
Percolation on a Square Lattice

Undergraduate Dissertation by

Gareth Flowers

1st Supervisor - ***Louise Richards***

2nd Supervisor - ***Bernard Rayner***

Statement of Academic Integrity

I declare that the information contained in this report is the work of the author. Any other work or information used has been fully referenced to the persons responsible.

Signed..... Date.....

Printed.....

Abstract

Percolation theory has been used by mathematicians and physicists for over forty years to describe the effects of disordered mediums. The analysis of these mediums will undoubtedly require a great deal of knowledge in the mathematical field of probabilities and algorithm. The generation of a lattice capable of simulating a physical situation can be, at the best of times, tedious and time consuming.

The generation of a computer simulation that will visually express and imitate the effects of percolation on a given situation has previously been explored, but in little detail. This project aims to construct a simulation package that will create, analyse and visually explore a square lattice using percolation theory. Stages of design and implementation will be detailed, and the success of the product will be evaluated.

Acknowledgements

There are numerous people that I would like to thank for their continued support throughout my time at university and especially throughout my final year.

First and foremost I would like to express how grateful I am to Louise Richards, who with her extensive knowledge has guided me through this dissertation on percolation theory. Her help made me determined to persevere through the difficult stretches of this dissertation.

I would also like to thank my mum and dad for their support of my schooling and their commitment to help me succeed. Their constant enthusiasm, encouragement and motivation has made a huge difference in the final outcome of this dissertation.

A very big thank you also goes to my girlfriend who has been there throughout, with her constant devotion and commitment. Her understanding towards my heavy work load will always be appreciated.

Table of Contents

<i>Statement of Academic Integrity</i>	<i>II</i>
<i>Abstract</i>	<i>III</i>
<i>Acknowledgements</i>	<i>IV</i>
<i>Table of Contents</i>	<i>V</i>
<i>Table of Figures</i>	<i>IX</i>
<i>Introduction</i>	<i>1</i>
<i>Context (Literature Review)</i>	<i>2</i>
1.1 Percolation Theory	2
1.1.1 What is Percolation Theory?	2
1.1.2 Why use Percolation Theory?	4
1.1.3 Analysis of Clusters	5
1.2 Monte Carlo Method	6
2 Identifying Problems and Solutions	8
2.1 The need for a Solution	8
2.2 Existing Solutions	9
2.3 Objectives	11
3 Simulation Design	12
3.1 Specifications	12
3.1.1 GUI Specifications	13
3.1.2 Lattice Specifications	13
3.2 Assumptions	14
3.3 Methodology	15
3.4 Programming Languages	16
3.4.1 C++	16
3.4.2 Java	17
3.4.3 Visual Basic	17
3.4.4 Choosing the Right Language	18

3.5	Software Tools	19
3.6	Designing the System	20
3.6.1	Lattice Generation & Percolation	21
3.7	User Interface	25
3.7.1	Output (Result Reporting)	27
3.7.2	Graphs	28
3.7.3	2-Dimensional Lattice	28
3.7.4	3-Dimensional Lattice and Cluster	28
3.7.5	Dialogs	29
4	Implementation	30
4.1	Target Computer System	30
4.2	Lattice Creation	30
4.3	Interface	33
4.3.1	Output	33
4.3.2	Graphs	34
4.3.3	2-Dimensional View	35
4.3.4	3-Dimensional View	36
4.4	Techniques Used	37
4.5	Testing	37
4.5.1	Lattice and Cluster Testing	38
4.5.2	Interface Testing	39
4.5.3	Performance Testing	39
5	Evaluation	41
5.1	Evaluation Techniques	41
5.2	Choosing the right Evaluation Technique	42
5.2.1	Cognitive Walkthrough Evaluation	42
5.2.2	Observation Evaluation	43
5.3	Results of Evaluation	43
5.4	Does it meet the Specification	45
5.5	Summary	45
	Bibliography	46
	Appendix A - Project Proposal	50

Percolation Theory	50
Project Aims/Objectives	50
Deliverables	50
Research Areas	51
Resource Requirements	51
<i>Appendix B - Project Work Schedule</i>	<i>52</i>
<i>Appendix C - Project Progress Note</i>	<i>53</i>
<i>Appendix D – Project Log</i>	<i>54</i>
<i>Appendix E – Professional Development Logs</i>	<i>55</i>
Log 1 – First Formal Meeting	55
Log 2 – Second Formal Meeting	57
Log 3 – Final Project Log	58
<i>Appendix F – Project Supervision Meeting Records</i>	<i>59</i>
Meeting 1	59
Meeting 2	60
Meeting 3	61
Meeting 4	62
<i>Appendix G – Design Diagrams</i>	<i>63</i>
UML Class Diagrams	63
Skeletal View	63
Lattice and Lattice App	64
MainFrame and Settings	64
ViewCluster and ViewGraph	65
ViewLattice	65
Layout Designs	66
Main Screen	66
Tab Pane Layouts	67
<i>Appendix H – Source Code</i>	<i>68</i>

Lattice.java	68
LatticeApp.java	76
MainFrame.java	77
ViewCluster.java	88
ViewGraph.java	94
ViewLattice.java	97
Settings.java	101
<i>Glossary</i>	<i>105</i>

Table of Figures

<i>Figure 1, a square lattice before percolation</i>	2
<i>Figure 2, a square lattice after percolation</i>	2
<i>Figure 3, Site percolation</i>	3
<i>Figure 4, Bond percolation</i>	3
<i>Figure 5, a sample of regular lattices.</i>	4
<i>Figure 6, Linear Cluster.</i>	8
<i>Figure 7, Hyper-branched Cluster.</i>	8
<i>Figure 8, Dendrimer Cluster.</i>	8
<i>Figure 9, 2D percolation simulation</i>	10
<i>Figure 10, 2D percolation simulation</i>	10
<i>Figure 11, Waterfall Model</i>	16
<i>Figure 12, JBuilders syntax colouring and error reporting</i>	20
<i>Figure 13, Simple class diagram</i>	21
<i>Figure 14, Hierarchical Task Analysis for building a lattice</i>	24
<i>Figure 15, Interface structure chart</i>	27
<i>Figure 16, Generated lattice (size 30 by 30)</i>	32
<i>Figure 17, Output of results</i>	34
<i>Figure 18, Graph view of the simulation</i>	34
<i>Figure 19, the simulations 2D View</i>	35
<i>Figure 20, the simulations 3D view of a cluster</i>	36
<i>Figure 21, diagram of a virtual universe</i>	36
<i>Figure 22, a sample lattice and its results</i>	38
<i>Figure 23, Simulation performance times</i>	40

Introduction

As more research is being done, the need for a computer simulation that demonstrates the effects of percolation theory is growing. Percolation theory is being implemented in increasingly more complex situations that require an increased amount of computation.

This project will attempt to outline the construction of a simulation that may be used by researches in the area of percolation theory. Focus will be given to the design and construction of a square lattice (both in 2 and 3-dimensional forms) that can be used to create a series of randomly formed clusters, by the implementation of percolation theory. The stages of analysis and design will be carried out to come up with an appropriate solution of conveying a broad range of data associated with the lattice and its clusters.

Attempts will be made to create a graphical user interface that will enable a user to generate a series of lattices of different sizes, from which results can be processed. The creation of a visual display of the lattice and cluster will also be tried.

Context (Literature Review)

1.1 Percolation Theory

1.1.1 What is Percolation Theory?

“Percolation theory was developed to mathematically deal with disordered media” (Bentz, D.P. and Garboczi, E.J., 1991)

The theory of percolation has been used by mathematicians to look at the connectivity features of a disordered material. The start of the theory is usually associated with Broadbent and Hammersley’s 1957 publication that introduced the mathematical aspects of using geometrical and probabilistic concepts (Stauffer and Aharony, 1994). To describe percolation theory it is often best to present an example that illuminates the most important features, from which one can identify with the insights gained – obtaining a more formal definition. Consider a large array of squares, known by physicists as a square lattice, which most can physically identify with a large sheet of graph paper. On this grid a random fraction of squares can be filled with a dot, the remainder of which remain empty.

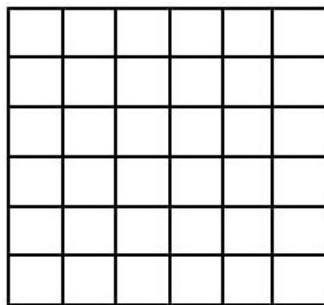


Figure 1, a square lattice before percolation

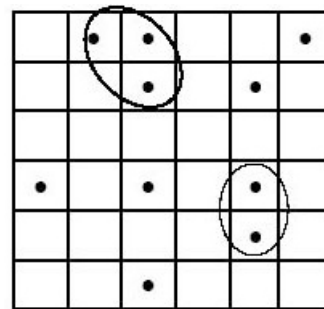


Figure 2, a square lattice after percolation

From this we can see that some of the squares are situated next to each other by a common side, these are nearest neighbour sites, whereas squares connected to each other by a corner are only next nearest neighbours. A cluster can then be defined as a chain of nearest neighbour squares that are occupied by the dots. Analysis of these clusters, particularly the numbers and properties of each, form the basis of percolation

theory. A cluster that spans the entire width, height or depth of a lattice is known as a percolating cluster.

It is assumed that the initial distribution of points within the lattice is done completely at random, and that changing the randomness will have an effect on the distribution and subsequent formation of clusters within the lattice. We can call the chances of a given site having a dot in it probability p , and so the number of points occupied within the lattice will be pN (where N is the total number of squares). Percolating clusters are created when the probability reaches a certain point, for example a square lattice such as this will create a percolating cluster when p reaches 0.59 ($p_c \geq 0.59$) (Stauffer and Aharony, 1994).

There are two main types of percolation – bond percolation, and site percolation (Sahami, 1994). We have already dealt with site percolation (Figure 3), this has been used in the previous example, where a whole square (or section of lattice) is covered and connected to another section by one (or more) sides. Bond percolation (Figure 4) is very similar we are concerned with only lines of the lattice – a dot is placed on the cross of two lines and a nearest neighbour is joined by a connecting line, opposed to a side.

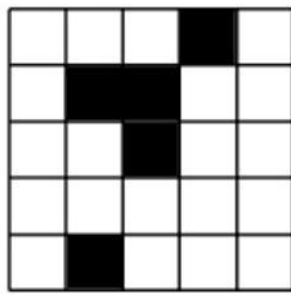


Figure 3, Site percolation

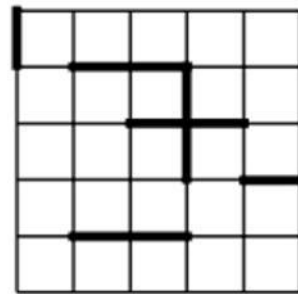


Figure 4, Bond percolation

Of course not all lattices are square and two-dimensional, this was just an example. A lattice is unrestricted in size and shape – depending on the application of the theory (refer to section 1.1.2). Examples of several different lattices are shown below in Figure 5.

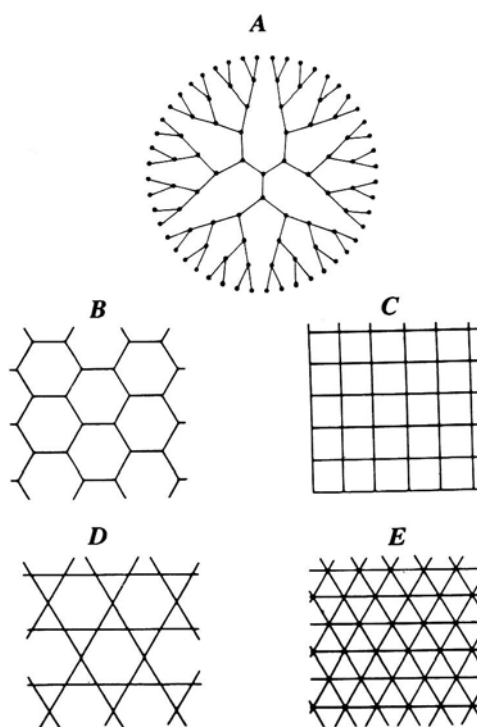


Figure 5, a sample of regular lattices.

(A) The Bethe lattice. (B) The honeycomb lattice. (C) The square lattice. (D) Kagomé lattice.
(E) The triangular lattice. (Diagram source: Sahami, 1994)

1.1.2 Why use Percolation Theory?

The original mathematical literature (Broadbent and Hammersley, 1957) dealt with the concept of the spread of hypothetical fluid particles through a random media (Sahami, 1994). The terms fluid and media were invoked in the most general sense, where fluid could be any liquid, gas, electric current etc., and the media being a porous rock or forest etc. Percolation theory therefore offers a means of mathematically evaluating and/or simulating a real world situation, for example the spread of a forest fire, the distribution of oil or gas in oil reservoirs (fractal oil fields) or the diffusion of two chemicals in a chemical reaction.

The application of percolation theory to a situation provides a significant number of benefits. The theory can be applied to a situation for which the cost of reproducing physically is too expensive, i.e. testing new chemical reactions and examining the

distribution of oil in a new oil reserve, or to the development of a new chemical that is highly dangerous or simply hard to reproduce. For example you could simulate a forest fire, starting from a fixed point (the initial fire) that grows exponentially from that point. The fire would grow from a point in the lattice, setting fire to another point next to it, then to another point next to that - forming a chain of fire that could spread across the whole forest (a chain of points across the whole lattice - i.e. a percolating cluster).

1.1.3 Analysis of Clusters

Once percolation theory has been applied to a given lattice, and that clusters have been generated, created or identified then it is possible to create a list of results that can be for analytical purposes. Each cluster within the lattice has a set of properties that can be found by performing some mathematical functions. The most common and useful of cluster analysis is detailed below.

- **Percolating Clusters** – Percolating clusters (or critical clusters as they are sometimes known) are clusters that span the entire lattice on which they are formed. A percolating cluster can easily be found by comparing the width, height or depth of the cluster to its corresponding value of lattice width, height or depth.
- **Cluster Size** – The cluster size is the total number of points within a cluster. This value is often referred to in other publications as cluster length or molecular weight.
- **Width, Height (and Depth)** – These values show the overall spread of a cluster. If used correctly they can give an impression of the shape of a cluster without a graphical representation.
- **Centre of Mass** – The centre of mass is a value that can be worked out for all three axis (x, y and d – or alternatively width, depth and height). This value is a representation of the point of the midway point, on the associated axis, where there is an equal number of points on either side. The centre of mass is a useful measure as it is used to find the radius of gyration, below. The centre

of mass of all three axis expressed as a vector (or coordinate) locates the precise point at which the entire weight of the cluster is considered to be situated.

Centre of mass can be calculated by finding an average of each x, y, and z component of each point in the cluster.

- **Radius of Gyration** – this is “the distance at which the entire area must be assumed to be concentrated in order that the product of the area and the square of this distance will equal the moment of inertia of the actual area about the given axis” (Wikipedia, 2005). It is essentially a value indicating the spread of points away from the centre of mass – a large value indicates the cluster is spread out over a large distance, whereas a small indicates that the clusters points are packed closely together. This can often be compared to the mathematical function, Standard Deviation.

1.2 Monte Carlo Method

Monte Carlo simulation was named for Monte Carlo, Monaco, where most primary attractions feature gambling and games of chance. Most games of chance exhibit random behaviour, and this is similar to how the Monte Carlo method selects values at random within a simulation. There are two main ‘types’ of Monte Carlo methods simple and sophisticated (Everett and Carter, 1996). Simple methods are used for the direct modelling of a random or chance process, whereas sophisticated methods (including classical, quantum, path-integral and volumetric) are generally used for more complex molecular computations.

The Monte Carlo method is often implemented in Percolation Theory because of the randomness of cluster formation. As you can see from section 1.1.1 percolation theory uses a random probability, p , of a site being occupied. The randomness of such probabilities is crucial to the successful deployment of these sites. In real situations there is often no way to determine how this randomness occurs, so in order

to model such situations effectively we need a method that reproduces true random numbers.

An example of Monte Carlo can be described in the roll of two dice:

If you roll two dice, then the chances of getting *twelve*, that is a number *six* on each die, are $1/36$. However, if you did not know the answer, you could use the Monte Carlo method to roll both of the dice hundreds of times, and adding up how many times you got a total of twelve. Eventually the ratio of *twelve's* to total number of rolls will also be $1/36$.

2 Identifying Problems and Solutions

2.1 *The need for a Solution*

Percolation theory is used extensively in the theory of chemical reactions (at the molecular level). When molecules of different chemicals are mixed together they react to form chains (or clusters) of molecules. These molecules can be of varying length and width (a large cluster size and distribution) and can be branched a number of times. The molecules produced have different physical and chemical properties, and are used for a variety of different applications.

There are three main types of molecules (clusters) produced:

- Dendrimers - Have maximum branching. Every point of the molecule is branched as much as possible.
- Hyper branched - Have some branching.
- Linear - Has no branching, it is just a straight chain of molecules.

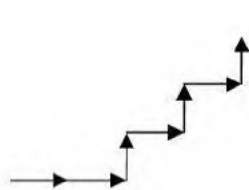


Figure 6, Linear Cluster.

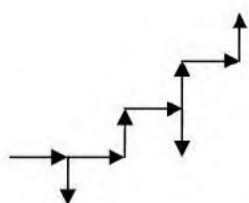


Figure 7, Hyper-branched Cluster.

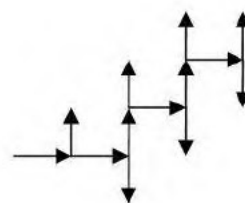


Figure 8, Dendrimer Cluster.

Dendrimers have special properties that make them good for use in things like one-coat paint or drug delivery. This makes these types of molecules very desirable, although the manufacture of such Dendrimers can be very expensive. At present most of the research that explores the properties of molecules created during chemical reactions has been done either mathematically or physically. Chemical reactions have been done in labs on a small scale, and mathematicians have produced theoretical predictions as to the types of clusters (molecules) formed and their subsequent

properties. In the process of a chemical reaction, there are generally two or more chemicals present that react to form clusters of molecules, often of varying length and size, the majority of these reactions using all available substances present during the reaction. In terms of percolation theory the reaction has run with the probability $p = 1$, this means that every site within the given lattice (in the case the chemical contents) has reacted and formed a bond with an adjacent site.

Although simulations based upon percolation theory have been produced, the majority of them base their designs upon a 2D square lattice. Most simulations are used to demonstrate the spread of forest fires, or similar situations, by using percolation probabilities (p), as shown in section 1.1.1. There are very few simulations that aim to visualise and apply the theory of percolation to an entire lattice in both two and three dimensions. It would be useful to have a program that simulates percolation theory - generating a lattice and analyse several aspects of the clusters formed within it. This program could then be used to collaborate results collected by mathematicians and physicists in order to prove or disprove theory's given about a particular reaction (or other situation).

2.2 Existing Solutions

There are only a few existing simulations that are based around percolation theory. A search through popular internet search engines revealed a number of existing simulations, largely based around JAVA and C++ languages. These simulations mostly deal with square lattices on a two dimensional format.

One of these simulations has been developed by Christoph Adami to demonstrate the probabilities at which a spanning cluster will appear in a two-dimensional square lattice (see Figure 9). This solution is effective in providing the user with a simple understanding in to the essence of percolation theory. Its layout is basic and simplistic, providing the user with only the required level of detail. This simulation

uses site percolation with a probability factor that the user can change - and allowing the user to interactively learn about the effects of probability change (variance in p).

