INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

The Cell Simulation

Demetrios Dardanis

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 2002

ABSTRACT

The Cell Simulation

Demetrios Dardanis

The Cell Simulation provides a software model of a biological cell, based on cell processes and substances. A study of the Cell Simulation project is made, focusing on providing new features and improving some of the existing design. The individual cell is given some indirect control over its processes through the use of process prioritization and as result is proven to be more stable, efficient and able to adapt to environmental changes. Populations of cells are created and shown to have a dependence on the process priorities and the environment factors such as external glucose levels. Through the use of populations, the software design of the older version is updated and modified to reflect the interaction between all entities in the simulation.

Table of Contents

List of Figures

List of Tables

1 Introduction

The Cell Simulation is an ongoing project in which a significant amount of work had been done. There have been three previous versions before the current one, each one addressing different aspects of cell simulation. The latest version concentrates on providing new functionality to the simulation as well as enhancements and improvements to the design of the previous versions.

1.1 Background

The simulation is a software model of the events in the life of a biological cell. A cell can have thousands of complex operations and many interdependencies and factors affecting their behaviour. In order to provide a model, it is necessary to simplify these operations and make certain assumptions about them.

In the next section of this report, a brief description of a biological cell is made: its main characteristics, structure and functionality in nature. While this has not been the main purpose of this project, it is useful to demonstrate the actual complexity that is involved in the life of real cell. Then, we go on to explore the assumptions and simplifications that were made for the purpose of this simulation. We can thus have a better idea of how close to reality our simulated cell really is and how well the software program can predict its behaviour.

2 A Biological Cell in Nature

A cell is the most basic unit of which all living systems are composed. It is the lowest level of organization of atoms and molecules where the characteristics of life begin to emerge. The living tissue of almost every organism is composed of cells.

According to the *cell theory*:

- Living matter is composed of cells
- Chemical reactions within organisms take place within cells
- Cells are born from other cells
- Cells carry within them the genetic information of the organism they belong to and pass it on to the next generation.

Very small organisms, such as bacteria, can consist of only a single cell that performs all life functions. Larger organisms are organized into groups of cells that co-operate to perform specific functions. For example, the cells that make up muscle tissue in animals have become specialized for movement while those that form bone tissue, for support. This leads to a high degree of *differentiation* among cells. The human body alone is constructed of about 200 different types of cells, each type specialized to perform specific functions. While diversity is the first remarkable fact about cells, the second (and even more remarkable fact) is their *similarity*. The conditions of life impose certain requirements on the composition and function of all living cells. Across all life forms, cells are composed of the same few kinds of molecules and atoms and their internal structure and organization is strikingly similar. This enables us to study cells by concentrating on the features that are common to all.

2.1 Cell Structure

Cells can be categorized into animal, plant or bacterial cells. The latter case is the simplest in terms of structure. Animal and plant cells are more interesting and they are basically similar in their structure. They all consist of the following essential components:

- Nucleus: The control centre of the cell where genetic information is stored, principally in terms of *deoxyribonucleic* acid (DNA). The nucleolus is also found here, which is rich in *ribonucleic* acid (RNA).
- Cell Membrane: Regulates the passage of materials into and out of the cell, and defines the cell as a separate entity from its surrounding environment.
- Cytoplasm: Does most of the work and it consists of a very elaborate system of membranes and enzymes, with specific functions. Here, a number of *organelles* can be found such as ribosomes, which consist of large RNA molecules and synthesize proteins.
- Cytoskeleton: An elaborate system of protein filaments that provide support for the cell and help it to maintain its shape.

In the case of plant cells, there is also the *Cell Wall* found outside the cell membrane. It controls both the rate and the direction of cellular growth in plants.

There are two distinct types of cells, prokaryotes and eukaryotes, which differ mostly in the way their genetic material is organized. Prokaryotes are much simpler in their structure and they were the first cells to appear in living organisms on our planet. Here, the DNA is a large single molecule that carries all genetic information. Modern day bacteria are similar to the first prokaryotic cells that appeared on earth. Eukaryotes

evolved after almost two billion years and they are much more complex in their structure. The components described above are all characteristic of eukaryotic cells. Here, the DNA is associated with proteins in complex structures known as chromosomes and it appears within the nucleus of the cell. All multicellular organisms are eukaryotes.

2.2 Energy Flow in the Cell

The living cell contains an enormous variety of organic molecules in constant and dynamic interaction. Most of these are classified as protein molecules. Enzymes are specific types of proteins that facilitate the metabolic reactions taking place within the cell. In the absence of enzymes, the metabolic processes in the cell do not take place at a rate that is high enough to maintain life and the cell dies.

Inside a living system, energy is supplied in the form of a single molecule, *adenosine triphosphate*, or ATP. Glucose and other carbohydrates also supply energy but in a stored form. They cannot be used directly by the biochemical processes of the cell, as is the case for ATP. Instead, they are stored for future use or may be transferred from cell to cell or between organisms. Carbohydrates are ultimately derived from photosynthesis, which takes place in plant cells. They are converted, though the use of enzymes, to energy (ATP) or into other kinds of molecules within the plant cells or within the animal cells that have ingested plant materials. Simply stated, for most chemical reactions to take place in the cell, ATP must be present and in order for the production of ATP, the cell must have ingested carbohydrates and must have adequate amounts of enzymes for conversion.

The genes in the DNA chromosomes of the cell control the synthesis of these enzymes. Genes contain information about many more enzymes than the cell uses at any given time. However, only the enzymes currently required are produced. This is how the cell can adapt to changes in the environment: if the conditions change, the genes will modify the production of enzymes. This in turn, will affect the type or amount of substances that are synthesized by the cell. For example, due to a change in environment, the cell may start to produce a different type of molecule or may cease to produce one that is no longer required. Based on this principle, differentiation of cells (described earlier) takes places in higher-level organisms.

2.3 Environment and Communication

All organisms are composed of organs with specific functions. Each organ is in turn made up of a specific type of tissue of cells. In order to function properly as a unit, the cells that form a tissue must communicate with each other.

Cells send each other chemical messages. These are signals that a cell is capable of sending and receiving. For example, healthy human cells placed in isolation, grow and move towards each other until they come into contact. Also, they multiply until every cell is in contact with another. At that point, reproduction stops, probably in response to some signal. There are cases where these signals do not seem to be present. In the case of cancer cells, for example, reproduction continues until all nutrients in the environment have been depleted.

When the cells are in contact, there are different types of junctions that form among them. Their function is to hold the cells of a tissue together but also to allow the exchange of nutrients or other materials, as well as electrical signals.

All cells move within their environment. For example, a cell may move towards a light source or as a result of division. There is also movement of the contents and substances taking place within the cell. Two principal types of protein are involved in movement and they are actin and myosin. These are known as "muscle" proteins but they can be found in almost all types of cells, including plant cells.

2.4 Reproduction

A cell grows by consuming materials from its environment and converting them into the energy or substances it needs. When it has reached a critical size and if the state of all its components allows, it will divide into two daughter cells. The growth cycle then begins again.

The daughter cells will contain about half of the constituents of the parent cell, and more importantly their genetic information, present as DNA, will be an exact replica of that of the parent. The process of replicating DNA is quite simple in prokaryotes however it is much more complex in eukaryotes due to the fact that their DNA is attached to proteins in the form of chromosomes. *Mitosis* is an elaborate process by which each daughter cell ends up with its own complete set of chromosomes. The remainder of cell contents, mostly contained in the cytoplasm, are about equally divided to the two children.

2.5 The Genetic Code and Protein Synthesis

The DNA molecule does not replicate by itself. For this, it requires the presence of enzymes known as DNA polymerases. More importantly, the proteins present in every living cell are synthesized based on information provided by DNA. Proteins are encoded sequences of amino acids (a total of 20). Also, DNA can be broken down into four distinct *nucleotide* molecules. These molecules (also expressed as a sequence of *codons*) provide the code for the basis of the production of protein.

There are four nucleotides, and twenty biologically important amino acids. Therefore, three nucleotides in sequence are required to adequately specify each amino acid ($4^3 = 64$ possible combinations). But how is the information stored in the DNA in terms of nucleotide sequences (or *codon triplets*) translated into sequences of amino acids (proteins)? This is a two-step process:

- Transcription. A specific type of RNA (messenger RNA) aligns itself with the DNA molecule and essentially copies the codon triplets. This process is controlled by an enzyme known as RNAPolymerase, which acts as a catalyst.
- Translation. Another type of RNA (transfer RNA) translates these sequences into a sequence of amino acids - a protein.

Each amino acid can be specified by more that one codon triplet. Of the 64 possible combinations, 61 have been translated to amino acid sequences whereas the remaining three are used as signals to stop protein synthesis.

Changes in the sequences or number of nucleotides in the DNA may result in changes in the corresponding amino acids after translation. This is the definition of a mutation.

3 The Simulated Cell

After having seen an – admittedly – very brief overview of a biological cell in nature we can go on to study the cell as it is represented in the simulation. Before focusing our attention on the project at hand, we may explore some of the reasons why a project of this kind is interesting and what kind of applications it may have.

3.1 Motivation

There is currently a significant amount of ongoing research in the field of biochemical modelling and simulation, and specifically cell simulation. The scope of most of these projects is quite ambitious: by taking advantage of the computational power of computers, to be able to master the complexity of biological systems. Some specific goals are:

- To predict the changing behaviour of living cells,
- To study the minimal conditions of survival of a particular cell,
- To enrich the knowledge about cellular processes, which may still not be completely understood experimentally, etc.

Many applications are found in genome engineering and they deal with simulating the genetic information to find a minimal set of genes for a living cell.

