Rik Muijs

Supervisor: Gustav Lindwall, Mathematical Sciences
Examiner: Jonah Brown-Cohen, Department of Computer Science and Engineering

Bachelor's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Simulate Bacterial Movement through Chemotaxis

Rasmus Durgé, Johan Ek, Jonny Fredriksson, Emil Logren, Mohamad Melhem, Rik Muijs

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis describes the development of an agent-based simulation of E. coli chemotaxis in C# and the Unity game engine. The agents use a mathematical model of the chemical pathway underlying chemotaxis to produce either forward-motion (running) or rotation (tumbling), in response to the concentration of ligand in their immediate environment. This model consists of a system of ODEs from Edgington and Tindall [1] and elements of survival analysis. A tool for analysing data from these simulations was also developed, and used to make quantitative comparisons between simulations. This is used to compare our model to a simplified model of chemotaxis, designed to always display chemotactic behaviour. It is concluded that both models display chemotactic movement, with the simplified model being more effective at finding the ligand source, but the ODE-based model being more adaptive.

Keywords: chemotaxis, computational biology, agent-based simulation, Unity engine

Sammandrag

Denna rapport beskriver utecklingen av en agentbaserad simulation av E. coli-kemotaxi i C# och spelmotorn Unity. Agenterna använder en matematisk modell av den kemotaktiska signalkaskaden som ger upphov till kemotaxi, uttryckt av antingen framåtrörelse (running) eller rotation (tumbling), baserat på ligandkoncentrationen i deras omedelbara omgivning. Denna modell består av ett system av ODEer från Edgington och Tindall [1] och delar av överlevnadsanalys. Ett verktyg för att analysera data från dessa simulationer utvecklades också som del av projektet och användes för att göra kvantitativa jämförelser mellan körningar. Denna data används för att jämföra vår modell med en simplifierad modell av kemotaxi, framtagen för att alltid uppvisa kemotaktiskt beteende. Slutsatsen av denna jämförelse är att båda modeller ger upphov till kemotaktiskt beteende, att den simplifierade modellen mest effektivt hittar ligandkällan, samt att den ODE-baserade modellen är den mest adaptiva.

Nyckelord: kemotaxi, beräkningsbiologi, agentbaserad simulation, Unity spelmotor

Acknowledgements

First and foremost, we wish to express our sincere thanks and gratitude to our supervisor Gustav Lindwall, a doctoral student at the department of Mathematical Sciences at Chalmers University of Technology, for his guidance, assistance and dedicated involvement in every step throughout the process. We would not have accomplished what we have without his constant feedback and insights.

We would also like to thank Erik Sintorn, Research assistant at the Computer Engineering division, Department of Computer Science and Engineering at Chalmers University of Technology, for devoting some time to meet with us. Lastly, we want to thank Becky Bergman from the Division for Language and Communication for her valuable comments on our report writing.

Contents

List of Figures

List of Tables

1 Introduction

Chemotaxis is the phenomenon that describes how certain bacteria navigate their environment towards sources of food through chemical reactions. Since bacteria consist only of single cells, they cannot make decisions based on an understanding of their surroundings, nor with the help of memory. For example, they only know if they have found some ligand, e.g. food, if they have managed to collide with it. They then have to somehow use this information to find more of the ligand, without the ability to compare past collisions. Some bacteria thrive under these restrictions; using only a handful of chemical reactions connected to form feedback loops — a chemical pathway — bacterial species like *Escherichia coli* (E. coli) exhibit great efficiency in ligand finding. The sensitive balance of the reactants and products of these reactions is shifted in response to the bacterium finding ligand, and produces differing responses in movement. Despite these reactions being simple in nature, their emergent effect when connected yields highly adaptive behaviour [3]. This simplicity also makes the problem suited for computational modelling, to see if similarly effective results can be achieved in silico. This project aims to implement an agent-based model, centred around a mathematical model developed by Matthew Edgington and Marcus Tindall [1], and to see how it compares to a model that relies on giving the bacteria enough information to guarantee chemotaxis. This process will also be simulated graphically, in the Unity game engine.

1.1 Purpose

The main purpose of this project is to create an agent-based model to simulate chemotaxis computationally. We also wish to introduce an interactive tool where a user can see how the agents behave with each other, with regards to user configurable settings.

1.1.1 Modelling

Commonly, models of chemotaxis are on the level of a population, using mathematical models describing colony migration [4]. With the relative simplicity of the internal chemistry of individual bacteria, our hope is that an agent based approach can achieve the same observed behaviour. In addition to hopefully simulating the phenomenon of chemotaxis, this approach carries the added benefits of connecting population level dynamics to individual internal processes. The degree to which different models used to represent the internals of the bacteria yield chemotactic behaviour can be used for evaluation of and comparisons between these models. With this in mind, we will implement two models of the chemotactic pathway, as described in 2.1. One of these is taken from Edgington and Tindall [1], attempting to model the internals accurately and without using information the bacteria should not have. The second uses positive changes in concentration to nudge the bacteria towards sources of some attractive ligand. These two models will be compared in terms of fitness, i.e. how well the bacteria navigate towards sources of ligand and how well they survive in the environment. An underlying goal of this project is to see if emergent complexity can arise and yield adaptive behaviour from simple and oblivious models. Having the second — chemotactic by design — model as a point of reference, we are able to quantify the extent to which adaptation and fitness emerges from our properly biological model.

1.1.2 Application

A secondary purpose is to build an interactive product around our agent-based simulation. This entails rendering our simulation in the Unity game engine, visualising the internal chemistry, individual behaviour, and population dynamics immersively. With advanced computer graphics, we hope to build an intuitive understanding of the phenomenon and its underlying mechanics. Our application will also allow the user to control parameters for the extracellular environment in which the bacteria act, and include a tool for analysing data from one or several simulations.

1.2 Scope

The final application is centred around a computational model of the internal chemistry of E. coli underlying chemotaxis. E. coli was chosen for its role as a model organism in the study of chemotaxis. The model will take a concentration of a single type of ligand as input, and produce a state of either running or tumbling as its output. This regulatory model is connected to graphical entities representing bacteria, which can either swim forward or stop completely and rotate. The movement will be at a constant speed, and in straight lines in the direction the bacterium is facing. This is an approximation of real E. coli motility, and will serve the purpose of showcasing chemotaxis well enough. In general, it is only the chemotactic pathway that is claimed to aim at biological accuracy; the rest of the bacterium's physiology is considered an abstraction. Furthermore, all bacteria will be restricted to moving in the plane, and not in three dimensions. It was determined that the costs of implementing three-dimensional models far outweighed their benefits with regards to demonstrating chemotactic behaviour.

These simplified bacteria will inhabit a two-dimensional field with a distribution of different concentrations of ligand. Parameters affecting the shape and layout of this distribution will be possible to change through user input before the simulation, along with population size and duration of the simulation. The bacteria themselves do not affect the distribution, e.g. by eating from it, but the intake of ligand at a given point can be set to decrease by the presence of neighbouring bacteria. An exception to this is made for a proof-of-concept with complete environmental dynamics developed in Matlab. Further abstractions for how the bacteria interact with their chemical surroundings are that they only have one type of receptor and that this is simply located at its centre of mass. The mechanics of ligands binding to the receptor and the bacterium absorbing and metabolising them is also omitted in this project. While chemotaxis also works for movement away from chemorepellents, such as toxins, these will not be included in our simulation. The extracellular environment will be visualised graphically and be of a fixed size, outside which bacteria cannot venture.

The simulation also includes simple cell division and -death. Just as with ligand intake, the realistic details of these are outside the scope of this project. They are included as a measure of the fitness of the bacteria as a population; by allowing our bacteria to thrive or fail in some quantifiable way, we can get some metric for biological success for a given population. Finished simulations can be analysed and compared through an external tool also part of this project. This tool extracts data from the simulation and performs statistical analyses and visualisations of one or more simulations.

2
Theory

In this chapter, we introduce the underlying theory used to develop our simulation. This theory includes the chemotactic pathway in biological detail; the extracellular environment of ligand the bacteria inhabit; ordinary differential equations used for chemical modelling; elements of survival analysis used to model triggering of events; agent-based modelling; and the Unity game engine used to render our simulation.

2.1 Chemotactic Pathway

The chemotactic pathway in E. coli describes the signal cascade and feedback loop of internal proteins that give way to chemotactic behaviour. The following section is a description of this pathway, based on Prescott's Microbiology [5].

E. coli bacteria move through the rotation of flagella, rotor like appendages sticking out through their outer membrane. These flagella can rotate either counter-clockwise (CCW) or clockwise (CW). When the flagella rotate CCW, the bacterium swims forward, in an approximately straight line. We call this state *running*. When the flagella instead rotate CW, the bacterium halts and rotates to face in a new direction. This state is called *tumbling*. It is by alternating between these states that E. coli navigate their environments. Adaptation to seek out favourable conditions comes from shifting the ratio of tumbling to running, and this shift emerges from intracellular signals in the chemotactic pathway.

These signals are transmitted through phosphorylation, which means the transfer of a phosphoryl group from one chemical compound to another. We can view phosphorylated chemicals as switched on, and those without as off. The signalling cascade is mainly made up of a closed system of proteins phosphorylating and dephosphorylating each other. The rate at which these processes occur is shifted based on the state of transmembrane receptors, or MCPs, which in turn are affected by the presence of chemicals in the environment. It is finally the balance of phosphorylated and unphosphorylated chemicals that affect the bacterium's tumbling bias. The result is that when the receptor is sensing low concentrations of ligand, the tumbling bias will increase, and vice versa. Figure 2.1 provides a graphical representation of the pathway [2].

In more detail, the proteins involved in the chemotactic pathway of E. coli are CheW, CheA, CheB, CheY, CheZ, and CheR. CheW is connected to the receptor, or MCP, and CheY is what affects tumbling bias. CheA autophosphorylates constantly, at a rate that is affected by the activity of the MCP. When activity is low in the receptor, the rate of autophosphorylation of CheA decreases. The activity of the

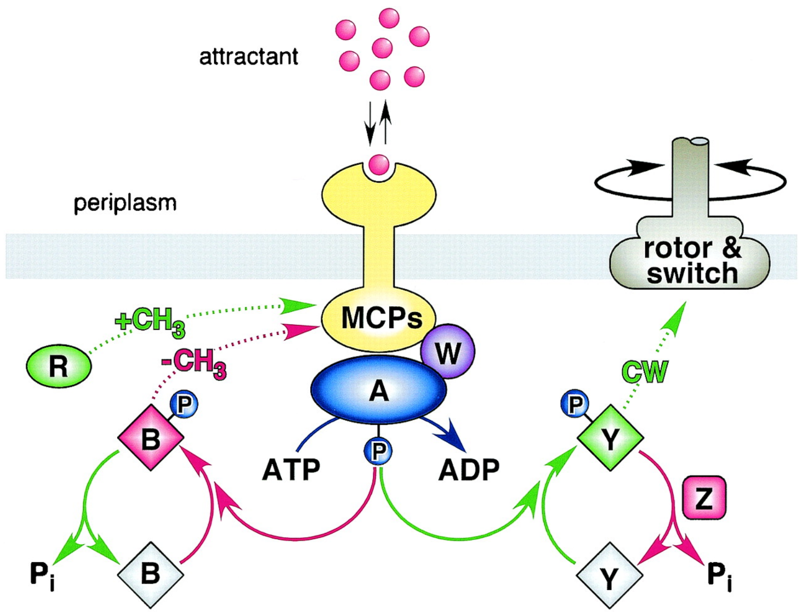


Figure 2.1: Graphical overview of the chemotactic pathway, showing the dynamics of the signalling proteins in a feedback loop. From [2]. Reproduced with permission.

MCP is, in part, decreased by binding with ligand or increased in their absence. Phosphorylated CheA, or CheA-P, spontaneously transfers its phosphoryl group to either CheB or CheY at set rates. The higher the levels of CheY-P, the higher the frequency of tumbling. It thus follows that, in the absence of ligand, CheA will autophosphorylate more rapidly, in turn yielding more CheY-P, finally leading to more frequent tumbling. To prevent the bacterium from tumbling indefinitely, CheY-P quickly loses its phosphoryl group both spontaneously and mediated by CheZ. This part of the chemotactic pathway explains how E. coli can be responsive to their environment. It is, however, lacking an explanation for how they manage to be adaptive and continually seek to improve their position.

Adaptation is explained by what happens to the phosphorylated CheB, which is the second byproduct of phosphotransfer from CheA-P, and CheR. In parallel to the signalling described so far, the protein CheR constantly adds methyl groups to the MCP. These methyl groups slowly increase the activity of the MCP, similar to the effect of a low-ligand environment. The purpose of this is to make the bacterium more prone to seek better conditions, by tumbling, at a constant concentration of ligand. Counteracting this constant methylation is the demethylation performed by CheB-P. When CheB-P removes methyl groups from the MCP, it lowers its activity, allowing for longer smooth runs. This force drives the cell from standing still after high frequencies of tumbling. Just like CheY-P is dephosphorylated constantly, CheB-P steadily loses its phosphoryl group.

The result is a handful of simple reactions, each nudging the internal state of the bacterium. When put together, the balance of these counteracting forces produces a highly efficient adaptation in the swimming behaviour of E. coli.

2.2 Chemical Modelling

In chemistry, the law of mass action states that the kinetic rates at which reactions occur are proportional to the concentrations of its reactants [6]. We can model chemical reactions mathematically with ordinary differential equations (ODEs). These would express the rates of change in concentration of reactants as a function of their concentrations. When regarding a chemical reaction network [7] with multiple reactions, reactants, and products, we can construct a system of ODEs to capture the dynamics of the entire network. One step further, there are cases in which the products of one such reaction serve as reactants in another, in which case the kinetic rate of the latter depends on the former. Generally, we can have networks of reactions in which the sets of reactants and products intersect on multiple counts. Since the kinetic rates of the reactions of such a network are interdependent, the entire system of ODEs has to be solved simultaneously, which is often done numerically.

2.2.1 Chemotactic Pathway Model

The chemotactic pathway of Section 2.1 can be mathematically modelled as a chemical reaction network. For this system, we are interested in expressing the quantity of each non-constant signalling protein — CheA, CheB, and CheY — that is phosphorylated, as well as the methylation level of the MCP. We should have four equations describing these four quantities. We see that CheY-P and CheB-P will depend on the amount of CheA-P. CheA-P will in turn depend on the methylation of the MCP, which depends on CheB-P, CheR, and external stimuli. The signalling proteins also phosphorylate and dephosphorylate spontaneously, depending on the current levels of phosphorylation. Edgington and Tindall [1] have developed just such a system of ODEs, which can be seen in (2.1). These equations describe methylation, followed by concentrations of CheA-P, CheY-P, and CheB-P.

$$\begin{aligned}
\frac{dm}{d\tau} &= \gamma_R(1-\phi) - \gamma_B b_p^2 \phi \\
\frac{da_p}{d\tau} &= \phi \overline{k}_1(1-a_p) - \overline{k}_2(1-y_p)a_p - \overline{k}_3(1-b_p)a_p \\
\frac{dy_p}{d\tau} &= \alpha_1 \overline{k}_2(1-y_p)a_p - (\overline{k}_4+\overline{k}_6)y_p \\
\frac{db_p}{d\tau} &= \alpha_2 \overline{k}_3(1-b_p)a_p - \overline{k}_5 b_p
\end{aligned} \tag{2.1}$$

$$\phi = \frac{1}{1+e^F}, \tag{2.2}$$

$$F = N\left(1 - \frac{m}{2} + \ln\left(\frac{1+L/K_a^{off}}{1+L/K_a^{on}}\right)\right) \tag{2.3}$$

ϕ can be seen as the overall receptor activity and F is the free energy of a receptor signalling team [1]. It is the free energy that is directly affected by current ligand concentration, L. Other than that, F depends on the number of receptors of the cell, N, current methylation, and ligand dissociation constants K_a^{on}/K_a^{off}.

The other factors of the first equation, γ_R and γ_B, are the constant rates at which CheR and CheB-P methylate and demethylate the MCP. The factors $\overline{k}_1 - \overline{k}_6$ are the kinetic rates at which the different reactions that make up the chemotactic pathway occur, and are, along with all other factors, explained in more detail in Table 3.1.

We can use this system of equations to express the underlying mechanics of chemotaxis, and then use the level of CheY-P to determine tumbling-bias. This is a detailed bottom-up approach, where the observed dynamics of the chemical reactions alone are responsible for reacting to the environment. It should also be noted that this ODE-system is a so-called stiff system of equations, which means that it is prone to sudden spikes that might be missed with traditional solvers [8]. Numerical solvers suitable for stiff problems take into account that step sizes have to be small enough to detect such rapid changes. For more details about differential equations and how to solve them, see Appendix C.

2.3 Survival Analysis

Survival analysis is a field of stochastic modelling used to investigate the time it takes for an event of interest to occur [9]. Survival analysis has applications in many fields, such as survival time for a patient after the onset of some disease, or the time until a component breaks in a production line. In this thesis, survival analysis is used, among other things, to investigate the time it takes for a bacterium to tumble after it has started running. There are two features to describe survival data: the survival probability and hazard rate. The survival probability, denoted $S(t)$, is defined as $\mathbb{P}(T > t), 0 < t < \infty$, the probability that something survives beyond a certain time t. The hazard rate, $h(t)$ can be interpreted as the frequency of failure per time unit.

Figure 2.2 shows an example depicting the probability of death for a human. In the graph, we can see that the most critical stages in a human's life are at the start- and endpoint, while there is a relatively low probability of dying in the middle. The hazard rate tells us that the probability of having died grows at a higher rate at the beginning and end of the age distribution, with a constant rate of growth between these two. Note that the hazard rate only has an impact on the overall survival probability at a certain time; the probability of death for a single human is cumulative over her lifespan.

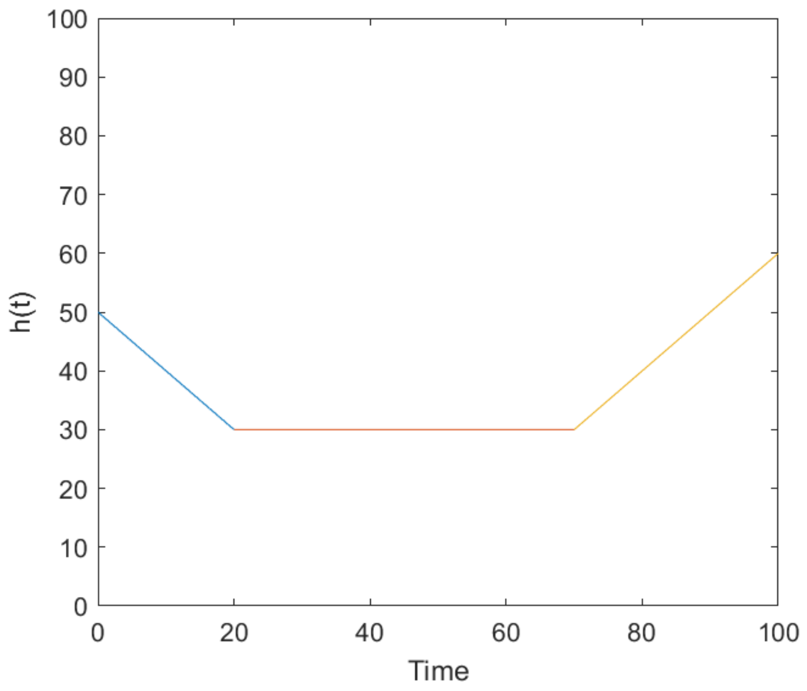


Figure 2.2: The hazard rate function $h(t)$ showing the probability of death for a human at any particular age.