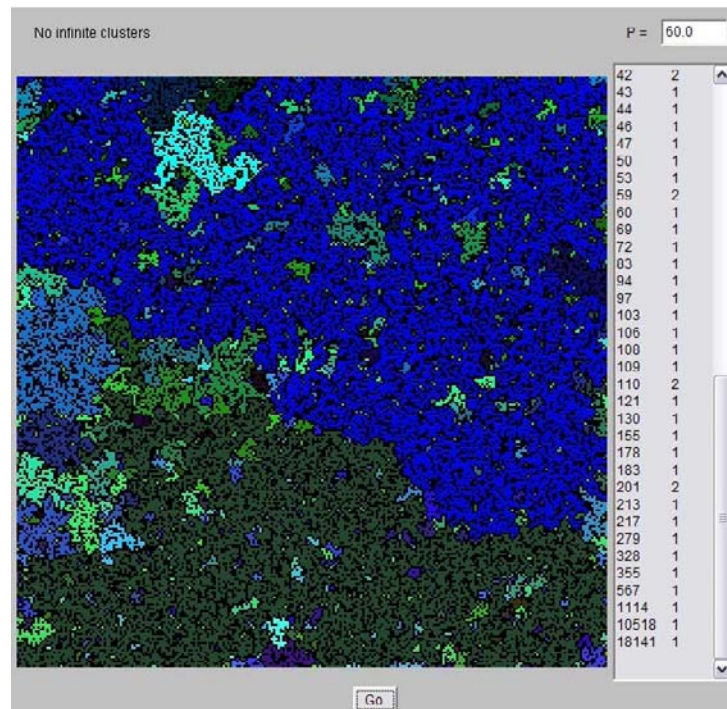


Figure 9, 2D percolation simulation

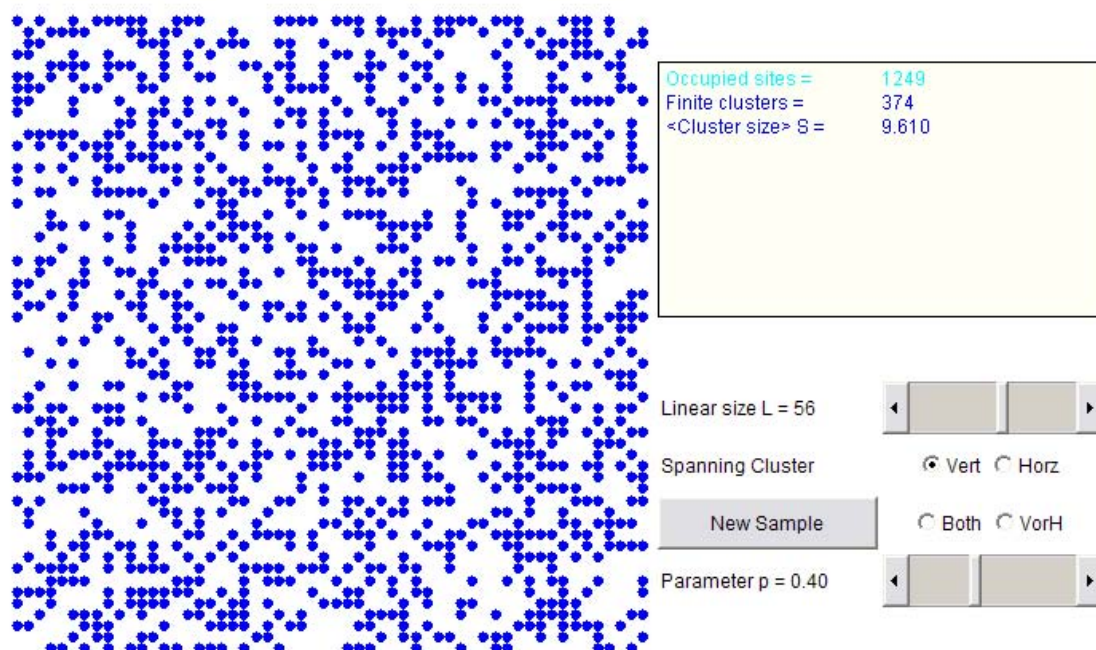


Figure 10, 2D percolation simulation

A second example, Figure 10, also of two-dimensional site percolation was created by Dr. R.J. Consalves, which provides a more detailed view. This java applet provides details of cluster sizes and probabilities, as well as having the extra options of dynamically resizable lattices and probabilities. This is beneficial because it enables the user to easily experiment with varying lattice sizes and probabilities, in turn finding out about spanning clusters. This simulation is personally more appealing as it has an increased level of intuitiveness whilst providing the user with more visual feedback and information.

2.3 Objectives

Simulations of percolation theory are mostly used by researchers and academics to backup theory and practical work done whilst investigating a situation. Whilst solutions have been made and constructed by other people (Adami, C. and Consalves, R.J.) all have been made using two-dimensional site percolation, that simply demonstrate the application of percolation theory in a given environment. This seems a common type of simulation to produce, and although effective in demonstrating simple percolation it draws short of appropriately analysing and outputting the results of cluster properties.

The aim of this project is to design and construct a simulation for percolation theory based upon a square lattice. The product should be able to generate a number of lattices at various sizes upon which some analysis can be performed. To produce a textual and/or graphical representation will greatly help any person attempting to apply percolation theory to a situation. Producing this simulation program should therefore hope to fill the gap by creating a meaningful simulation that can be used as an aid to researches.

3 Simulation Design

“A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.”
(Douglas Adams, 1981)

The design of any program has always remained the single most important factor when developing a system. A system which has been constructed with user interaction in mind “will significantly contribute to end-system usability” (Harmelan, 1999).

Throughout the following design and implementation there are several terms that should be noted. The term *node* refers to a point on a lattice at which the bond lines intersect, and the term *link* refers to a bond between nodes.

3.1 Specifications

In order to produce a product that conforms to the requirements of the user a specification must be drawn up. The specification is essentially a list of rules to which the final product must meet. A brief overview of the problem situation leads to the generalised specification list below.

- A program should be made that will generate a blank lattice, upon which percolation theory may be applied at a given probability. This would then create a series of clusters that could be used to as a basis for some analysis.
- Produce some analytical feedback to the user that will be both relevant and useful in the understanding and further research of percolation theory, e.g. radius of gyration, molecular weight distribution, using the previously developed lattice.

- Produce a 3-dimensional representation of the lattice so that a user can view various parts of lattice in a way that is a more realistic representation of the theory.
- This program would need to have a suitable graphical interface to enable users to easily view and understand the result set generated.
- The possibility of being able to load and save results to a file. This would enable large lattices to be generated and then saved for analysis at a later time.

These basic points can be expanded upon to provide a well-detailed specification that is to be used throughout the design stages.

3.1.1 GUI Specifications

The final specification as an enhancement of the general specification can be listed as follows:

- Generating a lattice of any size.
- Applying percolation theory to the lattice to create a series of clusters (see lattice specification below).
- Save the generated lattice to a file.
- Load a saved lattice.
- Analyse the clusters within the lattice – finding radius of gyration, molecular weights.
- Produce a textual output of the analysis.
- Produce a representation of the lattice in a 3-dimensional form that can be viewed easily by the user.

3.1.2 Lattice Specifications

The lattice used within the main program must have its own detailed specification. This is important to ensure that the lattice is constructed and clusters formed in a

specific way. This specification is different to how previous examples (Figure 9 and Figure 10) have generated lattices so it is important to realise the differences. The lattice specification is listed as follows:

- The lattice will be constructed using bond percolation (see page 3).
- A square lattice will be used.
- Each site location within the lattice must form a bond with another site, i.e. the lattice is run with probability $p = 1$.
- Each location can only create one *outgoing* bond.
- Each location can have a maximum of two incoming bonds.
- Closed loops are not allowed, that is a cluster cannot create a join back upon itself, in essence forming a loop.

3.2 Assumptions

Alongside the specification, there are a number of points that need to be addressed that could have an impact on the system. Setting a series of assumptions is important to reduce the risk of ambiguity and problems arising in further stages of development.

- Every time a random decision is supposed to be implemented then we can trust the programming languages 'random' function to conform to the simple Monte Carlo method. Every execution of the random function has an equal probability and an equal chance of producing each number. For example, if a random selection of four options is needed then each option has a one in four chance of being selected.
- Edges of lattices are negligible. In an ideal system there will be no edge to the lattice, the lattice will continue for billions of nodes in all directions. As this is a simulation a line must be drawn to produce a cut off point, and creating an edge to the lattice. At the edge of the simulated lattice it is assumed that there is one (in 2-dimensional form) or two (in 3-dimensional form) less nodes to link to. There is the possibility to further enhance the simulation at a later

stage by changing this assumption and possible wrapping an edge round to meet its opposing side, in essence creating a continual loop, although this may be beyond the initial scope of this project.

- Following on from the previous point, any node that is placed on an edge or corner still has the same equal probability of being linked as any other node contained within the lattice.

3.3 Methodology

A methodology is simply an organised and documented set of procedures for one or more phases of the software life cycle, such as analysis or design. There are a number of methodologies that can be used to control the design and implementation of a product, i.e. the Waterfall model, v-model, each offering its own benefits in the design stages. The waterfall method's key benefit is that "it provides a good starting point for discussing the logical flow of activity" (Dix, Finlay, Abowd, Beale, 2004), along with the fact that it allows for fast development (has short production time) and little implementation costs. This makes it an ideal candidate for use in this project.

Downfalls to this method include the lack of client input and evaluation of the product throughout the development, though this may not be an issue in this case as the project is not contracted to an outside client.

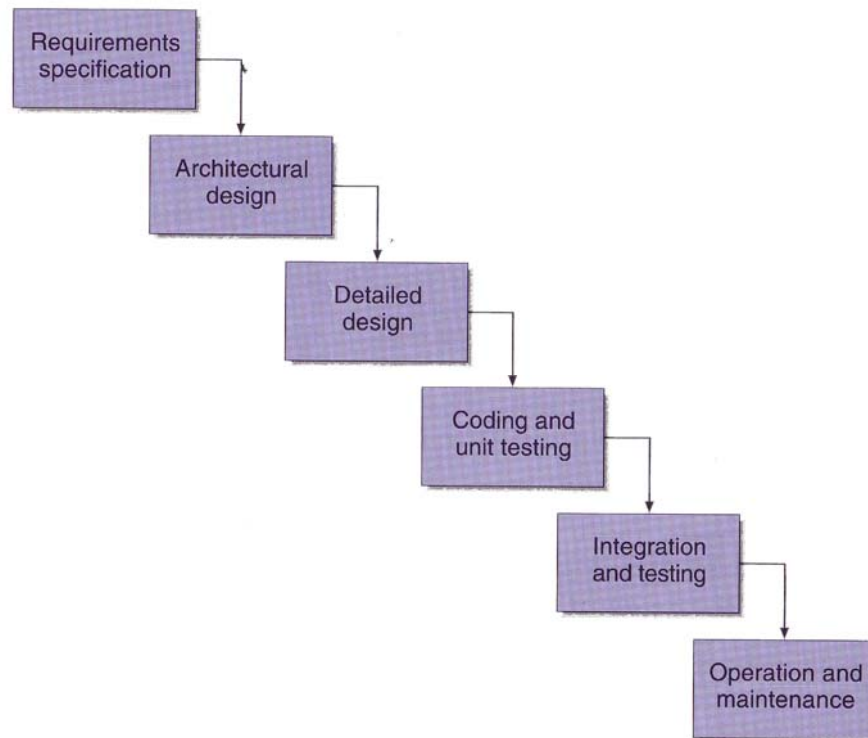


Figure 11, Waterfall Model

(Dix et al, 2004)

3.4 Programming Languages

To develop a product that conforms to all of the specification it is important to ensure that the right type of programming language is chosen. The range of programming languages available for use in computer program design is quite extensive although most choose to stick with the most popular and stable.

3.4.1 C++

“C++ is a general-purpose programming language with a bias towards systems programming that supports efficient low-level computation, data

abstraction, object-oriented programming and generic programming.”
(Stroustrup, 1999)

C++ is an advancement on the original C language. C++ was designed to use the power of C in Object Oriented Programming (OOP), whilst still maintaining speed and stability. C++ is a common choice for developing simulations and games, mostly due to its speed and power. It has a large base of methods and proven high quality libraries that help add stability and power to an application.

3.4.2 Java

“The Java programming language lets you write powerful, enterprise-worthy programs that run in the browser, from the desktop, on a server, or on a consumer device.” (java.sun.com)

Java is now an extremely popular and powerful language that enables the development of programs that can be run on servers, workstations, browsers and consumer devices. A Java developed program is run from within a Virtual Machine (VM) and as such can be run easily across most computer platforms (Windows, UNIX, Mac, Linux). Java provides a separate API for building 3-dimensional applications, which has been constructed especially to develop programs in 3D easily.

“Java 3D allows developers to focus on what to draw, not how to draw”
(Gehringer and Walsh, 2002)

3.4.3 Visual Basic

“Visual Basic is an Object-Oriented Programming (OOP) language and a Rapid Application Development (RAD) environment from Microsoft. Visual Basic provides tools for Internet programming, and helps developers quickly create and deploy enterprise client/server applications,

most often to access both local and remote databases.” (Information Technology Toolbox Inc, 2002)

Visual Basic is a programming language built by Microsoft upon the BASIC language. Although not a true OOP language, Visual Basic is often referred to as such because of its event driven procedures. It is a lot simpler to use than the previous two languages and is often used to develop quick and easy applications, due to the ease of its ‘drag and drop’ implementations, and because of this it is known as a Rapid Application Development (RAD) system.

3.4.4 Choosing the Right Language

Although not an exhaustive list, the languages above are three choices readily available for use at the time of development. Other languages could have possibly been used (i.e. C, BASIC, C#, J#) and documented as suitable choices, but it was felt that they would provide little or no significant improvement over using one of the languages listed.

The first option would be to strike off Visual Basic from the list. Despite its ease of development and the speed at which a program can be created, this language offers no real benefits to a simulation program. It is both slower and has little support for 3D programming.

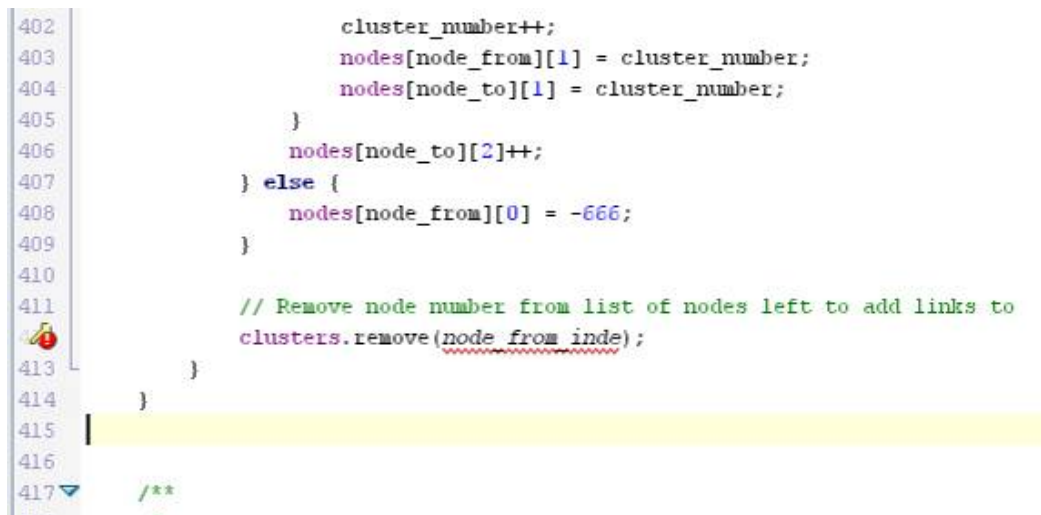
For most, choosing the appropriate language for developing a simulation program would be easy – C++, which would obviously produce a program capable of running a simulation at the fastest speeds. Speed, although the more common factor in a simulation is not just the only aspect of programming language that needs to be taken into consideration. The production of an easy to use, intuitive GUI is essential to ensuring that the program is successful and considering the use of a 3D visualisation of the lattice the use of a C language may not prove the most efficient. Java offers the most attractive proposition in such a case. The implantation of 3D java in combination with standard java will undoubtedly yield best results. As far as speed is

concerned, Java may not be significantly far behind when generating small lattices – but could possibly suffer when large lattices are needed.

Despite possibilities of speed difficulties in large simulation results it was decided that Java be used as the development language for the program. This language will be best because of the ability to combine the Java and Java 3D Application Program Interface (API) in order to create a suitable interface, whilst having capabilities of simulation and results analysis.

3.5 Software Tools

As the choice of programming language has now been selected, it is possible to select the tools used to help develop the simulation. With Java being chosen as the development language and Java 3D to provide the graphical aspects the use of a dedicated Java Integrated Development Environment (IDE) was thought to be most appropriate. The benefit of using such a tool is the ability of the program to use syntax colouring (highlighting different java syntax in different colours) to aid in program design, as well as an integrated compiler, and in some cases a debugger. There are a large number of both free and commercial software available that could assist this development by offering the features mentioned previously. As we do not want to be paying large sums of money out for this one simulation that choice was made to opt for a free IDE. A quick internet search (www.google.co.uk) was made for 'Java IDE' and the top results were briefly reviewed for content, usability and features. The best choices appeared to be JBuilder Foundation (www.borland.com/jbuilder), BlueJ (www.bluej.org) and JCreator (www.jcreator.com), and the final choice was to use JBuilder as this provides a large range of features and integrated compiling, and an added feature of error highlighting.

A screenshot of a code editor window. The code is written in Java and features syntax highlighting: keywords like 'cluster_number++', 'nodes', 'node_from', 'node_to', 'else', and 'remove' are in purple; string literals like '-666' and 'node from inde' are in red; and comments are in green. Line numbers 402 through 417 are visible on the left. An error report is shown on line 411, with a red icon and the text 'Remove node number from list of nodes left to add links to clusters.remove(node from inde);'. The code includes a loop structure with 'else' and 'remove' calls. A yellow highlight is present on line 415.

```
402         cluster_number++;
403         nodes[node_from][1] = cluster_number;
404         nodes[node_to][1] = cluster_number;
405     }
406     nodes[node_to][2]++;
407 } else {
408     nodes[node_from][0] = -666;
409 }
410
411 // Remove node number from list of nodes left to add links to
clusters.remove(node from inde);
413 }
414 }
415
416
417 /**
```

Figure 12, JBuilders syntax colouring and error reporting

Developing Java software requires the installation of a Java Development Kit (JDK) and a VM in order to build, compile and run any developed program. As this simulation will implement 3-dimensional java, the Java 3D API will also be required. The following versions were used for this development, accessible from java.sun.com.

- Java JDK 5.0 Update 2 (including JVM)
- Java 3D 1.3.1

3.6 Designing the System

The design of good system is vital to ensuring the simulation is a success. The simulation must adhere to all statements in the specification, as well as ensuring that it is usable and produces the required results.

The initial system can be split into three main sections:

- lattice construction and generation
- cluster analysis and result reporting
- 3-dimensional (and possibly 2-dimensional) lattice visualisation

All these sections should be run from a central interface as shown in the simple class diagram below.

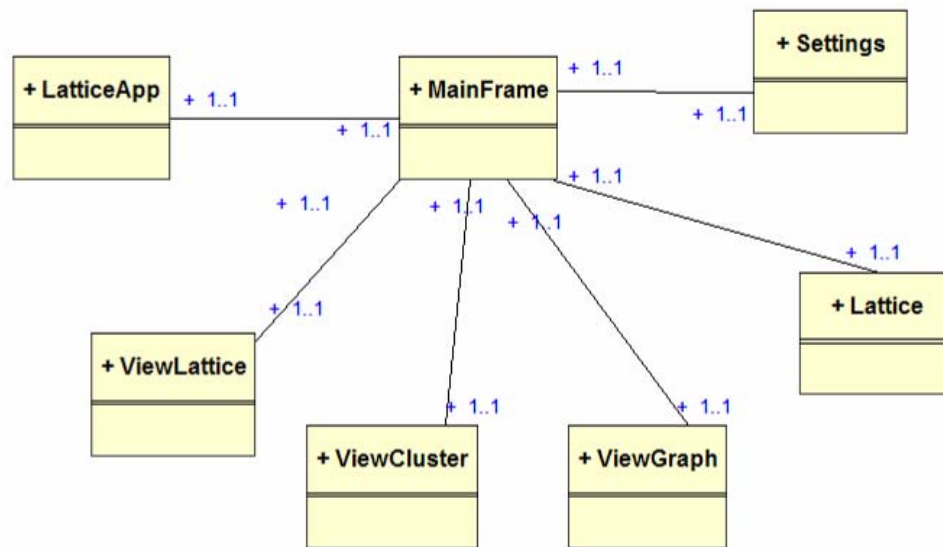


Figure 13, Simple class diagram

3.6.1 Lattice Generation & Percolation

The lattice design is probably the single most important feature in the development of this project. Since the entire simulation is based around a square lattice it is vital that a lattice be created that accurately depicts the state in which a lattice should be in relation to a real world situation.

To help aid the production of lattice the following hierarchal task analysis has been performed and can be seen in Figure 14.

Using the task analysis we can see the order of events that need to be performed to generate a lattice. It is important to observe the specifications constructed in section 3.1.2, previously, upon the generation of the lattice.

These events are listed below in detail (bulleted numbers refer to the event number in the hierarchical task analysis).