Using all the information that is known about a specific biological cell, a simulation can be built that models the cell's behaviour. By comparing the results from the simulation, to the behaviour of the real cell in biological experiments, we can gain useful information about other factors that may be influencing the cell. For example, the function of certain genes for a cell may still be unknown. A simulation of the cell based on all the

information we do know about it will provide a model, which when compared to the behaviour of the real cell may shed some light on the function of the unknown genes. The model can be refined based on the results and then re-used to better predict the behaviour.

Other applications have dealt with providing a model of the metabolic processes of the cell in order to enable scientists to chemically define a growth medium for the cell. Much of the metabolic requirements of a cell may still be unknown and the cell may require a growth medium (a culture), which has been determined empirically but there is no complete chemical definition for it. A simulation can model the gene sequence of the cell, as well as the components required for the metabolic processes of the cell (proteins, enzymes etc) and thus give a chemical definition of the environment in which the cell can survive and grow.

There are also graphical applications that model three-dimensional cellular architecture and are used in the field of computational biology to model specific physiological cell processes (for example nuclear envelope breakdown during mitosis).

For our project the scope is, of course, much more limited. We aim to provide a basis for studying cell behaviour as well as the effects of environmental changes. There are a number of assumptions that need to be made in order to simplify the cell functionality so that the behaviour can be simulated. At the same time we wish the simulation to be close enough to reality so it can be more interesting and provide a good model.

First, it must be determined which aspects in the life of a biological cell are actually relevant to the model and should be simulated. Our decision is based on whether a

certain aspect plays a decisive role in the life of a biological cell. The factors that were deemed to be most important and thus need to be modelled are:

- The Type of Cell: determines how the cellular contents are organized.
- The Cell Environment: a very important factor since all the nutrients necessary for life are found here.
- Genetic Information: according to the cell theory this is one of the identifying characteristics of cells, and thus must be modelled.
- The Cell Processes: what the cell actually does, of course, must also be modelled.

The above factors are described in more detail in the next few sections. Other aspects of biological cells (such as a specific cell process for example) can be added to the simulation as enhancements to the model. Note that for the remainder of this report the term "cell" is used to represent the simulated cell, unless a distinction is made.

3.2 Cell Classification

The first assumption that needs to be made for the simulation is regarding the type of biological cell to be simulated. We have seen that there is great diversity to be found in nature depending on the functionality and purpose of a biological cell. The simulation takes advantage of the similarity that exists in nature. Our cell can be of any of the types mentioned in the previous section. Based on the way genetic information is organized in the cell (DNA representation - to be discussed later) we can say that this cell belongs in the eukaryotes category and it is part of a multicellular organism. Also since, as we shall see, the cell structure is enclosed by a cell wall, we may conclude that this cell belongs to a plant organism. In any case, these assumptions do not change any of the fundamental

principles used in the simulation. The simulation can be applied equally as well to animal or plant cells.

3.3 Cell Environment

Biological cells of higher-level organisms live and may move about in three-dimensional space. They obtain nutrients from their environment and they also co-exist and communicate with each other. The simulated environment is much more simple. Our cell exists in a two-dimensional, 10X10 grid of squares. Opposite edges of the grid are connected, similar to a torus. The simulation runs in steps that are entered by the user and the cell moves only to adjacent squares, one at each step.

In the beginning of the simulation, a certain amount of glucose is randomly distributed on the grid. The cell can move to the adjacent square with the highest concentration of glucose. We do not simulate any other nutrients in the environment. Glucose (and carbohydrates in general) is the main form of stored energy used by the cell.

As far as inter-cellular communication is concerned, this is also not explicitly simulated. After a number of cell divisions, the grid starts to become populated, as we shall see, and it is possible to have more than one cell in any of the squares. We can visualize the grid as being part of the tissue of a specific organ, where all cells are of the same type and perform the same functions.

If many cells consume glucose in the same square, this may cause another cell in the square to move to another location with a higher glucose content. There is no external message sent explicitly to the cell to move but due to the actions of the other cells it may

have to move. So, the cells do influence each other but there is no other way that they communicate.

3.4 Genetic Information

The way that genetic information is stored in the cell is simpler than that of a real cell. The basic principle is the same however: protein synthesis is controlled by the information stored in the DNA molecule.

The DNA molecule appears as the genome structure in the cell. The genome is actually represented by a sequence of genes. A gene is a string of letters **a** to **f** and symbols < and >. Each gene controls the synthesis of a protein.

For example, gene **b<afed>** is correct, syntactically. There is no DNA replication or translation that takes place in the program, in order to determine the amino acids represented in the gene. In fact, each letter between the < and > represents by itself an amino acid. It is assumed that each protein is a sequence of four amino acids. So, if the gene **b<afed>** matches with a protein sequence (ie there is a protein with amino acid sequence **afed**) then that protein will be synthesized. If the match is exact we have an effectiveness factor of 1. Of course, if one or more letters do not match, the effectiveness factors drop.

Since there are seven proteins that are synthesized by the cell, we can say that the "normal" genome should contain a sequence of seven genes, each one providing a perfect match with a distinct protein. For the cell with this "normal" genome, it implies that every time a protein is synthesized, this is done at the maximum effectiveness.

The letter(s) appearing in front of the <, are regulators (letter **b** in our example). A regulator acts as an inhibitor during the synthesis of a protein.

3.5 Cell Processes and Substances

In the beginning of its life, a cell receives specific quantities of substances (proteins) necessary for its survival. As the cell grows, these substances are consumed and new ones produced continuously. At any point in its life the cell is kept busy doing a number of processes.

In a real cell, there are thousands of processes happening at all times and numerous substances that take part in them. In our simplified model we consider only six distinct processes and nine substances (see figure 1). All substances except glucose are produced in the cell. All substances are consumed by the cell. A typical cell is expected to:

- Ingest. The goal of ingestion is to provide the cell with enough glucose, which serves as food (stored energy). The cell does not produce glucose but rather it ingests glucose from its environment through the cell wall. Ingestion is facilitated by transportase, however if transportase is very low, glucose may still be ingested but at a slower (osmotic) rate.
- Generate. This process will produce the units of energy required by the cell to perform almost all of its life functions. The energy is in the form of ATP. The Generate process requires both sufficient glucose and energase to be present in the cell in order to produce ATP.
- Synthesize. This process is extremely important for the cell to function properly and retain a balance of substances. Synthesis produces transportase used for

ingestion, energase needed for (energy) generation, actin required by the Build process, cyclin and DNAPolymerase both necessary for the cell to reproduce and myosin, which the cell needs in order to move. For the cell to synthesize proteins it needs to consume RNAPolymerase, required for DNA replication and translation. Without RNAPolymerase it is not possible to have protein synthesis because the genetic information cannot be translated into amino acids. This substance is also itself a product of the Synthesize process. In the simulation, the substance to be synthesized at a particular step is chosen at random. Therefore, the Synthesize process does not guarantee that the cell will be supplied with a substance that it actually needs at that step.