The survival function is often modelled with the ordinary differential equation

$$h(t) = \frac{S'(t)}{1 - S(t)}, \tag{2.4}$$

where $h(t)$ returns the propensity of failure for a subject depending on the age it has reached.

In terms of chemotaxis, survival analysis is used for tumbling, division, and death. The reason survival analysis is useful in this project is because it returns a value that is then used in conjunction with a probability threshold. This threshold is the defining factor that decides if our objective remains in "status quo" or if it should tumble or divide/die depending on what is calculated in each time step. The survival function always grows towards this threshold, but at a rate that depends on the hazard function.

This feature provides a stochastic method for modelling chemotaxis and cell division and -death, events that are stochastic in nature. Finally, the hazard rate for tumbling and division for a bacteria grows higher in areas with greater ligand concentration, and similarly, the growth factor makes is so that even in areas with no concentration the bacteria will eventually tumble/divide given that they are still alive.

2.4 Extracellular Environment

A completely accurate description of the environment in which an actual E. coli bacterium would realistically live is far beyond the scope of this project, if even possible at all. Therefore, a series of simplifications and approximations were necessary.

Our underlying model scenario consists of an environment with a single type of ligand, distributed in a square-shaped petri dish. The medium in which the ligand is distributed does not affect bacterial movement. Even though the distribution of ligand is in part arbitrary, some specifications are necessary when modelling chemotaxis.

First of all, trivially, the distribution must vary spatially, or else it would be pointless. The phenomenon of chemotaxis is dependent on an environment where some areas are higher in ligand than others.

Secondly, when cell division is included, there must be dynamics in the distribution that depend on the population. A constant source of ligand would lead to an exponentially growing population of cells. This project examines different approaches with increasing complexity.

2.4.1 The Static Model

The model introduced by Edgington and Tindall [10], which is the main inspiration for the project, uses a static ligand distribution. The function used to describe the distribution in that model is

$$L(x,y) = l_0 + \exp\left(-\sqrt{\frac{x^2+y^2}{d}}\right) \tag{2.5}$$

where l_0 is an arbitrary minimum ligand concentration and d is a parameter that regulates the slope of the function. This equation describes a single centred source with a rotationally symmetric distribution, where $\sqrt{x^2+y^2}$ is the distance from the source. To account for multiple ligand sources, the equation can be extended to

$$L(x,y) = l_0 N + \sum_{i=1}^{N} \exp\left(-\sqrt{\frac{(x-x_i)^2+(y-y_i)^2}{d}}\right) \tag{2.6}$$

where N is the number of sources and (x_i, y_i) is the position of the i:th source.

This model entails that all bacteria are completely independent of each others' existence, in addition to the function $L(x,y)$ being time independent and thereby represents a constant source of ligand. While being perfectly adequate in the modelling of the phenomenon of chemotaxis motility-wise, it is insufficient when adding the possibility of cell division, for reasons discussed above.

2.4.2 The Dynamic Model

In reality, a spatially varying ligand distribution would even out, or diffuse, with time. By thereto adding consumers, e.g. bacteria eating from the environment, the ligand would ultimately become depleted. To model this more realistic environment is, at least from a mathematical perspective, a slightly more complicated task.

The concept of ligand diffusion can be described with a partial differential equation, or PDE, described in Appendix C.1.1. More specifically, the well-known *diffusion equation* [11].

$$\frac{\partial}{\partial t} L(t,x,y) = D\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) L(t,x,y) \tag{2.7}$$

where $L(t, x, y)$ is the ligand concentration at a position (x, y) at time t. D is the diffusivity constant, which describes the rate of diffusion in the medium. Both an initial condition, or IC, and a boundary condition, or BC, are needed to solve Equation (2.7). While the IC comes directly from Equation (2.5), the BC need to be properly specified. In the case of a solid boundary, like that of a petri dish, without flux into or out of the system, the BC can be formulated as

$$\frac{\partial}{\partial x}L(t,x,y) = \frac{\partial}{\partial y}L(t,x,y) = 0 \quad \text{for } (x,y) \text{ at the boundary} \tag{2.8}$$

which is a *homogeneous Neumann BC.*

To address the aspect of consumption there's a need to modify Equation (2.7) by adding a term that subtracts ligand given the position of a bacterium. This leads to

$$\frac{\partial}{\partial t}L(t,x,y) = D\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)L(t,x,y) - \sum_{i=0}^{n} f_i(t,x,y) \tag{2.9}$$

where n is the total number of bacteria in the system. The function $f_i(t, x, y)$ describes the consumption contribution by the i:th bacterium in respect to its position at time t.

The problem can now be formally described by using equations (2.6), (2.8) and (2.9)

$$\begin{cases} \frac{\partial}{\partial t}L(t,x,y) & = D\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)L(t,x,y) - \sum_{i=0}^{n} f_i(t,x,y) & (x,y)\in\Omega \\ L(0,x,y) & = l_0 N + \sum_{i=1}^{N} \exp\left(-\sqrt{\frac{(x-x_i)^2+(y-y_i)^2}{d}}\right) & (x,y)\in\Omega \\ \frac{\partial}{\partial x}L(t,x,y) & = \frac{\partial}{\partial y}L(t,x,y) = 0 & (x,y)\in\partial\Omega \end{cases} \tag{2.10}$$

where Ω is the region, i.e. the square petri dish, and $\partial\Omega$ denotes the boundary of the region.

2.5 Agent-Based Modelling

Agent-based modelling (ABM) is a modelling technique where a system can be described as a set of individual autonomous decision making units called agents. In an ABM, each agent assesses its own individual situation and performs an action based on a set of rules [12]. ABMs can exhibit complex behaviour patterns even with a simple model and provide valuable information about the dynamics of the system that it emulates and is used in fields such as biology, epidemiology, and finance to name a few.

One feature of ABM is that it can gauge one's comprehension about a complex system by modelling it on a microscopic level. An advantage of this approach is that the complexity of the model can be reduced by looking at individual parts instead of the entire system at once [13]. Being able to model the system through ABM also provides flexibility since each agent can follow a simple set of rules independently. This would not be the case in a complex system since there would be more parameters to take into account and where the selection of each parameter would have to be made with consideration of every part of the system.

Finally, ABM is particularly useful when modelling a biological system. This is because it yields emergent behaviour stemming from the individual modelling approach. One weakness with ABM however is that requires a lot of computational power since calculations need to be performed for each entity in the system, meaning that as the system grows larger so does the toll on the computer's RAM.

2.6 Continuum Description of Populations

Using an ABM approach, the population can only be controlled indirectly, through individual level models. Two parameters were used to investigate whether any population level pattern emerged from the ABM approach. These were the growth rate of the population, r: how fast the population grows in an environment with optimal conditions and the carrying capacity, K, which yields the value at which the population growth plateaus given a sufficient time interval. These parameters were used since they were quite easy to determine and also because they were sufficient to make some population level analysis. When the population had reached an equilibrium state, the carrying capacity was determined by calculating the mean population size. The growth rate r was calculated by simulating the evolution of a population and then fitting it with a logistic curve. A logistic model is a common approach for modelling a population where there is some scarcity of resources impairing its growth. Equation (2.11) is a differential equation describing the population dynamics given the parameters mentioned above,

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right) \tag{2.11}$$

where $P(t)$ is the size of the population at time t. Note that when P is low, the growth rate can be regarded as rP. However, as the population size increases, $\frac{P}{K}$ becomes larger and slows down the growth. Figure 2.3 shows an example of how the evolution of a population would enfold given that there is some scarcity in resources.

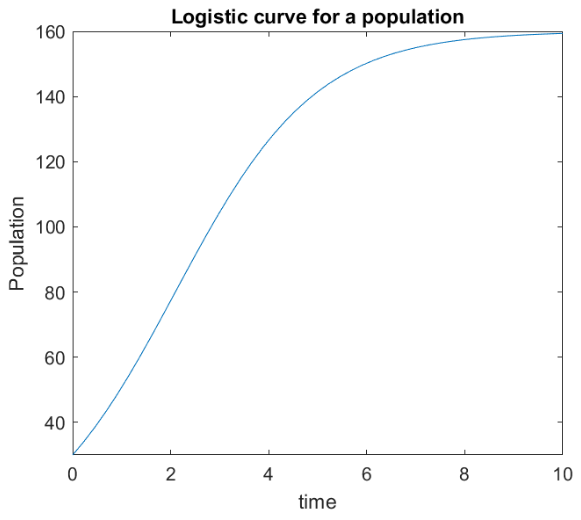


Figure 2.3: A logistic curve showing the evolution for a population.

2.7 Game Engines

To make a graphical simulation immersive, features such as 3D graphics, different textures and animations, and physics are almost necessary. By implementing the simulation like a computer game, incorporating these features is easier than one might think. Games get developed on top of a so-called game engine. These engines control the logic and rules which are needed for the game to run properly. These engines include things such as a graphics engine, physics engine, sound- and animation support, and much more. Game engines can be developed from the ground up or used as a tool to build games or other programs. As these engines can be quite complex to develop, using an already existing engine can save considerable time when developing games. This approach allows one to solely focus on the implementation of game specific details, while the engine handles the underlying aspects such as how the graphics are rendered to the screen.

3
Method

This chapter will describe techniques, based on our theoretical foundation, used to model and simulate chemotaxis, and to develop the application around this simulation. We also go over the two different algorithms used to control the bacteria. We finish the chapter with the description of a study conducted within our application, with the goal of measuring and comparing the different models of the chemotactic pathway.

3.1 ODE-Regulator

The ODE-Regulator is the model of the chemotactic pathway based on Edgington and Tindall's system of ODEs [1], presented in equations (2.1). Around these equations, an algorithm for deciding whether a bacteria should run or tumble was developed, taking as input some concentration of ligand. In conjunction with the system of ODEs, we use survival analysis with a hazard function that grows at a rate dependent on the concentration of CheY-P to trigger tumbling

$$S'(t,x) = h(x)(1 - S(t,x)), \tag{3.1}$$

$$h(x) = 0.02 + 0.5x, \tag{3.2}$$

where we use CheY-P for x. This is added to the system of ODEs from [1] shown in (2.1).

3.1.1 Parameters

The parameter values used in the system of ODEs in (2.1) were taken directly from [1], with the exception of $N = 30$.

$\overline{k}_1$	48.571	Autophosphorylation of CheA
$\overline{k}_2$	1385.714	Phosphorylation of CheY by CheA
$\overline{k}_3$	6	Phosphorylation of CheB by CheA
$\overline{k}_4$	8.686	CheY dephosphorylation by CheZ
$\overline{k}_5$	1	Autodephosphorylation of CheB-P
$\overline{k}_6$	0.121	Autodephosphorylation of CheY-P
α_1	0.814	Ratio of total CheA to total CheY
α_2	28.214	Ratio of total CheA to total CheB
γ_R	$8.57 \cdot 10^{-3}$	Methylation of MCP by CheR
γ_B	0.352	Demethylation of MCP by CheB-P
N	30	Number of receptors

Table 3.1: Parameter values used in ODE system taken from Edgington and Tindall [1]. These describe chemical kinetics and other properties of the bacteria and their chemical reactions.

3.1.2 ODE-Regulator Algorithm

We add the variable $U \in [0, 1]$ to serve as a threshold for tumbling. When our accumulating survival variable S exceeds U, a tumble occurs, and U is resampled uniformly.

Algorithm 1: ODE-Regulator Algorithm

```
Input: $L$, ligand concentration
Output: State *run* or state *tumble*
Use $L$ to solve for new $\phi$ according to (2.2)
Solve ODEs from (2.1) and (3.1) with current concentrations
Update concentrations of signalling proteins and $S$
if $S > U$ then
    $S := 0$
    $U \sim \text{Unif}[0, 1]$
    return tumble
else
    return run
```

3.2 Delta-Regulator

The Delta-Regulator is a simplified model of the chemotactic pathway, consisting of just a function that triggers tumbling based on the difference in ligand concentration, presented in Equation (3.3). This function grows with a step function, (3.4), depending on whether the ligand concentration increased or decreased from the previous position. This regulator will tumble more frequently when moving against the ligand gradient, which is the directly desired outcome of chemotaxis. This way, chemotactic behaviour is guaranteed, but without a description of its underlying mechanics.

$$S(t+1,x) = S(t,x) + h(x)(1 - S(t,x)), \tag{3.3}$$

$$h(x) = \begin{cases} 0.5 & \text{if } x < 0 \\ 0.09 & \text{otherwise} \end{cases} \tag{3.4}$$

Around these functions, an algorithm taking as input some ligand concentration and producing a state of either running or tumbling was developed.

3.2.1 Delta-Regulator Algorithm

Just as with the ODE-Regulator Algorithm in Algorithm 1, we use $U \in [0,1]$ as a threshold for tumbling; when S exceeds U, the bacteria tumbles.

Algorithm 2: Delta-Regulator Algorithm

```
Input: $L_{new}$, new ligand concentration
Output: State run or state tumble
Calculate $\Delta L = L_{new} - L_{old}$
Use $\Delta L$ to calculate $S$ according to (3.3)
$L_{old} := L_{new}$
if $S > U$ then
    $S := 0$
    $U \sim \text{Unif}[0,1]$
    return tumble
else
    return run
```

3.3 Bacterial Model

The bacterium can, at any point of the simulation, be in one of two different states: tumbling or running. Which of these states the bacterium is in is governed by the previously described regulators. The bacterium changes its directional angle while in the tumbling state and its positions while in the run state.

3.3.1 Running

The movement of a bacterium is governed by

$$\begin{aligned} \Delta x &= v\Delta t\cos(\theta) \\ \Delta y &= v\Delta t\sin(\theta) \end{aligned} \tag{3.5}$$

where θ is the directional angle of the bacterium, v is the velocity and Δt is the time step of the simulation. This means that the bacterium will continue moving with a constant velocity towards the same direction until it tumbles.

3.3.2 Tumbling

The calculation of the tumble angles is performed in a stochastic manner by sampling a value from the uniform distribution of values within the intervals [-98, -18] and [18, 98]. The sampled value is then added to the current angle. This choice of intervals is motivated by [10] where the authors chose to use the same intervals when calculating their turning angles.

3.3.3 Division and Death

Both the division and the death of the bacteria is modelled using survival analysis in a similar manner to the way that the tumbling is modelled in the Delta-Regulator. Their h-functions are shown in equations (3.6) and (3.7). The main difference to the previously described implementations of survival analysis that when the death hazard function reaches 1, the cell dies, terminating the process.

$$h_{division}(L) = 0.002 + 0.002L \tag{3.6}$$

$$h_{death}(L) = 0.05 - 0.001L \tag{3.7}$$

When a bacterium divides, the newly created bacterium inherits the position and internal state history of its "parent". Similarly, when a bacterium dies, all of its future positions and internal states are set to the current position and state.

3.4 Extracellular Environment

With the increasing complexity and therefore increasing computational load, the evolution from the initial static model to a dynamic model was simplified, thus leading to the implementation of a quasi-dynamic model. While the dynamic model was never implemented in the application, it was simulated in Matlab. It therefore serves as a proof-of-concept, highlighting the obvious next step in the further development of the product.

3.4.1 The Static Model

The desired possibility of regulating the amplitude of the ligand concentration, for optimisation purposes, issues the need of adding a scaling factor to Equation (2.6). Along with a squared distance term, the distribution becomes

$$L(x, y) = l_0 N + \Lambda \sum_{i=1}^{N} \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{d}\right), \tag{3.8}$$

where Λ is a parameter that regulates the maximum ligand concentration and (x_i, y_i) is the coordinates of the i:th ligand source.

These modifications result in a slightly different graph compared to Equation (2.5). These differences can be seen in Figure 3.1. The differences are mainly the effect of the squared distance term which leads to a normal distribution of the ligand concentration.

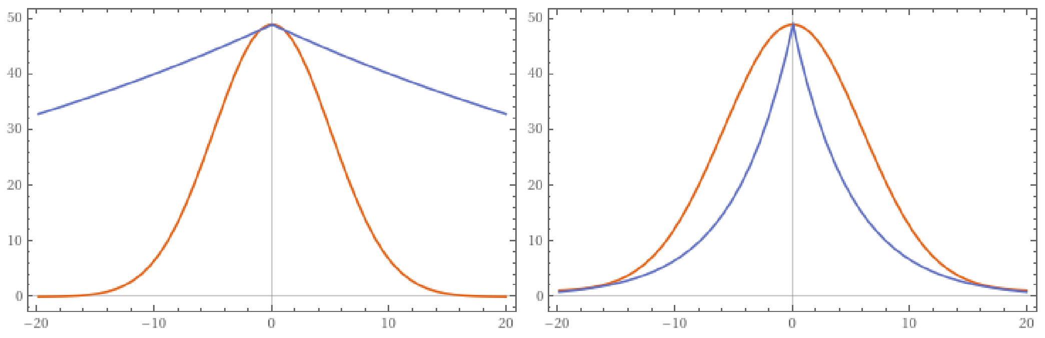


Figure 3.1: Comparison between (2.5) (blue) and (3.8) (orange) with a single concentric source. The figure on the left uses the same values for the constants for both of the functions, while the one on the right uses different values for the constants.

3.4.2 The Quasi-dynamic Model

The addition of death and division for the bacteria necessitates a dynamic environment to prevent the bacterial population from growing exponentially. However, this poses a problem as a fully dynamic model, for example the one described in Section 2.4.2, requires a large number of expensive calculations in each of the time steps. To avoid these expensive calculations a simplified model of a dynamic environment was chosen. The model, demonstrated in Equation (3.9), scales the ligand concentration in a given position based on the number of bacteria within a radius of 1 p.d.u. A *procedure defined unit*, or p.d.u., is an arbitrary unit defined by the unit of length in the Unity game engine. This method emulates the bacteria competing to consume the ligand that is available around them since it decreases the amount of ligand that is available for a given bacterium as the total population increases.

$$\tilde{L}_i(t,x,y) = \frac{L(x_i(t), y_i(t))}{n_i(t)} \tag{3.9}$$

where $n_i(t)$ is the number of bacteria within a given radius of the i:th bacterium at time t. $x_i(t)$ and $y_i(t)$ represents the respective coordinates of that bacterium at time t.

