Assignment 4

Deadline:	Hand in by 5pm on Friday 18th October 2013 (end of semester)
Evaluation:	40 marks – which is 40% of your final grade
***Late Submission**:*	5 Marks off per day late
Work	This assignment is to be done **individually** – your submission may be checked for plagarism against other assignments and against Internet repositories. If you adapt material from the Internet acknowledge your source.
Purpose:	To give you an opportunity to use advanced programming methods in a more extensive and complex simulations project.

Problem to solve:

Develop a simulation program that does something interesting. It can be graphical if you wish to build upon the ideas from 159709, but it need not necessarily be. Ideally it should help you investigate some phenomena that cannot be easily tackled using non-simulation methods.

Some suggestions are given overleaf, but you can also discuss your own mini-project proposal with the lecturer.

Requirements:

Develop your code in **C++** or **Java**, and make sure you document what you have done and how it is implemented. Be prepared to make a presentation to the class on your work.

Hand-in: Submit your **program and documentation** including discussion of your results (a zip file is acceptable) by email to d.p.playne@massey.ac.nz

Marks will be allocated for: correctness, fitness of purpose, utility, style, use of sensible **comments and program documentation**, and general elegance. Good comments will help me to award you marks even if your code is not quite perfect. You will also get marks for having writing up some discussion/conclusions – ie signs that you have really thought about the problem.

Additional Notes on the Technical Specification and Hints:

Be careful not to attempt something that is too complicated to implement using the programming techniques and resources available to you. Remember this assignment is worth 40% of a 15-credit paper, so limit the time you spend on this assignment accordingly.

On the other hand you will hopefully find developing computer simulation programs quite interesting and possibly addictive. This experience may even be the basis for further project work later in your research career.

In the example suggested projects overleaf you can write a graphical interactive simulation if you wish, but even if you do not want to use a complicated interactive graphics system you may find it useful to make your program dump out an image file to help you see what is going on in your simulated model. The PPM (Portable Pixmap) format is particularly easy to generate yourself. Search online for PPM format details.

Not that some of these projects are "open ended" that is they are essentially research problems that do not have definite known answers. In many cases you will find lots of material online and chasing up references and such like is part of the assignment and should appear in your write up.

If you have any questions about this assignment, please ask the lecturer.

Project Suggestion 1 – Simulation of the Q-State Potts Model

The Potts model is very like the Ising model discussed in lectures, but instead of just two states Spin-up and Spin-down – the Q-State Potts model has Q possible states. So the Ising model is the special case of Q=2.

You can set up a simulation of the Potts model in 2 dimensions using the same concepts of counting like-like bonds as for the Ising model. Investigate what happens to the critical temperature of the model for Q=3 and Q=4. A reasonable system size to us e would be of N = 256 x 256 sites, but a typical desktop will be able to run a bigger system than this. Initialise the system randomly and quench it to a finite temperature as discussed in lecture 4. You might want to look at temperatures similar to the one for the Ising model to start with.

You can make a graphical simulation if you wish – using eg OpenGL with C++ or using Java graphics. This can help debug your program, and for this small size of 2 dimensional system you should be able to make your program interactive so for example the temperature could be controlled by a slider or other graphical widget.

Can you colour the sites according to their Q value or according to which component cluster they are in? (Hint, generate a rainbow palette of colours to have plenty to assign to different clusters).

For extra credits try the model on a different mesh such as a hexagonal lattice.

Project Suggestion 2: Simulation of a Simple Lattice Gas Kawasaki Model

The Ising model can also be used as the basis of a lattice gas. In this model (Known sometimes as the Kawasaki model) you initialize the system with 50/50 spin up and down as normal, but instead of flipping spins, you choose two neighbouring sites and work out how the number of like-like bonds would change if they were swapped. That way you preserve the number of ups and downs. You can think of the whites as being atoms and the black as being vacancies or holes, so the fixed number of up whites move around. The model behaves similarly to the Ising model in that at low cold temperatures the atoms stick together whereas at high temperatures they all get jumbled up. If the system is too cold however there is not enough energy for the atoms to move around and you end up with a frozen "glass" like structure.