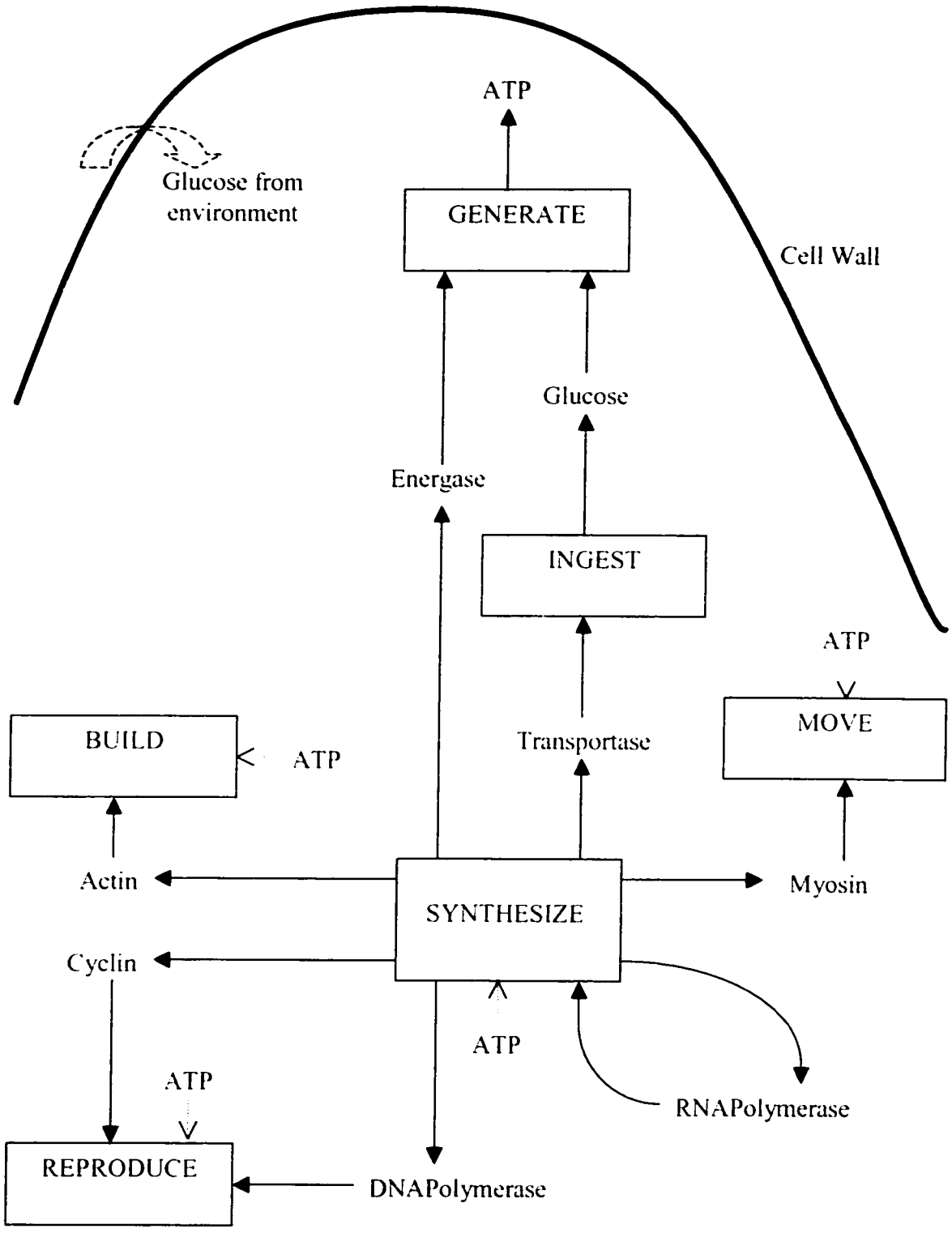

Figure 1: The Cell Processes and Substances

- Move. It is crucial that the cell be able to change its position in the grid and move towards areas (squares) of higher glucose concentration. If the cell does not move, it will eventually starve. Myosin is the protein that provides the contractile ability in muscle tissues and therefore enables the cell to move. The Move process requires and consumes myosin.
- Build. This process requires actin to be present in the cell. Closely related to the Move process, the Build process ensures the structural integrity of the cell. The presence of adequate amount of actin ensures that the cytoskeleton is stable enough for the cell to move.
- Reproduce. A mature cell may reproduce provided that there is enough DNAPolymerase and cyclin. The result is two cells with the identical genetic information. When a cell reproduces, its substances are halved and given to the child cell. In the case of DNAPolymerase and cyclin however, they are set to zero, for both cells. This ensures that the cell will not reproduce too quickly because this may be detrimental to it.

All of the cell processes except for Ingest and Generate consume energy in the form of ATP as shown in figure 1 by the dotted arrows.

In the simulation, substances exist in discrete arbitrary *units* that are not meant to represent quantities inside a real cell. For example, we can say that the cell contains 56 units of glucose. This is not meant to correspond to the amount of glucose that a real cell may contain.

When a substance is consumed, it simply disappears and its number of units decreased accordingly. For example, one unit of energase is consumed when the Generate process

takes place. The number of units of energase for the cell is decreased by one. No further degradation or excretion is modelled.

We can see that although the model simulates very few of the cellular processes and substances, there is a very delicate balance among them. Say, if for some reason the cell is not synthesizing transportase, the amount of glucose ingested dramatically drops. This results in a drop in the amount of ATP being produced, and since most processes require ATP, all functions of the cell will be affected and it will eventually die. It is desirable that the cell can have some indirect control over its processes.

4 Previous Versions of the Cell Simulation

We have seen the characteristics of a real biological cell as well as those of the simulated cell. In all versions of the simulation program (including the latest one), the cell characteristics have remained the same. A cell executes the same processes, using the same substances. It exists in the environment as described and the manipulation of its genetic information has not changed. Before studying the new features of the latest version, an overview of what was done previously would be useful.

4.1 User Interface

The user interface has remained virtually unchanged in the first three versions of the simulation. The information presented to the user is in the form of a graphical representation of the substances within the cell. This is best demonstrated by figure 2. Each curve has a distinct colour and corresponds to a substance on the left-hand side. The X-axis represents the number of steps that the simulation has executed. The Y-axis represents amount of a substance (number of units). So, in the figure we can see the amounts of any substance in the cell, at any step between 300 and 400. The sudden drop around step 315 indicates that the cell has just divided; all its contents are halved and DNAPolymerase and cyclin are set to zero until the cell starts to synthesize them again. The death of a cell is usually detected by the fact that the amounts of substances have stopped changing and we have the appearance of flat lines on the graph. This of course means that the cell is in a state in which it can no longer produce or consume anything and it is presumed dead.

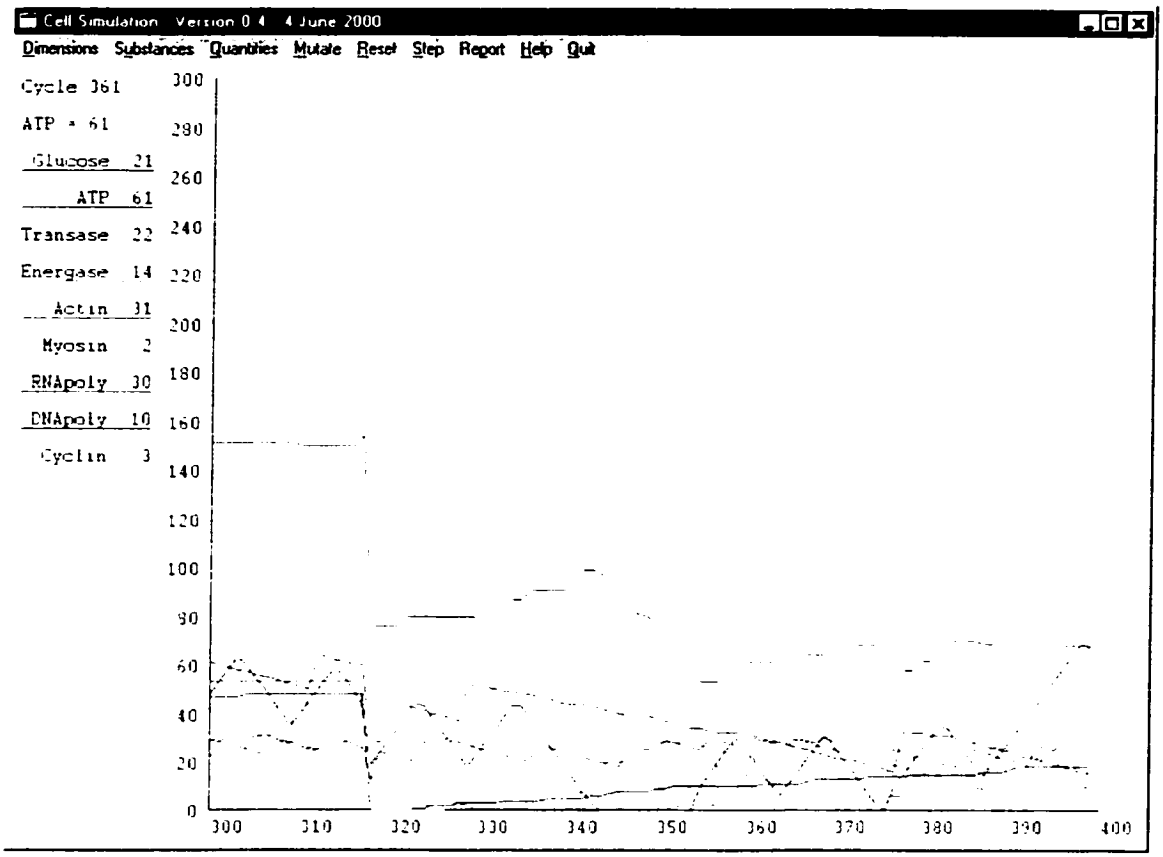

Figure 2 The Cell Simulation GUI (old version)

In the program, death is detected when it is highly unlikely or impossible that the cell will synthesize anything:

- RNAPolymerase is zero so no synthesis can happen, or
- Energase is zero and ATP is less than the gene length (five). It is unlikely that synthesis can take place in this state.

4.1.1 Menu Options

In brief, the menu has the following options:

- Dimensions. The user can modify the number of steps shown in the graph (X-axis) or the maximum amount of units shown (Y-axis).

- Substances. The characteristics of the substances such as Regulator factors and colours can be modified at run-time, for the current session only.
- Quantities. The amount of each substance can also be changed while the program is running. The numbers are expressed in units.
- Mutate. The genome can be modified to see the effect of mutations on the cell.
- Reset. All substances are set to their default values and all events are erased.
- Step. When clicked by the user it will run the simulation for a number of steps equal to the dimension setting for the X-axis.
- Report. The user can select which events they are interested in and have them printed to a file or on the screen.
- Help. A quick user guide reference is presented.
- Quit. Exits the program.

The menu button that actually does most of the processing in the simulation is, of course, Step. What happens when the user clicks on Step is described next.

4.2 Process Execution

Every time the Step button is clicked, it triggers the processing to take place in the code. The cell is updated for a number of times equal to the number of steps displayed. During each update, the cell is given the opportunity to execute all of the six cellular processes described earlier. Before executing a process, there is a check to see if it is actually feasible at this time. For example, if there is enough glucose and energase present, then the Generate process will be run. If the conditions allow it, all processes are executed within a step.

4.3 What Was Achieved

The goal of the Cell Simulation was to provide a good model of the very complex mechanisms of the behaviour of a single cell. In this, it was successful: it provided a software simulation of all of the aspects of biological cells that were deemed as relevant to a cell model (see section 3.1). Although not very intuitive when first using it, the user did have a good representation of the cell in terms of the graphs of substances. Cell functionality was well represented in terms of the cellular processes. Even in the new version, the basic life functions that a cell performs have remained unchanged despite the modifications/improvements that were made.

By simplifying some very complex operations (such as DNA replication and translation) the older versions gave us a good model without straying away too much from the way a real cell functions. Although DNA replication and translation are not explicitly modelled, protein synthesis is still controlled by the information stored in the genes.

The project has also been improving in each version. For example, the effectiveness of genes was dramatically improved from version 0.1 to 0.2.

4.4 Limitations

There are a few limitations that one may notice when using the simulation.

First, it appears that the cell is not very stable because it often dies very quickly and quite suddenly in most cases. Of course, this depends on the substances supplied in the beginning, and there are many cases where the cell does survive for a few thousand

cycles. However, overall, it does not live very long and sometimes dies soon after dividing.