3.4.3 The Dynamic Model

The dynamic problem (2.10) will now be specified. The choice of function $f(t,x,y)$ was inspired by topical educational literature from J.D. Murray [14]

$$f_i(t,x,y) = a_1\text{exp}(-a_2[(x-x_i(t))^2 + (y-y_i(t))^2]) \tag{3.10}$$

where a_1 and a_2 are parameters which regulates the intensity and area of consumption respectively. $(x_i(t), y_i(t))$ represents the position of the i:th bacterium at time t. For reasons explained in Section 3.4.1 the IC now comes from Equation (3.8).

These specifications constitutes the problem

$$\begin{cases} \frac{\partial}{\partial t}L(t,x,y) & = D\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)L(t,x,y)-\sum_{i=0}^{n} f_i(t,x,y) \quad (x,y)\in\Omega \\ L(0,x,y) & = l_0 N + \Lambda\sum_{i=1}^{N}\exp\left(-\frac{(x-x_i)^2+(y-y_i)^2}{d}\right) \quad (x,y)\in\Omega \\ \frac{\partial}{\partial x}L(t,x,y) & = \frac{\partial}{\partial y}L(t,x,y)=0 \quad (x,y)\in\partial\Omega \end{cases} \tag{3.11}$$

The region Ω is a square shaped petri dish of size 28x28 p.d.u.

The problem (3.11) was solved using the ADI-method, described in C.1.2. By applying (C.10) on (2.7), after some rearranging, the result for the first half time step becomes

$$-\alpha L_{i,j-1}^{n+1/2}+\beta L_{i,j}^{n+1/2}-\alpha L_{i,j+1}^{n+1/2}=\alpha L_{i-1,j}^{n}+\gamma L_{i,j}^{n}+\alpha L_{i+1,j}^{n} \tag{3.12}$$

where $\alpha = D\Delta t/(\Delta x)^2$, $\beta = 2(1+\alpha)$, $\gamma = 2(1-\alpha)$, n indexes the time step and (i,j) indexes the lattice points in the discretised region Ω, such that $L_{i,j}^n = L(n, x_i, y_j)$ where (x_i, y_j) are the coordinates of the node indexed (i,j). Similarly the second half time step follows from applying (C.11) to (2.7) resulting in

$$-\alpha L_{i-1,j}^{n+1}+\beta L_{i,j}^{n+1}-\alpha L_{i+1,j}^{n+1}=\alpha L_{i,j-1}^{n+1/2}+\gamma L_{i,j}^{n+1/2}+\alpha L_{i,j+1}^{n+1/2} \tag{3.13}$$

Note that in order to account for the Neumann BC it is implied that $L_{i-1,j} = L_{i+1,j}$ for $i \in \partial\Omega$, or equivalently $L_{i,j-1} = L_{i,j+1}$ for $j \in \partial\Omega$, as discussed in appendix C.1.2.

The bacterial consumption was implemented with the use of a matrix defined as

$$F_{ij} = \begin{cases} \hat{F} & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases} \tag{3.14}$$

where $\hat{F}$ is an operator inspired by Equation (3.10) such that

$$\hat{F}L_{i,j}^n = \sum_k f_k(n, x_i, y_j)$$

where k indexes bacteria and (x_i, y_j) is the coordinates related to the lattice point indexed (i,j).

The region Ω was discretised as a 60x60-grid. The result is one equation for each lattice point, that is $61 \cdot 61 = 3721$ equations per half time step, which can be expressed with matrix equations $Ax_1^{(n+1/2)} = (B-F)x_1^{(n)}$ and $Ax_2^{(n+1)} = (B-F)x_2^{(n+1/2)}$ respectively, where

$$x_1 = \begin{bmatrix} L_{1,1} & L_{1,2} & ... & L_{1,61} & L_{2,1} & L_{2,2} & ... & L_{61,60} & L_{61,61}\end{bmatrix}^T$$

for the the first half-time-step and

$$x_2 = \begin{bmatrix} L_{1,1} & L_{2,1} & ... & L_{61,1} & L_{1,2} & L_{2,2} & ... & L_{60,61} & L_{61,61}\end{bmatrix}^T$$

for the second half time step.

The matrices A, B and F stays the same though.

A has the following form

$$A = \begin{bmatrix} \beta & -2\alpha & 0 & & \cdot & & \cdot & & \cdot & & 0 \\ -\alpha & \beta & -\alpha & & & & & & & & \\ 0 & -\alpha & \beta & \cdot & & & & & & & \cdot \\ & & \cdot & \cdot & \cdot & & & & & & \\ \cdot & & & \cdot & \cdot & -\alpha & & & & & \cdot \\ & & & & -2\alpha & \beta & 0 & & & & \\ \cdot & & & & & 0 & \beta & -2\alpha & & & \cdot \\ & & & & & & -\alpha & \beta & \cdot & & \\ \cdot & & & & & & & \cdot & \cdot & \cdot & 0 \\ & & & & & & & & \cdot & \cdot & -\alpha \\ 0 & & \cdot & & \cdot & & \cdot & & 0 & -2\alpha & \beta \end{bmatrix} \tag{3.15}$$

Since A is tridiagonal and $\beta = 2(1+\alpha) > 2\alpha$ the problem was solved using the Thomas algorithm, as explained in Appendix C.1.2.

The matrix B has the form

$$B = \begin{bmatrix} \gamma & 0 & .. & 0 & 2\alpha & 0 & & \cdot & & \cdot & & 0 \\ 0 & \gamma & & & & 2\alpha & & & & & & \\ \vdots & & \cdot & & & & \cdot & & & & & \cdot \\ 0 & & & \cdot & & & & \cdot & & & & \\ \alpha & & & & \gamma & & & & \alpha & & & \cdot \\ 0 & \cdot & & & & \cdot & & & & \cdot & & \\ & & \cdot & & & & \cdot & & & & \cdot & 0 \\ \cdot & & & \alpha & & & & \gamma & & & & \alpha \\ & & & & 2\alpha & & & & \gamma & & & 0 \\ \cdot & & & & & \cdot & & & & \cdot & & \vdots \\ & & & & & & \cdot & & & & \cdot & 0 \\ 0 & & \cdot & & \cdot & & 0 & 2\alpha & 0 & .. & 0 & \gamma \end{bmatrix} \tag{3.16}$$

The mechanics behind the bacterial movement uses the same ODE system (2.1) as the other two models. The system was solved using the Matlab solver ode15s, which is specifically designed to solve stiff ODEs. The discrete ligand distribution is addressed by first determining in which grid square a bacterium is positioned and then calculating the average ligand concentration based on the value in the four corners, weighted by the distance to each corner. This can be formulated as

$$L(x,y) = \frac{y_1(x_2 L_{i-1,j-1} + x_1 L_{i-1,j}) + y_2(x_2 L_{i,j-1} + x_1 L_{i,j})}{(\Delta x)^2} \tag{3.17}$$

where x_1, x_2, y_1, and y_2 are defined as in Figure 3.2.

3.5 Model Evaluation

We concluded this project by conducting a study meant to evaluate the performance of the ODE-Regulator. Indicators of performance come both from how well the bacteria navigate towards sources of ligand, and from how well the population of

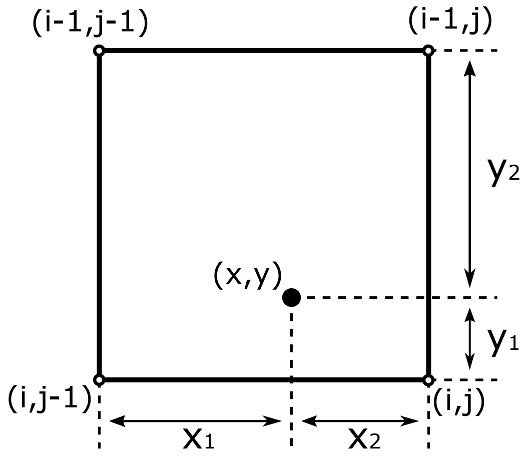


Figure 3.2: The bacterium is positioned at (x, y) located in the grid square of size $(\Delta x)^2$ defined by the corners $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$ and (i, j).

bacteria manages to grow with a given regulator. With this in mind, we split the study into two parts, one for each purpose. To compare just the motility of the bacteria, we switch off competition for ligand between individuals, as well as death and division. This means that the environment is static and all bacteria are independent of each other. For population growth, we keep these dynamics switched on. This split is appropriate as the dynamics of the environment and the population interferes with the individual bacterium's strive towards the ligand source.

For each experiment, we ran a batch of five simulations per regulator, for a total of 20 simulations. All simulations were initialised with parameters $N = 30$, $d = 100$, $l_0 = 0$. The statistics from each batch was aggregated to compensate for the possibility of randomised starting positions of bacteria affecting the result. These statistics show the progression over time of average distance to the ligand source and population size. The average distance is used to measure effectiveness in motility, while population size is used to measure survivability.

The evaluation of the fully dynamic model described in Section 3.4.3 is of an ocular nature. By plotting the bacterial migration responding to the temporal variations in the ligand distribution, the phenomenon of chemotaxis should become apparent.

4
Implementation

This chapter will focus on the implementation of the extracellular environment; the algorithms for running and tumbling, as well as cell division and -death. A description of the analytical tool will also be given. Lastly, we will have software-related decisions around tooling and development.

4.1 Bacterial Movement

The movement of each of the bacterium is governed by Algorithm 3 (which is also demonstrated in Figure A.1), which uses the previously described method to determine the state of the bacterium. If the bacteria is in the tumbling state, a new angle is calculated according to the method outline in Section 3.3.2 and the algorithm moves on to the position calculation stage. If the bacteria is not in the tumbling state, it directly moves to the position calculation stage, where the delta values for the x- and y-coordinates are calculated according to Equation (3.5). If adding these values to the current x- and y-coordinates would result in a position outside the simulation, a new angle is calculated and the process is repeated until the new position is inside the simulation area. When a valid position has been found the location for the bacteria is updated and the algorithm terminates.

Algorithm 3: Calculate next position

```
Input: (x,y), the current position for the bacterium
Output: The next position for the bacterium
L := ligand concentration for (x,y)
if Tumble given L then
    Calculate a new angle $\theta$
$\Delta x := v\Delta t\cos(\theta)$
$\Delta y := v\Delta t\sin(\theta)$
while $x+\delta x$ and $y+\delta y$ outside simulation area do
    Calculate a new angle $\theta$
    $\Delta x := v\Delta t\cos(\theta)$
    $\Delta y := v\Delta t\sin(\theta)$
x := x + $\Delta x$
y := y + $\Delta y$
return((x,y))
```

Algorithm 4 (see Figure A.2) is a slightly more complex version of Algorithm 3 that is used when running the simulation in the pre-calculation mode. In this version, Algorithm 3 is repeated N times, where N is the number of iterations that the simulation will run. The internal state and position for the bacterium are saved in each of the iterations and this data is later used when visualising the results of the simulation. Each of the tumbles and run segments take 1 iteration of time each, and both, therefore, result in a saved position and state.

Algorithm 4: Forward simulate movement

```
Input: n, the number of time steps that the simulation should run
i := 1
$position_0$ = the starting position of the bacteria
while $i < n$ do
    $position_i := position_{i-1}$
    L := ligand concentration for $position_i$
    if Tumble given L then
        Calculate a new angle $\theta$
        Save internal state
        i = i + 1
        if $i \geq n$ then
            return
        $position_i := position_{i-1}$
    $\Delta x := v\Delta t\cos(\theta)$
    $\Delta y := v\Delta t\sin(\theta)$
    while position.x+$\Delta x$ and position.y+$\Delta y$ outside simulation area do
        Calculate a new angle $\theta$
        $\Delta x := v\Delta t\cos(\theta)$
        $\Delta y := v\Delta t\sin(\theta)$
    position.x := x + $\Delta x$
    position.y := y + $\Delta y$
    Save internal state
    i = i + 1
return
```

4.2 Application

The code base of the application can be divided into two distinct parts: the model, which manages the data and performs the calculations that are required for the simulation, and the view layer, which manages the visualisation of the simulation. The view depends on the model since it can not visualise the simulation without it, while the model does not depend on the view or any Unity specific code. This makes it possible to use the existing model in another C#-based context, greatly increasing the value of the simulation. It is for this reason that this section will primarily focus on the model since it the most important part of the program.

4.2.1 Cell Object

The *Cell* object acts as the central representation of the E-coli bacteria in the model, the structure of which can be seen in Figure A.3. *Cell* delegates all of its functionality to an object of the type *IInternals* which manages the internal processes of the bacterium.

4.2.2 Internals

The *IInternals* object manages the internal processes of the *Cell* object, mainly by performing the calculations required to determine the next position that the bacteria should move to. There exists three concrete implementations of the *IInternals* interface (see Figure 4.1): *SmartInternals*, *Internals*, and *ForwardInternals*.

Internals is the most basic variant of the internals objects, it uses Algorithm 3 to calculate the next position that the bacteria should visit. This type of Internal is used when the program is run in the procedural mode, where it executes until the user terminates it.

ForwardInternals can be seen as an extension of *Internals* with the main difference being the fact that *ForwardInternals* uses Algorithm 4 instead of Algorithm 3. In other words, it calculates and stores all positions that the bacteria will visit before the simulation begins. *ForwardInternals* is also the only type of internals that implements cell division and -death.

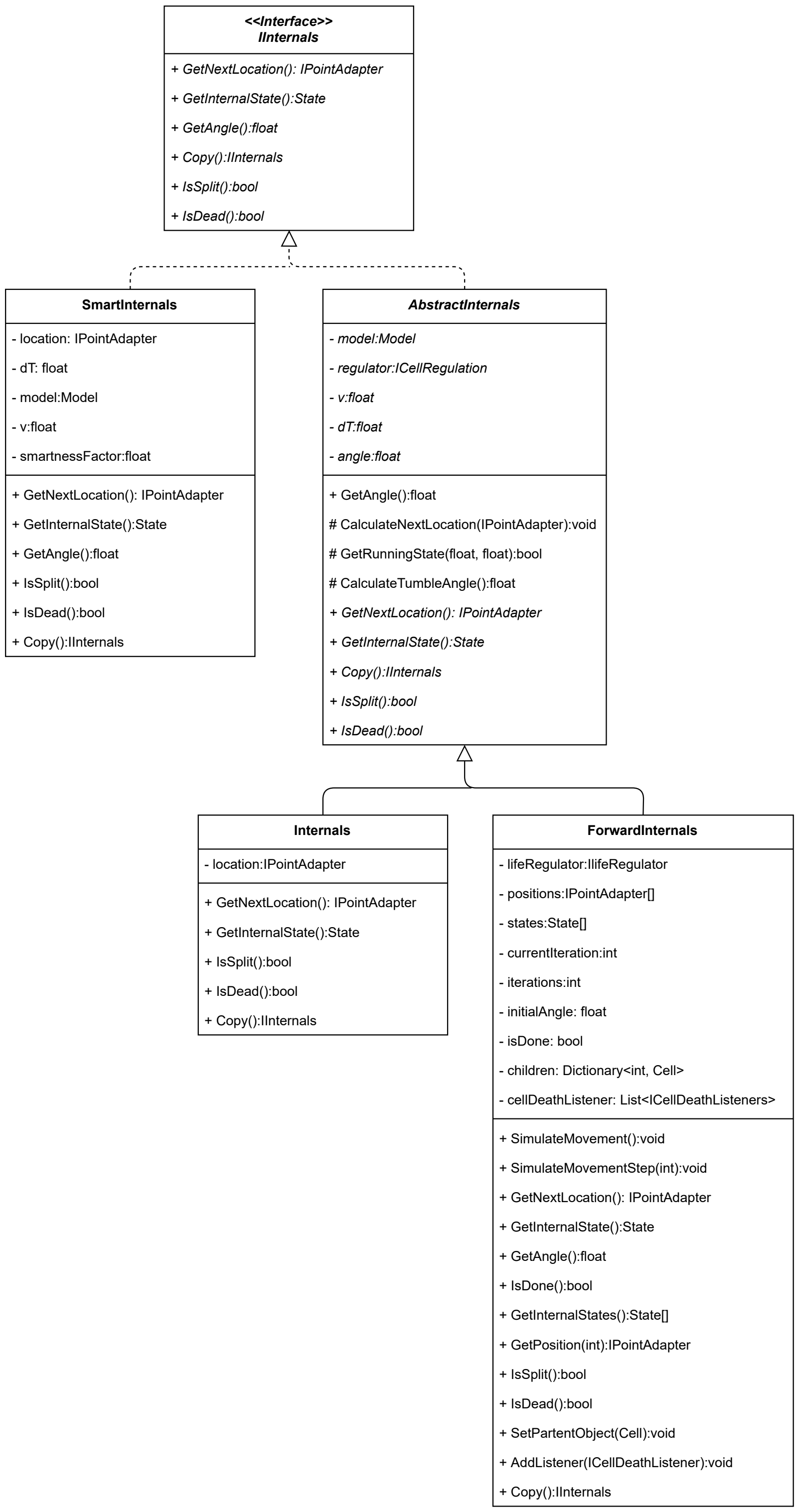
Lastly, *SmartInternals* uses a completely different movement algorithm to the previously described Internals. This object uses the ligand gradient to calculate the optimal movement angle. The angle is then disturbed using a value sampled from a normal distribution. Which results in the object performing a random walk towards the ligand source.

4.2.3 Regulators

The *IRegulation* interface is used to determine whether the bacterium should tumble or run at a given point in the simulation. Each of the different models for the internals of the bacterium, ODE- and Delta-Regulator, have a concrete implementation of this interface which is then used in *Internals* or *ForwardsInternals* when calculating the movement of the bacteria.