1. An initial lattice must be built. This should be an array of nodes (known subsequently as the nodes array) – initially set to the required lattice size squared for a 2-dimensional lattice, or to the lattice size cubed for a 3-dimensional lattice. As well as constructing a nodes array the construction of a clusters array, that is of equal size, is also required. The nodes array is used to store the *to* node, and the clusters array stores the cluster number to which the node is assigned.

e.g. for lattice size 200 the array would have to be

- i. 2D - $200^2 = 40000$
- ii. 3D - $200^3 = 8000000$

Each of the nodes within the array is a reference to a single node on the lattice

2. A link must be added between two nodes on the lattice by using the following steps.
 - 2.1. A node must be chosen at random. This random node should be chosen according to the rules of the simple Monte Carlo method (that is that there is an equal probability of any node being chosen), refer to section 1.2 for more information. If this node has not already got an outgoing link to another node then we can continue to stage 2.2.
 - 2.2. A suitable check needs to be made to see if any of the surrounding nodes are available to form a link to. On a 2-dimensional lattice the checks need to be made to the nodes directly to the left, right, upper and lower of the random nodes. In the 3-dimensional lattice the check needs to also to the nodes in front and behind the random node. The check is to see whether the value of the array at the random node value is empty or not. The check on each of these nodes is to search the entire nodes array to see how many *from nodes* link to that check node. If the number of *from nodes* linking to the check node is equal to or greater than the value detailed in the specification (two) then the check node is no good and must be discarded. Otherwise, if there are less than two nodes linking to it, the check node is good and can be used in the next stage.

- 2.3. After 2.2 has been performed there should be several good or bad nodes listed. A random choice is again needed to select one of the good nodes. The random choice should again conform to the simple Monte Carlo method and apply equal probabilities of each good choice being chosen.
 - 2.4. Once the chosen node has been selected then a link can be created. This is recorded in the nodes array by setting the value at the index of the *from* node to the *to* node.
e.g. if node 10 is connected to node 9 then: nodes array (10) = 9
3. Event number three assigns cluster numbers to each of the nodes, immediately after a link has been added. The cluster number determines the cluster group to which that specific node belongs.
 - 3.1. The first step of this group of sub events is to determine if the node has a cluster number assigned to it. Each node has a corresponding cluster number in the clusters array. If the value of clusters array at index *node from* is empty the node has not yet been assigned a cluster. If the clusters array contains a value then this is to be used as the current cluster number on event 3.3.
 - 3.2. If in event 3.2 the cluster number does not exist (the clusters array value is empty) then the highest cluster number within the clusters array must be chosen and incremented by one for use in event 3.3.
 - 3.3. Using the cluster number that was got from either event 3.2 or 3.3 the value should be assigned to the clusters array index *node from*.

As you can see for the task analysis events two and three loop continuously until each of the nodes has had a link and cluster number assigned to it. Following the whole process through should create a lattice with a series of randomly produced links (using Monte Carlo method), that form groups of links (clusters).

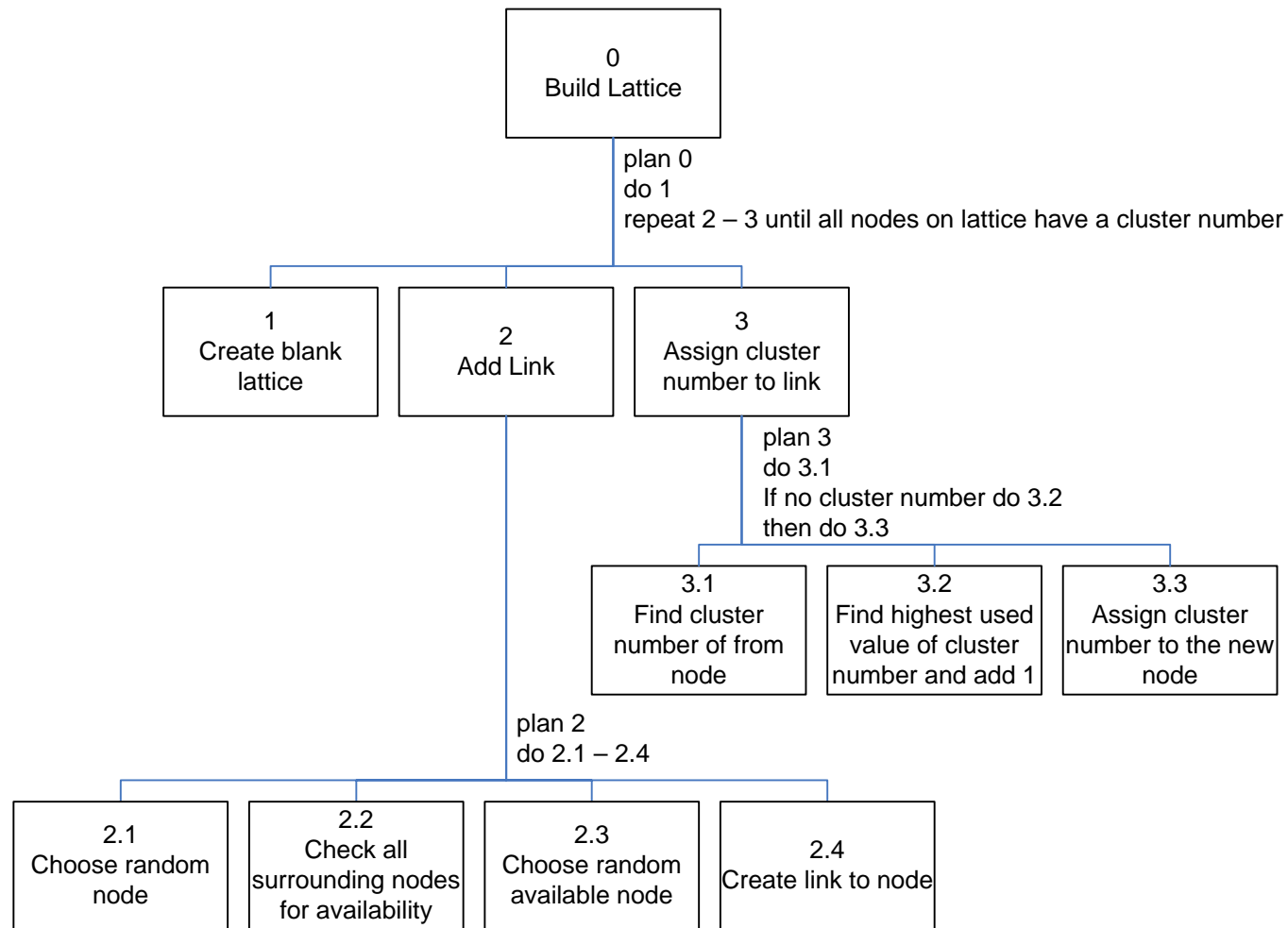


Figure 14, Hierarchical Task Analysis for building a lattice

3.7 *User Interface*

“The aim of HCI should be to build computer applications that are jargon free and easy to use to the degree that all the user sees is the task and not the computer system at all” (Faulkner, 1998)

A successful user interface is one that can be used easily by the user to complete the job in hand. Human Computer Interaction (HCI) is used to study the interaction of users and computers and to help determine good principles of design and can often provide suitable rules to adhere to. Schneiderman's (1998) golden rules of user interface design help provide designs with a clear-cut list of objectives that will help aid the design of an interface.

Schneiderman's Eight Golden Rules of Interface Design:

1. **Strive for consistency.**

Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.

2. **Enable frequent users to use shortcuts.**

As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

3. **Offer informative feedback.**

For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

4. **Design dialog to yield closure.**

Sequences of actions should be organized into groups with a beginning, middle and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of

relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.

5. Offer simple error handling.

As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

6. Permit easy reversal of actions.

This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry or a complete group of actions.

7. Support internal locus of control.

Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

8. Reduce short-term memory load.

The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced and sufficient training time be allotted for codes, mnemonics and sequences of actions.

The simulations main interface was designed as a series of storyboards that attempted to implement the majority of design rules laid out by Schneiderman. The story boards implements these rules because it was felt that they conveyed the most appropriate design methods and usability techniques. Of course, it took a few attempts to tweak the storyboard designs until they were at an appropriate level that implemented enough of the *golden rules* to make the interface successful. The final storyboards can be seen in appendix G.

The layout of the interface has been kept simple and easy to use. There are four different views available for the user to navigate between, which are:

- Output – General cluster results displayed in text format.
- Graphs – Cluster results displayed in simple graph form. The graph will be Radius of Gyration vs. Molecular Weight (cluster length).

- 2D lattice – A two dimensional representation of the entire lattice.
- 3D lattice – A three dimensional representation of the lattice and clusters contained within the lattice.

Navigation between each of the views is provided by a series of tabs along the top of the program. This form of tabbed navigation is common in other programs and so the user can use their own perceived mental model (Thimbleby, 1990) to recognise that this is the way to move through the program. Menus provide additional options for the generation of a new lattice or multiple lattices as well as the ability to load and save generated lattices to a file for later viewing. Shortcuts have been placed and associated with the most common menu functions allowing frequent users easy access to different parts of the program with easy key combinations (Schneiderman, 1998).

The layout of the interface can be viewed easily in the structure chart.

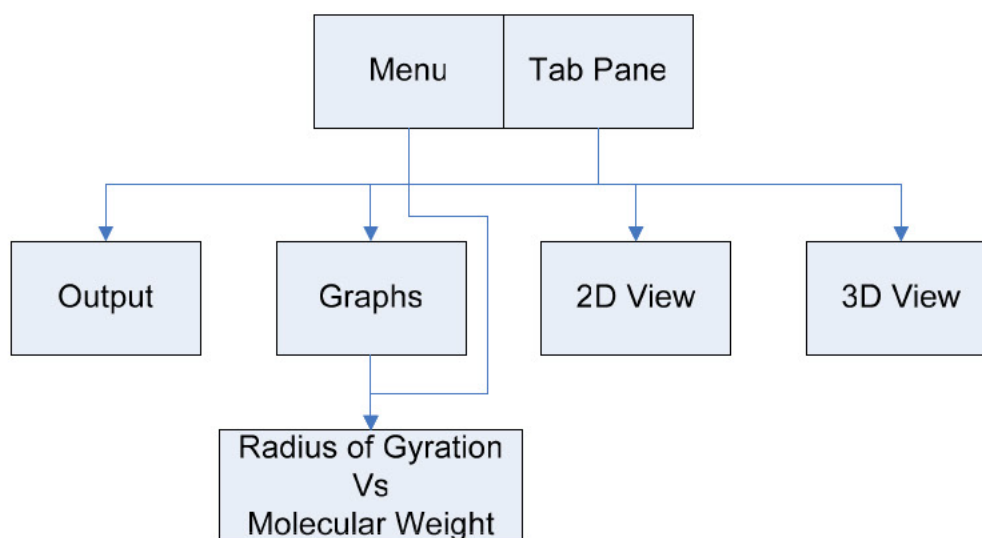


Figure 15, Interface structure chart

3.7.1 Output (Result Reporting)

The output section will contain a list of the results produced by the generated lattice(s) in text format. These results will be best formatted in tabular form so that the user can pinpoint results of particular clusters and lattices with ease. This form of results also

allows the user to copy results to another program for more enhanced analysis, which is beyond the scope of this project.

3.7.2 Graphs

The graphs will be displayed as a graphical image. The graph will display two sets of values (namely radius of gyration and molecular weight) in the form of a scatter graph. The scatter graph was chosen because it is initially the easiest type of graph to implement. Further development could include improvements to graph types and displays.

3.7.3 2-Dimensional Lattice

The visualisation of a 2-dimensional lattice will show an entire lattice in the main view with two list boxes to the left hand side. The list boxes will allow the updating of the main view with the selected lattice, and subsequent highlighting of the selected cluster. This view was placed in to give the user a visual impression of the lattice and the size and shapes of the clusters within.

3.7.4 3-Dimensional Lattice and Cluster

The 3D view of lattice and cluster, along with output, is the most crucial display within the simulation. This view is similar in layout to the previous 2D view, having two list boxes down the left hand side used for selection. The main view will be updated upon selection and display a 3D view of the lattice or cluster, and will allow rotations and/or movement by using the mouse. This view will give users a realistic 3D view of exactly how clusters will look.

3.7.5 Dialogs

The simulation will use three dialogs to perform a number of tasks within the program.

- Save Dialog – This will implement the operating systems (OS) default save dialog box. Using this the user will be able to save the generated lattice(s).
- Load Dialog – This will again implement the OS's load dialog box, allowing the user to retrieve a saved lattice.
- Settings Dialog – This is a custom constructed dialog that will enable users to change the lattice size, number of lattices generated and whether to produce a two or three-dimensional lattice.

Each of these dialogs will use error checking to ensure that values entered within the fields in the dialog are of the correct format. The settings dialog particularly needs to ensure that suitable numbers are entered within the text fields. The numbers should range from one to a set maximum (which will be determined later through testing). Error checking is essential to make sure users will only enter data that will not cause errors, and ensuring the program complies with Schneiderman's rules.

4 Implementation

4.1 *Target Computer System*

Before the implementation of the simulation can take place the decision as to the type of computer system the program should run on needs to be considered. As most simulations require a powerful environment to run in it is necessary to choose an appropriate system that is capable running at a moderately high CPU speed with a suitable level of memory. A recommended minimum specification has been drawn up below.

PC Hardware

- 1GHz Pentium III or higher Processor
- 256 MB RAM
- 500MB available hard drive space (for install, saved files, and required Java VM)

The computer system must have an appropriate version of Java installed in order to run as designed, refer to 3.5 Software Tools for more information on Java VM.

4.2 *Lattice Creation*

The lattice was created first, as this will be the basis upon which the simulation will be built. Following the stages stated in the lattice creation hierarchical task analysis (Figure 14, page 24) the lattice was constructed as a class.

Initial problems in creating the lattice included Java's lack of dynamic primitive array types. This meant that simple primitive data (i.e. `int`, `String`, `boolean`) could not be

used within an array that needed to be dynamically resized. This meant that where dynamic arrays were needed the higher level `ArrayList` would have to be used instead. `ArrayList`'s can only add objects so every primitive type that was to be stored had to be converted into its appropriate higher level object, e.g. converting `int` to an `Integer` object. This is demonstrated in the lattice constructor below.

```
// Creates new lattice
for(int i = 0; i < lattice_total_nodes; i++) {
    // Set up nodes with empty data (-1)
    nodes[i][0] = -1;
    nodes[i][1] = -1;
    nodes[i][2] = 0;
    clusters.add(new Integer(i));
}
```

This was definitely not a design choice as it may have implications on speed, especially in large lattices sizes.

After the initial creation of the lattice arrays, the `add_links()` function is called. This function is the core of creating random links and the assignment of cluster numbers within the lattice. This loops through every node in the array, every time choosing a random node using the function below (refer to 3.2 on assumptions about random functions):

```
private static int rand(int low, int high)
{
    // returns a random number between low and high
    return(Math.random() * (high - low + 1) + low);
}
```

On every pass of the loop a check is made to see if the surrounding nodes are available for linking. This is done by applying the `valid_coordinate` function to a point on the lattice (in x, y coordinate form).

```
if(valid_coordinate(point_x + 1, point_y, node_from)) {
    valid_points.add(new Integer(find_node(point_x + 1,
    point_y)));
}
if(valid_coordinate(point_x - 1, point_y, node_from)) {
    valid_points.add(new Integer(find_node(point_x - 1,
    point_y)));
}
```


Once a node is identified as being valid it is added to an arraylist of `valid_points`. A random entry is then chosen (again using the `rand()` function) and the node entered in the `array_nodes` table as having a link created.

The assignment of cluster numbers is performed after the link is set and is created by using a fairly complex routine of `if` statements. This routine checks to see if the `from` or `to` nodes are already assigned a number and uses that as the cluster number, or increments the current highest `cluster_number` and uses that instead.

```
// Assign a cluster number
if((nodes[node_from][1] > -1) && (nodes[node_to][1] > -1)) {
    if(nodes[node_to][1] > nodes[node_from][1]) {
        replace_nodes(nodes[node_to][1], node_from);
    } else if(nodes[node_to][1] < nodes[node_from][1]) {
        replace_nodes(nodes[node_from][1], node_to);
    }
} else if((nodes[node_from][1] > -1) &&
    (nodes[node_to][1] == -1)) {
    nodes[node_to][1] = nodes[node_from][1];
} else if((nodes[node_from][1] == -1) &&
    (nodes[node_to][1] > -1)) {
    nodes[node_from][1] = nodes[node_to][1];
} else {
    cluster_number++;
    nodes[node_from][1] = cluster_number;
    nodes[node_to][1] = cluster_number;
}
```

Once the loop has complete and every lattice is linked then the class will produce a random lattice of a given size, like the one shown in Figure 16.

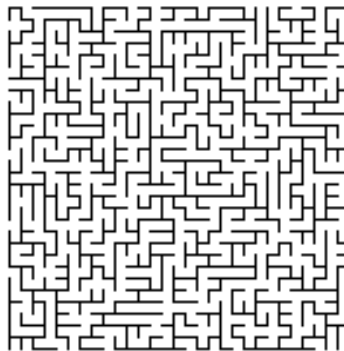


Figure 16, Generated lattice (size 30 by 30)

4.3 Interface

The interface was created using as a central class that implemented the JFrame component. The JFrame was expanded upon to include such components as Menu, JTabbedPane and Panels in order to create look as specified in the design. This frame implements new instances of the other classes to produce the necessary views. Each one of these is viewed on request through changes in the JTabbedPane or when the appropriate Menu entry is selected. The associated UML class diagram is shown in appendix G.

The GUI uses two functions to show and hide various graphical elements within the display upon creation of the lattice. As lattice generation could take a while the choice to hide all non-essential and display a 'please wait' message was added to the design. This provides feedback that helps the user to understand what is going on within the simulation.

4.3.1 Output

The output of the results of cluster and lattice analysis are displayed in a JTextArea component within one of the JTabbedPane views.

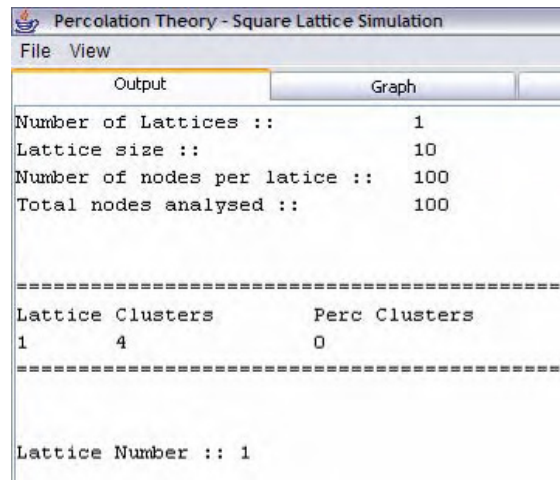


Figure 17, Output of results

4.3.2 Graphs

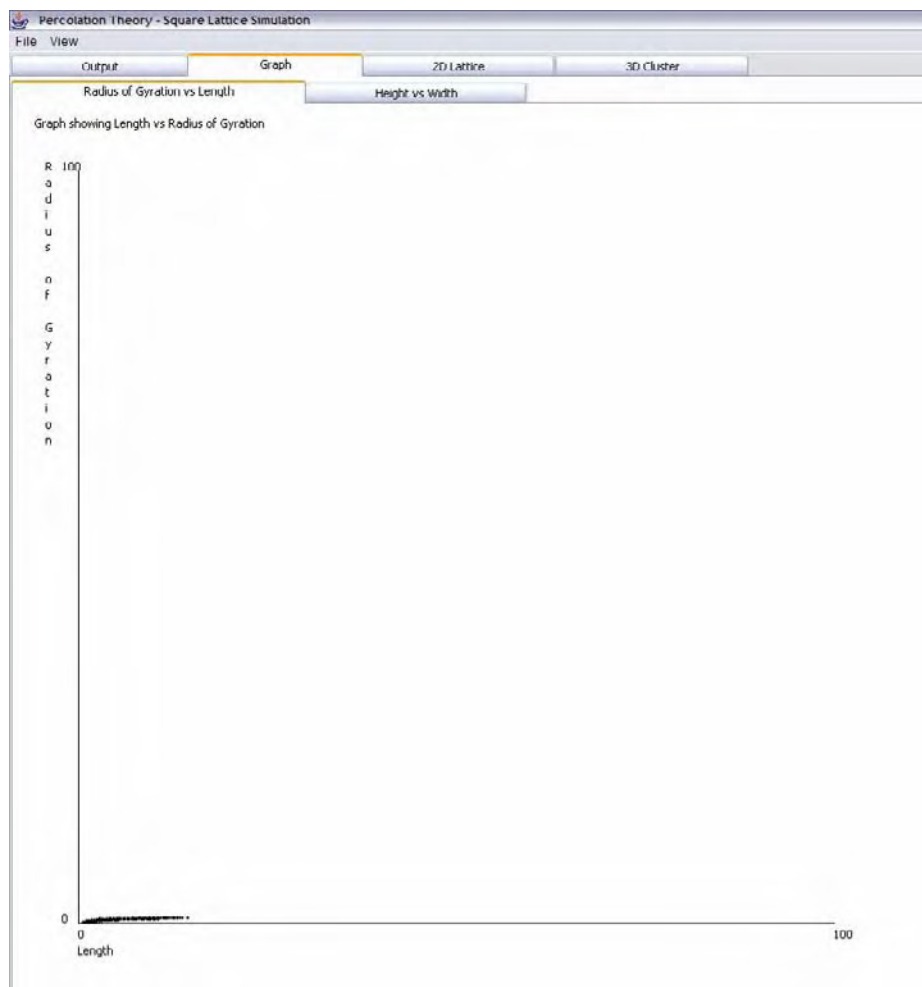


Figure 18, Graph view of the simulation

Java's Graphic component was used to draw the graph view as shown above. Common drawing shapes (i.e. lines and ovals) were used along with some text labels.