The movement of the cell on the grid is kept track of, in the code. It would be very helpful to have a visual representation of the grid and the position of the cell on it. Just by looking at the graph of substances, it is not obvious to the user if the cell is actually moving in the environment. The only way to know is by monitoring the amount of myosin, and whenever there is a drop we know that movement has taken place (because movement consumes myosin). However, we still have no way of knowing the current location on the grid or how the external glucose is distributed on the grid.

In the event of cell division, the only effect on the simulation is that the cell becomes smaller. It does not actually divide in two cells. Many times, as we saw, division can be traumatic and the cell dies soon after. At this point we are forced to restart the simulation since there are no actual child cells that we can monitor.

We may identify a specific set of limitations in the old version that became the goals of the latest version. Namely, these are:

1. Cell Stability
2. Modelling of Cell Division
3. Visual Representation of Cell Movement

The latest version of the cell simulation, attempts to address these limitations, and this is what we will explore in the rest of the report.

5 Process Priorities

To address our first goal of improving the stability of the cell, the new version incorporates the use of process priorities. Previously, each process was given an opportunity to run at every cycle. This seemed to work fine, however it did not take into account the actual "needs" that the cell may have at certain times. Since all processes would run unconditionally, the cell could spend time trying to execute processes that it did not need to do at the time. A better way to do this would be to use what is known about the current state of the cell and decide which processes to execute. This would give the cell some degree of *autonomy* since it can have some influence over its own process regulation.

5.1 The Idea

The cell should be able to improve its own stability by implementing some sort of self-regulated feedback in the simulation. It was proposed to select only those processes deemed necessary and execute them at each cycle. The necessity of a process is expressed by its priority. How do we decide the priority of a process?

To do this we look at the amounts of the various proteins in the cell and define a relationship between the amount of a certain protein and the process that produces it. For example, referring to figure 1, the amount of ATP in the cell is related to the priority of the Generate process: if ATP is decreasing, the priority of Generate must increase. To express the priority as a function of the amount of a protein we use the exponential function as outlined in [1]. Some processes do not produce any proteins. Then, we must

find another way to calculate the priority. Let's have a look at how the priority of each process is calculated, based on what the program actually does.

5.2 Calculation of Priorities

For any process p we use variations of the relation:

$$\text{Priority}_p = e^{-\lambda g}$$

Where g represents the amount of a substance and λ is a constant determined experimentally, different for each process. The priority can range from 0 to 1. We make the following calculations for each process:

- For Ingest we use the amount of glucose present in the cell as g. If it drops, the priority is increased.
- For Generate the amount of ATP is used, as above.
- The Move process does not produce any protein but it is clearly dependent on the amount of glucose in the cell and in its environment. So, we use an average value of the two as g in the above formula. This way, if the amount of glucose in the cell is very low, the priority of Move will be high, except if there is a large amount of glucose in the square where the cell is located. In the latter case, movement is discouraged so the cell can consume the glucose in its own square before moving to another square.
- The Reproduce process also does not produce any proteins; instead it will result in the amounts of all proteins being halved. The cell should reproduce only if other priorities are not too high and the average value of the amounts of cyclin and DNAPolymerase is high enough. So, we use this average value as g in the

formula. If it is equal to zero, we want the priority to also be zero and to increase as g increases. We use $(1 - e^{-\lambda g})$ to calculate the priority of the Reproduce process.

- The Build process is not very critical for the cell since it can survive even if it does not take place. However, Build consumes actin, which is required for cell movement, therefore we must ensure that this process only executes if the amount of actin is high enough. To calculate priority we use $(1 - e^{-\lambda g})$ with the amount of actin as g.
- The Synthesize process is like the powerhouse of the cell processes. Everything depends on proteins being synthesized when they are needed most. Except for energase and RNAPolymerase all other proteins have already been accounted for in the other priority calculations. So, synthesis must ensure that we do not run out of either one of these two substances. We can say that the priority of the Synthesize process should depend on the smaller of the two quantities. If, for example, energase is approaching zero and RNAPolymerase is some high value, the value of g is set to the amount of energase so that the priority is increased and synthesis is encouraged to take place. In the rare case that the two values are equal we use their average (actually either one will do).

The number of processes to execute at each cycle is set in the program and can be easily modified. The constant λ used in the calculation of priorities was determined experimentally for each process. Different values were attempted such that the resulting priorities would give a good range of values from 0 to 1. See the Appendix A for the

values of λ used. This constant can also be modified in the program to observe its effect on the priority of a process.

Fortunately, in our case, the constant λ is the only parameter that is determined experimentally. In many other simulation models there are a number of parameters that are used and would require a much more sophisticated method of determining their values. It would be much more challenging (if at all possible) to experimentally determine four or five variable parameters in the priority calculation.

5.3 User Interface

The user interface did not change dramatically once priorities were added to the simulation. The bigger changes took place in the actual processing and they are described in a later section. The GUI interface now has one new button in the menu called *Priorities* with the option of using or not using priorities. From the point of view of the user however, there are some very notable differences as the program executes.

5.4 Effects of Using Process Priorities

When the processes of the cell are prioritized it appears as if the cell is more careful about selecting its next process to execute. As expected, it executes processes because they are required at the time and not just because it is able to do them, as was the case in previous versions.

Looking at the graphical representation of a cell using priorities (see figure 3, described in detail in the section 6), we notice that there is kind of a cyclical appearance of the substance curves. In other words, it appears as if the quantity of each substance ranges

between a maximum and minimum value. These values correspond respectively to the minimum and maximum priority of the producing process and they are a direct result of the self-regulated feedback of the cell. To use an example, the maximum amount of ATP in the graph, is also the point at which the priority of the Generate process is at its lowest. If there is a high value of ATP in the cell, it is not really required to generate more at that time. As the ATP gets consumed the priority will increase. This range of values (maximum, minimum ATP amount in our example) depends on the constant λ used in the priority calculation.

5.4.1 Lifespan

As a result of prioritization, the lifespan of the cell is increased dramatically. Doing some trial runs of the simulation for a single cell, with no priorities, reveals that the average lifespan rarely exceeds 4,000 cycles and in many cases the cell dies much earlier. With priorities however, the cell is much more stable and is easily able to survive up to 40,000 or more cycles. In fact, as will be shown later, in almost all cases, the death of a *prioritized* cell, occurs when it is sharing the grid with other cells and has to compete for the consumption of glucose.

5.4.2 Reproduction

Another noticeable difference of using priorities is that the cell does not reproduce as often. This may be partly the reason why its lifespan is so much longer.

In the previous version, cell reproduction takes place whenever possible, so we have a much higher rate of cell division but both the parent and child cell are much more unstable. In many cases, the cell dies right after reproduction.

As mentioned earlier, the priority assigned to Reproduction ensures that it takes place only when there are no other higher priority processes and the amounts of cyclin and DNAPolymerase are high enough. The cell divides only if it is safe to do so and thus, Reproduction is much less traumatic than before.

5.4.3 Movement

Similar to the effects on the Reproduction process, the effect of using priorities on the movement of the cell is that the cell moves when it has to and not just whenever it is able to do so. Previously, the Move process did not take into account the amount of external glucose in the square occupied by the cell: the cell could move even if there was still a large quantity of un-ingested glucose in its square (and in fact, most of the time it did). Since the priority calculation takes into account the external glucose, we can ensure that the cell will not move before it has ingested the external glucose. This will become more obvious when we discuss the visual representation of the cell in the grid. We will see that a cell that uses priorities leaves behind a trail of empty (white) squares, indicating more complete ingestion of the external glucose.

5.4.4 Number of Processes

It is interesting to point out the effect of changing the number of processes to execute per cycle. This number can be from one (only the highest priority process executes at each cycle) to six (this is equivalent to using no priorities).

When this number is set to one, the cell struggles to survive and it does not make it. It is not able to move and can only ingest or synthesize something for a few cycles before dying. Setting this number to two processes per cycle, the cell can survive quite nicely, it

lives a long life and it can move about on the grid, however, it never reproduces. The priority of the Reproduce process is never high enough.

Using three processes per cycle, the cell becomes very stable (as was our goal, outlined in section 4.4) and it can perform any process that is required for survival, including Reproduce and Build. This is the default setting, and it has been used for all results in this report. Higher settings tend to approach the no-priorities case.

An interesting discussion can be made regarding the number of processes to execute per cycle in such a system. It is clear that self-directed prioritization of cellular processes helps to make the cell more stable, however, is there an "ideal" number of processes to execute per cycle? This depends on many factors one of them being the total number of processes being simulated. If, for example, there are hundreds of processes modelled in the system, it is very unlikely that selecting only the three highest priority ones would provide a good model: the cell would probably die very soon. Unless, of course, only three of the processes are necessary for survival. In that case, the cell would always select the same three processes. The other processes may never have a priority that is high enough. There is a second factor to consider: the priority calculation must ensure that all processes have a good range of priority values. Otherwise, even if the number of processes per cycle is quite high, there may be processes that never get selected to execute.

In general, we can say that for a good model, the number of processes per cycle should be the smallest number that will allow all processes to have a chance to execute when their priority is high enough.

6 A Population of Cells

To address our goal of providing a model for Cell Division (second goal in section 4.4) we need to consider a population of cells. Up to this point in the Cell Simulation we have been concerned with only a single cell. However, now that the cell is stable enough, it is interesting to see how it can reproduce, resulting in child cells, which occupy the same environment and may also divide at later stages. We can study how cells populate the grid, and how the cells can – indirectly - have an effect on each other by sharing a common environment.

6.1 The User Interface

The main change in the GUI interface for population is the appearance of the Grid (see figure 3 for a pictorial view of the GUI as it appears in the new version). The priorities function as before. We note that in the figure, the cell displayed has just divided. Contrast this with the cell division shown in figure 2 for a cell that does not use priorities. We notice how much higher the levels of Cyclin and DNAPolymerase are before reproduction actually happens.