4.2.4 Forward-Simulation

The expensive calculations required by the ODE-Regulator made it impractical to run the calculations in real time, since they had a considerable impact on the frame rate of the simulation. To amend this problem, we decided to switch over to a forward-simulation approach where the entire simulation is calculated before being played out to the user. This was initially quite simple since we only needed to store all of the positions that the bacteria should visit as well as their internal states. However, the addition of cell division and -death complicated matters significantly since we also had to store which bacteria should be alive in any given time step. This was handled by storing the bacteria in a matrix where the object on the i:th row



<<Interface>>
IInternals
+ GetNextLocation(): IPointAdapter
+ GetInternalState():State
+ GetAngle():float
+ Copy():IInternals
+ IsSplit():bool
+ IsDead():bool
SmartInternals
- location: IPointAdapter
- dT: float
- model:Model
- v:float
- smartnessFactor:float
+ GetNextLocation(): IPointAdapter
+ GetInternalState():State
+ GetAngle():float
+ IsSplit():bool
+ IsDead():bool
+ Copy():IInternals
AbstractInternals
- model:Model
- regulator:ICellRegulation
- v:float
- dT:float
- angle:float
+ GetAngle():float
# CalculateNextLocation(IPointAdapter):void
# GetRunningState(float, float):bool
# CalculateTumbleAngle():float
+ GetNextLocation(): IPointAdapter
+ GetInternalState():State
+ Copy():IInternals
+ IsSplit():bool
+ IsDead():bool
Internals
- location:IPointAdapter
+ GetNextLocation(): IPointAdapter
+ GetInternalState():State
+ IsSplit():bool
+ IsDead():bool
+ Copy():IInternals
ForwardInternals
- lifeRegulator:IlifeRegulator
- positions:IPointAdapter[]
- states:State[]
- currentIteration:int
- iterations:int
- initialAngle: float
- isDone: bool
- children: Dictionary<int, Cell>
- cellDeathListener: List<ICellDeathListeners>
+ SimulateMovement():void
+ SimulateMovementStep(int):void
+ GetNextLocation(): IPointAdapter
+ GetInternalState():State
+ GetAngle():float
+ IsDone():bool
+ GetInternalStates():State[]
+ GetPosition(int):IPointAdapter
+ IsSplit():bool
+ IsDead():bool
+ SetPartentObject(Cell):void
+ AddListener(ICellDeathListener):void
+ Copy():IInternals

Figure 4.1: Class diagram showing the relationship between the different types of internals.

and j:th column is the j:th bacteria that is alive in the i:th time step. The forward-simulation simulates all of the bacteria one time step at a time to adequately capture any interactions between bacteria as well as to handle cell division and -death in as good of a manner as possible.

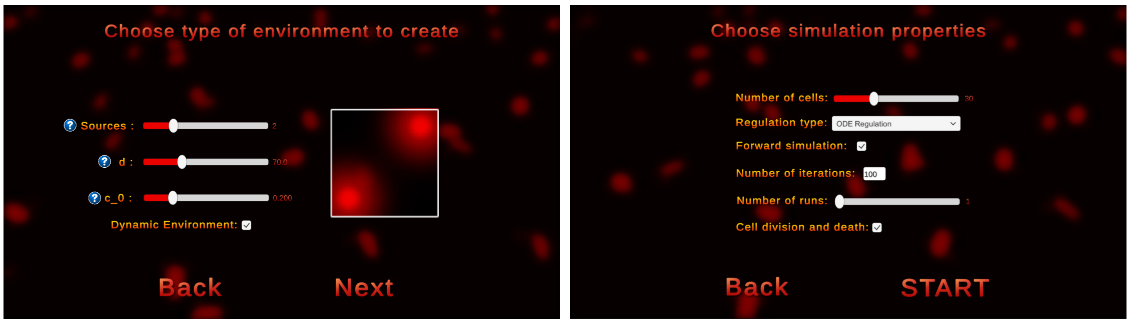
4.2.5 Unity GameObjects

There are two main GameObjects that are handled by scripts. These are the cell itself and the heat map. The cell is constructed out of several GameObjects. These are the body of the cell which is a basic cylinder as well as four custom modelled flagella. At the start of the simulation, these cell GameObjects are instantiated with a random position in the environment.

The heat map is made out of a 120x120 grid that spans across the whole environment. Each square in the grid is given a value based on the concentration value the specific square is in. Each square is then given a colour with varying intensity, the higher the concentration value, the higher the intensity of the colour.

4.2.6 GUI

The Graphical User Interface (GUI) is designed with simplicity in mind, only displaying necessary information. Before starting a new simulation, the user has to go through an initial simulation wizard shown in Figure 4.2. Here, the user can set various parameters with the help of sliders and input fields. The simulation can be started when these initial conditions have been set. When a bacterium is selected during the simulation, the UI view shown in Figure 4.3 is displayed. It displays detailed information about the selected bacterium, such as the various chemical concentration values and more.


Choose type of environment to create

Sources : 2

d : 70.0

c_0 : 0.200

Dynamic Environment: ✓

Choose simulation properties

Number of cells: 30

Regulation type: ODE Regulation

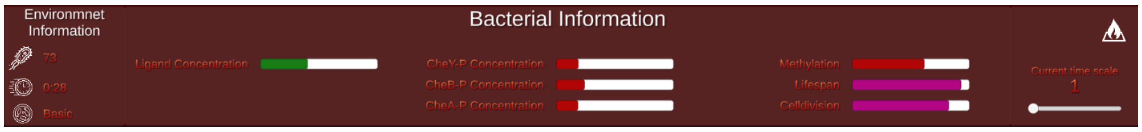
Forward simulation: ✓

Number of iterations: 100

Number of runs: 1

Cell division and death: ✓

Figure 4.2: The two setup screens of the simulation wizard. Here the user can set desired parameters which are taken into account when simulating.


Environmnet Information

73

0:28

Basic

Bacterial Information

Ligand Concentration

CheY-P Concentration

CheB-P Concentration

CheA-P Concentration

Methylation

Lifespan

Celldivision

Current time scale 1

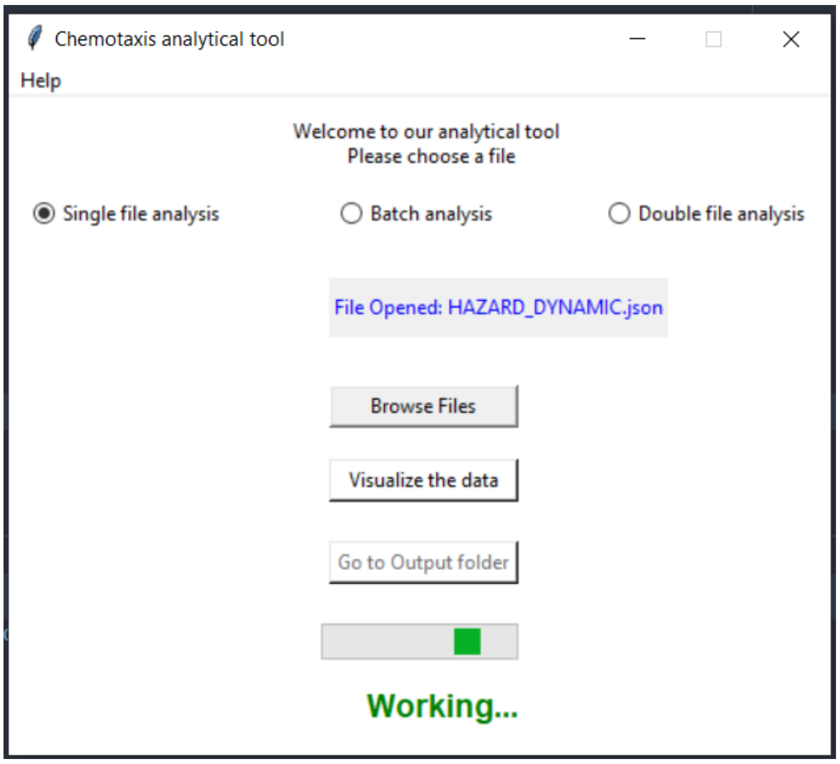
Figure 4.3: User interface shown at the bottom of the screen when the simulation is running and a cell is selected.

4.3 Analytical Tool

Early in the process, we identified the need of analysing data generated from the application in order to evaluate and test the performance and accuracy of our models. Our initial thought was to perform the data analysis using the Unity engine. However, as the complexity of our application kept on growing, and the Unity engine having little to no support for data plotting we decided to perform the data analysis using a separate tool. The choice of what programming language the analytical tool is going to be developed with landed on Python. Python, with its built-in and open source libraries, provides extensive and immense support for data parsing, analysing and plotting. The exported data from the Unity application is fed into the analytical tool to be parsed. Using open-source libraries like Numpy [15], Scipy [16] and matplotlib [17], we were able to analyse, illustrate, and visualise the parsed data. The analytical tool supports different types of analysis, more specifically:

- Single file analysis: Generate plots for a single file,
- Double file analysis: Generate plots comparing two files, and
- Batch analysis: Generate plots for a batch with multiple files along with average plots for the whole batch.

To make the analytical tool more user friendly and easier to use, we developed a fairly simple and responsive GUI using Python built-in library Tkinter. The GUI allows the user to specify what type of analysis to perform and browse for files or folders with proper data (see Figure 4.4).



Welcome to our analytical tool
Please choose a file

File Opened: HAZARD_DYNAMIC.json

Working...

Figure 4.4: Analytical tool GUI showing the different analysis options represented using radio buttons. Multiple buttons with relevant functionality are included, as well as progress bar and status label to show the progress of the simulation.

4.4 Tools

This section will cover the tools and programming languages used during development. The structure of the workflow of the group will also be explained.

4.4.1 Game Engine and 3D Modelling

The game engine Unity [18] was used as it allowed the development to start off quickly without the need of having to create an engine from scratch. It was chosen as it is well-documented and there are many useful tutorials available. Furthermore, Unity has a fairly gentle learning curve. This fits the project well, as no group member had any previous experience with game engines, making Unity the obvious choice. The modelling software Blender [19] was used in order to create more complex 3D models and animations like the flagella of the cell.

4.4.2 Programming Languages

C# was the main programming language used, as Unity offers a primary scripting API in C#. The programming language Python was used to create the external data analytical tool. Initially, Matlab was used to develop a proof-of-concept of the ODE-Regulator, before converting it to C#. The reason is simply that we have more experience using Matlab for mathematical modelling, and it would therefore be too time consuming to get us acquainted with another language. The dynamic model described in 2.4.2 was implemented solely in Matlab.

4.4.3 Workflow

Three main working areas were identified at the start of the project: Intracellular, Extracellular and Graphics. Since group members came from different fields of expertise, it was decided to equally divide the members among these areas in order to maximise work efficiency. As the project progressed, problems arose which did not fit into any of the previous working areas. In these cases, tasks were allocated to the most appropriate members with flexibility.

Due to the project being performed in a pandemic, communication between the group members was of utmost importance. Weekly meetings with the supervisor and group members were performed over Zoom. These meetings would generally include discussions about the current state of the project as well as future development possibilities. Furthermore, a detailed weekly journal was written, summarising what had been accomplished and discussed during the week. Discord was also used, as it allowed for the creation of communication channels in which written discussions could be easily organised and saved. A GitHub repository was created, which allowed for easy version control as well as the use of branches. Two primary branches were used, the main branch which always contained a stable version of the application and the development branch in which all new development was performed.

5 Results

This chapter will focus on the data from our study evaluating the fitness of the ODE-Regulator as compared to the Delta-Regulator. We also go over what we have achieved in the implementation of the application we have developed around the simulation. Using the analytical tool to analyse the experiments described in 3.5 we were able to obtain several plots and figures which were used to compare the performance and fitness of the different model types. Amidst the various plots obtained from the analytical tool, we will mainly focus on population level plots, namely average distance from the ligand source and population changes.

As the phenomenon of chemotaxis describes how the bacteria population migrates towards the ligand source, we can utilise an average distance from ligand source plot to study the performance of chemotaxis. The desired outcome is a shorter distance to the ligand source as the simulation progresses. This should carry on until the simulation reaches a steady state where the distance for the bacteria population fluctuates at a specific interval, as the population moves closer to the ligand source.

The population change plot can be used to study the evolution of the bacteria population for the quasi-dynamic models. Given different starting parameters, such as regulator type and minimum ligand concentration and -distribution steepness, it is intriguing to analyse the evolution of the bacteria population. Specifically, the growth rate and the carrying capacity for a simulation with a specific parameter configuration are of interest. That being said, the amount of time it takes the bacteria population to reach its equilibrium state, and what carrying capacity it can possibly have, is completely governed by the start parameters of the simulation. Therefore, we can compare the bacteria population evolution for the different types of regulators we have.

Along with the aforementioned plots, screenshots from the Unity simulation will be shown as well. These screenshots show the initial and final positions of all bacteria in the population. This is a simple yet effective way to observe the outcome of the migration process and population size evolution.

5.1 Results from static models

In this section we will show data, plots and graphics from running the different regulators in a simulation with neither a quasi-dynamic environment nor any population changes. The regulators will be evaluated based on the bacteria's average distance from the ligand source. The differences between the regulators performance will also briefly be discussed.

5.1.1 ODE-Regulator

Figure 5.1 shows how the bacteria have positioned themselves more highly concentrated in the centre of the simulated environment, where ligand concentration is higher, at the end of the simulation.

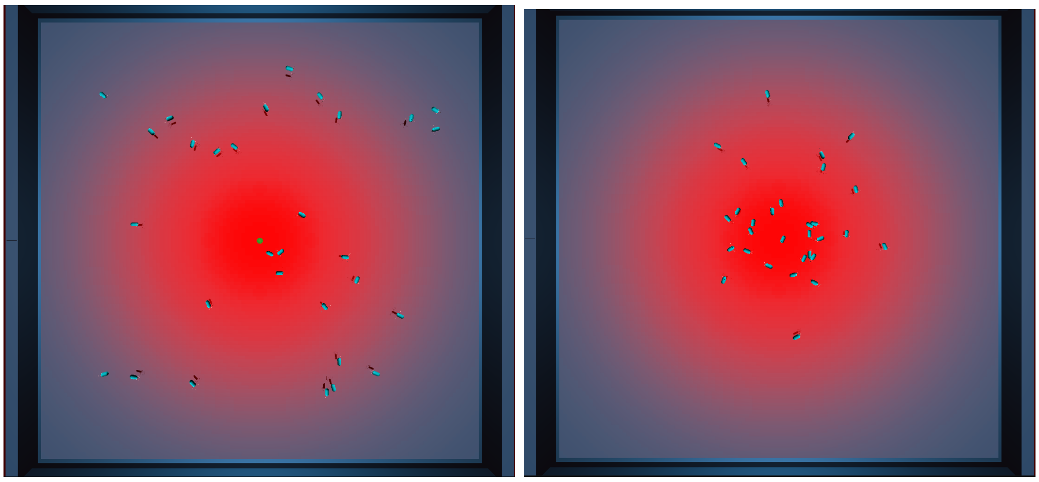


Figure 5.1: Initial (left) and final (right) states of the static simulation with the ODE-Regulator.

In Figure 5.2, we see the average distance to the ligand source for five separate runs as well as a fitted mean curve. In the figure, we can see that in all simulations the average distance to the ligand source reduces as time goes on and this aligns with Figure 5.1 where we saw the final positions for the cells in the population.

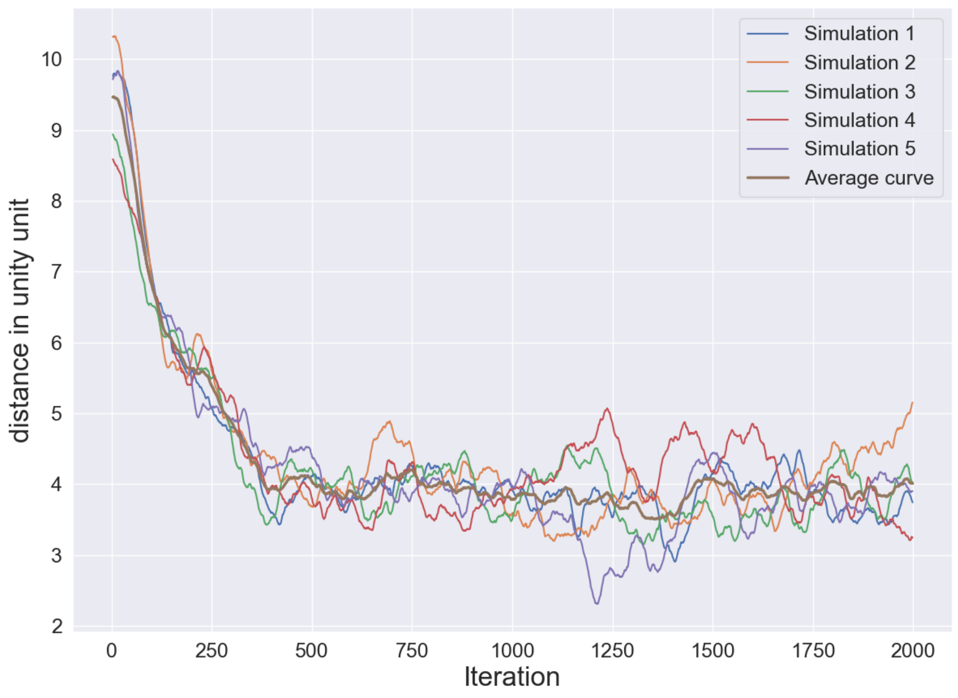


Figure 5.2: Average distance to ligand source for five separate runs and a fitted mean curve with the ODE-Regulator.

5.1.2 Delta-Regulator

Figure 5.3 shows that the population of bacteria are more highly concentrated in the centre of the simulated environment at the end of the simulation. Most notably, fewer bacteria are positioned at the outskirts of the field.