4.3.3 2-Dimensional View

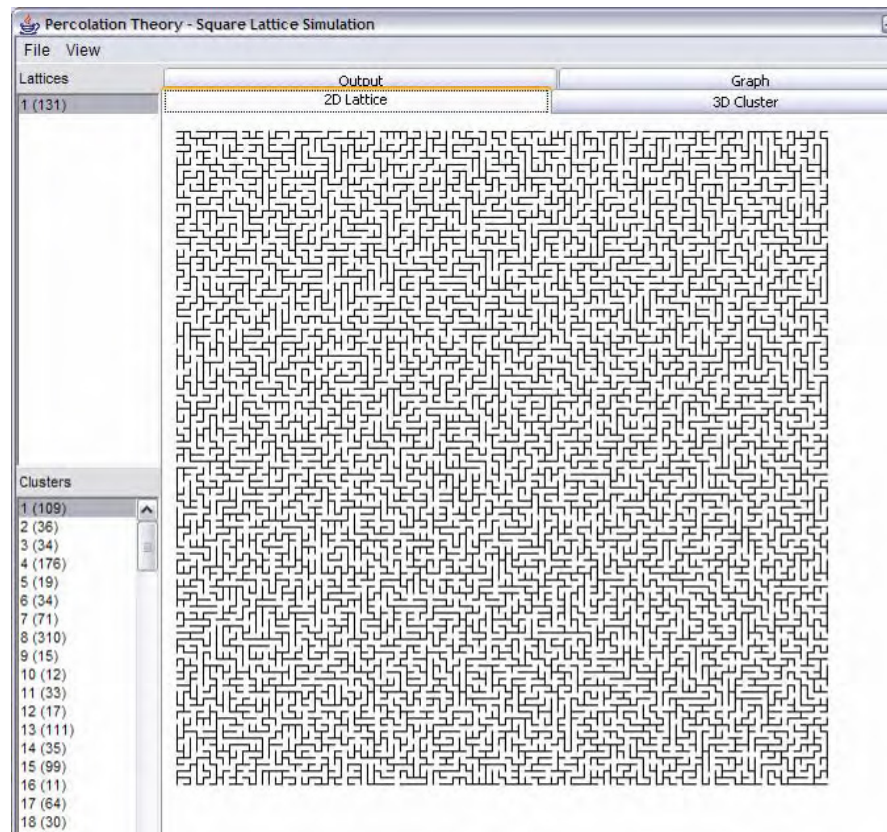


Figure 19, the simulations 2D View

The graphic component was again used to draw lines to create the 2D lattice. This component also implements a mouse event handler so that the user click on areas on the lattice and the corresponding cluster will be highlighted.

4.3.4 3-Dimensional View

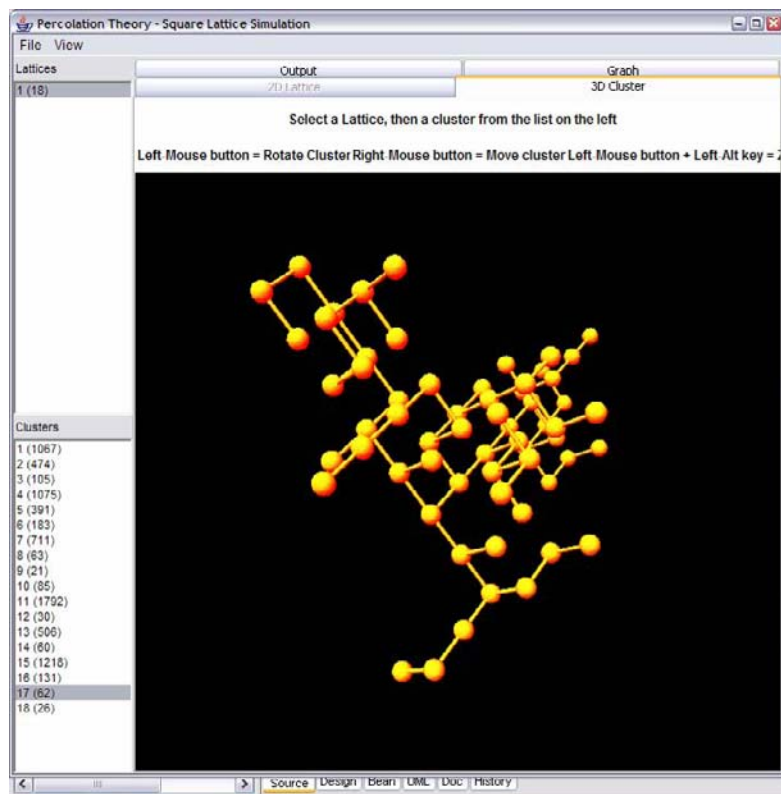


Figure 20, the simulations 3D view of a cluster

The 3D view uses the 3D API to construct a virtual universe environment. The following diagram helps to convey the relationship of components needed to construct a universe.

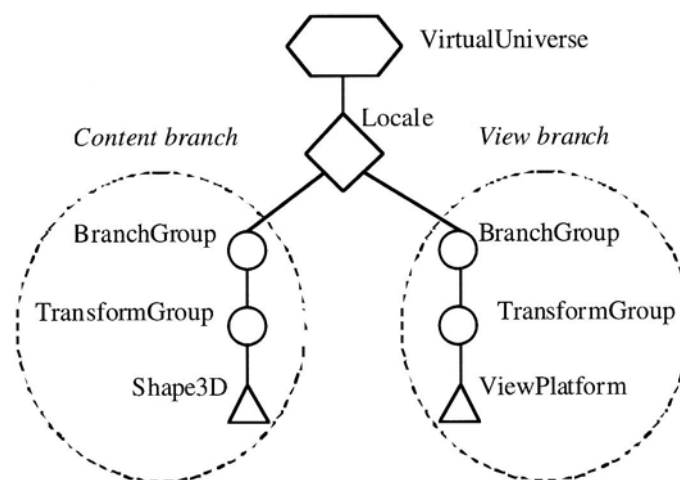


Figure 21, diagram of a virtual universe

(Palmer, 2001)

To construct a 3D virtual universe a content and view branch must be created. The content branch is to contain the objects that are to be viewed (shown below adding a sphere),

```
Transform3D trans_sphere = new Transform3D();
trans_sphere.setTranslation(new Vector3f(xc, yc, zc));
TransformGroup spheres = new TransformGroup(trans_sphere);
spheres.addChild(new Sphere(sphere_radius, appearance));
group.addChild(spheres);
```

and the view branch sets up how the content should look (shown below as tilting the view on a vector(0,0,5)).

```
BranchGroup view = new BranchGroup();
Transform3D view_transform = new Transform3D();
view_transform.set(new Vector3f(0.0f, 0.0f, 5.0f));
TransformGroup group = new TransformGroup(view_transform);
ViewPlatform platform = new ViewPlatform();
PhysicalBody body = new PhysicalBody();
PhysicalEnvironment environment = new PhysicalEnvironment();
```

4.4 *Techniques Used*

There were several coding styles used in the simulation development.

- **For Loops** – In order to cycle through every node within a lattice, or to cycle through every lattice for loops where used as these provide a slight speed improvement rather than using `do...while` loops.

4.5 *Testing*

To test the system appropriately there are three main areas that need to be concentrated on lattice generation/results, and interface design. Each of these can be tested using simple methods described in the following sections.

4.5.1 Lattice and Cluster Testing

Testing for lattices and clusters can be done visually and by using the outputted results created by the program. It is easy to manually test small lattices for cluster sizes, dimensions (width, height, depth), percolating clusters.

Cluster	Length	Mass X	Mass Y	Mass Z	Width	Height	Depth	Rad.G	Percolating
1	8	8.6	7.4	0	2	5	1	1	0
2	15	5.6	7.8	0	6	4	1	1.7	0
3	10	0.9	7.8	0	3	4	1	1	0
4	34	3.3	4.4	0	8	6	1	2.2	0
5	23	7	1.7	0	6	6	1	2	0
6	10	1.5	0.8	0	4	3	1	1	0

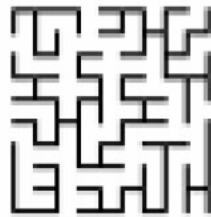


Figure 22, a sample lattice and its results

As you can see from the table of results, gathered from the simulation, along with the 2D view of the lattice, it is possible to check that every piece of data in the table is correct with visual representation. Working through the top left cluster we can compare information and then match it to a cluster number in the table.

Length	=	10	
Width	=	4	
Height	=	3	
Mass X	=	$(0+0+1+1+1+2+2+2+3+3) / 10$	= 1.5
Mass Y	=	$(0+0+0+0+1+1+1+1+2+2) / 10$	= 0.8

It is therefore easy enough to tell that this cluster number six, and that the results are correct.

4.5.2 Interface Testing

The interface was checked thoroughly for errors after implementation. Due to problems in implementation a number of bugs had arisen that need attention. All problems are listed below.

- The graph displays (radius of gyration vs. molecular weight) do not display correctly when the lattice size goes above 9 (in 3D) or 26 (in 2D).
- There is currently no way of stopping the generation of a lattice once it has been started. If a new large lattice were to be generated i.e. greater than a few hundred, then it takes a while (possibly over a few hours) to create the lattice and there is no method of termination.
- When changing between views the 3-dimensional view of the lattice and clusters does not automatically update and may retain an old cluster until a new one is selected from the list.
- The program may crash upon creation of a new large lattice.

As you can see there are several issues that need to be addressed in order for the simulation to work planned. These will be dealt with in the evaluation.

4.5.3 Performance Testing

The speed at which the simulation will run has been tested and the results can be seen below. Simulations were carried out for various lattice sizes, and using multiple numbers of lattices.

Lattice Size	Number of Lattices	3D	Time for Generation
10	1	No	~1 seconds
10	1	Yes	~1 second
10	100	No	~1 second
10	100	Yes	~2 seconds
100	1	No	~1 second
100	1	Yes	
100	10	No	4 minutes
100	10	Yes	1 hour 35 minutes
500	1	No	25 minutes
500	10	No	5 hours
1000	1	No	23 hours

Figure 23, Simulation performance times

(Tests performed on a PC with the following specification: AMD 2300+ CPU, 756MB RAM)

5 Evaluation

5.1 Evaluation Techniques

Many different evaluation techniques can be used to evaluate a system or piece of software. Each one varies in the way it carries out its evaluation and in doing so each has its strengths and weaknesses.

The best methods for evaluation, are either the expert or cognitive walk-through. These are performed by both having a systems expert or an evaluator going through the system systematically and identifying the usability issues or faults (Dix et al 2004). This can often produce the most effective problem solving methods as the evaluators will have good knowledgeable experience with similar systems or computer systems in general that will aid their intuitivism to the system. Questionnaires are a useful evaluation method in nearly every developed product. They enable producers to identify features that users find easy/hard or like/dislike, despite limitations when it comes to identifying the reasons (feedback can be limited). Some specific HCI questionnaires have been developed and a number of respectable sources favour them as they return better results than standard questionnaires. Heuristic evaluation depends on a panel of evaluators independently testing the system against a list of ten or so heuristics. This list covers the majority of the system and can be quite useful. The ‘Break me if you can’ technique is a simple technique that is performed exactly as described. The evaluator simply plays with the system and tries to bring it down – break it. This can be a good test of the resilience of a system to users that are not very skilled with computers, although has some drawbacks as areas of the system could be missed. Another method of evaluation is to observe users using the system either by taking notes, recording the evaluation (by video recorder), or getting the users to ‘think aloud’ and write down ideas themselves. This method uses co-operative evaluation, the user and evaluator work together, to assess the state of the system.

5.2 Choosing the right Evaluation Technique

When it came to evaluating the prototype system there were many evaluation techniques that would assist in the evaluation of the admissions system. The preferred method chosen was the cognitive walkthrough method. Having colleagues who are on similar computing and business computing courses meant that this method could be reliable and produce a good set of feedback for evaluation. The observation technique will also be implemented by a separate group of people, also with varying degrees of computer knowledge. The combination of these two methods would provide a good range of results of the suitability of the system, and success of the designed user interface.

5.2.1 Cognitive Walkthrough Evaluation

A set of simple tasks was given to each evaluator that they were to carry out unaided. Each task was then evaluated based on difficulty any difficulties found whilst performing. The following tasks were asked to be performed by each evaluator:

- Generate 10 new lattices of size 50 in 2D.
- Save these results to a file – which should be named “lattice results”.
- Generate five new lattices of size 10 in 3D.
- Find how many of the lattices have percolating clusters.
- Load the results file “lattice results”
- View the results in the 3D view and rotate round the clusters.

5.2.2 Observation Evaluation

The evaluators were left alone with the simulation for a period of 15 minutes. In that period they were asked to use the simulation as much as possible and to report back any problem or difficulty they had.

5.3 *Results of Evaluation*

With the cognitive walkthrough and observation methods completed an evaluation of the simulation can be performed. Although the evaluations were carried out by a small number of people (8 to be exact), it is believed that their thoughts will accurately depict a representation of the larger population. There were a few main issues raised by the evaluators.

Firstly, it was thought the lattice takes too much time to generate when using large sizes. This problem has previously arisen in the testing that was done earlier (section 4.5.3). The apparent slowness of lattice creation could be down to a number of things – speed of processor on which the simulation is running, efficiency of the implemented algorithms or the speed of the programming language. Of these options it is probably a combination of algorithm design and Java's processing ability. It was clear from research done earlier (3.4.4 Choosing the Right Language) that Java was not essentially the most appropriate language for constructing simulations, but because of visualisation purposes this language was implemented anyway. Optimization techniques have been applied throughout the program where appropriate to try and reduce generation time, but to no significant gain. A selection of applied methods are as follows, as shown by Shirazi (2003):

- Changing for loops to compare to 0. A comparison to 0 is quicker than any other. e.g. changing `for(int i = 0; i < size; i++)`
to `for(int i = size; --i >= 0;)`

- Removing multiple instances of the same code and replacing by a variable, eliminating sub-expressions.

```
e.g. change      x1 = y * Maths.abs(z) + y;
                  x2 = y * Maths.abs(z) + y;
to               x1 = y * Maths.abs(z) + y;
                  x2 = x1;
```

The second problem was the lack of ability to stop the generation of a lattice halfway through. This again is a problem found out by general testing and one that hopefully is fixable. At the moment the simulation is not aspiring to good rules of HCI design by not providing the user with option of stopping the running task (Faulkner, 1997). In light of this problem the inclusion of a *stop* button has been added to the *wait* dialog seen during the lattice generation process. This button is implemented through an action listener but will only take effect before or after a lattice has been created. This means that while a large lattice is being generated it has no useful purpose but it can stop the generation of a large number of smaller lattices. Including this button has improved the interface slightly but not enough to make a significant difference. Therefore this problem will need addressing further.

The graph displays have also been noted as problematic. These displays work fine for tiny lattice sizes but not for larger ones. This problem was therefore taken on board and necessary changes were made to the program to remedy it. After close scrutiny it was found that the `graph_size` variable (which acts as a multiplication factor to enlarge or shrink the graph so it will fit on screen) was configured as an integer. This meant that large lattice sizes, that produced a factor in the region of $0 > i < 1$, were being rounded down to 0 (and squashing the graph down to nothing). This was easily fixed by changing the variable to a `float`, and typecasting subsequent variables as needed.

```
// Draw x and y axis
graphics.drawLine((int)xc, (int)yc, (int)xc, (int)(yc -
(lattice_size * graph_size)));
graphics.drawLine((int)xc, (int)yc, (int)(xc + (lattice_size *
graph_size)), (int)yc);
```

5.4 Does it meet the Specification

Comparisons of the developed simulation against the initial specification shows that it meets nearly every criteria that was initially laid out. The only downfall is on the very first statement (see section 3.1.1) which details the ability to ‘Generate a lattice of any size’. The lattice size is changeable but the size limit is dependant upon the amount of memory available on the computer that is running the simulation. If the VM is configured properly then a 256MB PC can create a lattice up to around 5000x5000 whereas as 512MB machine can generate a 7500x7500.

5.5 Summary

The evaluation and testing has inevitably shown up some ‘bugs’ in the simulation that required attention. Some problems have had solutions applied and have successfully been resolved, but a few remain and require further improvement. Despite this evaluators have shown that the simulation is fairly simple to use and the main interface is clear and easy to learn. The program does what initially set out to do and can therefore be viewed as a success.

Bibliography

ADAMI, C., *Introduction to Artificial Life* [Internet], Percolation Chapter 7.

Available from: <http://www.krl.caltech.edu/~adami/CD1/AL.html#avida> [Accessed 10th November 2004]

ANON, 2004. *Java Programming Language* [Internet], Java Technology Overview.

Available from: <http://java.sun.com/overview.html> [Accessed 4th January 2005]

ANON, *Percolation* [Internet], Microsoft Power Point. Available from:

<http://ariadne.mse.uiuc.edu/498/498-02.pdf> [Accessed 4th January 2005]

ANON, 2002. *Lecture 3: Percolation* [Internet], Introduction to Modeling and

Simulation. Available from: http://ocw.mit.edu/NR/rdonlyres/Nuclear-Engineering/22-00JIntroduction-to-Modeling-and-SimulationSpring2002/EF797BCF-4274-4E98-A9F3-6DBC300CC67B/0/lecture_3.pdf [Accessed 15th December 2004]

ANON, 2002. *Visual Basic Overview* [Internet], Information Technology Toolbox Inc. Available from:

<http://visualbasic.ittoolbox.com/browse.asp?c=VBPeerPublishing&r=%2Fpub%2Fvb%5Foverview%2Ehtm> [Accessed 10th January 2005]

ANON, 2005. *Wikipedia The Free Encyclopedia* [Internet]. Available from:

http://en.wikipedia.org/wiki/Main_Page [Accessed 15th December 2004]

ASHCROFT, M., 2000. *Percolation Theory* [Internet], Solid State Physics.

Available from: <http://www.iu-bremen.de/course/c020009/32215/> [Accessed 4th November]

BARNES, D.J., 2000. *Object-Oriented Programming with Java: An Introduction*.

New Jersey: Prentice Hall.

BENTZ, D.P., 2000. Fibres, Percolation and Spalling of High Performance Concrete: Percolation Theory [Internet], ACI Material Journals,
Available from: <http://www.bfrl.nist.gov/861/vcctl/software/hcss/firefiber/node3.html>
[Accessed 7th November 2004]

BENTZ, D.P., & GARBOCZI, E.J., 1991. Cement and Concrete Research.
Percolation of phases in a three dimensional cement paste microstructure model
[Internet], 21(325-344). Available from:
<http://ciks.cbt.nist.gov/garbocz/paper22/paper22.html> [Accessed 7th November 2004]

BERKOWITZ, B. & EWING, R.P., 1998. Surveys in Geophysics. *Percolation Theory and Network Modelling Applications in Soil Physics* [Internet], 19 (23-72),
Available from: http://www.agon.iastate.edu/soilphysic/a677_berk.pdf [Accessed 12th December 2004]

CADENHEAD, R., 2002. *Sams Teach Yourself Java 2 in 24hours*. 3rd Ed. Indiana: Sams Publishing.

CALDARELLI, G., FRONDONI, R., GABRIELLI, A., MONTUORI, M.,
RETZLAFF, R. & RICOTTA, C., 2001. Europhysics Letters. *Percolation in Real Wild Fires* [Internet], 56 (4) pp 510-516. Available from:
<http://pil.phys.uniroma1.it/~gcalda/doc/035.pdf> [Accessed 27th November 2004]

CARTER, E.F., 1996. *Random Walk, Markov Chains and the Monte Carlo Method* [Internet], Taygeta Scientific Inc. Available from:
<http://www.taygeta.com/rwalks/rwalks.html> [Accessed 13th December 2004]

DIX, A.J., FINLAY, J., ABOWD, G.D. & BEALE, R., 2004. *Human-Computer Interaction*. 3rd Ed. New Jersey: Prentice Hall.

FAULKNER, C., 1997. *The Essence of Human-Computer Interaction*. London: Pearson Education Limited.

GONSALVES, R.J., 1999. *Java Applet: Percolation* [Internet],
Available from: <http://www.physics.buffalo.edu/gonsalves/Java/Percolation.html>
[Accessed 16th December 2004]

HARMELEN, M.V., 1999. *Object Modelling and User Interface Design: Designing Interactive Systems*. Harlow: Addison-Wesley.