For a population, it becomes necessary to have a way to monitor the substances and movement of all cells on the grid. Thus we can also achieve our third goal of section 4.4 (visual representation of cell movement). The user now has the option of viewing the grid by clicking on the *Cell Grid* button. A new window appears encompassing a 10X10 grid, which represents the habitat of the cell(s). A square on the grid may have a number on it; this indicates how many cells are occupying that square at the time.

6.1.1 Cell Information

On the left-hand side we have some new information provided to the user. The *Population* field shows how many cells are currently occupying the grid.

The graphical representation has remained unchanged. However, we can now monitor the substances for any of the living cells on the grid. By clicking on a square containing a number, we select the cell whose graph we want to be displayed. If there are more that one cells in that square, they can all be viewed by clicking repeatedly. The *Cell* field

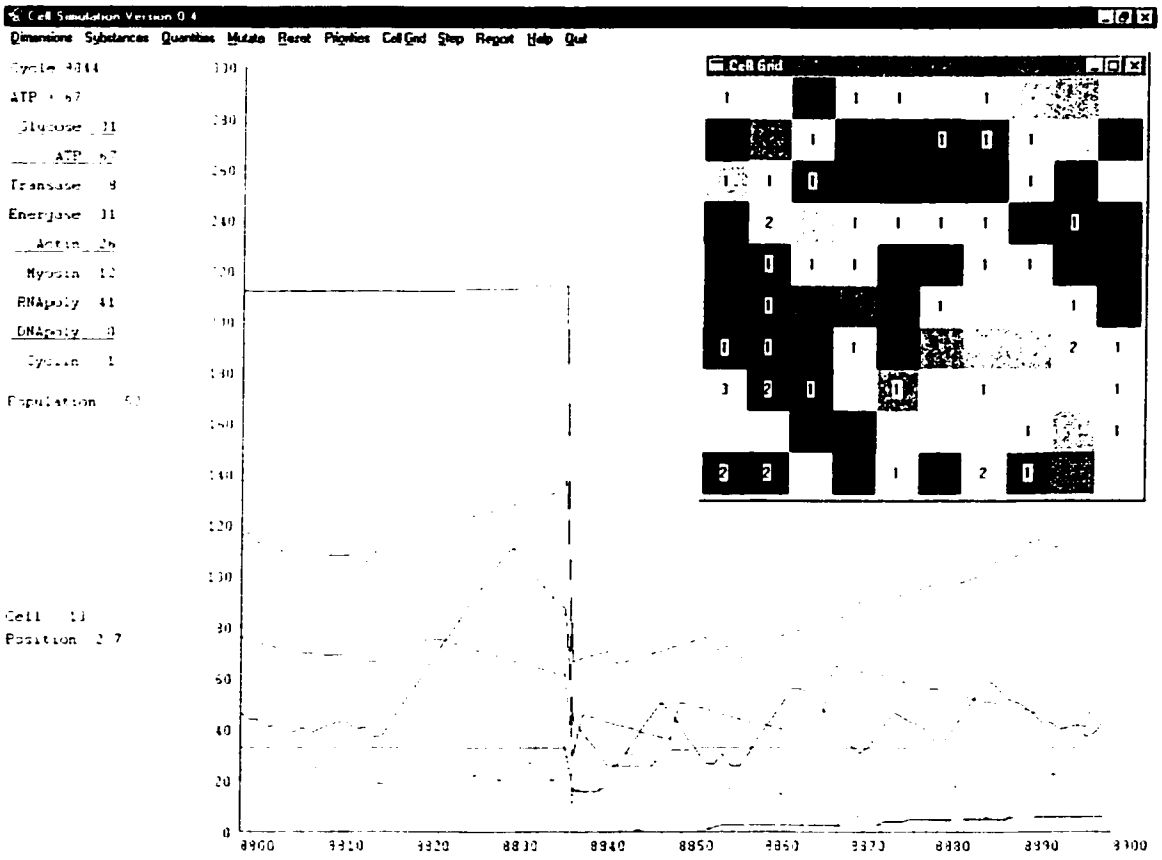

Figure 3: The New Cell Simulation GUI

shows the cell that is currently being viewed. To distinguish among them, each cell is assigned a name at the time of birth. The name is actually a number, which is made up from the name of the parent followed by the order of birth of the cell. The first cell is always cell 1. The second child of cell 1 is 12. The cell in the figure is 13, so it's the third child of cell 1. Its first child will be 131 and so on. Consequently, from the length of the cell name we can tell the generation the cell belongs to.

As cells move on the grid it is difficult to distinguish them and follow the path of a single cell. However, the *Position* field shows where the displayed cell is located on the grid (starting from 0, at the top left corner).

The user may lower the number of steps executed and displayed on the graph (by using the Dimensions button). This helps to get a better picture of how the cells move about at each interval.

6.1.2 Amount of Glucose

Another feature of the grid is the amount of glucose located in each square. This represents the external glucose available for the cell to ingest through the cell wall. Squares with a darker shade contain more glucose and vice-versa. If the number of steps is lowered, the user will notice that the shade of each occupied square gradually becomes lighter, indicating consumption of glucose by the cell(s). When priorities are used, cells move only once the external glucose is exhausted, leaving a trail of empty (white) squares behind. Of course, these get replenished eventually, as glucose is randomly re-distributed on the environment.

6.2 Beginning to Populate

Each cell will reproduce when the conditions are favourable. If priorities are used, these conditions are more restrictive but they do ensure that the parent and child cell will have a better chance for survival. Otherwise, reproduction takes place much sooner and more frequently, resulting often in the death of the parent or the child cell.

The population that does not use priorities will usually grow faster, but it will require more generations of cells, since the cells often die right after dividing. This is evident if we have a look at the Cell name shown in the Cell Simulation window. The generation of a cell is equal to the length of its name. As the population grows, we notice that we have only very long Cell names in the grid, since all the "old" cells have died.

A population that does use priorities however, will grow a bit slower initially but almost all cells will survive until there starts to be a shortage of glucose. It is not unusual for the very first cell (Cell 1) to still be alive when the grid has filled up.

6.3 Stabilizing

The cell population will not grow indefinitely of course, and indeed after a certain number of cycles we notice that it is no longer growing but it usually starts to fluctuate about some average value. This value depends on a number of factors the most important ones being the initial amount of glucose distributed on the grid and the priority settings. The cell population goes through periods of many new births taking place followed by periods when many cells seem to starve and die. This is causing the fluctuations

6.3.1 Cell Deaths in a Population

A minor modification was done in the determination of death for a cell. Previously we mentioned that in some cases, the substances would appear as flat lines in the graphical representation. The cell was at a state where death had not been detected, but it was essentially dead, since it could not do anything. Most of the time the cause of this were the following conditions occurring simultaneously:

- A very low value of ATP. The cell cannot synthesize any protein because the amount of ATP is less that the length of the required gene.
- Zero glucose. The cell is not able to generate any new ATP.
- Zero transportase. The cell cannot obtain any new glucose from the environment at a pace that is fast enough (only through osmosis).

For a population, it is crucial that cells that are no longer active are presumed dead and promptly removed from the grid. Otherwise there is a risk of having many cells remaining on the grid but not being able to actually do anything useful. We would have the occurrence of many cells with a graph of flat lines.

The above state was identified as a *Starvation* condition and it has been added to the reasons for cell death.

6.4 Consumption of Glucose

The first few times the simulation was run using populations we noticed a curious event: once we had reached about 100 cells (roughly 1 cell per square), instead of reaching some stable condition as expected, the population increase seemed to accelerate! Cells multiplied faster and also by looking at the graph, they were much heavier and did not

move to adjacent squares. Each cell had huge amounts of ATP and especially glucose. The large amount of glucose explained why the cells did not need to move but where was all the glucose coming from?

After some investigation the question was answered. In the old versions of the Cell Simulation, as soon as the cell ingested some glucose from the environment, that same amount of glucose was distributed randomly into the environment again. The problem is that now, the total amount of glucose in the environment plus that of the cell is actually higher than the amount of glucose we started out with. For example, if the grid initially contains 4,000 units of glucose and the cell at a certain point ingests 5 units, it will also distribute 5 units somewhere on the grid. Now the external environment still has 4,000 units but the total amount in the whole system is 4,005 units. This may not have caused a problem for one cell or even for small populations, but it was devastating once we had growing numbers of cells. If there are 100 cells on the grid and say half of them ingest 5 units of glucose each during a step, then we would have a total of 4,250 units in the whole system. The external glucose remains constant at 4,000 units, but now most of the squares on the grid are occupied, the cells contain their own internal glucose and they are basically ingesting more glucose and then "throwing" around the same amount to each other.

For one or a few cells on the grid, we do not notice this effect because chances are that the re-distributed glucose will randomly go to a square which is not occupied and it will remain unused for a while. For 50 cells or more, the glucose will probably go to a square occupied by a cell and will be ingested and re-distributed right away.

What needs to be considered is having an equilibrium for the whole system: glucose in the environment plus glucose inside all living cells should not exceed the initial amount. This is a more realistic model and gives better results.

What the program does now is instead of distributing the glucose as soon as it is ingested, each cell distributes glucose back to the environment in two cases:

- After it has been converted to ATP. If n units get converted to ATP during the Generate process, then n units will be given back to the environment.
- After the death of the cell. If the cell contains n units of glucose at its time of death, then this will be distributed. It is possible not to return the glucose of dying cells to the grid and still have an interesting model. In that case, the external glucose does not get replenished at a rate that is fast enough. The population grows, then some cells start to starve and die without returning their not-converted glucose. Consequently, the living cells start to starve until the whole population disappears after some cycles.

Looking at the user interface, we notice that for higher populations, almost all squares appear white (no external glucose). This is expected since higher populations should deplete the nutrients in the environment. If communication among cells was modelled in the simulation, this would be the point where cells could send signals to stop multiplying. This is what happens in normal healthy human cells, as discussed earlier.