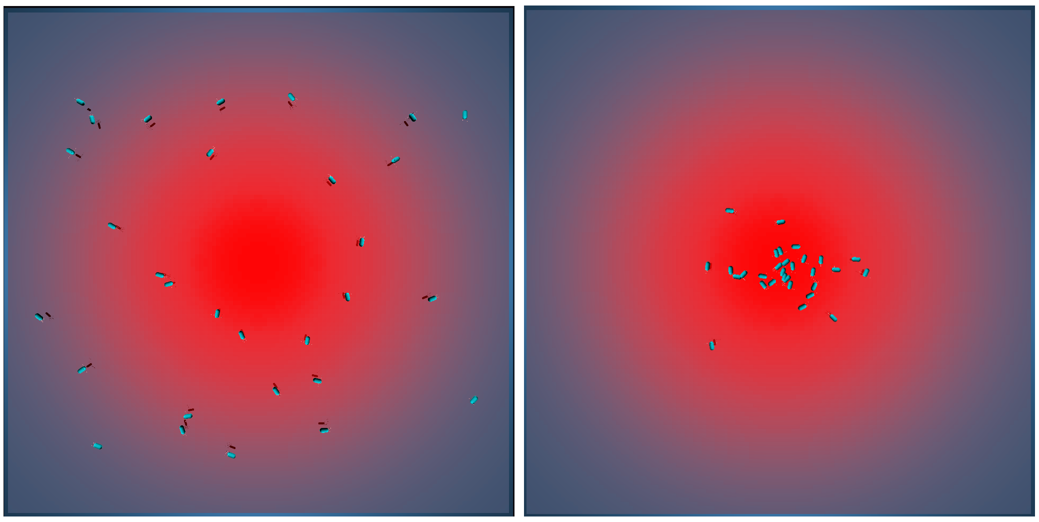


Figure 5.3: Initial (left) and final (right) states of the static simulation with the Delta-Regulator

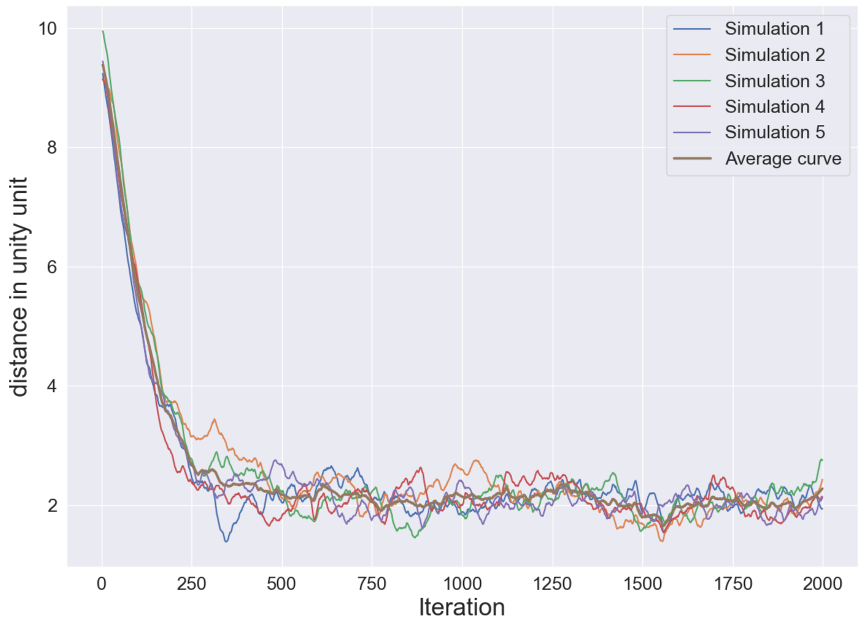


Figure 5.4: Average distance to ligand source for five separate runs and with a fitted mean curve with the Delta-Regulator.

In Figure 5.4, we again see a downward trend for all the simulations, meaning they are moving towards the ligand source. We also notice that there appears to be less variation for each simulated population, meaning that once they reach the ligand source they tend to stay closer to it for the remainder of the simulation.

5.1.3 Comparison

In Figure 5.4, we can see that the bacteria reach a distance of roughly 3 p.d.u. at 250 iterations, while in Figure 5.2 the bacteria are only at around 5.5 p.d.u. This means that the bacteria more quickly get to the ligand source with the Delta-Regulator. We can also see that the Delta-Regulator bacteria stays around 2 p.d.u. from the ligand source with little variation as the simulation progresses. On the other hand, the ODE-regulator bacteria stays around 4 p.d.u. with more variation between simulations.

5.2 Quasi-dynamic Models

In this section, we will show data, plots, and graphics from running the different regulators in a simulation with a quasi-dynamic environment and cell division and -death. The regulators will be evaluated based on the bacteria's average distance from the ligand source as well as the population size. The differences between the regulators performance will also briefly be discussed.

5.2.1 ODE-Regulator

In Figure 5.5, we see how the population has increased from the 30 initial bacteria over the course of the simulation. The bacteria also appear to radiate outwards from the centre of the simulated environment.

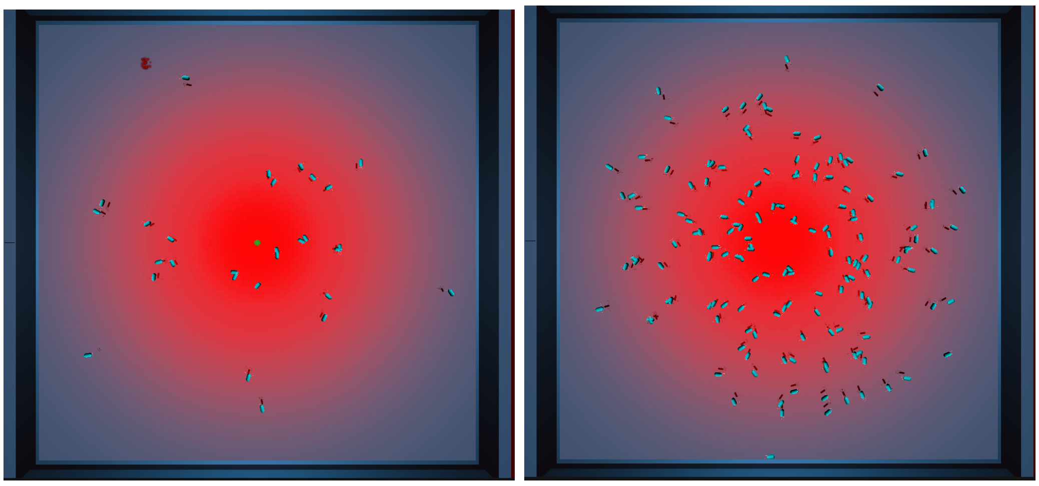


Figure 5.5: Initial (left) and final (right) states of the dynamic simulation with the ODE-Regulator.

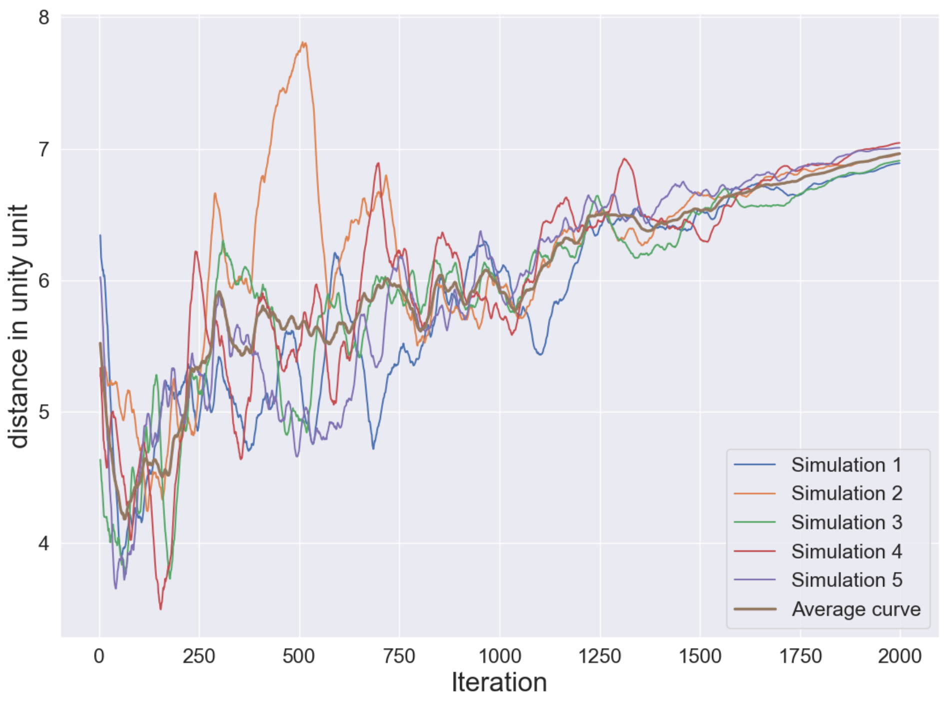


Figure 5.6: Average distance to ligand source for five separate runs and with a fitted mean curve with the ODE-Regulator.

Figure 5.6 shows us the average distance to the ligand source for the entire population. We can see that all the curves follow a general trend, but there are also big oscillations for some populations. Towards the end, all curves seem to stabilise and no longer show any oscillations like in the earlier stages.

Lastly, Figure 5.7 shows how the size of the population climbs quickly, only to taper off and oscillate around 160-175 individuals.

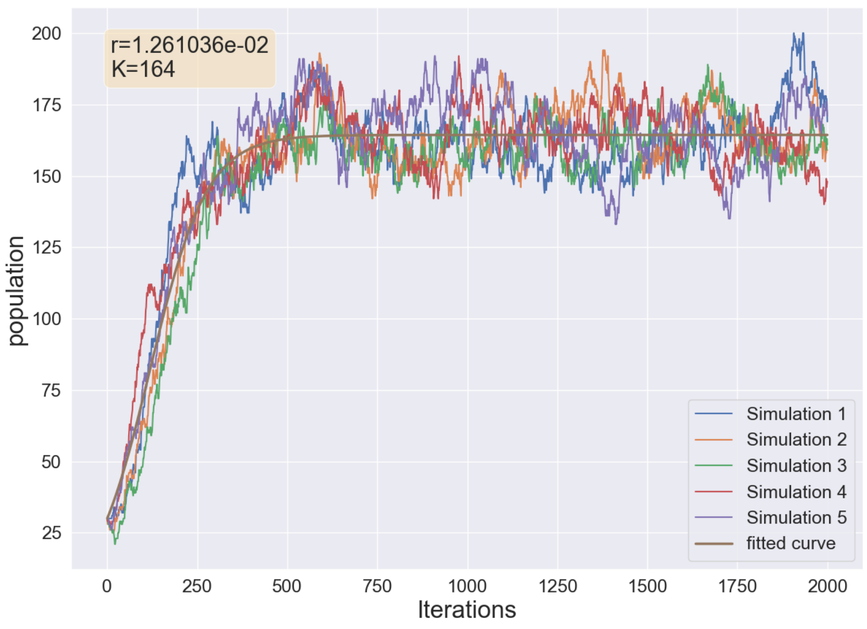


Figure 5.7: Population size evolution for five separate runs fitted with a logistic curve.

5.2.2 Delta-Regulator

Figure 5.8 shows a population boom within the centre of the simulated environment, radiating outwards.

In Figure 5.9, we can see the average distance to the ligand source. We notice that the general downward trend is there, though the population seem to deviate further away from the ligand source and there are also large oscillations.

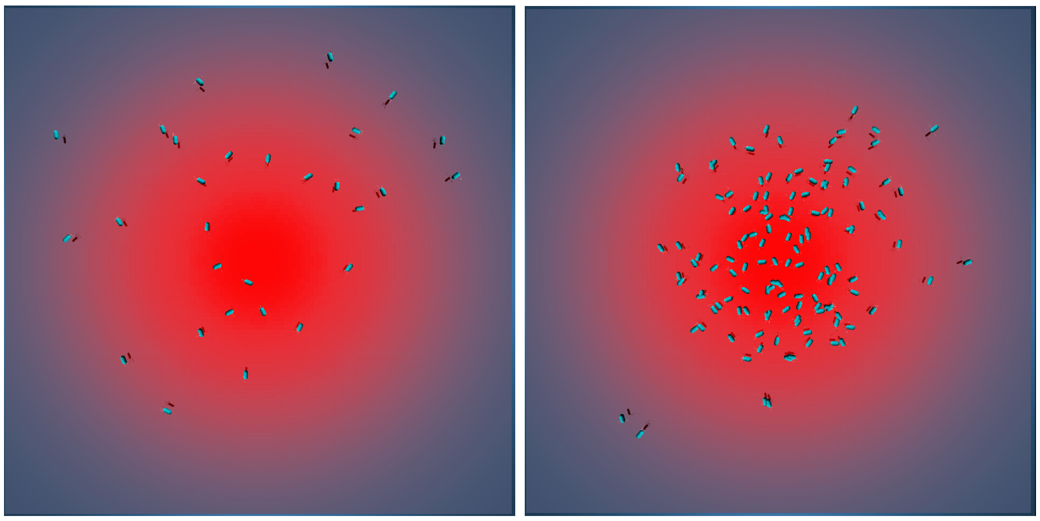


Figure 5.8: Initial (left) and final (right) states of the quasi-dynamic simulation with the Delta-Regulator.

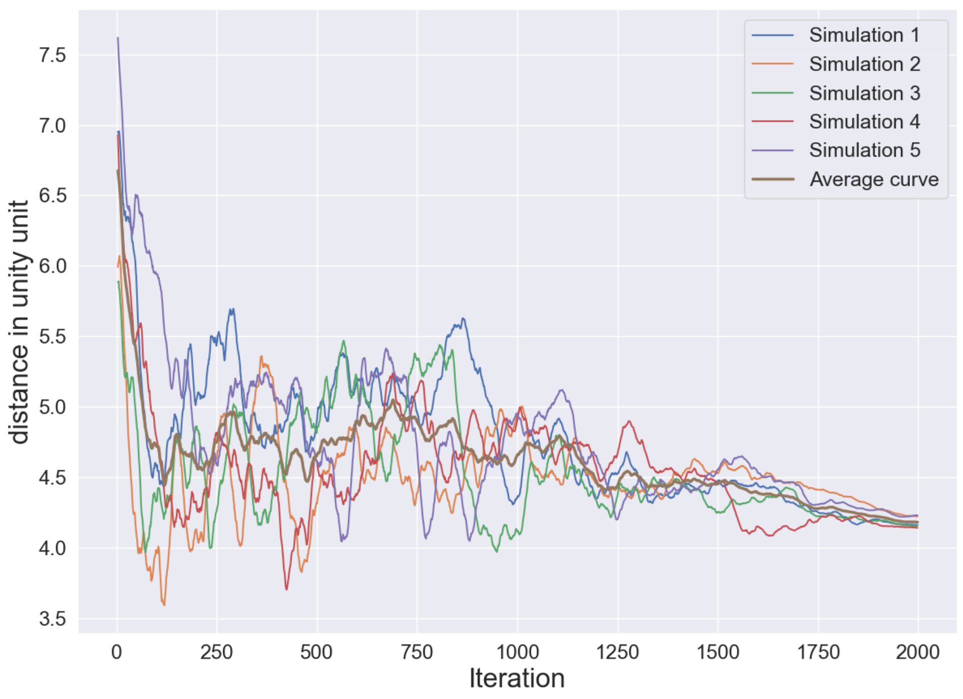


Figure 5.9: Average distance to ligand source for five separate runs and with a fitted mean curve with the Delta-Regulator.

In 5.10, we see an initial rapid increase in population, followed by a somewhat steady population size of between 100 and 140 individuals.

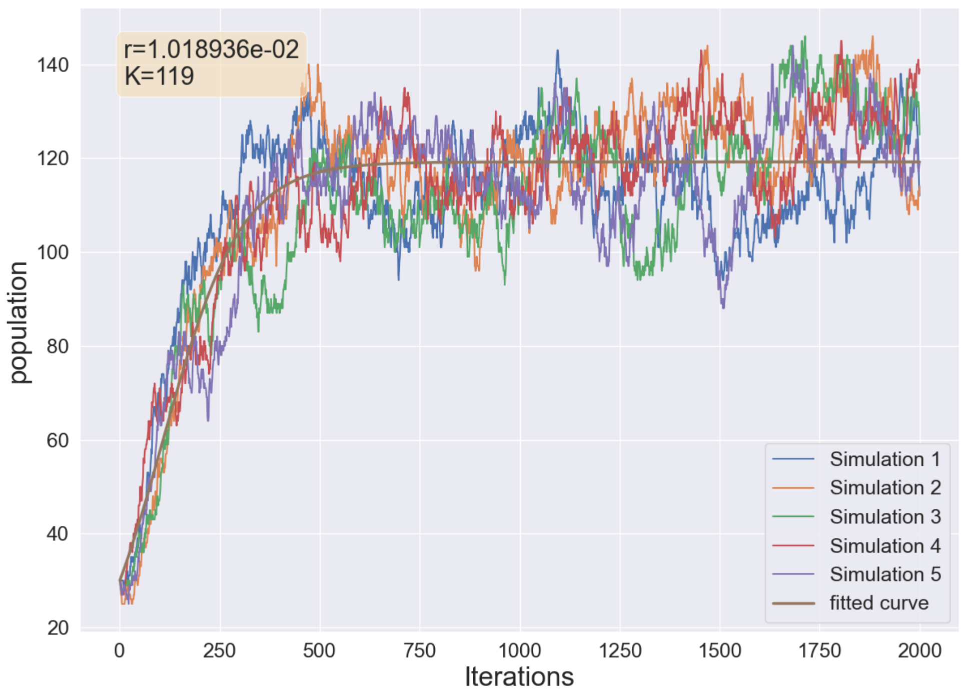


Figure 5.10: Population size evolution for five separate runs fitted with a logistic curve.

5.2.3 Comparison

Comparing the two population sizes shown in Figure 5.10 and Figure 5.7 we can note that the carrying capacity differ. We can make a rough estimation of the carrying capacity for the ODE-Regulator to be $K_{ODE} \sim 160-175$ and for the Delta-Regulator we get $K_{Delta} \sim 100 - 140$. We can thereby conclude that the ODE-Regulator can sustain a higher number of cells than the Delta-Regulator. Furthermore, comparing the results in Figure 5.6 with Figure 5.9 we can see a substantial difference. Looking at the distance graphs in the ODE-Regulator we can note that it has a general uptrend, whilst the Delta-Regulator has a general downtrend. However, the two regulators seem to both have large oscillations at the start of the simulation whilst stabilising towards the end.

5.3 Proof-of-concept: Dynamic Environment

This section aims to illustrate the ODE-Regulator's adaptation to a dynamic environment where diffusive ligand distribution as well as bacteria induced consumption are implemented. The model was only implemented in Matlab and therefore, these results serves as a proof-of-concept, illustrated through Matlab-plots.

The results emerge from solving a system of equations, $AL^{(n+1)} = (B - F)L^{(n)}$, where A and B are matrices which regulate the diffusion through dependence on the time step Δt, the spatial step Δx and the diffusion rate D, as described in Section 3.4.3. The matrix F describes the total consumption by the whole population of bacteria. Each bacterium's consumption depends on its current position, the consumption area a_1 and the consumption rate a_2. $L^{(n)}$ is a vector where each element corresponds to the ligand concentration in a certain point in the discretised region Ω, at time step n. The initial ligand distribution depends on number of sources, the minimum l_0, the amplitude Λ and the steepness coefficient d, such that the maximum ligand concentration for a single source is $l_0 + \Lambda$, positioned in its centre (see Figure 5.11).

All calculations were executed with $\Delta t = 1$ s, on a 60x60 grid of size 28x28 p.d.u., that is with $\Delta x = 28/60 = 0.467$ p.d.u.