KIRKPATRICK, J. & CHALMERS, K., 2003. *Go and Percolation Theory* [Internet].
Available from:
http://gobase.org/information/computers/go_and_percolation_studies.doc [Accessed
16th December 2004]

KODESOVA, R., 2003. *Percolation Theory and it's Application for Interpretation of Soil Water Retention Curves* [Internet], Department of Soil Science and Geology, Czech University of Agriculture, Prague, Czech Republic. Available from:
http://www.ictp.trieste.it/~pub_off/lectures/Ins018/19Kodesova1.pdf [Accessed 27th
November 2004]

LIANG, Y.D., 2003. *Introduction to Java Programming*. 4th Ed. New Jersey: Prentice Hall.

PALMER, I., 2001. *Essential Java 3D Fast*. London: Springer.

RUBINSTEIN, R.Y., 1981. *Stimulation and the Monte Carlo Method*. Canada: John Wiley & Sons, Inc.

SAHIMI, M..., 1994. *Applications of Percolation Theory*. London: Taylor & Francis Ltd.

SCHNEIDERMAN, B., 1998. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3rd Ed. Harlow: Addison-Wesley.

SHIRAZI, J. 2003. *Java Performance Tuning*. 2nd Ed. California: O'Reilly & Associates Inc.

STAUFFER, D. & AHARONY, A., 1994. *Introduction to Percolation Theory*. 2nd Ed. London: Taylor & Francis Ltd.

STROUSTRUP, B., 1999. *An Overview of the C++ Programming Language* [Internet], AT&T Laboratories. Available from:
<http://www.research.att.com/~bs/crc.pdf> [Accessed 7th January 2005]

THIMBLEBY, H., 1990. *User Interface Design*. New York: ACM Press Frontier Series.

TOPPER, R.Q., 2002. *Molecular Monte Carlo* [Internet], Welcome to the Molecular Monte Carlo Homepage. Available at:
<http://www.cooper.edu/engineering/chemechem/monte.html> [Accessed 15th December 2004]

WALSH, A.E., & GEHRINGER, D., 2002. *Java 3D: API Jump-Start*. New Jersey: Prentice Hall.

WEISS, M.A., 1999. *Data Structures & Algorithm Analysis in Java*. Harlow: Addison-Wesley.

Appendix A - Project Proposal

Percolation Theory

The theory of percolation deals with number and properties of clusters within a square lattice. In layman's terms a square lattice is simply a square of points, in which some of the points are connected to each other. Percolation theory is the study of these interconnected points (clusters) and the size and distribution of the clusters. Physical applications of this theory are most often referred to in diffusion, forest fires and fractal oil fields.

Project Aims/Objectives

My aims in this project are to look into the simulation and modelling of percolation theory for analytical purposes. The system modelled will be a 2D square lattice, clusters are formed by randomly making bonds between neighbouring sites. The simulation will produce analysis of the lattice's clusters and enable a user to apply percolation theory to this sort of system.

Deliverables

Throughout the development of my project I hope to be able to produce a number of deliverables.

- The generation of a program that will simulate the random growth of clusters within a square lattice. This program would need to have a suitable graphical interface to enable users to easily view and understand the result set generated.
- From this program I would hope to be able to produce some analytical feedback to the user that will be both relevant and useful in the understanding and further research of percolation theory, e.g. radius of gyration, molecular weight distribution.
- It may be possible to produce some kind of 3-dimensional representation of the lattice so that a user can view various parts of lattice in a way that is a more realistic representation of the theory.
- Documentation on this development work will be done to show the necessary techniques and tasks undertaken in order to produce the simulation as described.

Research Areas

In order to produce a good project I would need some research from a few key areas. The main area of knowledge would have to be in percolation theory itself. I will need to know how and why the theory is used to simulate and model various given situations, as well as how it is used in analysis and research. Other areas of importance would include the development of 3D programs and/or the production of suitable analysis, e.g. algorithm design, graphing methods.

Resource Requirements

Requirements needed for this project would include:

- Programming language - C++ or Java would be most suitable as these are the languages I have already covered. Java would possibly be a better choice due to its 3D capabilities.
- Desktop Computer – Any pc capable of running the development software, obviously the faster the pc the more preferable.

Appendix B - Project Work Schedule

To be completed by the student before first formal meeting and signed off at the first formal meeting. Please attach the project proposal including plan and return the completed form to the Project Co-ordinator.

Project Title: **Percolation on A Square Lattice**

Student Name: **Gareth Flowers**

Supervisors: **Louise Richards**
Bernard Rayner

Aims and Objectives:

See attached project proposal

Deliverables:

See attached project proposal

Resources:

See attached project proposal

I have considered the possible ethical issues arising from my project, and have/have not* completed and attached an ethics release form as appropriate. If I modify my project in such a way as to affect the ethical status, I understand it is my responsibility to notify my supervisor and take the necessary steps to reflect this.

Student's Signature: _____ Date:

1st Supervisor's Signature: _____(Project **is** / **is not*** Feasible and Resources available) Date:

2nd Supervisor's Signature: _____(Project **is** / **is not*** Feasible and Resources available) Date:

*Delete as appropriate

Checklist :

Attach project proposal; project plan; PDP log sheet 1; Ethics release form (if appropriate).

Appendix C - Project Progress Note

To be completed by the student before the second formal meeting and agreed and signed off at the second formal meeting.

Project Title: **Percolation on A Square Lattice**

Student Name: **Gareth Flowers**

Supervisors: **Louise Richards**
Bernard Rayner

List of Agreed Deliverables:

Generation of a program that will simulate the growth of clusters within a square lattice.

Analytical feedback of the results gathered from the simulation, presented in a meaningful manner that the user can interpret. For example listing radius of gyration and percolating clusters.

Produce documentation of test procedures and evaluation methods undertaken, along with the full development of the project, including UML class modelling and other techniques.

Amendments to specific objectives (with reasons):

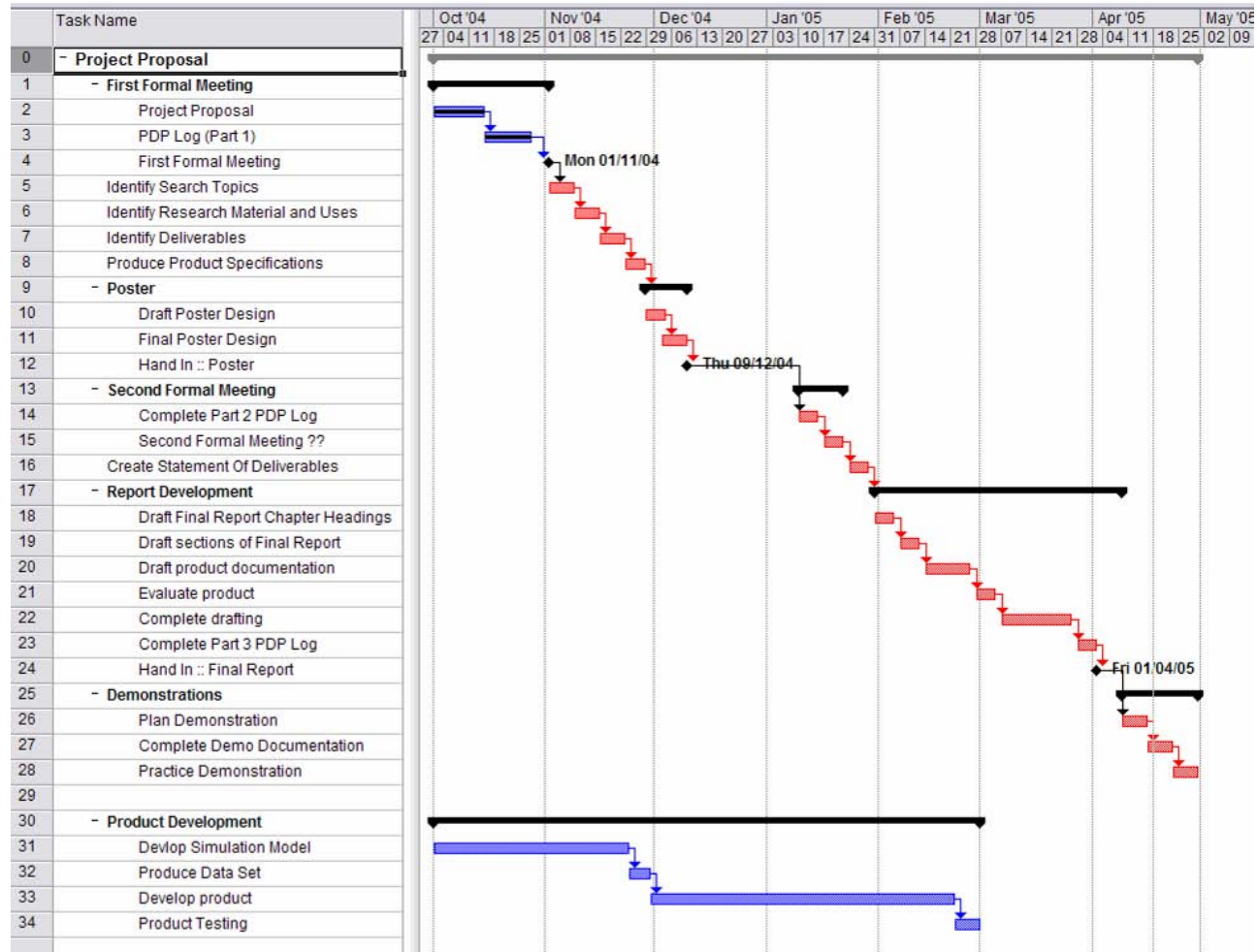
Please provide (overleaf, or on an attached piece of paper), a modified project plan showing the remaining activities (including writing of project) with their estimated durations and weightings.

Student's Signature: _____ Date:

1st Supervisor's Signature: _____ Date:

2nd Supervisor's Signature: _____ Date:

Appendix D – Project Log



Appendix E – Professional Development Logs

Log 1 – First Formal Meeting

1. Identify the knowledge and skills that you have learnt and developed during the previous 2 or more years of your course (e.g. modules studied, programming and methodology skills developed, organisational skills developed etc) that are likely to be most useful in helping to successfully complete THIS project:

I have learnt a range of skills in various modules I have previously studied. Professional Skills in my first year helped me to develop better organisational and time keeping skills. These skills were further developed and added to when I did the Group Project, from which I learnt a lot about project planning and designing. Programming modules are going to be very useful to me as I come to producing the end product of my project. I have picked up useful knowledge about Object Oriented Software and Graphical User Interfaces that will help a great deal in designing a final product. Thankfully, I have previously done both Java and C++ modules so I have got a good choice between programming languages if it comes to it.

2. Identify the knowledge and skills that you have learnt and developed **outside of your course** (e.g. previous or part-time employment, previous courses, voluntary work, hobbies, family responsibilities etc) that are likely to be most useful in helping you to successfully complete this project:

I have had a regular part time job for a long time so I grown accustomed to being good at timekeeping and working to schedules. My sandwich year was spent working at an engineering firm at which I developed a vast amount of my personal and professional skills. I have learnt to manage small projects that were kept to strict deadlines as well as controlling other people. I have developed a greater knowledge of programming skills that may be useful to any further development. All the work I have done to date has helped me to become more resourceful and self-motivated in any work that I do, I can work efficiently within time limits and under pressure.

3. Identify any knowledge and skills that you feel you do not currently have or are weak in that you feel you might need to help you successfully complete this project. After discussion with your supervisor PLAN how you might tackle these shortcomings, within the constraints of your final year studies (available timeframe, other modules and responsibilities):

Obviously the first area of knowledge that I have a lack of is percolation theory. The knowledge of this area will be important to the understanding of the overall project – although an advanced knowledge is not needed. Programming skills will have to be enhanced in order to produce a successful product. My knowledge of Java far out ways my knowledge of C++ so deciding to go down the C++ might prove risky if I cannot improve my knowledge of the language quickly.

4. BRIEFLY comment on the following ETHICAL considerations that you have assessed in relation to this project –

- a) The remit and feasibility of the study.
- b) The nature of recruitment and participation of participants
- c) Possible harm to participants, researchers and facilities to deal with it
- d) Procedures for providing explanation to participants – including information sheet if appropriate.
- e) Procedures for obtaining informed consent from participants or where necessary from their parents/guardians
- f) Procedures for respecting confidentiality
- g) Procedures for operating with data protection legislation
- h) Procedures for anonymous report writing.
- i) Safety requirements – consideration of Health & Safety.

Users of the final product will be using a desktop pc some normal computer health & safety is required.
- j) Insurance and Indemnity arrangements
- k) Risk assessment
- l) Resource allocation

Log 2 – Second Formal Meeting

1. The Gantt Chart/Project Plan included in your Project Proposal agreed at your First Formal meeting and section 3 of your Professional Development Log 1 indicated the GOALS that you set yourself **for this stage** in your project. Have you achieved these goals?

I am well on my way to producing the simulation program as my final product. I have the simulation constructed and am working on producing further results that can be analysed in order to perform the more advanced analysis I would like. This part of the project is a few weeks behind schedule and would preferably be finished by the beginning of March.

As of yet I have not achieved a suitably appropriate graphical interface that I hoped would be a more favoured outcome of the program. Although not completely necessary at this stage, it was a feature that I wanted to build in to the program for better ease of use and graphical representation of the results.

2. If not, why not? (Give concrete reasons rather than ‘excuses’).
How do the goals need to be readjusted now? How will you restructure your time/use of time to ensure that you can achieve the restructured goals before the end of the project? What help or resources would positively contribute to you achieve your readjusted goals? (e.g. human resources – staff, colleagues; reading and research; generic LMU resources such as Skills for Learning; web based resources)

The simulation has taken me a while to due to lack of knowledge about the JAVA language. It has taken me longer than anticipated to gather the information and knowledge required to achieve the right results, and although I am progressing quite quickly with the program now most of my time was taken up in the first few months of development.

Due to the amount of time taken producing the program I need to careful look at the amount of time left to test and evaluate the project, as well as produce the appropriate documentation. It is very important to get the program finished off as soon as possible so I can spend the remaining 3 or 4 weeks documenting the project, 2 weeks or so to be used in the evaluation stage (an extra week more than originally planned).

3. If yes, then what challenges did you overcome to achieve the set goals? Do you think there will be any future obstacles to successfully achieving all your goals by the end of the project? Would it be appropriate to readjust your goals, perhaps to set your targets higher?

Most of my goals are still ongoing, aside from the completion of the Project Proposal and the Poster. The main obstacles to get over in the near future are the completion of the project, and then evaluation and testing. I feel that this is a big part of the project and I want to concentrate a lot of my time and effort into doing this.

Log 3 – Final Project Log

1. What have you done well in your project so far? How could you evidence that achievement? (Be specific, don't just refer to successful completion of project)

I think the coding up of the lattice simulation has been very successful. I have been able to produce a model of a square lattice, both in 2 and 3-dimensional format, that properly applies the theory of percolation. This development has enabled me to learn a great deal about the Java programming language, especially in the realms of 3-dimensional environments. The evidence for this is justified in the visual elements of the simulation program.

2. What could you have improved on in your performance on your project? HOW could you have achieved that improvement? (give concrete plans, with small achievable steps – general resolutions to do better, work harder, rarely achieve much)

Improvements could have been made to the amount of written work and research done whilst developing the program. I put too much effort into the initial construction of the program and neglected slightly the amount of work needed to produce the final report. Sticking to the initial project plan would have been the best thing to do whilst producing the project. This would have enabled me to hit clear deadlines that could have pushed the development of the program to a slightly higher level.

Appendix F – Project Supervision Meeting Records

Meeting 1

Date Of Meeting: 12th October 2004

Students Name: Gareth Flowers

Supervisor: Louise Richards

Agenda: Overview of Percolation Theory
What am I going to be doing??

Details of Discussion

Discussed the theory of percolation
General overview of the project

Actions and Decisions

Research into percolation

Supervisor Signature:

Meeting 2

Date Of Meeting: 19th October 2004

Students Name: Gareth Flowers

Supervisor: Louise Richards

Agenda: Ideas for Project

Details of Discussion

Talked about ideas suitable for project

Discussed research material found.

Actions and Decisions

Draft project proposal

Draft Project timeplan

Supervisor Signature:

Meeting 3

Date Of Meeting: 2nd November 2004

Students Name: Gareth Flowers

Supervisor: Louise Richards

Agenda: Set up meeting for First Formal
Ensure draft proposal is acceptable

Details of Discussion

Checked draft proposal – few amendments needed

Choice of coding language – Java because of 3D capabilities

Arranged First Formal date

Actions and Decisions

More research needed

Experimental coding

Supervisor Signature:

Meeting 4

Date Of Meeting: 14th December 2004

Students Name: Gareth Flowers

Supervisor: Louise Richards

Agenda: Assess current state of project
Prepare for second Formal

Details of Discussion

Project simulation coding tips and assistance
Discussed the direction in which to develop code
Poster design ideas

Actions and Decisions

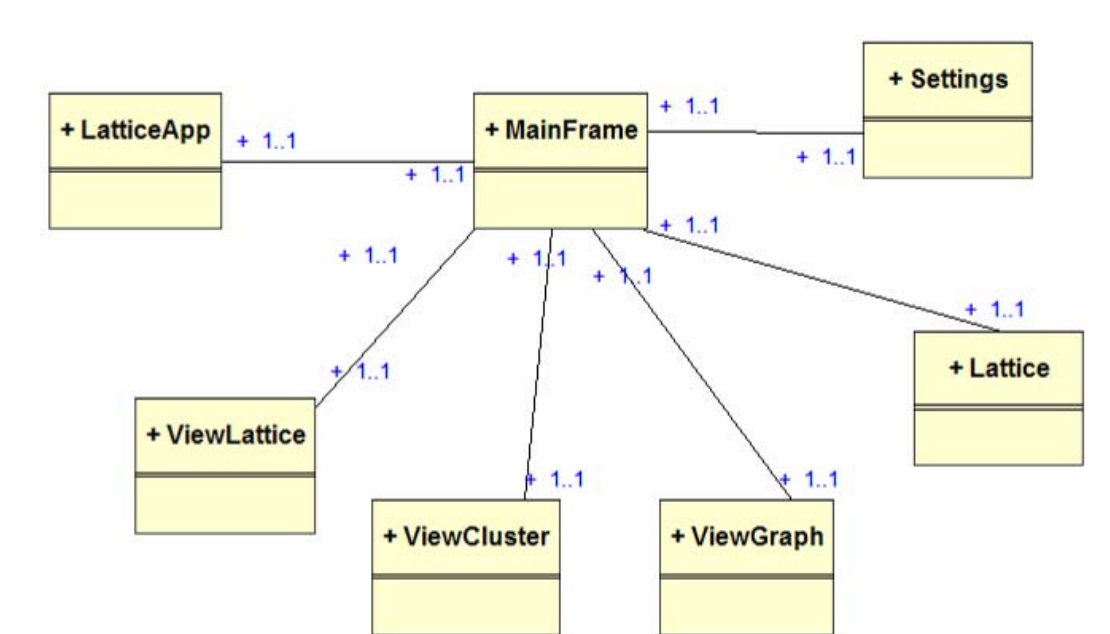
Prepare for second formal meeting
Develop Poster

Supervisor Signature:









































Appendix G – Design Diagrams





UML Class Diagrams

Skeletal View









































































Lattice and Lattice App

Lattice	
	 cluster : double[][]
	 is_3d : boolean
	 lattice_size : int
	 lattice_total_nodes : int
	 nodes : int[][]
	 Lattice() : void
	 generate() : void
	 get_nodes() : int[][]
	 get_results() : double[][]
	 add_links() : void
	 find_node() : int
	 find_node() : int
	 find_x() : int
	 find_y() : int
	 find_z() : int
	 jblnit() : void
	 rand() : int
	 replace_nodes() : void
	 valid_coordinate() : boolean
	 valid_coordinate() : boolean

LatticeApp	
	 main() : void
	 LatticeApp() : void

MainFrame and Settings

MainFrame	
	 array_nodes : int[][]
	 array_results : double[][]
	 is_3d : boolean
	 lattice_numbers : int
	 lattice_size : int
	 percolating_cluster : int
	 view_cluster : ViewCluster
	 wait_dialog : Dialog
	 actionPerformed() : void
	 itemStateChanged() : void
	 MainFrame() : void
	 stateChanged() : void
	 create_views() : void
	 empty_objects() : void
	 get_lattice_info() : void
	 get_lattice_results() : void
	 hide_gui() : void
	 load_file() : void
	 output() : void
	 save_file() : void
	 show_gui() : void

Settings	
	 apply : JButton
	 cancel : JButton
	 change : boolean
	 is_3d : JCheckBox
	 lattice_numbers : int
	 lattice_size : int
	 ls : JTextField
	 nl : JTextField
	 actionPerformed() : void
	 Settings() : void
	 get_changed() : boolean
	 get_is_3d() : boolean
	 get_num() : int
	 get_size() : int
	 open() : void