6.5 Characteristics of Population

It is interesting to see how different factors can influence the cell population, especially the way the population grows and how it reaches an average value. By varying the

amount of initial glucose distributed on the grid we get the following results for maximum population:

	Priorities			No Priorities		
Init.Gluc	Max.Pop.	Ave.Pop.	Ave.Gluc.	Max.Pop.	Ave.Pop.	Ave.Gluc.
1000	1	-	-	1	-	-
2000	26	17	118	2	-	-
3000	61	40	75	13	4	750
4000	98	65	61	24	10	400
5000	135	96	52	38	19	263
10000	220	186	53	139	82	121
20000	428	392	51	223	178	112

Table 1: Effect of Initial Glucose on Population

In the table above, maximum population is the highest number of cells recorded while running the simulation, at a specific amount of initial glucose. The average population is the number around which the population seemed to fluctuate, after running the program for many cycles (>40,000). Of course, in the cases where there were no cells alive after a few cycles, there is no average value. Finally, the average glucose is the amount of glucose per cell.

As expected, the cells using priorities are much more efficient and able to grow to larger populations. They require smaller amounts of glucose to survive. The difference is more dramatic at smaller amounts of glucose where the no-priority cells are struggling to survive. At an amount of 20,000 units of glucose we can have almost 2 cells occupying each square and more than double that if we use priorities.

The graph in figure 4 demonstrates the pace of population growth in each case. The results were taken from just one sample test run, with 5,000 units of initial glucose. Clearly, a population that uses priorities grows at a slower pace for about the first 10,000 cycles. The no priorities case has already levelled off at that point and it has started to fluctuate.

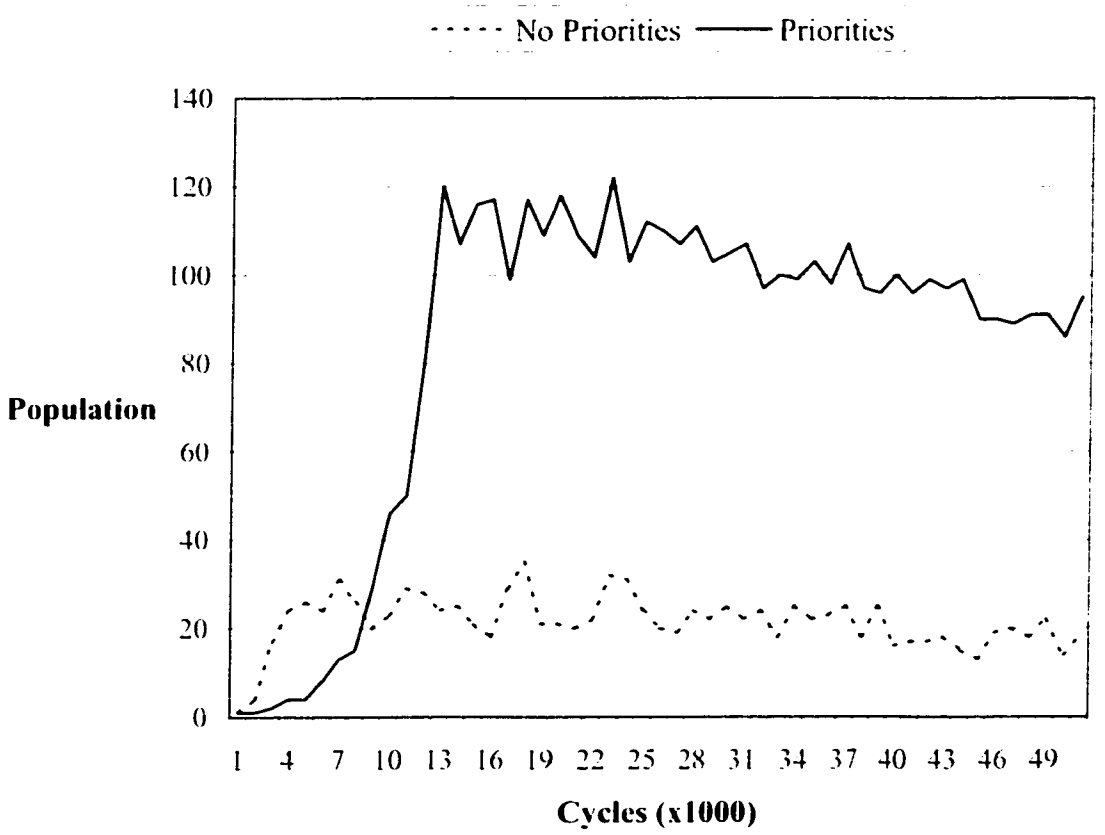

Figure 4: Effect of Priorities on Population Growth

In both cases we notice that there is a period of many births and few deaths during which the population grows abruptly. This occurs sooner if priorities are not used. After that, we have periods of births followed by starvation of some cells.

As a final point, we note that if the simulation was run for an even greater number of cycles there was a tendency for the population to gradually decrease. We can see this by extrapolating the curves in the graph. This means that we are probably losing some glucose as time goes by. From investigating the code, it appears that the cause of this was that since the amount of glucose is represented by an integer, during reproduction, it's possible to lose some glucose when dividing in half. Each time the amount of glucose of the parent cell was an odd number, and the cell divided, one unit of glucose was lost in the division. The effect of this was magnified with larger populations and the total glucose decreased, resulting in a population that tended to decrease with time as well. Modifying the code by ensuring that no glucose is lost during cell division has now rectified this problem.

7 Design Modifications

At this point we have a good idea about the cell functionality, its processes and the behaviour of the system. In this section, we will study how the program itself was modified during this stage of the project.

The most radical change in this version is that we are no longer dealing with a single cell but with a population. We need to monitor the changes taking place in a whole population of cells and update all cells accordingly. We will have a look at changes done for populations, priorities and some improvements to the old program.

7.1 The Cell Simulation Design

The cell simulation is based on the Object Oriented paradigm and makes use its fundamental principles such as encapsulation, inheritance and polymorphism. For the purpose of better explaining the design, we have identified three hierarchical levels of classification based on function. In each of these, we have a number of classes or modules that may co-operate within their level to accomplish a task required by a higher level. This should become clear in the discussion that follows. Going from top to bottom, the levels are described next (Note that the classes and functions described here refer to the old version. The changes to the design will be described later)

7.1.1 The User Interface Level

At this level we have modules that receive input directly from user commands. Every time a button on the GUI is clicked, the required action is handled here. The two

modules we can identify in the code are *Sim* and *Viewer*. Sim is responsible for tasks such as initializing the simulation in response to the user starting the program, or resetting everything in response to a click on the Reset button and so on. It is not required to know exactly how these tasks will be executed. It is sufficient to trigger the correct operations in the lower level (Cell Processing). For example, the following function is found in the Sim module:

```
void doCycles ()
{
    // Update the cell and record the quantity of each substance.
    for (int time = 0; time < maxSteps; time++)
    {
        cell->update();
        for (int s = 0; s < NUM_SUBSTANCES; s++)
            subs[s]->setAmount(time, cell->getAmount(s));
    }
    stepsSoFar += maxSteps;
}
```

The function executes when the user has clicked the Step button. It triggers the cell to update its data though the `cell->update()` statement. The cell is updated a number of times equal to the number of steps on display in the GUI.

The Viewer module handles the look and feel of the report window in response to the user having clicked the Report button. It does co-operate with Sim, however no commands to lower level modules are issued by the Viewer.

7.1.2 The Cell Processing Level

This is where all the interesting work happens in response to messages sent from the Sim module. From our discussion of cell life so far we may identify the classes that exist in this level. A *cell* is a separate entity that exists in an *environment* that has its own

characteristics. The interactions between the cell and its environment are *events* in the life of the cell. The three classes found here are:

- Cell class. Arguably the most important class of the program. Here we have a good example of the use of encapsulation. The characteristics of the cell such as mass, substances it contains as well as how it executes the cellular processes are all found in the private data of this class. These are only a concern of the cell itself. The public functions enable other objects to send messages to the Cell class. Using the previous example, `update()` is a public member function which enables Sim to send messages to the Cell object.
- Environ class. This class represents the environment of the cell. In its private data, it keeps track of the amount of (external) glucose in each square on the grid. The public functions enable other objects to get the amount of glucose in a square or to take glucose from a square during the Ingestion process.
- Events class. This class is used to provide the lists of events that take place during the lifetime of a cell. Here we have the use of polymorphism. There is a generalized Event class, which specifies the interface that derived classes will have. Since there are many different events (such as the Move event or Death event etc) each derived class (such as MoveEvent or DeathEvent) can specify the actual method of implementation.

The environment and events are both private associations of the Cell class so that the cell may update its own environment and events. In this level the classes described co-operate with each other in order to accomplish the tasks input from the Sim module. To do this, they need to employ the services of other functions at a lower level.

7.1.3 The Implementation Level

Here we have a number of smaller classes and modules that do not seem very interesting on their own but are used to provide the implementation details of how cell data is stored. The *Coord* class provides the methods and data needed to keep track of cell movement in terms of coordinates on the grid. The *Genome* class specifies the representation and operations possible on the cell's DNA, whereas the *Locus* class does the same for a single gene. Finally, the *Substance* class defines how the program keeps track of substance data. Each object of this class represents a different cell substance.

7.2 Design for Priorities

The design changes required by the implementation of priorities in the program were quite simple and straightforward. The Cell class was modified by adding a new private method that calculates priorities and a private variable to store the actual priority values for each process:

```
double priority[NUM_PROCS];
double calcPriority(int currprocess);
```

The implementation of `calcPriority()`, is taken directly from the previous discussion of how priorities were implemented in the program (see section 5.2). The exponential function was used in the calculation with the values for the constant λ.