Study this model by counting the number of separate clusters. You could also study this system as an interactive graphics program if you wish.

For extra credits investigate this model when there are more than two species present - the Q-State Kawasaki Model. How does the behaviour change?

Project Suggestion 3: Simulation of the Sznajd Opinion Model

Lattice simulation models can be used to study opinion formation amongst crowds of people. Create a square lattice of L by L sites where L might be round 100, and apply periodic boundary conditions. Place "individuals" on the sites where each individual has one of two opinions. Start off with a random mixture of different opinioned individuals.

At each iteration an individual and one of its neighbours is chosen at random. If the two individuals have the same opinion then the opinion of all sixe nearest neighbours of the pair are set to that of the pair. This is reminiscent of the fact that people are more likely to change their opinions through peer pressure to that of those near to them.

Simulate this model and show that consensus is always reached eventually. How does the (average) time to reach consensus vary with different starting mixture ratios of opinions (eg 45/55, 46/54, 47/53, 48/52, 49/51)? You can use similar analysis techniques as discussed for the other models – look at (and plot) how the numbers of individuals with the different opinions changes with time; look and count clusters of individuals with separate opinions.

You can make your simulation an interactive graphical one if you wish.

For extra credits what happens when the individuals are arranged on a random graph, rather than on a plain square mesh? What happens when there are more than just 2 possible opinions present in the population? What happens if some individuals are deliberately contrary and seek to adopt the opposite opinion to that of their neighbours?

Project Suggestion 4: Daisyworld Albedo/Climate Change Model
Daisyworld is a simulation model for a simplified planet orbiting a radiant sun. There are only two "life forms" black and white daisies – the white daisies reflect light, the black ones absorb it. This can be modelled as array of cells that are updated by a cellular automaton rule and where the daisies crowd one another and the changing albedo provides a negative feedback loop to regulate the amount of black/white daisies even if the solar radiation levels change. There is a lot of material available online on "Daisyworld" including demo Applets for 2d square lattice systems. An interesting graphical variation would be to implement on a spherical mesh with 3D graphics and a large and fast simulation running in C/C++. [This project would suit students who have already done OpenGL in 159.709, although you could also use Java and Java2D graphics.] The Daisyworld model can be extended in a number of ways - you could try more than just two species of daisy - additional species could emit and absorb "greenhouse gases" for example so that albedo could be controlled by more than one effect. Or a predator species could "harvest the daisies.

Give screendumps and explain your model. See Wikipedia/google on "daisyworld" and give references etc.

Project Suggestion 5: Random Walk with Obstacles

In one dimension a random walk is quite easily understood analytically. Simulations are needed for more complicated geometries however. Imagine a 2D grid and that a particle starts off at the centre. At each iteration or time step of the simulation it moves one position, eg north, south, east or west. The probability of the particle "diffusing" to some definite distance D away from its starting position falls off according to a normal like distribution. In other words the particle is most likely to be found close to its starting point, but could in principle travel a long way. You can set up a simulation to colour or shade the cells according to how often the particle has visited them after a large number of steps and plot a density profile along an axis to verify the distribution. An interesting variation is to locate some obstacles and se e how they modify the density distribution. A long thin barrier or a single slit between two barriers or double slits between three barriers would all be interesting. Or a box with a hole in its wall might also give rise to interesting patterns. Or you could use a font to generate a barrier based on the word "Massey" and see how that affects the particle diffusion pattern. Or you could try this on a 3D mesh – although it is harder to visualize.

Give screendumps and explain your model. See Wikipedia/google on "random walk" or "diffusion" and give references etc.

Project Suggestion 6: Self Avoiding Walk – Maze generation

A variation of the random walk is the self-avoiding walk where the diffusing particle is not allowed to cross its own path. The particle can get stuck however, fencing itself into an impossible position. What is the distribution of path lengths like? How does it vary if the particle is allowed to jump to next nearest as well as just nearest neighbouring cells? How does it vary in a 3D system? What about a 4D system?
Can you devise rules for the particle to stop it getting stuck? Can you use the path generated to make a maze pattern?