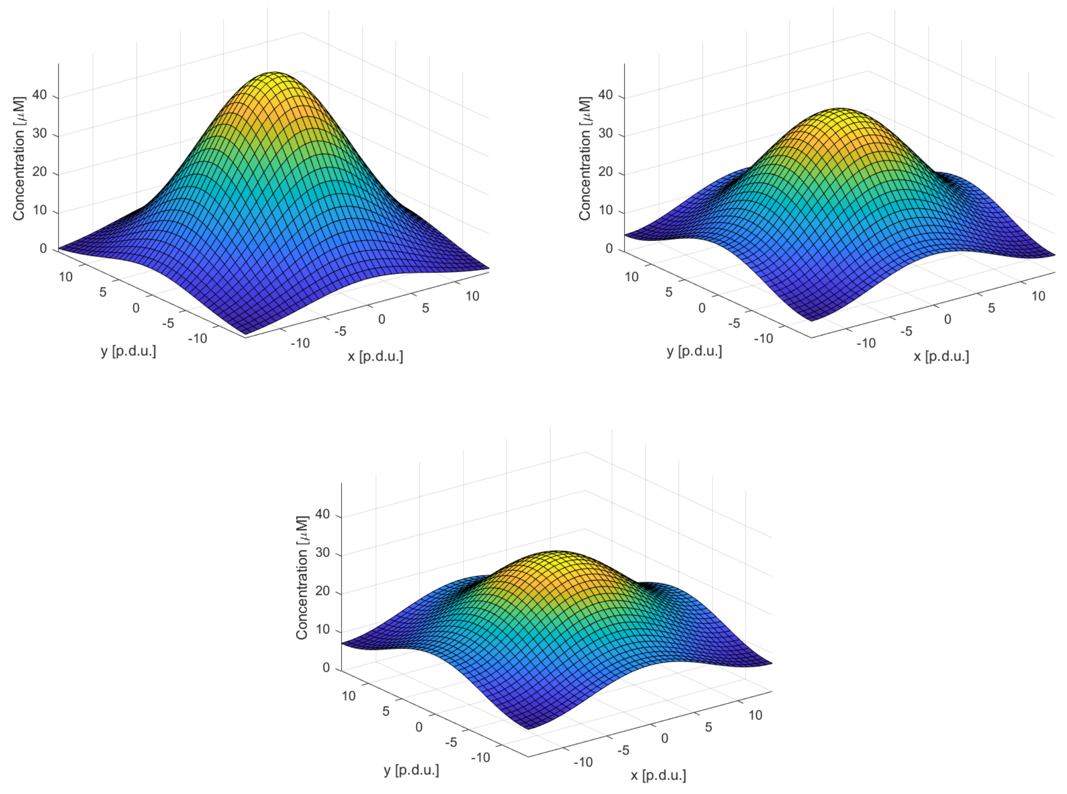


Figure 5.11: Illustration of diffusion with a single centred ligand source with $\Lambda = 49$, $d = 100$, $l_0 = 0$, $D = 0.5$ and $a_1 = 0$, that is without bacterial consumption. The diffusion is shown for $t = 0$, $t = 125$ and $t = 250$ time steps respectively.

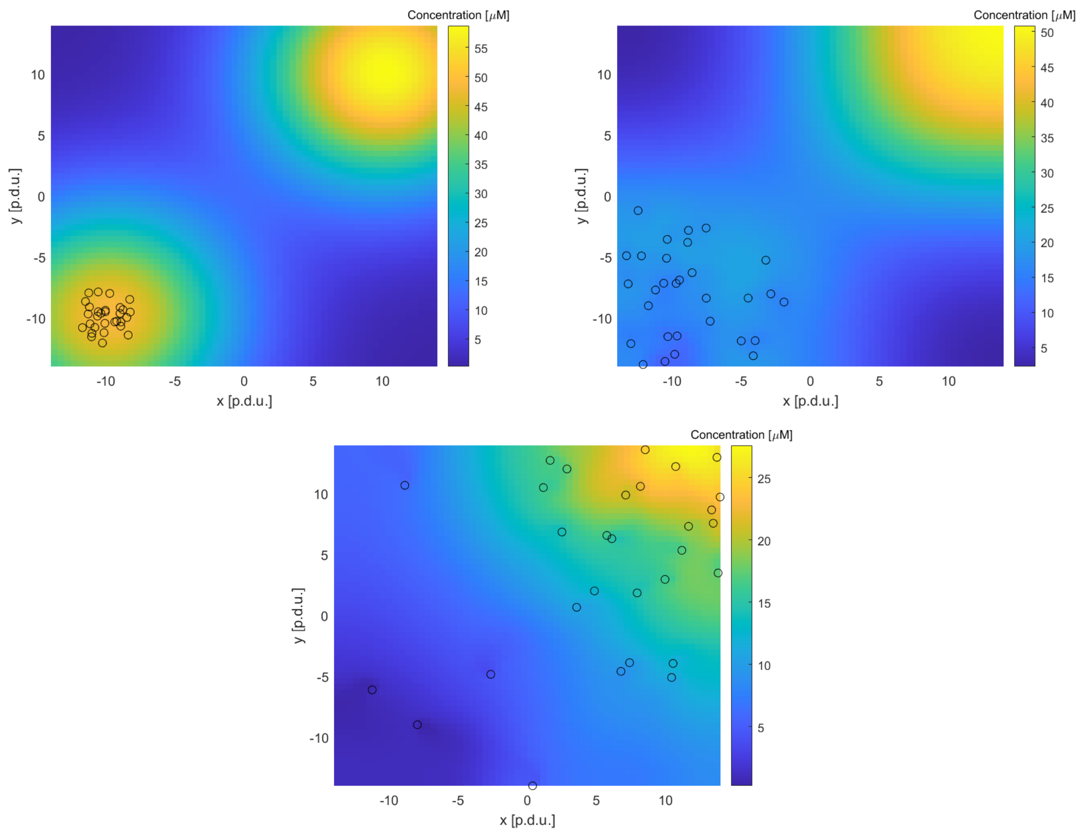


Figure 5.12: Illustration of effective bacterial migration through chemotaxis, with diffusing ligand distribution and consuming bacteria. One source S_1 was placed in (-10,-10) with $\Lambda = 49$ and a second source S_2 was placed in (10,10) with $\Lambda = 58.8$, both with $l_0 = 0$, $D = 0.2$ and $d = 95$. The consumption parameters were $a_1 = 0.3$ and $a_2 = 6$. The initial positions of all 30 bacteria were within 2 p.d.u. of S_1, illustrated on the first figure on the top left. The figure on the top right, at $t = 500$ time steps, illustrates the depletion of S_1 and the start of bacterial migration. The last figure on the bottom, at $t = 2000$ time steps, illustrates the bacterial migration at the end of the simulation.

5.4 Application

The application consists of three main parts: an initial setup screen, the simulation itself and lastly the end screen and analytical tool. The setup screen contains a user interface in which different parameters can be set which will be taken into account when simulating. The different parameters are the following:

- **Number of sources:** The user can choose up too 5 ligand sources which have a fixed position in the simulation area,
- **d & c_0:** Governs the characteristics of the ligand sources,
- **Dynamic Environment:** Determines if the environment should be dynamic or static,
- **Number of cells:** Specifies how many cells should be created at the start of

the simulation,

- **Regulation Type:** The two regulators can be chosen here are Delta- and ODE-Regulator,
- **Forward simulation:** Determines if all the calculations should be done in real time or before the simulation starts,
- **Number of iterations:** Defines how many iterations the simulation should run for,
- **Number of runs:** Specifies how many times the simulation should run, and
- **Cell division and -death:** Determines if the cells should be able to die and divide.

A preview of how the concentration is distributed is also shown in this view to give a more intuitive feeling when changing the characteristics of the ligand sources. When the user has set the desired parameters they can choose to start the simulation. Note that none of the parameters can be changed when the simulation has started.

When the simulation has started it can be seen in real time, see Figure 5.13, it contains the bacteria themselves, see Figure 5.14, the boundary in which they reside, as well as a concentration heat map. This heat map shows the ligand concentrations spread out over the simulation space. One can move the position of the camera freely around the environment with the help of the keyboard and mouse. It is also possible to press G on the keyboard to get a top-down view of the whole simulation area. Furthermore, the simulation can be paused as well as sped up to 25 times the normal speed. Specific information about the simulation is displayed at the bottom of the screen while it is running. Information such as current amount of bacteria, elapsed simulation time, environment type and simulation speed is shown. The different bacteria that exists in the simulation can be selected with the cursor to show bacterium specific information. Details like concentration of the internal chemicals CheY-P, CheA-P, CheB-P, and methylation level are shown, as well as current ligand concentration. Furthermore, if cell division and -death has been enabled, the bacterium's lifespan and time until cell division is also shown in this view.

Lastly, when all iterations have been simulated, the simulation stops and an end screen is shown. This screen shows the results of the simulation, displaying basic data such as number of existing cells at the end of the simulation, number of iterations run, simulation time elapsed, and average ligand concentration. A graph is also shown which displays ligand concentration over time. A higher value of the ligand concentration means that the bacteria where closer to the ligand source. It is also possible to export the data collected from the simulation onto a JSON (JavaScript Object Notation) file. This file can then be analysed using the analytical tool described in Section 4.3. This external tool allows the data to be compiled into useful plots which can be used for further analysis.

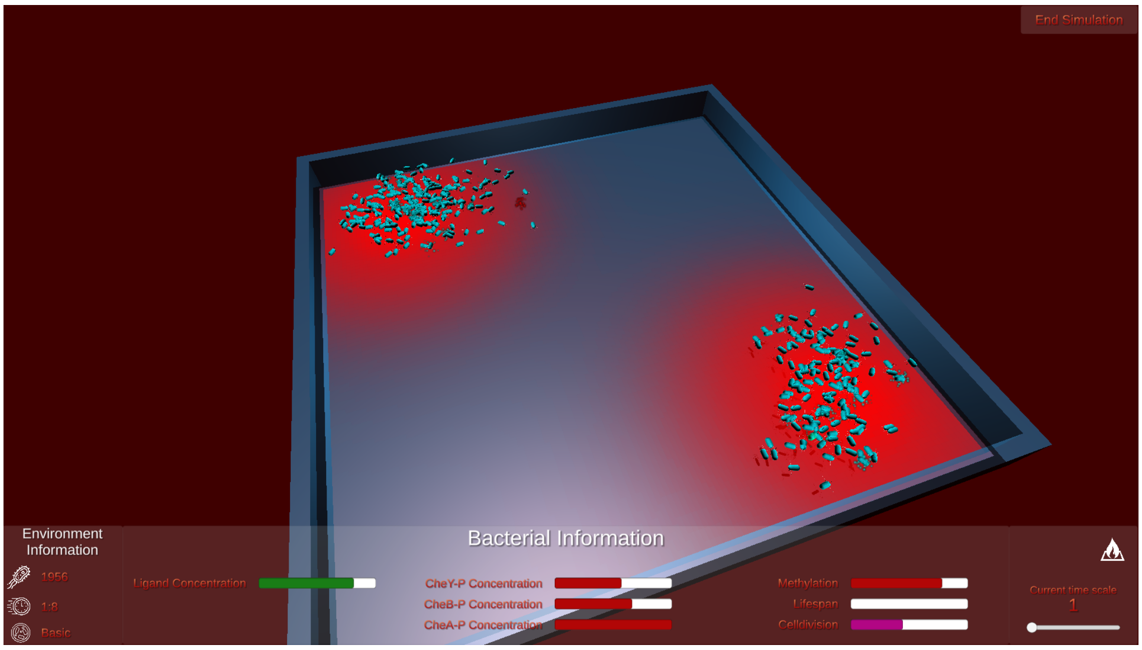


Figure 5.13: Screenshot of the main simulation scene.

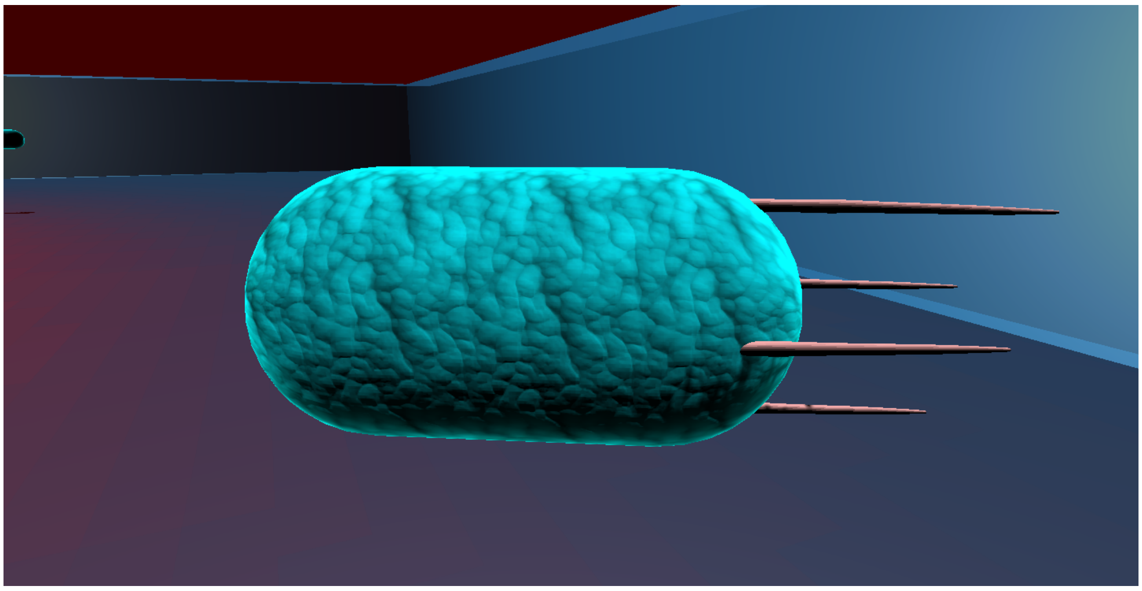


Figure 5.14: Close-up view of one of the many bacteria which can be seen in the simulation scene.

6 Discussion

In this chapter, we reflect on the results from the evaluation study, as well as insights that have come up during the course of the project.

6.1 ODE-Regulator Performance

The ODE-Regulator displays clear chemotactic behaviour, which is supported by the data. From a uniform distribution over the environment initially, the bacteria quickly find their way towards where the ligand concentration is at its highest. We use a static environment to investigate chemotactic behaviour since the bacteria have a clear goal to strive towards and then stay close to. In Figure 5.2, we see that the bacteria stay at approximately 4 p.d.u. from the peak of the ligand concentration stably. One might wonder why they do not decrease this distance continually, to come to a rest at the absolute epicentre. The answer lies in the fact that the bacteria do in fact continually seek to improve their positions. Once they have reached an optimal position, they will try to find something even better by tumbling and running in a new direction. The stability around 4 p.d.u. comes from the dynamic equilibrium around which the bacteria try to do this and the time it takes for them to realise that their position has worsened. In a radial distribution like the one in our simulation, every position outside of the ligand source is worse than the source, so there is no direction that the bacteria can travel to get better conditions. Hence, the fact that the results do not reveal optimal ligand finding is supportive of chemotaxis having emerged.

The data also suggests that the ODE-Regulator is a favourable tool to have for simulated bacteria in our given environment. Figure 5.7 shows a steep population growth that tapers off and oscillates around a population size roughly five times larger than the initial population.

Looking only at bacteria with the ODE-Regulator, we can surmise that it does give rise to chemotaxis to some degree. This is shown in a static simulation in which only the motility of bacteria is measured, without competition for ligand. We then have evidence that suggests that this degree of chemotaxis is favourable in terms of survivability in an environment with competition. Without any point of comparison, however, this merely suggests that it is better for the bacteria to have chemotaxis than it is to not have it.

6.2 Comparison Between Regulators

The results of our two experiments reveal seemingly conflicting conclusions. The static simulations show that bacteria with the Delta-Regulator are better at finding and staying close to the ligand source. The equilibrium of the movement pattern of circling the ligand source is about 2 p.d.u. for the Delta-Regulator, compared to 4 p.d.u. for the ODE-Regulator. The different simulations also show less variation between different populations of Delta-regulated bacteria. Keeping in mind that the Delta-Regulator was implemented specifically to be very good at navigating towards ligand-sources, these results are not very surprising. If anything, one might argue that it is a wonder that the ODE-Regulator comes as close to the Delta-Regulator as it does. The great coherence of the populations with Delta-Regulator also suggests that this optimisation for ligand finding is the main, singular influence for the motility of these bacteria.

The most surprising results are the ones from the quasi-dynamic experiment. While the Delta-Regulator is clearly better at finding the ligand source, the ODE-Regulator is superior for population growth. The populations grow at similar rates, with $r_{ODE} \approx r_{Delta}$. We believe that this comes from the implementation of cell division being the dominant factor when determining the rate. The biggest difference between the regulators are the different carrying capacities, with $K_{ODE} = 164$ and $K_{Delta} = 119$. The most likely explanation for this is that the ODE-Regulator is more flexible and adaptable than the Delta-Regulator. As previously mentioned, the bacteria with the Delta-Regulator seem to be more strongly inclined to find the ligand source than the ODE-bacteria. When there is competition between bacteria for ligand availability, however, it is not favourable to concentrate all bacteria at the same position. While ODE-regulated bacteria adapt to these new conditions, the Delta-bacteria seem to get stuck closer to the ligand source. Looking at figures 5.6 and 5.9, we see that the average distance to the ligand source increases for the ODE-population, while it decreases for the Delta-population. This discrepancy shows the dangers of looking only at movement patterns: the emergent adaptability of ODE-Regulator trumps the movement-wise better performing Delta-Regulator in terms of survivability. This is a case for bottom-up modelling in general, as comprehensive understanding of components can yield surprising results in novel circumstances.

6.3 ODE-Regulator in the Dynamic Model

The successful implementation of both a diffusive ligand distribution and consuming bacteria is likely the clearest testimony to the adaptive properties of the ODE-Regulator. The fact that a significant portion of the bacterial population never manages to migrate towards another source of ligand, highlighting its stochastic nature, only adds to its applicability. This in particular makes the possible implementation of cell division and -death tremendously more interesting. These exciting results immediately spark ones imagination and inspires a desire to keep experimenting with even more complex environments. By, for example, dramatically increasing the size of the region and consistently adding new ligand sources as time progresses, the survivability and adaptability of bacterial populations would become immensely more intriguing.

6.4 Stochastic Regulator

The ODE-based approach to chemical modelling adopted in this project views the chemical reaction network as a whole, and describes observed relationships between the reactants. An alternative approach to modelling coupled chemical reactions is to view them as stochastic processes. What is needed for this type of method is a collection of individual reactant particles, a set of reactions, and some probabilistic method of choosing a reaction to occur. An initial distribution of reactants would then morph according to a series of reactions. Typically, the rules for discriminating between reactions would involve both that reaction's isolated kinetic rate and the concentration of its constituent reactants. This way, reactions that happen quickly and reactions with abundant reactants would occur with larger probability. The available reactants get updated after each reaction, which shifts the probabilities of each reaction to occur. Examples of this approach are the Stochastic Simulation Algorithm of Gillespie [7], and surface CRN simulations [20].