Also, in the Sim module the required interface code was added for the Process Priorities option in the GUI.

7.3 Design for a Population

Adapting the existing design for a single cell to one for a population of cells was the biggest challenge in the project. We need to be able to monitor all processes and substances just as before, but now these must be applied to many cells. We also need to keep track of the locations and movements of all cells on the grid. New considerations come into play such as removing dead cells from the grid as soon as death is detected and so on.

The first decision to be made was how to represent the population of cells. The most natural way seemed to be using a tree where the root would be the oldest cell and the leaves the youngest. However, a tree would require the use of recursion and since much of the functionality for one cell already existed, it would mean almost a total redesign of the system. Also, since each cell can have more than one child, this would not be a binary tree of course, and the complexity becomes even higher. Consequently, it was decided to use a simple linked list of cells added in chronological order.

What the program does now is that each time a cell reproduces, a new Cell object is instantiated, and added to the end of the list of cells while ensuring that it contains the right amounts of all substances. As the list grows, there are some new operations that need to be applied to the whole population of cells and some old operations which no longer apply to just a single cell. An example of the latter case is when the Sim module sends a message to a single cell to update itself – as we saw in function `doCycles()`. This now needs to be applied to a whole population of cells. We would require to place code in the Sim module that traverses the list and calls `cell->update()` at each cell. Although this would work fine, it would mean that more of the processing code would

become mixed with the GUI code. This is contrary to the goal we set out to achieve in the beginning of the project. We would like as much as possible to stick to the three levels of classification discussed.

Finally, for new operations such as the removal of dead cells from the grid, it was not obvious where they should be placed. Is this something that the Cell class should take care of? A cell object cannot remove other cells from the list, so this is not possible. The operation could be placed in Sim but we run into the same problem as before: too much cell processing becoming a part of the GUI code.

7.3.1 Functionality and Data

To solve this problem we must realize that these operations are not part of the User Interface code (for obvious reasons) and they are not part of the Cell Processing code either (they do not deal with only one cell). They seem to belong to a level that is in between the two. This is a level that contains "global" information about the whole population of cells. We may call it the *Population Processing Level*.

The Population class was created to provide all of this "global" functionality. Visually, we can say that a Population object represents the grid of cells. Thus, only one instance of it is required throughout the program. During initialization of the simulation we now have:

```
void initialize ()
// Initialize the simulation.
{

    // Create a new population of cells
    Sim = new Population();
}
```

To decide whether some new operation should belong to the Population class or the Cell class, we need to ask if it is common to all cells or does it depend on the cell. The same

is true for data. For example, the calculation of priorities is dependent on the cell concerned because it needs to know the amounts of substances of the cell. So, looking at figure 6, `calcPriority()` is still a member of the Cell class. Only the cell knows and is able to calculate the priorities of its processes. However, the decision on whether to use priorities does not depend on the cell; it is common for all cells, so we make it a member of the population (variable `usePriorities` in Figure 5). Any future extensions to the program can use this way to decide if the Population or the Cell class should be modified. Using a Population class provides an elegant solution to the problems mentioned before. The `doCycles()` function in the Sim module is now :

```
void doCycles ()
{
    // Update the cell and record the quantity of each substance.
   Sim->doCycles(maxSteps);
   stepsSoFar += maxSteps;
}
```

All that is required to do is simply to call the `doCycles()` function of the Population object in statement `Sim->doCycles()`. The population object will take care of updating all the cells in the list.

As for the removal of dead cells, it should now be obvious that it is the responsibility of the Population class (see private member function `removeDeadCells()` in figure 5.)

7.3.2 Who Should Know What?

The single cell needed to know just about all the information available in the program: the cell environment, the events taking place etc. With a population of cells, much of this information is no longer the concern of just one but of all cells. The cell itself now has more limited knowledge. It only knows information directly related to it, like its genetic

information, or its substances. For this reason the old private associations of the cell to the environment and list of events have now been removed. Looking again at figure 5, these are now the responsibility of the Population class. We can say that the Environ and the Events classes are now part of the Population Processing Level since all cells share the same environment and events. The following classification can be made (again, going from top to bottom):

- The User Interface Level containing the Sim and Viewer modules.
- The Population Processing Level containing the Population, Environ and Events classes.
- The Cell Processing Level, which now includes only the Cell class and
- The Implementation Level, which has not changed from the earlier description.

There is a drawback that arises when individual cells need to add events to the List of Events. This can no longer be done directly by a cell; it is done only through accessing the population object and appending a new event to the list. The cells do not know what other events are in the list: only the population knows that. Making Cell a friend of the Population class would be another solution that would allow direct access to the environment, however we opted to preserve encapsulation.

On the other hand, what does the population object need to know about the list of cells? The population does not need to know the small details such as the substances that an individual cell contains. All that it requires to know is the first cell in the list (i.e. the oldest cell) so that it can traverse the list when needed. Also, it needs to keep track of the cell that is currently displayed in the graph and the number of cells located in each square

of the grid. All of these are now part of the private data of the Population class (shown in bold):

```
class Population
{
    // A simplified model of a population of biological cells

    public:  // All public data goes here

    private:
        void removeDeadCells();

        // The first (oldest) cell in the population
        Cell *firstCell;
        // The displayed cell in the graph
        Cell *graphedCell ;

        Environment *env;
        ListEvents *events;

        // The grid containing cells in each square
        int Grid[DIVISIONS][DIVISIONS];
        bool usePriorities ;
};
```

The Population class could have been made a friend of the Cell class to be able to manipulate cells directly, but again, to preserve encapsulation this was not done.