Give screendumps and explain your model. See Wikipedia/google on "self avoiding walk" and give references etc.

Project Suggestion 7: Directed Percolation

Ordinary percolation can be investigated in 2D or 3D by randomly filling cells in a lattice with some probability p. You can use a component-labeling algorithm to determine if there is a route from one side of the lattice to the other through only filled cells. Vary p and you will see there is a definite transition at a particular value dependent upon dimension d. You may need to do several repeat experiments with different random configurations for each p in your p-scan to establish uncertainties or error bars on your data. In addition to a yes/no path existence measurement, you can use other graph metrics such as the path length or number of hops to get from side to side.

Directed percolation is an interesting variation whereby you are allowed to jump across gaps in the mesh but only in preferred directions. So for example you might allow jumps in x direction only but not y. A graphical simulation would help you develop a model and an interesting outcome would be to see how the mean path length varies with p and preferred direction.

Give screendumps and explain your model. See Wikipedia/google on “directed percolation” and give references etc.

Project Suggestion 8: Object-Oriented Matrix Manipulation

Matrices can be stored in a number of different manners including dense format where every entry is store d even the zero values, as well as compressed row, column and skyline formats for sparse storage – where only the non-zeroes are stored.

Experiment with a parameterised Matrix class in C++ that can handle floats of doubles or complex matrices and can convert between different (hidden) storage formats and can do operations like matrix-matrix multiplication. Is such a system feasible? For what other operations might it work (eg matrix inversion?) Test your system with some sparse matrices eg a banded matrices. Can you set up some operator overloads in a C++ version to specify matrix algebra expressions like A = B * C, where A, B, and C are matrices? How will your system check the storage and size compatibilities?

A good interface might be able to “hide” parallel matrix computations that are actually done using a library system like LAPACK or ScaLAPACK Thes e are systems that are available for download and could be used on one of the Massey compute clusters..

See the Numerical Recipes book chapters on linear algebra and matrices, and see also google/wikipedia searches on “sparse matrix storage,” “LAPACK,”…

Project Suggestion 9: 3D Mandelbrot Visualisation

The well-known Mandelbrot fractal is generated by checking to see how many iterations are necessary to pass a test at each point on a region of the complex plane. This is usually done as a 2D colour picture, but the number of iterations could be used to generate a depth and you could make a 3D rendered chasm with fractal cliffs base d on the Mandelbrot number of iterations at each point in the mesh. You might need to render using a logarithmic scale however. Experiment with your assignment 1 complex numbers class and OpenGL graphics to render the Mandelbrot set in an unusual 3D way.

In addition to the copious Mandelbrot information available on the Web, Google, Wikipedia etc, see also the link: `http://www.massey.ac.nz/~kahawick/159235/examples/MandelbrotTool.java` for basics on the Mandelbrot algorithm, and refer to 159709 material on OpenGL for 3D rendering techniques eg as cubes on a mesh. Can you think of a good way to navigate down the fractal cliff so a user can re-find an area of interest and make screendumps etc?

Project Suggestion 10: Spatial Rock-Paper-Scissors-Spock-Lizard

The well known rock paper scissors game can be played by agents on a spatial mesh and they can be programmed with different strategies. A spatial variant is to have agents play against their neighbours and have the best strategies propagate. You can then colour the players and look for interesting spatial patterns

An unusual variation is to consider the 5-outcome game variant of rock-paper-scissors-spock-lizard, where the five form a closed 5-cycle. Implement a 2D array version - you can have wrap around boundaries for simplicity - in Java/Java2D or C++/OpenGL.

What do you find when there are 3, 4, 5 play variants?

Give screendumps/ population plots with simulation time etc and explain your model. See Wikipedia/google on “rock paper scissors spock lizard” or “spatial game theory” and give references etc.

Project Suggestion 11: Propose something exciting and interesting yourself!

But come talk to me first! Something that involves “advanced programming for simulations” with or without graphics, would be appropriate.