ViewCluster and ViewGraph

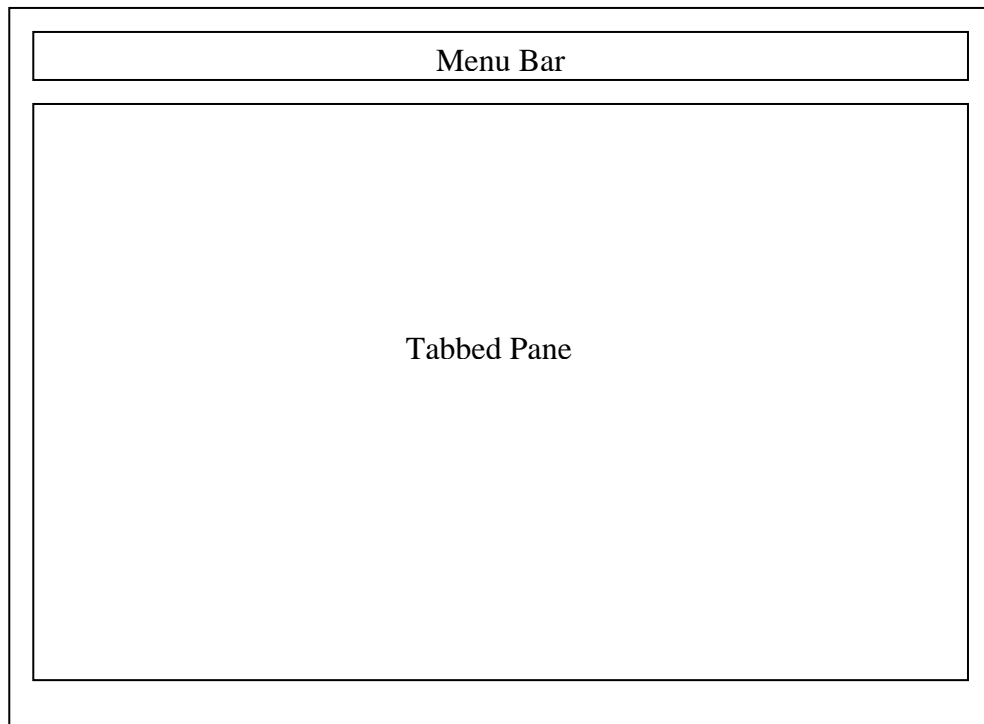
ViewCluster	ViewGraph
<ul style="list-style-type: none"> array_nodes : int[][] cluster : int is_3d : boolean lattice_size : int 	<ul style="list-style-type: none"> array_results : double[][] graph_size : float graph_type : int graphics : Graphics lattice_size : int list_array_results : ArrayList
<ul style="list-style-type: none"> ViewCluster() : void build_view() : BranchGroup addLights() : void build_content() : BranchGroup find_x() : int find_y() : int find_z() : int 	<ul style="list-style-type: none"> paint() : void ViewGraph() : void set() : void axis() : void draw_graph() : void draw_string_vert() : void

ViewLattice

ViewLattice
<ul style="list-style-type: none"> array_nodes : int[][] graphics : Graphics lattice_size : int percolating_cluster : int
<ul style="list-style-type: none"> mouseClicked() : void mouseEntered() : void mouseExited() : void mousePressed() : void mouseReleased() : void paint() : void ViewLattice() : void draw_lattice() : void get_cluster_number() : int set() : void set_cluster_number() : void display_x() : int display_y() : int find_x() : int find_y() : int

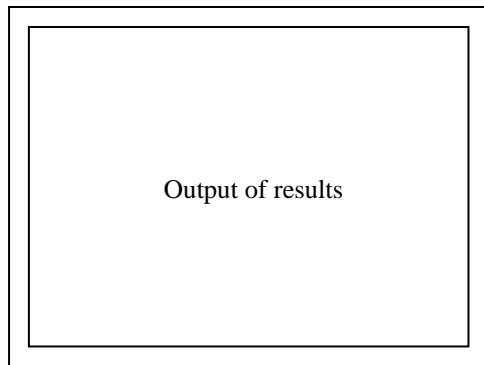
Layout Designs

Main Screen



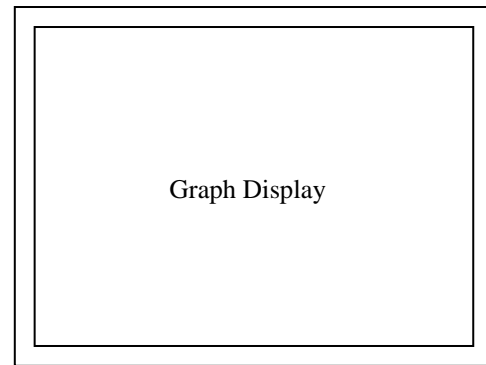
The initial page will open with a menu bar across the top and a tabbed pane in the centre. The tab pane will navigate through the four views – output, graphs, 2d view and 3d view.

Tab Pane Layouts



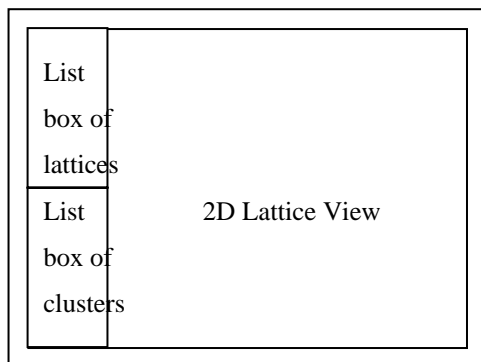
Tabbed pane output view

Full pane is taken up by text area of results



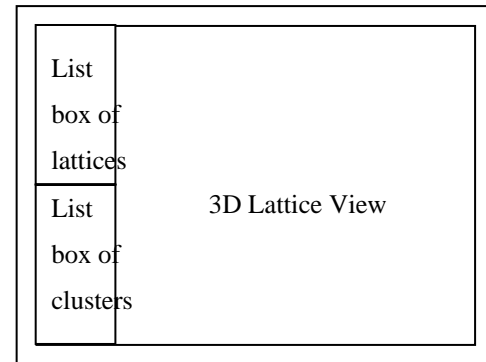
Tabbed pane Graph view

Full pane is taken up with a graph



Tabbed pane 2D view

Two list boxes down left hand side for lattices and clusters. When lattice is clicked the view updates to show the lattice. When cluster is clicked the cluster is highlighted within the lattice.



Tabbed pane 3D view

Two list boxes down the left hand side. When a lattice is selected the cluster list updates with the lattices associated clusters. When a cluster is selected it is shown in 3D in the main view. The mouse can be used to rotate round the 3D cluster.

Appendix H – Source Code

Lattice.java

```
// Lattice

// Import Required Libraries
import java.util.ArrayList;

/**
 * <p>Title: Lattice</p>
 * <p>Description: Generates a new lattice and adds random clusters
to it.</p>
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
 * <p>Company: Leeds Metropolitan University</p>
 * @author Gareth Flowers
 * @version 1.0
 */
class Lattice
{
    public Lattice()
    {
        try {
            jbInit();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    final private int lattice_incoming_nodes = 2; // Total incoming
nodes per node
    private int lattice_size; // Size of lattice in nodes
    private int lattice_total_nodes;
    private boolean is_3d;
    private int[][] nodes;
    private double[][] cluster;
    private ArrayList clusters = new ArrayList(0);

    /**
     * When run creates a new lattice and adds links to it
     * @param size int
     * @param d3 boolean
     */
    protected void generate(int size, boolean d3)
    {
        lattice_size = size;
        lattice_total_nodes = size * size;
        is_3d = d3;
        if (is_3d) {
            lattice_total_nodes *= size;
        }
    }
}
```

```

        nodes = new int[lattice_total_nodes][3];

        // Creates new lattice
        for(int i = lattice_total_nodes; --i >= 0; ) {
            // Set up nodes with empty data (-1)
            nodes[i][0] = -1;
            nodes[i][1] = -1;
            nodes[i][2] = 0;
            clusters.add(new Integer(i));
        }

        // Add links between nodes
        add_links();
    }

    /**
     * Returns the nodes array
     *
     * <br> nodes[0] = Node Number Connected To
     * <br> nodes[1] = Cluster Number
     *
     * @return int[][]
     */
    protected int[][] get_nodes()
    {
        return nodes;
    }

    /**
     * Returns a random number between low and high
     * @param low int
     * @param high int
     * @return int
     */
    private static int rand(int low, int high)
    {
        return (int)(Math.random() * (high - low + 1) + low);
    }

    /**
     * Return the X coordinate for the position number
     * @param node int
     * @return int
     */
    private int find_x(int node)
    {
        return node % lattice_size;
    }

    /**
     * Return the Y coordinate for the position number
     * @param node int
     * @return int
     */
    private int find_y(int node)
    {

```

```
        if(is_3d) {
            return(node -
                ((lattice_size * lattice_size) *
                (node / (lattice_size * lattice_size)))) /
                lattice_size;
        } else {
            return node / lattice_size;
        }
    }

/**
 * Return the Z coordinate for the position number
 * @param node int
 * @return int
 */
private int find_z(int node)
{
    return node / (lattice_size * lattice_size);
}

/**
 * Return the node number from the x,y coordinates
 * @param point_x int
 * @param point_y int
 * @return int
 */
private int find_node(int point_x, int point_y)
{
    return(point_y * lattice_size) + point_x;
}

/**
 * Return the node number from the x,y coordinates
 * @param point_x int
 * @param point_y int
 * @param point_z int
 * @return int
 */
private int find_node(int point_x, int point_y, int point_z)
{
    return(point_z * lattice_size * lattice_size) + (point_y *
lattice_size) +
        point_x;
}

/**
 * Check for a valid coordinate
 * @param px int
 * @param py int
 * @param node_from int
 * @return boolean
 */
private boolean valid_coordinate(int px, int py, int node_from)
{
    int node_to = find_node(px, py);
```

```
        try {
            if((px >= lattice_size) || (py >= lattice_size) || (py <
0) ||
                (px < 0) || (nodes[node_to][0] == node_from) ||
                (nodes[node_to][2] >= lattice_incoming_nodes) ||
                ((nodes[node_from][1] == nodes[node_to][1]) &&
                (nodes[node_to][1] != -1))) {
                return false;
            } else {
                return true;
            }
        } catch(Exception e) {
            return false;
        }
    }

    /**
     * Check for a valid coordinate
     * @param px int
     * @param py int
     * @param pz int
     * @param node_from int
     * @return boolean
     */
    private boolean valid_coordinate(int px, int py, int pz, int
node_from)
    {
        int node_to = find_node(px, py, pz);

        try {
            if(((nodes[node_from][1] == nodes[node_to][1]) &&
                (nodes[node_to][1] != -1)) || (px >= lattice_size) ||
                (py >= lattice_size) || (py < 0) || (px < 0) ||
                (nodes[node_to][0] == node_from) ||
                (nodes[node_to][2] >= lattice_incoming_nodes)) {
                return false;
            } else {
                return true;
            }
        } catch(Exception e) {
            return false;
        }
    }

    /**
     * Returns the cluster results array
     *
     * <br> cluster[0] = Cluster Number
     * <br> cluster[1] = Weight (number nodes)
     * <br> cluster[2] = Mass X
     * <br> cluster[3] = Mass Y
     * <br> cluster[4] = Mass Z
     * <br> cluster[5] = Width
     * <br> cluster[6] = Height
     * <br> cluster[7] = Depth
     * <br> cluster[8] = Radius Gyration
     * <br> cluster[9] = Percolating
     *
     */
```



```
* @param nodes int[][]
* @param size int
* @return double[][]
*/
protected double[][] get_results(int[][] nodes, int size)
{
    lattice_size = size;
    clusters.clear();

    for(int i = nodes.length; --i >= 0; ) {
        if(!clusters.contains(new Integer(nodes[i][1]))) {
            clusters.add(new Integer(nodes[i][1]));
        }
    }

    int cluster_size = clusters.size();
    cluster = new double[cluster_size][10];

    for(int i = cluster_size; --i >= 0; ) {
        int total = 0, number_nodes = 0;
        int min_x = -1, min_y = -1, min_z = -1;
        int max_x = -1, max_y = -1, max_z = -1;
        int point_x = 0, point_y = 0, point_z = 0;
        double sum_x = 0, sum_y = 0, sum_z = 0;
        double mass_x = 0, mass_y = 0, mass_z = 0;
        cluster[i][0] = ((Integer)clusters.get(i)).intValue();

        for(int j = nodes.length; --j >= 0; ) {
            if(nodes[j][1] == (int)cluster[i][0]) {
                point_x = find_x(j);
                point_y = find_y(j);
                point_z = find_z(j);

                sum_x += point_x;
                sum_y += point_y;
                sum_z += point_z;
                number_nodes++;

                if(max_x == -1) {
                    min_x = point_x;
                    min_y = point_y;
                    min_z = point_z;
                    max_x = point_x;
                    max_y = point_y;
                    max_z = point_z;
                }

                if(point_x > max_x) {
                    max_x = point_x;
                } else if(point_x < min_x) {
                    min_x = point_x;
                }
                if(point_y > max_y) {
                    max_y = point_y;
                } else if(point_y < min_y) {
                    min_y = point_y;
                }
                if(point_z > max_z) {
                    max_z = point_z;
                } else if(point_z < min_z) {
```

```
        min_z = point_z;
    }
}

mass_x = sum_x / number_nodes;
mass_y = sum_y / number_nodes;
mass_z = sum_z / number_nodes;
cluster[i][1] = number_nodes;
cluster[i][2] = mass_x;
cluster[i][3] = mass_y;
cluster[i][4] = mass_z;
cluster[i][5] = max_x - min_x + 1;
cluster[i][6] = max_y - min_y + 1;
cluster[i][7] = max_z - min_z + 1;

// Radius of Gyration
for(int j = nodes.length; --j >= 0; ) {
    if(nodes[j][1] == (int)cluster[i][0]) {
        total += ((find_x(j) - mass_x) * (find_x(j) -
mass_x)) +
                ((find_y(j) - mass_y) * (find_y(j) -
mass_y));
        if(is_3d) {
            total += ((find_z(j) - mass_z) * (find_z(j) -
mass_z));
        }
    }
}
// Square results for radius of gyration
cluster[i][8] = Math.sqrt(total / number_nodes);

// Record a '1' if a percolating cluster
// otherwise '0' if not
if((min_x == 0) && (max_x == lattice_size - 1)) ||
((min_y == 0) &&
    (max_y == lattice_size - 1)) {
    cluster[i][9] = 1;
} else {
    cluster[i][9] = 0;
}
clusters.clear();

return cluster;
}

/**
 * Creates new links between nodes and assigns the relevant
cluster number
 */
private void add_links()
{
    // function to search random nodes and add links
    int cluster_number = 0;

    while(clusters.size() > 0) {
        // Find a node from list of nodes left to have links
added
```

```
int node_from_index = rand(0, clusters.size() - 1);
int node_from =
((Integer)clusters.get(node_from_index)).intValue();

ArrayList valid_points = new ArrayList(0);

int point_x = find_x(node_from);
int point_y = find_y(node_from);

// Check for valid surrounding nodes
if(is_3d) {
    int point_z = find_z(node_from);
    if(valid_coordinate(point_x + 1, point_y, point_z,
node_from)) {
        valid_points.add(new Integer(find_node(point_x +
1, point_y,
point_z)));
    }
    if(valid_coordinate(point_x - 1, point_y, point_z,
node_from)) {
        valid_points.add(new Integer(find_node(point_x -
1, point_y,
point_z)));
    }
    if(valid_coordinate(point_x, point_y + 1, point_z,
node_from)) {
        valid_points.add(new Integer(find_node(point_x,
point_y + 1,
point_z)));
    }
    if(valid_coordinate(point_x, point_y - 1, point_z,
node_from)) {
        valid_points.add(new Integer(find_node(point_x,
point_y - 1,
point_z)));
    }
    if(valid_coordinate(point_x, point_y, point_z + 1,
node_from)) {
        valid_points.add(new Integer(find_node(point_x,
point_y,
point_z + 1)));
    }
    if(valid_coordinate(point_x, point_y, point_z - 1,
node_from)) {
        valid_points.add(new Integer(find_node(point_x,
point_y,
point_z - 1)));
    }
} else {
    if(valid_coordinate(point_x + 1, point_y, node_from))
    {
        valid_points.add(new Integer(find_node(point_x +
1, point_y)));
    }
    if(valid_coordinate(point_x - 1, point_y, node_from))
    {
        valid_points.add(new Integer(find_node(point_x -
1, point_y)));
    }
}
```

```
        if(valid_coordinate(point_x, point_y + 1, node_from))
        {
            valid_points.add(new Integer(find_node(point_x,
point_y + 1)));
        }
        if(valid_coordinate(point_x, point_y - 1, node_from))
        {
            valid_points.add(new Integer(find_node(point_x,
point_y - 1)));
        }
    }

    if(valid_points.size() > 0) {
        // If there is a valid node
        int node_to = ((Integer)valid_points.get(rand(0,
            valid_points.size() - 1))).intValue();

        nodes[node_from][0] = node_to;

        // Assign a cluster number
        if((nodes[node_from][1] > -1) && (nodes[node_to][1] >
-1)) {
            if(nodes[node_to][1] > nodes[node_from][1]) {
                replace_nodes(nodes[node_to][1], node_from);
            } else if(nodes[node_to][1] <
nodes[node_from][1]) {
                replace_nodes(nodes[node_from][1], node_to);
            }
        } else if((nodes[node_from][1] > -1) &&
            (nodes[node_to][1] == -1)) {
            nodes[node_to][1] = nodes[node_from][1];
        } else if((nodes[node_from][1] == -1) &&
            (nodes[node_to][1] > -1)) {
            nodes[node_from][1] = nodes[node_to][1];
        } else {
            cluster_number++;
            nodes[node_from][1] = cluster_number;
            nodes[node_to][1] = cluster_number;
        }
        nodes[node_to][2]++;
    } else {
        // if there is no valid node
        nodes[node_from][0] = -666;
    }

    // Remove node number from list of nodes left to add
links to    clusters.remove(node_from_index);
    }
}

/**
 * Replaces all instances of 'from' with 'to' in the 'nodes'
array
 * @param from int
 * @param to int
 */
private void replace_nodes(int from, int to)
{


---


```

```
        for(int i = lattice_total_nodes; --i >= 0; ) {
            if(nodes[i][1] == from) {
                nodes[i][1] = nodes[to][1];
            }
        }

    }

    private void jbInit() throws Exception
    {
    }
}
```

LatticeApp.java

```
// LatticeApp

// Import Required Libraries
import javax.swing.UIManager;

/**
 * <p>Title: Lattice</p>
 * <p>Description: Starts a new instance of Lattice application</p>
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
 * <p>Company: Leeds Metropolitan University</p>
 * @author Gareth Flowers
 * @version 1.0
 */
final class LatticeApp
{
    /**
     * Starts a new instance of 'MainFrame'
     */
    private LatticeApp()
    {
        try {
            // UI Manager used to set look of program to the current
OS
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            // Create new MainFrame
            new MainFrame();
        } catch(Exception exception) {
            exception.printStackTrace();
        }
    }

    /**
     * Main
     * @param arguments String[]
     */
    public static void main(String[] arguments)
```

```
{  
    // Create application frame  
    new LatticeApp();  
}
```