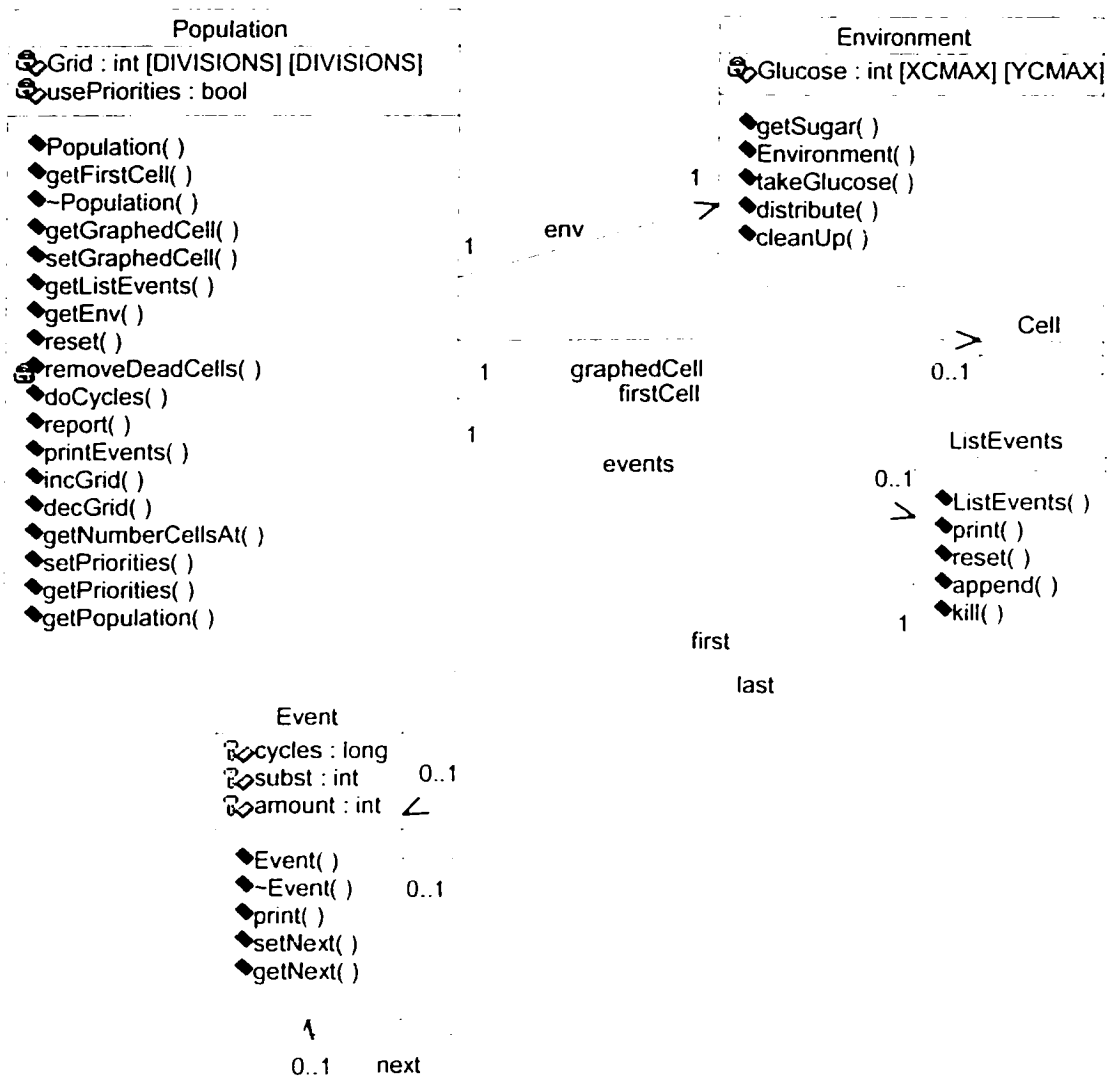

Figure 5: The Population Class and Associations

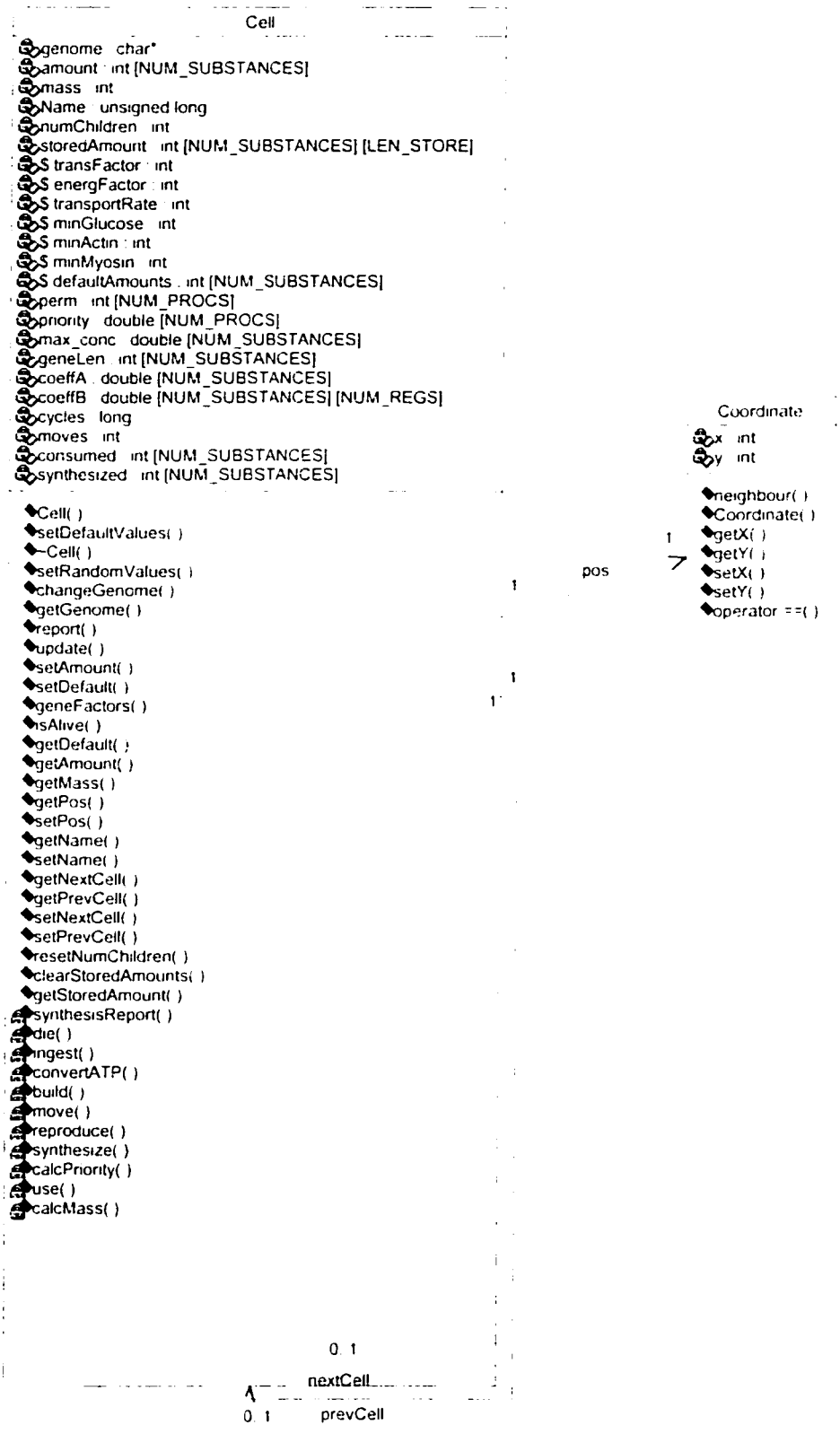

Figure 6: The Cell Class

7.4 Design Changes to the Old Version

We have already seen some of the effects that the addition of a population has had on the previous design. By only allowing access through the population object, we not only avoid having to place a lot of cell processing code in the GUI code, but in some cases we have removed much of the processing that was there. To use an example, the old version executed the following function each time the user reset the simulation (from the Sim module):

```
void reset (bool randomValues)
    // Reset everything in response to 'reset' button.
{
    stepsSoFar = 0;
    for (int s = 0; s < NUM_SUBSTANCES; s++)
    {
        subs[s]->clearAmounts();
    }
    if (randomValues)
        cell->setRandomValues();
    else
        cell->setDefaultValues();
    events->kill();
    delete [] viewText;
    viewText = NULL;
    totLines = 1;
}
```

The highlighted code does not really belong to the User Interface Level. It deals mostly with cell functions and substances and the cell events are also accessed. One can imagine that resetting the simulation for a population of cells within the Sim module would be even more complex. Instead, the function now looks like this:

```
void reset (bool randomValues)
    // Reset everything in response to 'reset' button.
```

```
{
    stepsSoFar = 0;
    Sim->reset(randomValues);
    delete [] viewText;
    viewText = NULL;
    totLines = 1;
}
```

That section of code has now been replaced by a single call to the population object. There is no need to access the events, or substances from this level.

The GUI no longer has direct access to the environment, or the events. These are instantiated by the constructor of the Population class. During initialization the population object is created and it creates the first cell in the population, the environment common to all cells and the list of events that all cells can update.

When the user clicks on the grid to select a cell to view, a message is sent to the population to inform it of the user's selection. Our goal was to limit the actions performed by the GUI code and replace them with messages sent to the population, whenever possible. This should give cleaner separation in the code of the program.

We also have separation in terms of the tasks performed by each class in the application. The Cell class for example, is only concerned with the individual tasks of one cell. These tasks use only entities that are "smaller" than the cell, such as the genome or the substances classes. So, these are the only modules included in the Cell class. Looking at the old Cell class we had:

```
#include "defs.h"
#include "events.h"
#include "environ.h"
#include "locus.h"
#include "subst.h"
#include "coord.h"
```

Now the same class includes:

```
#include "defs.h"
#include "locus.h"
#include "subst.h"
```

We don't need to have as many `#include` statements in our class definitions and we can say that this is one way to have lower coupling among modules.

Finally, some additional changes were done mostly to the GUI code. These had nothing to do with the simulation application itself, but were related to Windows API functionality. Much of this has changed since the first version of the project and needed to be updated. For example, the syntax of the callback functions is different in many cases. The old version looked like this:

```
// Main window callback function.
long APIENTRY WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
```

whereas now it has been modified to :

```
// Main window callback function.
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
```

Hopefully, this will make the program more portable across different platforms.

8 Future Enhancements

The simulation currently provides a good basis for studying cell populations. We have seen how the cell has been made more stable through the usage of priories. Also, we saw how the cells reproduce and we now have a pictorial representation of the cell population as it appears on the grid.

One limitation of the current design is that the Synthesize process has no control of which substance will be synthesized. This is chosen at random. It would be useful to have priorities used within the Synthesize process itself. For example, if the cell is very low on RNAPolymerase, the priority of Synthesize is increased; however, there is no guarantee that RNAPolymerase will be synthesized in the next cycle. In fact, chances are that some other protein will be synthesized which may not be needed at the time. It would be useful to have some way to promote a certain needed substance during the Synthesize process.

Also, in the future, it would be interesting to go beyond the point of studying just the growth of population, but also to study the way the population evolves. By varying the genetic information, and giving each cell the capacity to change and adapt to new environments we should get some interesting results. For example, the way λ is determined now is not very scientific. Instead, λ values could be specified by the genome of the cell itself. Thus, in combination with cellular evolution, the genome could actually optimize the cell priorities and the cell would be better able to survive in a changing environment.

9 Conclusion

The goal of this report was to provide an in-depth analysis of the Cell Simulation project, at the latest stage of its development. This is a project closely related to many on-going large-scale projects in the field of biochemical modeling. We have seen the great complexity that exists in the life of a real biological cell in nature: Some of its processes and substances still remain undefined and are being actively researched.

A study of the older versions of the project was made and how some of their limitations were addressed in the new version. Process Priorities have now been added to the cell and it was shown that the cell is now much more stable and able to survive for longer periods of time. Also, we now have a simulation that models a population of cells living in the same environment and sharing the same nutrients from the surroundings. It was shown that new concerns must now come into play especially from the coding aspect of the simulation. Changes to the old design were done to accommodate the new requirements for a growing number of cells. The user interface code has been updated to become a little more intuitive especially where populations are concerned. At the same time, we have demonstrated how some improvements to the old program help to reduce coupling of modules and provide cleaner code.

Overall, I believe this project has been a success and we have achieved what we set out to accomplish. Many lessons were learned not only in terms of the design and implementation specific to such a project, but also in terms of the complexity that a real living system encompasses and the challenge of simulating life processes.

References

1. P.Grogono, The Cell Simulation, Versions 0.1 – 0.4, May 2000

2. J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lorensen, Object-Oriented Modeling and Design, Prentice Hall, New Jersey, 1991

3. R.Lee, W.Tepfenhart, UML and C++, Prentice Hall, New Jersey, 1997

4. H.Curtis, Biology , Worth Publishers Inc., New York, 1983

5. M.Tomita et al, E-CELL: software environment for whole-cell simulation, 1999 Oxford University Press

Appendix A

Determination of λ in the priorities calculation

In the priorities calculation we used variations of the equation $Priority_p = e^{-\lambda q}$ for a process p. The values of λ were determined experimentally by trying different values and analyzing the results. To give realistic results, λ should be a value between 0 for a maximum priority and 1 for a priority of 0. Here are the values actually used in the program and all results in the report:

Process	λ value
Ingest	1/1000
Generate	1/1000
Build	1/30
Move	1/200
Reproduce	1/100
Synthesize	1/1500

Table 2 The Values of λ

We can see that based on those values the Synthesize process is the one most promoted by the value of λ. It is followed by Ingest and Generate, while the Build process is the least promoted by the λ value followed by Reproduce and Move.