The benefits of simulating a chemical reaction network on the level of individual molecules mirrors the benefits of agent-based modelling. In a sense, it is a more realistic approach: chemicals cannot solve differential equations. Rather than trying to capture and express the complexity of a system, the modelling effort is directed at its basic components. This approach runs the risk of emergent behaviour being inaccurate compared to what is observed. The claim, however, is that the interaction between comprehensively modelled components can give rise to nuances and complexity not otherwise attainable. Gillespie makes a similar claim when comparing ODEs to his Stochastic Simulation Algorithm: the latter captures fluctuations that the former does not [7]. An interesting experiment would be to implement a stochastic regulator and compare its performance to our ODE-Regulator. We attempted just this, but found it hard to get the implementation done right within the timeframe of the project.

6.5 Credit to Biology

The good performance of the ODE-Regulator begs the question of how something seemingly oblivious to its surroundings could adapt to them so well. The rather crude, but adaptive-by-design Delta-Regulator is, in our study, outperformed in terms of overall score in fitness metrics. This achievement suggests that there are nuances within the simple chemical network of the chemotactic pathway that allow these bacteria more complex responses to their environment. The sensitivity of CheY to be phosphorylated leads to the possibility of rapid changes in behaviour, while the methylation and demethylation of the MCPs gives the bacterium a rudimentary memory. The intricacies of how these chemicals work and are intertwined is likely beyond the scope of our simulation, but their observed changes and effects let us access their emergent complexity. One big difference between our computational model and real E. coli is the cost of this emergent complexity. Solving the ODEs is computationally expensive, and still most likely misses important details in the natural reactions. In bacteria, these reactions happen spontaneously, and there is no mechanism for monitoring and controlling the system as a whole. Instead, the balance between all components making up a bacterium are, through evolution, so carefully balanced that no coordination is needed: it all just happens. As has been mentioned before, these spontaneous reactions are also simple chemically; a single transfer of a phosphoryl group between two reactants is not very complex. Rather than extending the responsibility of each component in the pathway, E. coli have evolved to let complexity arise through their interactions. These single celled organisms are truly great examples of making the most out of very little. Both economic efficiency and biological fitness is achieved by letting go of control and leaning on simplicity, rather than micromanagement and complex rule systems. This effect is paralleled on a higher level of abstraction is the adoption of a method like ABM. It is tempting to model the entire system with top-down understanding of observed behaviour, but in letting our agents act independently, we see similarly complex results, and at times are even surprised by the unexpected.

6.6 Possible Use

Our application can be of use mainly for education and research purposes. We have striven to make the simulation as immersive as possible, to highlight the link between intracellular processes and population level phenomena. We believe that seeing chemotaxis from the view of individual bacteria in congregation helps connect the biological theory with intuition. With regards to research, both environmental factors and alternative regulators can be investigated with our provided tools. The distribution of ligand can be configured before each simulation; insofar as these settings are of interest, their effect on the bacteria can be compared directly in the application. Of more interest is the evaluation of different models of the chemotactic pathway, or other types of regulators. The structure of the application makes it easy to implement new regulators and have our simulated bacteria use them to guide their movement. With the addition of our analytical tool, different regulators can also be compared quantitatively. An interesting research question would be to compare an implementation of the Stochastic Regulator to our ODE-Regulator.

6.7 Further Development

The application contains several areas that would benefit from further development. First and foremost, the previously described model for a dynamic environment should also be implemented in the Unity application, since it would add further value to the simulation. In the same vein, anything that makes the general simulation application richer should be implemented. This includes additional models of chemotaxis, like the Stochastic Regulator from 6.4, for the purpose of comparison and deeper insight. The goal is to reach as good of an understanding of the biology as possible. Parallel to the chemotaxis modelling, other bacterial functions could be made more realistic, most notably division and death.

Secondly the currently used ODE-solver is extremely slow, which results in very long loading times for larger simulations. This also acts a limiter on the complexity of the simulations that can be run in the application. The previously mentioned dynamic environment would almost be impossible to run using the current ODE-solver since it would add many extra calculations for it to handle which would further lengthen the loading times. It would therefore be highly advantageous to change to faster solver if one is available or optimise the ODEs so that they run faster using the current solver. Alternatively, efforts into making the regulators of different bacteria run in parallel would be useful. Some thought went into this with regards to using compute shaders in Unity, but translating the ODE solvers into shader language would be too much of an undertaking.

Additionally, the current approach that is used when exporting data from the simulation results in large amounts of duplicate data. Since child bacteria share their pre-birth movement history with their parent and dead bacteria have their death position saved for all future time steps in the simulation. This means that in a simulation of 200 time steps, a bacterium that is created in time step 100 and dies in 101 would have 200 saved positions, even though it should only have 1. This results in longer loading times, since the data is exported during the loading, and substantially larger export files which in turn slows down the data analysis.

Lastly, further aesthetic improvements to the current cell models and its animations could be implemented. The cells could be made more realistic by, for example, adding dynamically sized and positioned flagella. In the current simulation, all cell models and their colours are the same. This could be improved by altering each cell's appearance slightly to make them more realistic. Furthermore, the tumble- and run animations can be enhanced by letting the cell rotate CW and CCW around its own axes respectively. More realistic custom animations could also be added for when the cell divides as well as dies.

6.8 Workflow

Creating the three different groups at the start of the project turned out to be very rewarding. The group members could solely focus on their work area resulting in the development to speed up substantially. However, these groups where not kept the same during the whole project. As the development progressed, areas like the extracellular quickly became fully finished, and it was thus important to always arrange the tasks in such a way that everyone in the group had things to do. The weekly meetings with the supervisor was a big help for the group and for the project as a whole. There where many situations where knowledge and experience was lacking from the group members; many of these problems where solved when discussed with the supervisor, which resulted in the project being led in the right direction. The weekly group meetings also worked very well: it gave everyone in the group a chance to talk about what they had done in the week and to discuss about the project in general. Two of these weekly meetings were scheduled throughout the week, although if there where any urgent deadlines approaching, extra meetings would be arranged. Structuring the meetings in this way was very healthy for the project as it kept the development going and important deadlines where always met.

6.9 Ethical and Societal Aspects

Neither the process of developing this simulation and application, nor their results, carries the risk of harm unto others. Given the currently ongoing pandemic, it is important not to provide sources of potential misinformation. Since this simulation is so narrowly focused on the movement of individual cells, and since no claims about viruses or large-scale migration patterns can be made from our data, we determine that the two issues are far enough separated. The extent of the impact this product could have is within the field of understanding bacterial chemotaxis. Most of this potential, we determine to be positive; researchers can use our tool for data analyses, and students can use it to build intuitive and quantitative understanding. Insofar as negative consequences could arise, they would have to do with these data and visualisations misrepresenting the phenomenon or underlying mechanics of chemotaxis. Given the transparency between the underlying model and observed behaviour, we feel that these risks are outweighed by the potential good a tool like ours could bring about. Furthermore, we have built the product in such a way as to make it easy to extend with other models of the chemotactic pathway. Ultimately, this means that our product could be used to measure and compare different models of the pathway, deepening our understanding of how to simulate it computationally.

7 Conclusion

At the core of this project, we have implemented a computational model of the chemotactic pathway. This model — called ODE-Regulator — is based on the mathematical model introduced by Edgington and Tindall [1], and is implemented in the programming language C#. Alongside the model of the chemotactic pathway, we implemented a simplified model — called Delta-Regulator — that uses differences in concentration to achieve chemotaxis. We have also created an agent-based simulation in the Unity engine, in which simulated E. coli-bacteria governed by these models navigate an environment with spatial differences in ligand concentration. Comparing the results simulating these models, Delta-Regulator is more effective at finding the ligand source, while ODE-Regulator performs much better in terms of population growth. Both display chemotaxis, but the more comprehensive ODE-Regulator appears more flexible to environmental challenges like competition for ligand. To analyse the data from our simulations, we have also developed a tool in the programming language Python to run statistics.

We surmise that our detailed bottom-up model of the chemotactic pathway yields population level phenomena corresponding to observed chemotaxis and population dynamics. In doing so, we provide a connection between population level dynamics and the internal dynamics of the oblivious individual. We also make the case that our agent-based model displays emergent complexity akin to population level simulations. The tools we have developed around our model can be used in the future to evaluate models of chemotaxis both with immersive visuals and with statistics.

Bibliography

[1] M. P. Edgington and M. J. Tindall, “Mathematical analysis of the escherichia coli chemotaxis signalling pathway,” *Bulletin of Mathematical Biology*, vol. 80, pp. 758-787, 2018.

[2] J. S. Parkinson, “Bacterial chemotaxis: a new player in response regulator dephosphorylation,” *Journal of Bacteriology*, vol. 185, no. 5, pp. 1492–1494, 2003.

[3] C. H. Hansen, R. G. Endres, and N. S. Wingreen, “Chemotaxis in escherichia coli: A molecular model for robust precise adaptation,” *PLOS Computational Biology*, vol. 4, no. 1, 2008.

[4] E. F. Keller and L. A. Segel, “Model for chemotaxis,” *Journal of theoretical biology*, vol. 30, pp. 225-234, 1971.

[5] J. M. Willey, L. M. Sherwood, and C. J. Woolverton, *Prescott’s Microbiology*, pp. 71-55, 339-341. New York, USA: McGraw-Hill, 10th ed., 2016.

[6] P. Érdi and J. Tóth, *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*, p. 3. Manchester University Press, 1989.

[7] D. T. Gillespie, “Exact stochastic simulation of coupled chemical reactions,” *The journal of physical chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[8] I. Gustafsson and K. Holmåker, *Numerisk Analys.* Stockholm, Sweden: Liber AB, 2016.

[9] C. Kartsonaki, “Survival analysis,” *Diagnostic Histopathology*, vol. 22, no. 7, pp. 263–270, 2016. Mini-Symposium: Medical Statistics.

[10] M. P. Edgington and M. J. Tindall, “Understanding the link between single cell and population scale responses of escherichia coli in differing ligand gradients,” *Computational and Structural Biotechnology Journal*, vol. 13, pp. 528-538, 2015.

[11] W. A. Strauss, *Partial Differential Equations, An Introduction.* Hoboken, USA: John Wiley & Sons, 2008.

[12] E. Bonabeau, “Agent-based modeling: Methods and techniques for simulating human systems.” `https://www.pnas.org/content/99/suppl_3/7280`, 2002. Accessed: 2021-04-19.

[13] “Agent based modelling: Introduction.” `http://www.geog.leeds.ac.uk/courses/other/crime/abm/general-modelling/index.html`. Accessed: 2021-04-20.

[14] J. D. Murray, *Mathematical biology: I. An introduction*, vol. 17. Springer Science & Business Media, 2007.

[15] “NumPy,” May 2021. [Online; accessed 13. May 2021].

[16] “SciPy.org — SciPy.org,” May 2021. [Online; accessed 13. May 2021].

[17] “Matplotlib: Python plotting — Matplotlib 3.4.2 documentation,” May 2021. [Online; accessed 13. May 2021].

[18] “Unity engine.” `https://unity.com/`. Accessed: 2021-03-25.

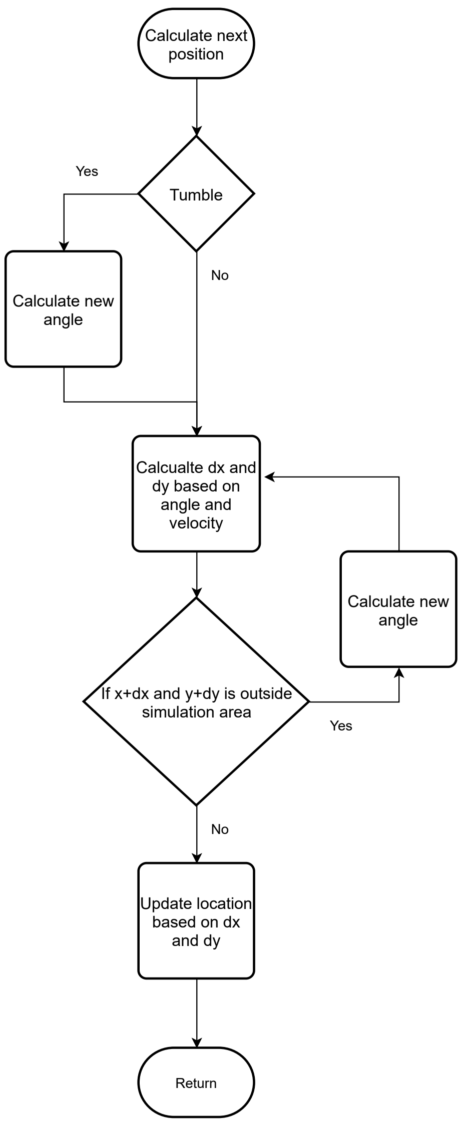
[19] “Blender.” `https://www.blender.org`. Accessed 2021-04-14.

[20] S. Clamons, L. Qian, and E. Winfree, “Programming and simulating chemical reaction networks on a surface,” *Journal of the Royal Society Interface*, vol. 17, no. 20190790, 2020.

[21] H. Logemann and E. P. Ryan, *Ordinary Differential Equations, Analysis, Qualitative Theory and Control.* Springer Science & Business Media, 2014.

A

Appendix 1: Figures



Calculate next position
Yes
Tumble
No
Calculate new angle
Calcualte dx and dy based on angle and velocity
Calculate new angle
If x+dx and y+dy is outside simulation area
Yes
No
Update location based on dx and dy
Return

Figure A.1: Flowchart for the calculate next position algorithm.

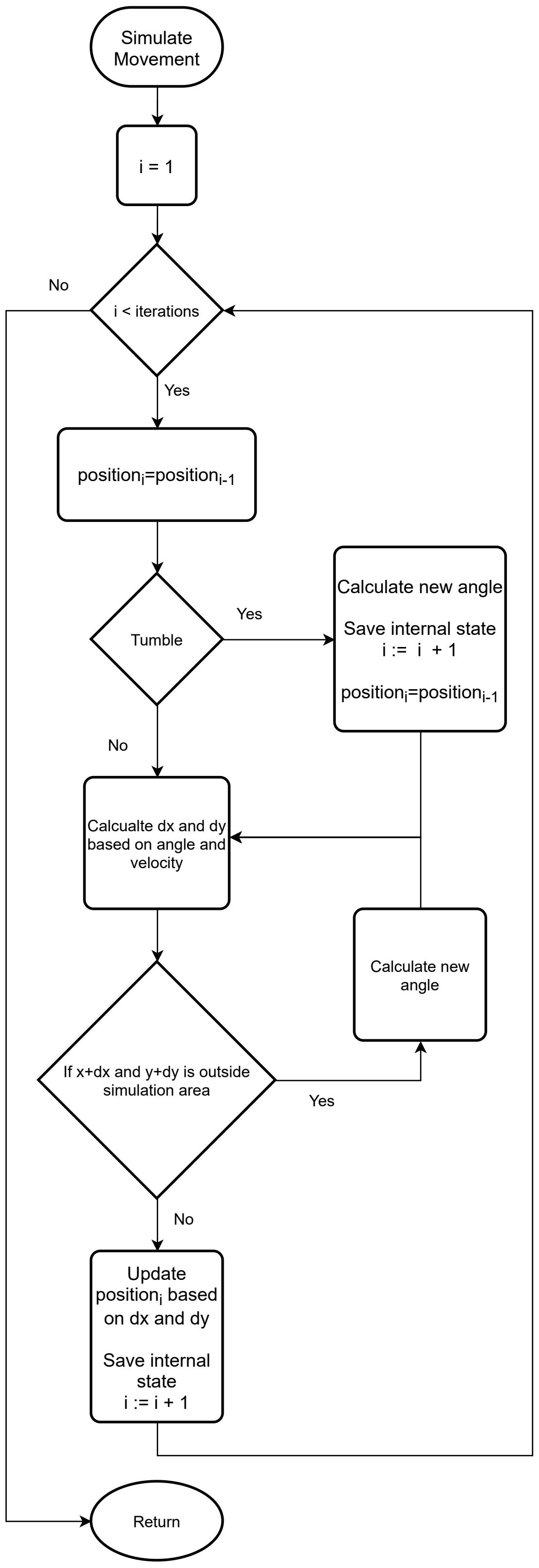

Simulate Movement
i = 1
i < iterations
No
Yes
position$_i$=position$_{i-1}$
Tumble
Yes
Calculate new angle
Save internal state
i := i + 1
position$_i$=position$_{i-1}$
No
Calcualte dx and dy based on angle and velocity
Calculate new angle
If x+dx and y+dy is outside simulation area
Yes
No
Update position$_i$ based on dx and dy
Save internal state
i := i + 1
Return

Figure A.2: Flowchart for the pre-calculation movement algorithm.

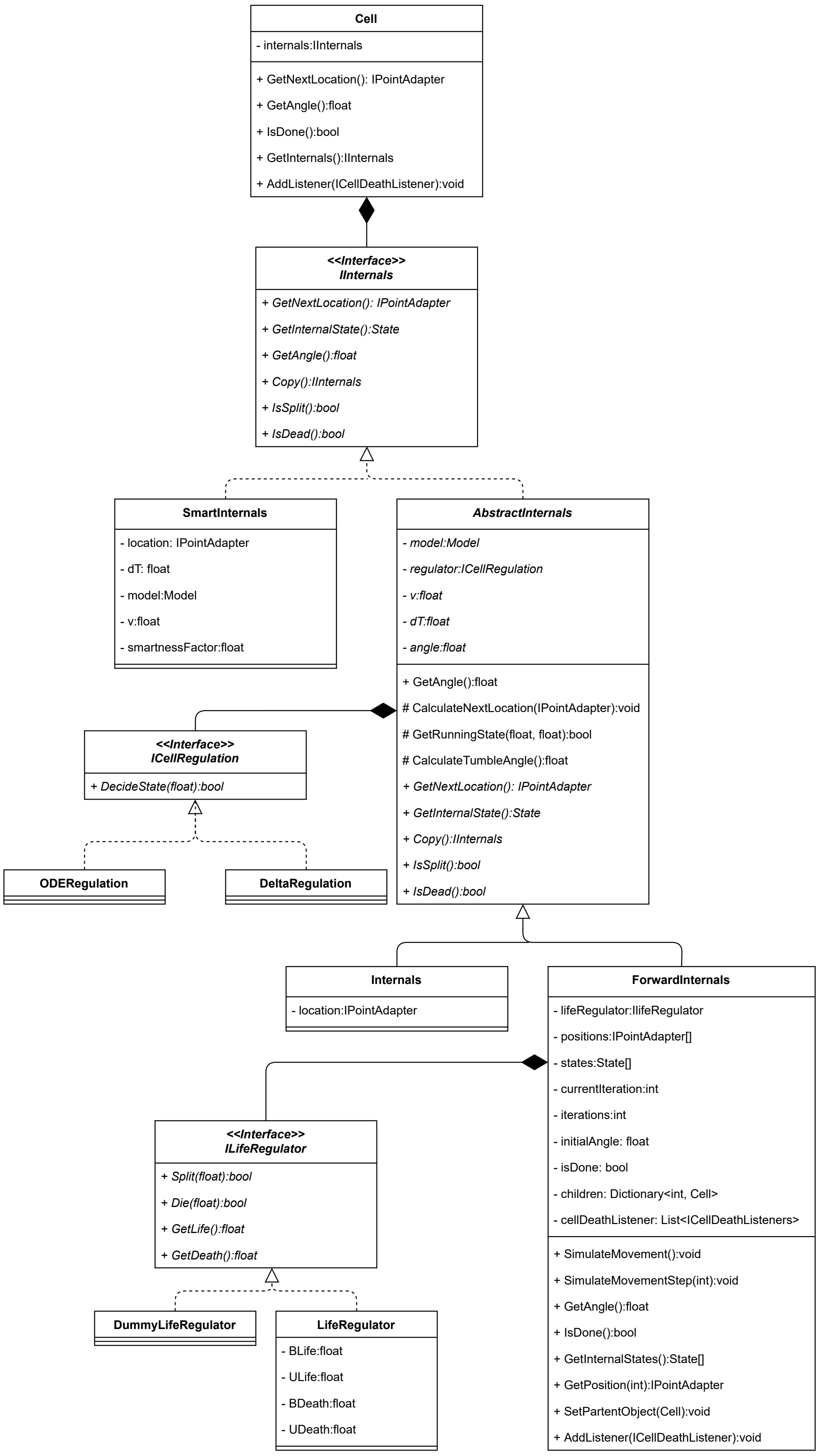


Figure A.3: Class diagram showing the structure of the cell object.