MainFrame.java

```
// MainFrame  
  
// Import Required Libraries  
import java.awt.BorderLayout;  
import java.awt.Button;  
import java.awt.Dialog;  
import java.awt.FileDialog;  
import java.awt.Font;  
import java.awt.GridLayout;  
import java.awt.Label;  
import java.awt.List;  
import java.awt.Menu;  
import java.awt.MenuBar;  
import java.awt.MenuItem;  
import java.awt.MenuShortcut;  
import java.awt.Panel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.awt.event.KeyEvent;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.FileReader;  
import java.io.PrintStream;  
import java.text.DecimalFormat;  
import java.util.ArrayList;  
  
import javax.swing.JDialog;  
import javax.swing.JFrame;  
import javax.swing.JScrollPane;  
import javax.swing.JTabbedPane;  
import javax.swing.JTextArea;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
  
/**  
 * <p>Title: MainFrame</p>  
 * <p>Description: Main frame for all views</p>  
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>  
 * <p>Company: Leeds Metropolitan University</p>  
 * @author GarethFlowers  
 * @version 1.0  
 */
```

```

class MainFrame extends JFrame implements ActionListener,
ItemListener,
    ChangeListener
{
    // Create new variables and objects
    private int lattice_size;
    private int lattice_numbers;
    private int percolating_cluster;
    private boolean is_3d;
    private int[][] array_nodes;
    private double[][] array_results;
    private ArrayList list_array_nodes = new ArrayList(0);
    private ArrayList list_array_results = new ArrayList(0);
    private Lattice lattice = new Lattice();
    private ViewCluster view_cluster;
    private ViewLattice view_lattice = new ViewLattice();
    private ViewGraph view_graph_1 = new ViewGraph();
    private ViewGraph view_graph_2 = new ViewGraph();

    // GUI Objects
    private JTextArea view_output = new JTextArea();
    private JScrollPane scroll = new JScrollPane(view_output,
                                                JScrollPane.
VERTICAL_SCROLLBAR_ALWAYS,
                                                JScrollPane.
HORIZONTAL_SCROLLBAR_ALWAYS);
    private Panel list_panel = new Panel(new GridLayout(2, 1));
    private Settings settings = new Settings(this, 10, 1);
    private List list_lattices = new List();
    private List list_clusters = new List();
    private MenuBar menu_bar = new MenuBar();
    private Menu menu_file = new Menu("File");
    private MenuItem menu_file_new = new MenuItem("Generate New
Lattices",
                                                new
MenuShortcut(KeyEvent.VK_N));
    private MenuItem menu_file_load = new MenuItem("Load Results",
new MenuShortcut(KeyEvent.VK_L));
    private MenuItem menu_file_save = new MenuItem("Save Results",
new MenuShortcut(KeyEvent.VK_S));
    private MenuItem menu_file_exit = new MenuItem("Exit",
new MenuShortcut(KeyEvent.VK_E));
    private Menu menu_view = new Menu("View");
    private MenuItem menu_view_cluster = new MenuItem("Cluster",
new MenuShortcut(KeyEvent.VK_C));
    private MenuItem menu_view_lattice = new MenuItem("Lattice");
    private MenuItem menu_view_output = new MenuItem("Show Output",
new MenuShortcut(KeyEvent.VK_O));
    private Menu menu_view_graph = new Menu("Graphs");
    private MenuItem menu_view_graph_1 = new MenuItem(
"Radius of Gyration / Length of Cluster",
new MenuShortcut(KeyEvent.VK_1));
    private MenuItem menu_view_graph_2 = new MenuItem("Height /
Width",
new MenuShortcut(KeyEvent.VK_2));

    private JTabbedPane main_tab_pane = new JTabbedPane();

```

```
private JTabbedPane graph_tab_pane = new JTabbedPane();
private String spacer = " ";
private Dialog wait_dialog;
private Button wait_stop = new Button("Stop Generating!");

/**
 * Builds a new instance of the main frame user interface
 */
public MainFrame()
{
    try {
        // Add Menu Items
        menu_file_new.addActionListener(this);
        menu_file.add(menu_file_new);
        menu_file.addSeparator();
        menu_file_load.addActionListener(this);
        menu_file.add(menu_file_load);
        menu_file_save.addActionListener(this);
        menu_file.add(menu_file_save);
        menu_file.addSeparator();
        menu_file_exit.addActionListener(this);
        menu_file.add(menu_file_exit);
        menu_bar.add(menu_file);
        menu_view_graph_1.addActionListener(this);
        menu_view_graph.add(menu_view_graph_1);
        menu_view_graph_2.addActionListener(this);
        menu_view_graph.add(menu_view_graph_2);
        menu_view.add(menu_view_graph);
        menu_view_lattice.addActionListener(this);
        menu_view.add(menu_view_lattice);
        menu_view_cluster.addActionListener(this);
        menu_view.add(menu_view_cluster);
        menu_view.addSeparator();
        menu_view_output.addActionListener(this);
        menu_view.add(menu_view_output);
        menu_bar.add(menu_view);

        // Create wait dialog
        wait_dialog = new JDialog(this, "");
        Panel wait_panel = new Panel(new GridLayout(2, 1));
        Label label = new Label(
            "Please wait while the lattice information is
being generated...");
        label.setFont(new Font("Arial", Font.BOLD, 14));
        wait_panel.add(label);
        // Added after evaluation
        wait_stop.addActionListener(this);
        wait_panel.add(wait_stop);
        wait_dialog.add(wait_panel);
        wait_dialog.pack();

        // Add lists down the left hand side for lattices and
clusters
        Panel lattice_panel = new Panel(new BorderLayout());
        lattice_panel.add(new Label("Lattices"),
BorderLayout.NORTH);
        lattice_panel.add(list_lattices);
        list_lattices.addItemListener(this);
        Panel clusters_panel = new Panel(new BorderLayout());
```



```

        clusters_panel.add(new Label("Clusters"),
BorderLayout.NORTH);
        clusters_panel.add(list_clusters);
        list_clusters.addItemListener(this);
        list_panel.add(lattice_panel);
        list_panel.add(clusters_panel);
        add(list_panel, BorderLayout.WEST);
        list_panel.setVisible(false);

        // Create tabs for different views
        main_tab_pane.addTab(spacer + "Output" + spacer, scroll);
        main_tab_pane.addTab(spacer + "Graph" + spacer,
graph_tab_pane);
        graph_tab_pane.addTab(spacer + "Radius of Gyration vs
Length" +
                                spacer,
                                view_graph_1);
        graph_tab_pane.addTab(spacer + "Height vs Width" +
spacer,
                                view_graph_2);
        main_tab_pane.addTab(spacer + "2D Lattice" + spacer,
view_lattice);
        main_tab_pane.addTab(spacer + "3D Cluster" + spacer,
view_cluster);
        main_tab_pane.setSelectedIndex(0);
        main_tab_pane.addChangeListener(this);
        add(main_tab_pane);

        // Apply settings to the frame
        setMenuBar(menu_bar);
        setTitle("Percolation Theory - Square Lattice
Simulation");
        setResizable(true);
        setExtendedState(MAXIMIZED_BOTH);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        get_lattice_info(10, 1, false);
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

/**
 * Perform action when list selection changes
 * @param itemevent ItemEvent
 */
public void itemStateChanged(ItemEvent itemevent)
{
    int index = 0;

    // Run when an item in one of the list boxes is selected
    if (itemevent.getSource() == list_lattices) {
        index = list_lattices.getSelectedIndex();
        list_clusters.select(0);

        array_nodes = (int[][][])list_array_nodes.get(index);
        get_lattice_results(index);
    }
}

```

```
        // Update current view with new lattice
        view_lattice.set(lattice_size, percolating_cluster,
array_nodes);
        view_cluster = new ViewCluster(array_nodes, lattice_size,
(int)array_results[0][0],
is_3d);
    } else if(itemevent.getSource() == list_clusters) {
        index = list_clusters.getSelectedIndex();
        // Update current view with new lattice

view_lattice.set_cluster_number((int)array_results[index][0]);
        view_cluster = new ViewCluster(array_nodes, lattice_size,
(int)array_results[index][0], is_3d);
    }

    int current_view = main_tab_pane.getSelectedIndex();
    main_tab_pane.remove(3);
    main_tab_pane.add(view_cluster, spacer + "3D Cluster" +
spacer);
    main_tab_pane.setSelectedIndex(current_view);

}

/**
 * Run when a button or menu item is clicked
 * @param actionevent ActionEvent
 */
public void actionPerformed(ActionEvent actionevent)
{
    // Run when a menu item is selected
    if(actionevent.getSource() == menu_file_exit) {
        System.exit(0);
    } else if(actionevent.getSource() == menu_file_new) {
        settings.open();
        if(settings.get_changed()) {
            get_lattice_info(settings.get_size(),
settings.get_num(),
settings.get_is_3d());
        }
    } else if(actionevent.getSource() == menu_file_save) {
        save_file();
    } else if(actionevent.getSource() == menu_file_load) {
        load_file();
    } else if(actionevent.getSource() == menu_view_output) {
        main_tab_pane.setSelectedIndex(0);
    } else if(actionevent.getSource() == menu_view_lattice) {
        main_tab_pane.setSelectedIndex(2);
    } else if(actionevent.getSource() == menu_view_cluster) {
        main_tab_pane.setSelectedIndex(3);
    } else if(actionevent.getSource() == menu_view_graph_1) {
        main_tab_pane.setSelectedIndex(1);
        graph_tab_pane.setSelectedIndex(0);
    } else if(actionevent.getSource() == menu_view_graph_2) {
        main_tab_pane.setSelectedIndex(1);
        graph_tab_pane.setSelectedIndex(1);
    } else if(actionevent.getSource() == wait_stop) {
        System.exit(0);
    }
}
```

```
}

/**
 * Run when the tabbed pane is changed
 * @param changeevent ChangeEvent
 */
public void stateChanged(ChangeEvent changeevent)
{
    JTabbedPane pane = (JTabbedPane)changeevent.getSource();
    int index = list_clusters.getSelectedIndex();

    if(index < 0) {
        index = 0;
    }
    if(pane.getSelectedIndex() == 3) {
        view_cluster = new ViewCluster(array_nodes, lattice_size,
(int)array_results[index][0], is_3d);
        view_cluster.repaint();
        list_panel.setVisible(true);
    } else if(pane.getSelectedIndex() == 2) {
        list_panel.setVisible(true);
    } else {
        list_panel.setVisible(false);
    }
}

/**
 * Empties cuurent arraylists and lists
 */
private void empty_objects()
{
    percolating_cluster = -1;
    list_lattices.removeAll();
    list_clusters.removeAll();
    list_array_nodes.clear();
    list_array_results.clear();
}

/**
 * Hides all GUI before lattice is generated
 */
private void hide_gui()
{
    wait_dialog.setLocation((getWidth() / 2) -
(wait_dialog.getWidth() / 2),
                        (getHeight() / 2) -
                        (wait_dialog.getHeight() / 2));
    wait_dialog.setFont(new Font("Arial", Font.BOLD, 12));
    wait_dialog.setResizable(false);
    wait_dialog.setVisible(true);
    menu_file_new.setEnabled(false);
    menu_file_load.setEnabled(false);
    menu_file_save.setEnabled(false);
    menu_view.setEnabled(false);
}
```

```
list_panel.setVisible(false);
main_tab_pane.setVisible(false);
this.repaint();
}

/**
 * Shows all GUI after lattice is generated
 */
private void show_gui()
{
    menu_file_new.setEnabled(true);
    menu_file_load.setEnabled(true);
    menu_file_save.setEnabled(true);
    menu_view.setEnabled(true);
    list_clusters.select(0);
    list_lattices.select(0);
    main_tab_pane.setVisible(true);
    main_tab_pane.setSelectedIndex(0);

    if(is_3d) {
        main_tab_pane.setEnabledAt(2, false);
        menu_view_lattice.setEnabled(false);
    } else {
        main_tab_pane.setEnabledAt(2, true);
        menu_view_lattice.setEnabled(true);
    }
    list_panel.setVisible(false);
    wait_dialog.setVisible(false);
}

/**
 * Creates new lattices and gets all results using values from
Settings()
 * @param size int
 * @param num int
 * @param dim boolean
 */
private void get_lattice_info(int size, int num, boolean dim)
{
    hide_gui();
    // Reset lattice variables to default
    lattice_size = size;
    lattice_numbers = num;
    is_3d = dim;
    empty_objects();

    for(int i = 0; i < lattice_numbers; i++) {
        // Add lattice information to arraylists
        lattice.generate(lattice_size, is_3d);
        array_nodes = lattice.get_nodes();
        list_array_nodes.add(array_nodes);
        array_results = lattice.get_results(array_nodes,
lattice_size);
        list_array_results.add(array_results);
        list_lattices.add((i + 1) + " (" +
array_results.length + ")");
    }
}
```

```
        create_views();
        show_gui();
    }

    /**
     * Opens a save file dialog, then saves the lattice arrays to
file
     */
    private void save_file()
    {
        // Open save dialog
        FileDialog filedialog = new FileDialog(this,
                                                "Save Lattice
information ...",
                                                FileDialog.SAVE);

        filedialog.setLocation(50, 50);
        filedialog.setVisible(true);

        if(filedialog.getFile() != null) {
            try {
                File file = new File(filedialog.getFile() +
".lattice");
                FileOutputStream output = new FileOutputStream(file);
                PrintStream printstream = new PrintStream(output);

                // print information to file
                printstream.println(lattice_numbers);
                printstream.println(lattice_size);
                printstream.println(is_3d);

                for(int i = lattice_numbers; --i >= 0; ) {
                    array_nodes = (int[][])list_array_nodes.get(i);
                    for(int j = array_nodes.length; --j >= 0; ) {
                        printstream.println(array_nodes[j][0]);
                        printstream.println(array_nodes[j][1]);
                    }
                }

                // Close file after all information is in
                printstream.close();
            } catch(Exception exception) {
                exception.printStackTrace();
            }
        }
    }

    /**
     * Opens a load file dialog then populates nodes array from file
information
     */
    private void load_file()
    {
        // Open a load file dialog
        FileDialog filedialog = new FileDialog(this, "Load a saved
Lattice...",
                                                FileDialog.LOAD);

        filedialog.setFile("*.lattice");
        filedialog.setLocation(50, 50);
```

```

        filedialog.setVisible(true);

        if(filedialog.getFile() != null) {
            try {
                hide_gui();

                // Reset lattice variables to default
                empty_objects();

                FileReader filereader = new
FileReader(filedialog.getFile());
                BufferedReader bufferedreader = new
BufferedReader(filereader);
                ArrayList aList = new ArrayList(0);

                // Load information from file
                lattice_numbers =
Integer.parseInt(bufferedreader.readLine());
                lattice_size =
Integer.parseInt(bufferedreader.readLine());
                int lattice_total_nodes = lattice_size *
lattice_size;
                if(bufferedreader.readLine().equals("true")) {
                    is_3d = true;
                    lattice_total_nodes *= lattice_size;
                } else {
                    is_3d = false;
                }

                array_nodes = new int[lattice_total_nodes][2];

                for(int i = lattice_numbers; --i >= 0; ) {
                    for(int j = array_nodes.length; --j >= 0; ) {
                        array_nodes[j][0] =
Integer.parseInt(bufferedreader.
                            readLine());
                        array_nodes[j][1] =
Integer.parseInt(bufferedreader.
                            readLine());
                    }

                    list_array_nodes.add(array_nodes);
                }
                list_array_results.add(lattice.get_results(array_nodes,
                    lattice_size));
                list_lattices.add(" Lattice - " + (i + 1) + " ");
            }

            bufferedreader.close();
            filereader.close();

            create_views();
        } catch(Exception exception) {
            exception.printStackTrace();
        }
    }
    show_gui();
}

```

```
/**
 * Creates new instance of each view
 */
private void create_views()
{
    get_lattice_results(0);
    array_nodes = (int[][][])list_array_nodes.get(0);

    view_cluster = new ViewCluster(array_nodes, lattice_size,
                                   percolating_cluster, is_3d);
    view_lattice.set(lattice_size, percolating_cluster,
array_nodes);
    if(is_3d) {
        view_graph_1.set(lattice_size * lattice_size *
lattice_size,
                        graph_tab_pane.getHeight(), 1,
list_array_results);
    } else {
        view_graph_1.set(lattice_size * lattice_size,
                        graph_tab_pane.getHeight(),
                        1, list_array_results);
    }
    view_graph_2.set(lattice_size, graph_tab_pane.getHeight(), 2,
list_array_results);

    output();
}

/**
 * Gets lattice results from ViewLattice class
 * @param num int
 */
private void get_lattice_results(int num)
{
    double temp_size = 0;
    array_results = (double[][][])list_array_results.get(num);
    list_clusters.removeAll();

    for(int i = 0; i < array_results.length; i++) {
        if((array_results[i][9] > 0) && (array_results[i][1] >
temp_size)) {
            percolating_cluster = (int)array_results[i][0];
            temp_size = array_results[i][1];
        }
        list_clusters.add((i + 1) + " (" +
(int)array_results[i][1] + ")");
    }
}

/**
 * Outputs results to textarea
 */
private void output()
{
    DecimalFormat dp1 = new DecimalFormat("0.##");
    DecimalFormat dp0 = new DecimalFormat("0");
    String bl = "\n";
    String tb = "\t";
```

```

String line = "-----" +
              "-----";
String dline = "===== " +
               "===== ";

view_output.setText("Number of Lattices :: " + tb + tb +
                    lattice_numbers + bl + "Lattice size :: "
+ tb + tb +
                    lattice_size + bl +
                    "Number of nodes per latice :: " + tb +
                    (lattice_size * lattice_size) + bl +
                    "Total nodes analysed :: " + tb +
                    (lattice_size * lattice_size *
lattice_numbers) +
                    bl + bl + bl + dline + bl + "Lattice" +
tb +
                    "Clusters" + tb + "Perc Clusters");

for(int i = 0; i < lattice_numbers; i++) {
    int perc_cluster = 0;
    array_results = (double[][][])list_array_results.get(i);

    for(int j = array_results.length; --j >= 0; ) {
        if(array_results[j][9] > 0) {
            perc_cluster++;
        }
        view_output.append(bl + (i + 1) + tb +
array_results.length + tb +
                        tb +
                        perc_cluster);
    }

    view_output.append(bl + dline + bl + bl);

    for(int i = 0; i < lattice_numbers; i++) {
        array_results = (double[][][])list_array_results.get(i);
        view_output.append(bl + "Lattice Number :: " + (i + 1) +
bl + bl +
                        "Cluster" + tb + "Length" + tb + "Mass
X" + tb +
                        "Mass Y" + tb + "Mass Z" + tb +
"Width" + tb +
                        "Height" + tb + "Depth" + tb + "RadG"
+ tb +
                        "Percolating");
        for(int j = 0; j < array_results.length; j++) {
            view_output.append(bl + (j + 1) + tb +
dp0.format(array_results[j][1]) +
tb +
dp1.format(array_results[j][2]) +
tb +
dp1.format(array_results[j][3]) +
tb +
dp0.format(array_results[j][4]) +
tb +
dp0.format(array_results[j][5]) +
tb +
dp1.format(array_results[j][6]) +
tb +

```



```

                                dp1.format(array_results[j][7]) +
tb +                                dp1.format(array_results[j][8]) +
tb +                                dp0.format(array_results[j][9]));
                                }
                                view_output.append(bl + bl + line + bl + bl);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

ViewCluster.java

```

// ViewCluster

// Import Required Libraries
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;

import javax.media.j3d.Appearance;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GraphicsConfigTemplate3D;
import javax.media.j3d.Locale;
import javax.media.j3d.Material;
import javax.media.j3d.PhysicalBody;
import javax.media.j3d.PhysicalEnvironment;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.View;
import javax.media.j3d.ViewPlatform;
import javax.media.j3d.VirtualUniverse;
import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Sphere;

/**

```

```
* <p>Title: ViewCluster</p>
* <p>Description: Produces a 3D view of a cluster</p>
* <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
* <p>Company: Leeds Metropolitan University</p>
* @author Gareth Flowers
* @version 1.0
*/
class ViewCluster extends Panel
{
    private int lattice_size;
    private int cluster;
    private int[][] array_nodes;
    private boolean is_3d;

    /**
     * Creates new instance of ViewCluster and sets up
     VirtualEnvironment
     * @param nodes int[][]
     * @param size int
     * @param num int
     * @param d3 boolean
     */
    public ViewCluster(int[][] nodes, int size, int num, boolean d3)
    {
        array_nodes = nodes;
        lattice_size = size;
        cluster = num;
        is_3d = d3;

        GraphicsConfigTemplate3D gcTemplate = new
GraphicsConfigTemplate3D();
        GraphicsEnvironment local = GraphicsEnvironment.
            getLocalGraphicsEnvironment();
        GraphicsDevice screen = local.getDefaultScreenDevice();
        GraphicsConfiguration configuration =
screen.getBestConfiguration(
            gcTemplate);
        Canvas3D canvas_3d = new Canvas3D(configuration);

        Locale locale = new Locale(new VirtualUniverse());
        locale.addBranchGraph(build_view(canvas_3d));
        locale.addBranchGraph(build_content());
        setLayout(new BorderLayout());
        setBackground(Color.WHITE);

        Panel info_panel = new Panel(new GridLayout(2, 1));
        info_panel.setBackground(Color.WHITE);
        info_panel.setFont(new Font("Arial", Font.BOLD, 12));
        Panel info_panel_1 = new Panel();
        info_panel_1.add(new Label(
            "Select a Lattice, then a cluster from the list on
the left"));
        info_panel.add(info_panel_1);
        Panel info_panel_2 = new Panel(new GridLayout(1, 3));
        info_panel_2.add(new Label("Left-Mouse button = Rotate
Cluster"));
        info_panel_2.add(new Label("Right-Mouse button = Move
cluster"));
        info_panel_2.add(new Label(
```

```
        "Left-Mouse button + Left-Alt key = Zoom In or
Out"));
    info_panel.add(info_panel_2);
    add(BorderLayout.NORTH, info_panel);
    add(BorderLayout.CENTER, canvas_3d);
}

/**
 * Returns the X coordinate of the given node
 * @param node int
 * @return int
 */
private int find_x(int node)
{
    return node % lattice_size;
}

/**
 * Returns the Y coordinate of the given node
 * @param node int
 * @return int
 */
private int find_y(int node)
{
    if(is_3d) {
        // Return the 2D Y coordinate for the position number
        return(node - ((lattice_size * lattice_size) *
            (node / (lattice_size * lattice_size)))) /
            lattice_size;
    } else {
        // Return the 3D Y coordinate for the position number
        return node / lattice_size;
    }
}

/**
 * Returns the Z coordinate of the given node
 * @param node int
 * @return int
 */
private int find_z(int node)
{
    // Return the Z coordinate for the position number
    return node / (lattice_size * lattice_size);
}

/**
 * Sets up a view for the 3D viewing
 * @param canvas Canvas3D
 * @return BranchGroup
 */
protected BranchGroup build_view(Canvas3D canvas)
{
    BranchGroup view = new BranchGroup();
    Transform3D view_transform = new Transform3D();
```

```
view_transform.set(new Vector3f(0.0f, 0.0f, 5.0f));
TransformGroup group = new TransformGroup(view_transform);
ViewPlatform platform = new ViewPlatform();
PhysicalBody body = new PhysicalBody();
PhysicalEnvironment environment = new PhysicalEnvironment();

group.addChild(platform);
view.addChild(group);
View my_view = new View();
my_view.addCanvas3D(canvas);
my_view.attachViewPlatform(platform);
my_view.setPhysicalBody(body);
my_view.setPhysicalEnvironment(environment);
return view;
}

/**
 * Sets up a view for the 3D content
 * @return BranchGroup
 */
private BranchGroup build_content()
{
    int max_x = -1, max_y = -1, max_z = -1;
    int min_x = -1, min_y = -1, min_z = -1;
    BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,
0.0, 0.0),
        100.0);
    Appearance appearance = new Appearance();
    BranchGroup content = new BranchGroup();
    TransformGroup group = new TransformGroup(new Transform3D());
    group.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    group.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

    appearance.setMaterial(new Material(new Color3f(1f, 0f, 0f),
        new Color3f(1f, 0f, 0f),
        new Color3f(1f, 1f, 1f),
        new Color3f(1f, 0f, 0f),
80f));

    for(int i = 0; i < array_nodes.length; i++) {
        if(array_nodes[i][1] == cluster) {
            if(max_x == -1) {
                max_x = find_x(i);
                max_y = find_y(i);
                max_z = find_z(i);
                min_x = find_x(i);
                min_y = find_y(i);
                min_z = find_z(i);
            }
            if(find_x(i) > max_x) {
                max_x = find_x(i);
            }
            if(find_x(i) < min_x) {
                min_x = find_x(i);
            }
            if(find_y(i) > max_y) {
                max_y = find_y(i);
            }
            if(find_y(i) < min_y) {
```

```
        min_y = find_y(i);
    }
    if(find_z(i) > max_z) {
        max_z = find_z(i);
    }
    if(find_z(i) < min_z) {
        min_z = find_z(i);
    }
}

}

float x_set = min_x + ((max_x - min_x) / 2.0f);
float y_set = min_y + ((max_y - min_y) / 2.0f);
float z_set = min_z + ((max_z - min_z) / 2.0f);
float cylinder_radius = 0.007f;
float sphere_radius = 0.035f;
float distance = 0.175f;
float half_distance = 0.0875f;

for(int i = 0; i < array_nodes.length; i++) {
    if((array_nodes[i][1] == cluster)) {
        float xc = (find_x(i) - x_set) * distance;
        float yc = (find_y(i) - y_set) * distance;
        float zc = 0;

        if(is_3d) {
            zc = (find_z(i) - z_set) * distance;
        }

        // Create a sphere to represent nodes
        Transform3D trans_sphere = new Transform3D();
        trans_sphere.setTranslation(new Vector3f(xc, yc,
        zc));

        TransformGroup spheres = new
        TransformGroup(trans_sphere);
        spheres.addChild(new Sphere(sphere_radius,
        appearance));
        group.addChild(spheres);

        // Creates a cylinder to represent link
        // Rotate cylinder to correct orientation
        if(array_nodes[i][0] != -666) {
            Transform3D trans_cylinder = new Transform3D();
            if(array_nodes[i][0] <= i - (lattice_size *
        lattice_size)) {
                trans_cylinder.setTranslation(new
        Vector3f(xc, yc,
                zc - half_distance));
                trans_cylinder.setRotation(new AxisAngle4f(0,
        1, 1,
                3.14159f));
            } else if(array_nodes[i][0] >=
                i + (lattice_size * lattice_size)) {
                trans_cylinder.setTranslation(new
        Vector3f(xc, yc,
                zc + half_distance));
                trans_cylinder.setRotation(new AxisAngle4f(0,
        1, 1,
```

```

        9.42477f));
    } else if(array_nodes[i][0] < i - 1) {
        trans_cylinder.setTranslation(new
Vector3f(xc,
        yc - half_distance, zc));
        trans_cylinder.setRotation(new AxisAngle4f(1,
1, 0,
        6.28318f));
    } else if(array_nodes[i][0] > i + 1) {
        trans_cylinder.setTranslation(new
Vector3f(xc,
        yc + half_distance, zc));
    } else if(array_nodes[i][0] < i) {
        trans_cylinder.setTranslation(new Vector3f(xc
-
        half_distance, yc, zc));
        trans_cylinder.setRotation(new AxisAngle4f(1,
1, 0,
        9.42477f));
    } else if(array_nodes[i][0] > i) {
        trans_cylinder.setTranslation(new Vector3f(xc
+
        half_distance, yc, zc));
        trans_cylinder.setRotation(new AxisAngle4f(1,
1, 0,
        3.14159f));
    }

    TransformGroup cylinder = new
TransformGroup(trans_cylinder);
    cylinder.addChild(new Cylinder(cylinder_radius,
distance,
        appearance));
    group.addChild(cylinder);
}
}

content.addChild(group);

// Create the rotate behavior node
MouseRotate behavior = new MouseRotate();
behavior.setTransformGroup(group);
content.addChild(behavior);
behavior.setSchedulingBounds(bounds);

// Create the zoom behavior node
MouseZoom behavior2 = new MouseZoom();
behavior2.setTransformGroup(group);
content.addChild(behavior2);
behavior2.setSchedulingBounds(bounds);

// Create the translate behavior node
MouseTranslate behavior3 = new MouseTranslate();
behavior3.setTransformGroup(group);
content.addChild(behavior3);
behavior3.setSchedulingBounds(bounds);

// Add lights and compile (for speed)
add_lights(content);
```

```
        content.compile();

        return content;
    }

    /**
     * Add a directional light to the 3D scene
     * @param branchgroup BranchGroup
     */
    private static void add_lights(BranchGroup branchgroup)
    {
        DirectionalLight light = new DirectionalLight(new
        Color3f(0.1f, 1.4f,
                0.1f), new Vector3f(4.0f, -7.0f, -12.0f));
        light.setEnable(true);
        light.setInfluencingBounds(new BoundingSphere(new
        Point3d(0.0, 0.0, 0.0),
                100.0));
        branchgroup.addChild(light);
    }
}
```

ViewGraph.java

```
// ViewGraph

// Import Required Libraries
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Panel;
import java.util.ArrayList;

/**
 * <p>Title: ViewGraph</p>
 * <p>Description: Draw a graph using input results</p>
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
 * <p>Company: Leeds Metropolitan University</p>
 * @author Gareth Flowers
 * @version 1.0
 */
class ViewGraph extends Panel
{
    private int graph_type;
    private float graph_size;
    final private float graph_offset = 60f;
    private int lattice_size;
    private ArrayList list_array_results;
    private double[][] array_results;
    private Graphics graphics;

    /**
```

```
    * Creates new instance of ViewGraph
    */
    public ViewGraph()
    {
        try {
            setBackground(Color.WHITE);
            setLayout(new BorderLayout());
        } catch(Exception exception) {
            exception.printStackTrace();
        }
    }

    /**
     * Set the graph variables
     * @param size int
     * @param height int
     * @param type int
     * @param results ArrayList
     */
    protected void set(int size, int height, int type, ArrayList
results)
    {
        // Set variables
        lattice_size = size;
        list_array_results = results;
        graph_size = (height - (3f * graph_offset)) / lattice_size;
        graph_type = type;
    }

    /**
     * Draw graph on graphics component
     */
    private void draw_graph()
    {
        float xc = graph_offset;
        float yc = (lattice_size * graph_size) + graph_offset;
        int size = 0;

        switch(graph_type) {
            case 1:

                // Display Length vs Radius Gyration Graph
                axis((int)xc, (int)yc, "Length", "Radius of
Gyration");

                size = list_array_results.size();
                for(int i = size; --i >= 0; ) {
                    array_results =
(double[][][])list_array_results.get(i);
                    int length = array_results.length;
                    for(int j = length; --j >= 0; ) {
                        graphics.fillOval((int)(xc +
array_results[j][1]),
                                            (int)((yc -
array_results[j][8]) -
                                            1.5),
                                            3, 3);
                    }
                }
            }
        }
    }
}
```



```
        break;
    case 2:

        // Display Width vs Height Graph
        axis((int)xc, (int)yc, "Width", "Height");
        size = list_array_results.size();
        for(int i = size; --i >= 0; ) {
            array_results =
(double[][][])list_array_results.get(i);
            int length = array_results.length;
            for(int j = length; --j >= 0; ) {
                graphics.fillOval((int)(xc +
                                array_results[j][5] *
                                graph_size),
                                (int)(yc -
                                array_results[j][6] *
                                graph_size),
                                3, 3);
            }
        }
        break;
    default:
    }
}

/**
 * Draw and label axis
 * @param xc int
 * @param yc int
 * @param x_label String
 * @param y_label String
 */
private void axis(float xc, float yc, String x_label, String
y_label)
{
    // Draw x and y axis
    graphics.drawLine((int)xc, (int)yc, (int)xc,
                      (int)(yc - (lattice_size * graph_size)));
    graphics.drawLine((int)xc, (int)yc,
                      (int)(xc + (lattice_size * graph_size)),
(int)yc);

    // Add Labels to axis
    graphics.drawString(x_label, (int)xc, (int)(yc + 30f));
    draw_string_vert(y_label, xc - 30f, graph_offset);

    // Add values to axis
    graphics.drawString("0", (int)xc, (int)(yc + 15f));
    graphics.drawString(" " + lattice_size,
                      (int)(xc + (lattice_size * graph_size)),
                      (int)(yc + 10f));
    graphics.drawString("0", (int)(xc - 15f), (int)yc);
    graphics.drawString(" " + lattice_size, (int)(xc - 15f),
                      (int)(yc - (lattice_size * graph_size)));
    graphics.drawString("Graph showing " + x_label + " vs " +
y_label, 20,
                                20);
}
```

```
    /**
     * Converts a string into separate letters and writes each letter
in a
     * vertical line (from top to bottom)
     * @param text String
     * @param xc int
     * @param yc int
     */
    private void draw_string_vert(String text, float xc, float yc)
    {
        //
        float font_height =
graphics.getFontMetrics(getFont()).getHeight() + 1f;
        int j = 0;
        int k = text.length();

        while(j < k + 1) {
            if(j == k) {
                graphics.drawString(text.substring(j), (int)xc,
                                   (int)(yc + (j * font_height)));
            } else {
                graphics.drawString(text.substring(j, j + 1),
(int)xc,
                                   (int)(yc + (j * font_height)));
            }
            j++;
        }
    }

    /**
     * Update Graphics
     * @param graphic Graphics
     */
    public void paint(Graphics graphic)
    {
        // On object repaint
        graphics = graphic;
        draw_graph();
    }
}
```

ViewLattice.java

```
// ViewLattice

// Import Required Libraries
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
```

```
/**
 * <p>Title: ViewLattice </p>
 * <p>Description: Panel object added to the main frame
 *   Draws a 2d lattice and allows cluster highlighting</p>
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
 * <p>Company: Leeds Metropolitan University</p>
 * @author Gareth Flowers
 * @version 1.0
 */
class ViewLattice extends Canvas implements MouseListener
{
    private final int display_node_distance = 5; // Distance between
node points (pixels)
    private final int display_offset = 10; // Offset from top left
(pixels)
    private int view_cluster_number = -666;
    private int lattice_size;
    private int percolating_cluster;
    private final int select_distance = 3;
    private Graphics graphics;
    private int[][] array_nodes;

    /**
     * Creates new instance of ViewLattice
     */
    public ViewLattice()
    {
        try {
            // Initialise object
            setBackground(Color.WHITE);
            addMouseListener(this);
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    /**
     * Set Lattice view variables
     * @param size int
     * @param perc_cluster int
     * @param nodes int[][]
     */
    protected void set(int size, int perc_cluster, int[][] nodes)
    {
        // Set variables
        array_nodes = nodes;
        lattice_size = size;
        percolating_cluster = perc_cluster;
        repaint();
    }

    /**
     * Run on mouse click
     * @param mouseevent MouseEvent
     */
    public void mouseClicked(MouseEvent mouseevent)
```

```
{
    // Set cluster as the one nearest the mouse click
    for(int i = array_nodes.length; --i >= 0; ) {
        if((mouseevent.getX() > display_x(i) - select_distance)
&&
            (mouseevent.getX() < display_x(i) + select_distance)
&&
            (mouseevent.getY() > display_y(i) - select_distance)
&&
            (mouseevent.getY() < display_y(i) + select_distance))
        {
            view_cluster_number = array_nodes[i][1];
        }
    }
    repaint();
}

/**
 * Run when mouse button pressed
 * @param mouseevent MouseEvent
 */
public void mousePressed(MouseEvent mouseevent)
{}

/**
 * Run when mouse button released
 * @param mouseevent MouseEvent
 */
public void mouseReleased(MouseEvent mouseevent)
{}

/**
 * Run when mouse leaves area
 * @param mouseevent MouseEvent
 */
public void mouseExited(MouseEvent mouseevent)
{}

/**
 * Run when mouse enters area
 * @param mouseevent MouseEvent
 */
public void mouseEntered(MouseEvent mouseevent)
{}

/**
 * Return the X coordinate from the node
 * @param node int
 * @return int
 */
private int find_x(int node)
{
    // Return the X coordinate for the position number
    return node % lattice_size;
}
```

```
/**
 * Return the Y coordinate from the node
 * @param node int
 * @return int
 */
private int find_y(int node)
{
    // Return the Y coordinate for the position number
    return node / lattice_size;
}

/**
 * Return the display position of X from the node
 * @param node int
 * @return int
 */
private int display_x(int node)
{
    // Return the X coordinate onscreen display position for the
position number
    return display_offset + (find_x(node) *
display_node_distance);
}

/**
 * Return the display position of Y from the node
 * @param node int
 * @return int
 */
private int display_y(int node)
{
    // Return the Y coordinate onscreen display position for the
position number
    return display_offset + (find_y(node) *
display_node_distance);
}

/**
 * Draw the lattice on the graphics area
 * @param cluster int
 */
protected void draw_lattice(int cluster)
{
    for(int i = array_nodes.length; --i >= 0; ) {
        if(array_nodes[i][0] != -666) {
            // Draw lines as links between nodes
            if(array_nodes[i][1] == cluster) {
                // Draw selected cluster in red
                graphics.setColor(Color.RED);
                graphics.drawLine(display_x(i), display_y(i),
                                display_x(array_nodes[i][0]),
                                display_y(array_nodes[i][0]));
            } else if((array_nodes[i][1] == percolating_cluster))
{
                // Draw percolating cluster in blue

```

```

        graphics.setColor(Color.BLUE);
        graphics.drawLine(display_x(i), display_y(i),
                           display_x(array_nodes[i][0]),
                           display_y(array_nodes[i][0]));
    } else {
        // Draw all other clusters in black
        graphics.setColor(Color.BLACK);
        graphics.drawLine(display_x(i), display_y(i),
                           display_x(array_nodes[i][0]),
                           display_y(array_nodes[i][0]));
    }
}

/**
 * Update the graphics area
 * @param graphic Graphics
 */
public void paint(Graphics graphic)
{
    // Called when this object is repainted
    graphics = graphic;
    draw_lattice(view_cluster_number);
}

/**
 * Return the current cluster number
 * @return int
 */
protected int get_cluster_number()
{
    // Return the current selected cluster
    return view_cluster_number;
}

/**
 * Set the current cluster number
 * @param cluster_number int
 */
protected void set_cluster_number(int cluster_number)
{
    // Set the current selected cluster
    view_cluster_number = cluster_number;
    repaint();
}
}

```

Settings.java

```
// Settings
```

```
// Import Required Libraries
import java.awt.Dialog;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/**
 * <p>Title: Settings</p>
 * <p>Description: Settings dialog used when regenerating
lattices</p>
 * <p>Copyright: Gareth Flowers Copyright (c) 2005</p>
 * <p>Company: Leeds Metropolitan University</p>
 * @author Gareth Flowers
 * @version 1.0
 */
class Settings extends Dialog implements ActionListener
{
    private int lattice_size;
    private int lattice_numbers;
    private JTextField nl = new JTextField(10);
    private JTextField ls = new JTextField(10);
    private JCheckBox is_3d = new JCheckBox();
    private JButton apply = new JButton("Recreate New Lattices...");
    private JButton cancel = new JButton("Cancel");
    private boolean change = false;

    /**
     * Creates new instance of settings dialog
     * @param frame Frame
     * @param size int
     * @param num int
     */
    public Settings(Frame frame, int size, int num)
    {
        super(frame, true);
        lattice_size = size;
        lattice_numbers = num;

        Panel row1 = new Panel();
        row1.add(new Label("Lattice Size (width in nodes)"));
        ls.setText("" + lattice_size);
        row1.add(ls);
        add(row1);

        Panel row2 = new Panel();
        row2.add(new Label("Number of Lattices"));
        nl.setText("" + lattice_numbers);
```

```

        row2.add(n1);
        add(row2);

        Panel row3 = new Panel();
        row3.add(new Label("3 Dimensional"));
        is_3d.setSelected(false);
        row3.add(is_3d);
        add(row3);

        Panel row4 = new Panel();
        apply.addActionListener(this);
        row4.add(apply);
        cancel.addActionListener(this);
        row4.add(cancel);
        add(row4);

        Dimension screen_size =
Toolkit.getDefaultToolkit().getScreenSize();
        setTitle("Generate New Lattice");
        setResizable(false);
        setSize(290, 180);
        setModal(true);
        setLocation((screen_size.width / 2) - (getWidth() / 2),
                    (screen_size.height / 2) - (getHeight() / 2));
        setLayout(new FlowLayout(FlowLayout.RIGHT));
        setVisible(false);
    }

    /**
     * Called by buttons on the dialog
     * @param actionevent(ActionEvent)
     */
    public void actionPerformed(ActionEvent actionevent)
    {
        if(actionevent.getSource() == apply) {
            try {
                if((get_num() <= 0) || (get_num() > 9999) ||
                    (get_size() <= 0) ||
                    (get_size() > 9999)) {
                    JOptionPane.showMessageDialog(this,
value between 1 and 9999.",
                    "Error",
JOptionPane.ERROR_MESSAGE);
                } else {
                    change = true;
                    setVisible(false);
                }
            } catch(Exception exception) {
                JOptionPane.showMessageDialog(this,
value between 1 and 9999.",
                    "Error",
JOptionPane.ERROR_MESSAGE);
            }
        } else {
            setVisible(false);
        }
    }

```



```
    }  
}  
  
/**  
 * Returns the lattice size  
 * @return int  
 */  
protected int get_size()  
{  
    String ls_value = ls.getText();  
    return Integer.parseInt(ls_value);  
}  
  
/**  
 * Returns number of lattices  
 * @return int  
 */  
protected int get_num()  
{  
    String nl_value = nl.getText();  
    return Integer.parseInt(nl_value);  
}  
  
/**  
 * Returns the dimension (3D)  
 * @return boolean  
 */  
protected boolean get_is_3d()  
{  
    return is_3d.isSelected();  
}  
  
/**  
 * Returns true if the apply button is pressed  
 * @return boolean  
 */  
protected boolean get_changed()  
{  
    return change;  
}  
  
/**  
 * Reset variables and displays dialog  
 */  
protected void open()  
{  
    change = false;  
    setVisible(true);  
}  
}
```

Glossary

API	Application Program Interface
CPU	Central Processing Unit
GUI	Graphical User Interface
HCI	Human-Computer Interaction
IDE	Integrated Development Environment
JDK	Java Development Kit
JVM	Java Virtual Machine
Linux	Is a free Unix-type operating system originally created by Linus Torvalds
RAD	Rapid Application Development
UNIX	An operating system that supports multitasking and is ideally suited to multi-user applications (such as networks)
VM	Virtual Machine (see JVM)
Windows	Operating System Developed by Microsoft Corporation (www.microsoft.com)