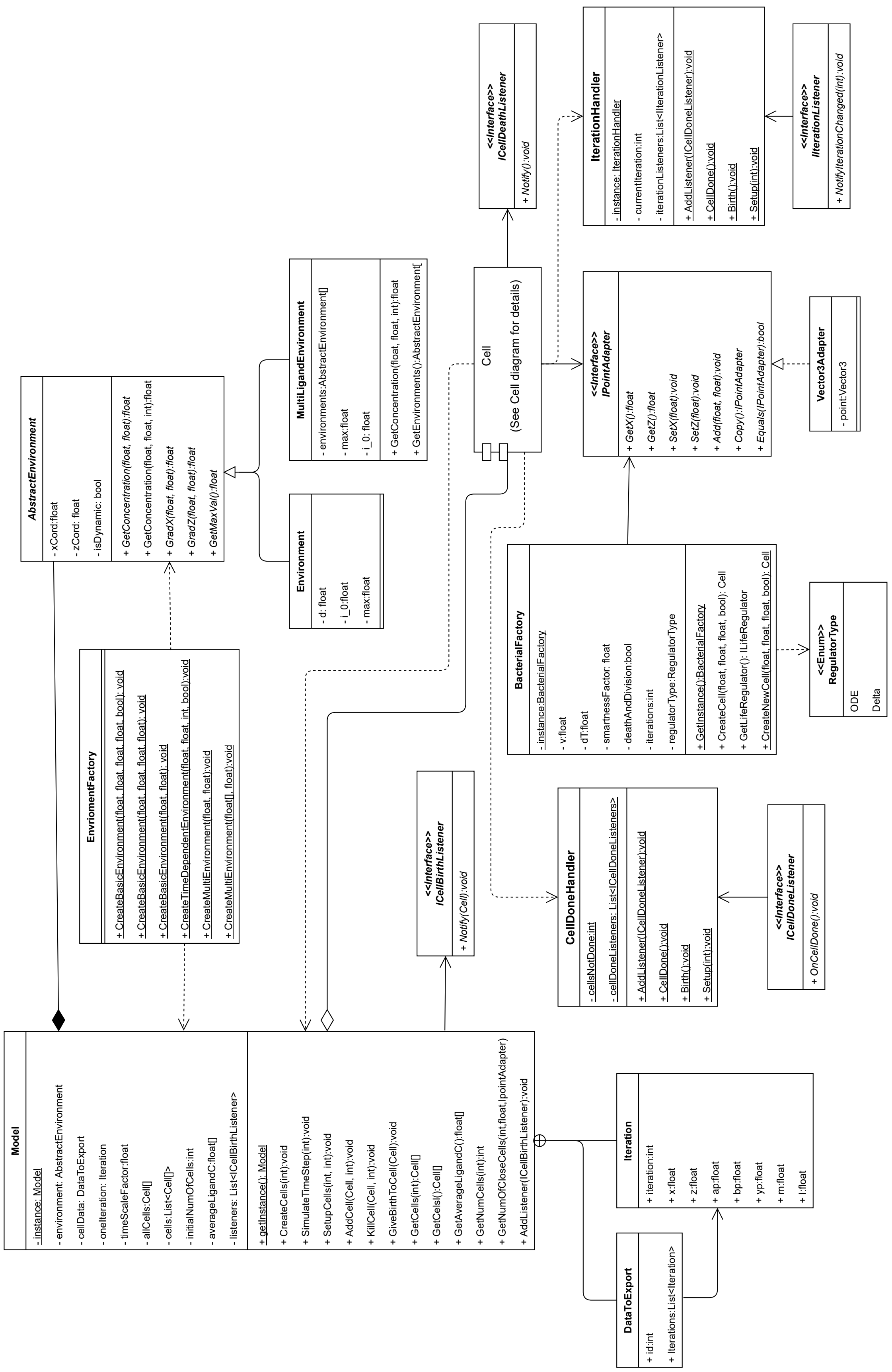

AbstractEnvironment
- xCord:float
- zCord: float
- isDynamic: bool
+ GetConcentration(float, float):float
+ GetConcentration(float, float, int):float
+ GradX(float, float):float
+ GradZ(float, float):float
+ GetMaxVal():float
MultiLigandEnvironment
- environments:AbstractEnvironment[]
- max:float
- i_0:float
+ GetConcentration(float, float, int):float
+ GetEnvironments():AbstractEnvironment[]
Environment
- d: float
- i_0:float
- max:float
Cell
(See Cell diagram for details)
<<Interface>>
ICellDeathListener
+ Notify():void
IterationHandler
- instance: IterationHandler
- currentIteration:int
- iterationListeners:List<IIterationListener>
+ AddListener(ICellDoneListener):void
+ CellDone():void
+ Birth():void
+ Setup(int):void
<<Interface>>
IIterationListener
+ NotifyIterationChanged(int):void
<<Interface>>
IPointAdapter
+ GetX():float
+ GetZ():float
+ SetX(float):void
+ SetZ(float):void
+ Add(float, float):void
+ Copy():IPointAdapter
+ Equals(IPointAdapter):bool
Vector3Adapter
- point:Vector3
EnviromentFactory
+ CreateBasicEnvironment(float, float, float, float, bool): void
+ CreateBasicEnvironment(float, float, float, float): void
+ CreateBasicEnvironment(float, float): void
+ CreateTimeDependentEnvironment(float, float, int, bool):void
+ CreateMultiEnvironment(float, float):void
+ CreateMultiEnvironment(float[], float):void
BacterialFactory
- instance:BacterialFactory
- v:float
- dT:float
- smartnessFactor: float
- deathAndDivision:bool
- iterations:int
- regulatorType:RegulatorType
+ GetInstance():BacterialFactory
+ CreateCell(float, float, bool): Cell
+ GetLifeRegulator(): ILifeRegulator
+ CreateNewCell(float, float, bool): Cell
<<Enum>>
RegulatorType
ODE
Delta
<<Interface>>
ICellBirthListener
+ Notify(Cell):void
CellDoneHandler
- cellsNotDone:int
- cellDoneListeners: List<ICellDoneListeners>
+ AddListener(ICellDoneListener):void
+ CellDone():void
+ Birth():void
+ Setup(int):void
<<Interface>>
ICellDoneListener
+ OnCellDone():void
Model
- instance: Model
- environment: AbstractEnvironment
- cellData: DataToExport
- oneIteration: Iteration
- timeScaleFactor:float
- allCells:Cell[]
- cells:List<Cell[]>
- initialNumOfCells:int
- averageLigandC:float[]
- listeners: List<ICellBirthListener>
+ getInstance(): Model
+ CreateCells(int):void
+ SimulateTimeStep(int):void
+ SetupCells(int, int):void
+ AddCell(Cell, int):void
+ KillCell(Cell, int):void
+ GiveBirthToCell(Cell):void
+ GetCells(int):Cell[]
+ GetCelsl():Cell[]
+ GetAverageLigandC():float[]
+ GetNumCells(int):int
+ GetNumOfCloseCells(int,float,IpointAdapter)
+ AddListener(ICellBirthListener):void
Iteration
+ iteration:int
+ x:float
+ z:float
+ ap:float
+ bp:float
+ yp:float
+ m:float
+ l:float
DataToExport
+ id:int
+ Iterations:List<Iteration>

Figure A.4: Simplified class diagram of the model.

B

Appendix 2: Tables

Constant	Value	Description
d	100	Determines the slope of the ligand concentration
Λ	49	Determines the maximum ligand concentration
l_0	0	Base amount of ligand concentration in each position
n	30	Number of cells at the start of the simulation
iterations	2000	The number of iterations that the simulation will run
$n_{sources}$	1	The number of ligand sources in the simulation
r	1	Determines the radius within which the cells impact each other

Table B.1: The parameters that where used in the simulations with descriptions of their purpose

C

Appendix 3: Mathematical Theory

C.1 Differential Equations

C.1.1 Background

A differential equation is an equation consisting of one or more functions and their respective derivatives. Differential equations is a very powerful mathematical tool and is usually implemented when we want to describe the rate of change of an object rather that its absolute value at some point [21]. There are many types of differential equations and (C.1) is an example of a simple differential equation.

$$y'(t) = f(t, y(t)) \tag{C.1}$$

where $y'(t) = \frac{dy}{dt}$ and $f(t, y(t)$ is a arbitrary given function. A differential equation can also be divided into different categories depending on what the right and left hand side of the equation states and what we wish to describe. Equation (C.2) shows a differential equation of order n, where n is order of derivative on y, also if $Q(x) = 0$ we say that it is a homogeneous differential equation and non-homogeneous otherwise.

$$a_n(x)y^{(n)} + a_{n-1}(x)y^{(n-1)} + ..a_1(x)y' = Q(x) \tag{C.2}$$

We can also distinguish differential equations into partial and ordinary equations where (C.3) is a partial differential equation, or PDE, since it depends on several variables: t, x, y, z and each respective partial derivative is present in the equation. An ordinary differential equation only depends on one variable and (C.1) is an example of a ordinary differential equation or ODE.

$$\frac{\partial^2 u}{\partial t^2}(t,x) = v^2\left(\frac{\partial^2 u}{\partial x^2}(t,x) + \frac{\partial^2 u}{\partial y^2}(t,x) + \frac{\partial^2 u}{\partial z^2}(t,x)\right) \tag{C.3}$$

We can also have system of ordinary differential equations. Below we can see (C.4) which is a first order linear ODE where $A(t)$ is a nxn coefficient matrix, b is a source vector and $y(t)$ is the unknown function vector.

$$y'(t) = A(t)y(t) + b(t) \tag{C.4}$$

$$A(t) = \begin{bmatrix} a_{11}(t) & . & . & a_{1n}(t) \\ . & . & . & \\ . & . & . & \\ a_{1n}(t) & . & . & a_{nn}(t) \end{bmatrix}, b(t) = \begin{bmatrix} b_1(t) \\ . \\ . \\ b_n(t) \end{bmatrix}, y(t) = \begin{bmatrix} y_1(t) \\ . \\ . \\ y_n(t) \end{bmatrix}$$

C.1.2 Numerical solutions

Often in practise the task of analytically solving a system of differential equations is difficult, if not impossible. There are however numerical methods which approximates the solution with adequate precision. Numerical differentiation can be accomplished with *finite difference approximations* [11].

Observe the definition of derivative as a limit, shown in Equation (C.5).

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{C.5}$$

If Δx has a fixed non-zero value the result is Equation (C.6), *the forward finite difference approximation.*

$$f'(x) \approx f'_{forward}(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{C.6}$$

By evaluating the function at x and $x - \Delta x$ the result is *the backwards finite difference approximation*, both of which has an error proportional to Δx. By evaluating the function at $x + \frac{1}{2}\Delta x$ and $x - \frac{1}{2}\Delta x$ instead the result is Equation (C.7), *the central finite difference method.*

$$f'(x) \approx f'_{central}(x) = \frac{f(x+\frac{1}{2}\Delta x) - f(x-\frac{1}{2}\Delta x)}{\Delta x} \tag{C.7}$$

which has an error proportional to $(\Delta x)^2$. Trivially, the precision increases with smaller steps Δx. To approximate the second order derivative one simply applies the central finite differential approximation again, which results in Equation (C.8)

$$f''(x) \approx f'_{central}[f'_{central}(x)] = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} \tag{C.8}$$

An appropriate method of numerically solving the diffusion equation, see (2.7), is *the alternating-direction implicit method*, or *ADI.* In fact it was developed specifically to solve the two-dimensional diffusion equation in a square region using finite differences. It's derived from *the implicit Crank-Nicolson method* [11] which uses Equations (C.7) and (C.8) to produce Equation (C.9)

$$\frac{c_{ij}^{n+1} - c_{ij}^{n}}{\Delta t} = \frac{1}{2(\Delta x)^2}(\delta_x^2 + \delta_y^2)(c_{ij}^{n+1} + c_{ij}^{n}) \tag{C.9}$$

where the region Ω has been discretised with a square grid, i.e. $\Delta x = \Delta y$, and a lattice-point (i, j) is denoted ij. δ_x^2 and δ_y^2 are operators defined through (C.8) as

$$\begin{aligned} \delta_x^2 c_{ij} &= c_{i+1,j} - 2c_{i,j} + c_{i-1,j} \\ \delta_y^2 c_{ij} &= c_{i,j+1} - 2c_{i,j} + c_{i,j-1} \end{aligned}$$

The concept of the ADI-method is splitting (C.9) into two half time-steps

$$\frac{c_{ij}^{n+1/2} - c_{ij}^{n}}{\Delta t/2} = \frac{\delta_x^2 c_{ij}^{n+1/2} + \delta_y^2 c_{ij}^{n}}{(\Delta x)^2} \tag{C.10}$$

$$\frac{c_{ij}^{n+1} - c_{ij}^{n+1/2}}{\Delta t/2} = \frac{\delta_x^2 c_{ij}^{n+1} + \delta_y^2 c_{ij}^{n+1/2}}{(\Delta y)^2} \tag{C.11}$$

the first half with the x-derivative taken explicitly ((C.10)) and the second half with the y-derivative taken explicitly ((C.11)), hence alternating-direction. The result is a tri-diagonal system of equations, or a matrix-equation, that can be easily be solved, e.g. with the *Thomas algorithm.*

A tridiagonal matrix A has the form

$$A = \begin{bmatrix} b_1 & c_1 & 0 & . & . & 0 \\ a_2 & b_2 & c_2 & & & . \\ 0 & a_3 & b_3 & . & & . \\ . & & . & . & . & 0 \\ . & & & . & . & c_{n-1} \\ 0 & . & . & 0 & a_n & b_n \end{bmatrix} \tag{C.12}$$

with the lower diagonal $l_A = \begin{bmatrix} a_2 & a_3 & ... & a_n \end{bmatrix}$, the middle diagonal $m_A = \begin{bmatrix} b_1 & b_2 & ... & b_n \end{bmatrix}$ and the upper diagonal $u_A = \begin{bmatrix} c_1 & c_2 & ... & c_{n-1} \end{bmatrix}$. The Thomas algorithm is basically a simplified *Gaussian elimination*, that's applicable on tri-diagonal matrices. The algorithm for solving a matrix-equation of the type $Ax = b$ is as following

$$c_i' = \begin{cases} \frac{c_i}{b_i} & \text{for } i = 1 \\ \\ \frac{c_i}{b_i - a_i c_{i-1}'} & \text{for } i = 2, 3, ..., n-1 \end{cases}$$

$$d_i' = \begin{cases} \frac{d_i}{b_i} & \text{for } i = 1 \\ \\ \frac{d_i - a_i d_{i-1}'}{b_i - a_i c_{i-1}'} & \text{for } i = 2, 3, ..., n \end{cases} \tag{C.13}$$

$$x_i = \begin{cases} d_i' & \text{for } i = n \\ \\ d_i' - c_i' x_{i+1} & \text{for } i = n-1, n-2, ..., 1 \end{cases}$$

The solution is acquired in $\mathcal{O}(n)$ calculations, compared to $\mathcal{O}(n^2)$ for Gaussian elimination. The algorithm isn't stable in general, however it is for diagonally dominant matrices. For the case of a tri-diagonal matrix, as observed in (C.12), it is diagonally dominant if $|b_i| \geq |a_i| + |c_i|$ for $i = 1, 2, .., n$, where $a_1 = c_n = 0$

C.2 Matrices for reproduction purposes

The dynamic problem in Section 3.4.3 is formulated as $Ax = (B - F)x$, i.e. as a matrix-equation.

The diagonals of A is

$$u_A[61k+1:61(k+1)] = \begin{cases} \begin{bmatrix} -2\alpha & -\alpha & -\alpha & ... & -\alpha & 0 \end{bmatrix} & \text{for } k = 0,1,2,...,59 \\ \\ \begin{bmatrix} -2\alpha & -\alpha & -\alpha & ... & -\alpha \end{bmatrix} & \text{for } k = 60 \end{cases}$$

$$m_A = \begin{bmatrix} \beta & \beta & ... & \beta \end{bmatrix}$$

$$l_A[61k+1:61(k+1)] = \begin{cases} \begin{bmatrix} -\alpha & -\alpha & ... & -\alpha & -2\alpha & 0 \end{bmatrix} & \text{for } k = 0,1,2,...,59 \\ \\ \begin{bmatrix} -\alpha & -\alpha & ... & -\alpha & -2\alpha \end{bmatrix} & \text{for } k = 60 \end{cases}$$

The matrix B has the following form

$$B_{ij} = \begin{cases} \gamma & \text{for } i = j \\ \\ 2\alpha & \text{for } (i,j) = \begin{cases} (k, k+61) \\ (61(61-2)+k, 61(61-1)+k) \end{cases} \qquad k = 1,2,...,61 \\ \\ \alpha & \text{for } (i,j) = (k, k \pm 61), k = 62, 63, ..., 61(61-2) \\ \\ 0 & \text{elsewhere} \end{cases}$$