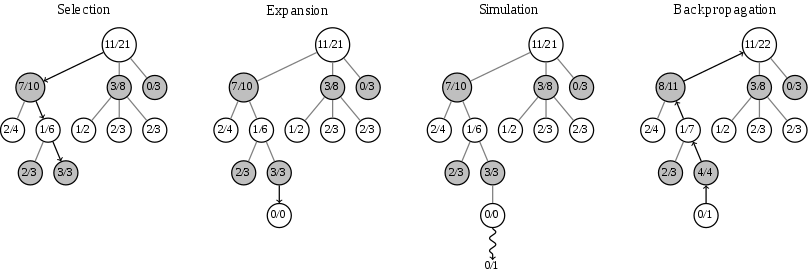
Monte Carlo Tree Search C++ Tutorial

1. Download **Monte Carlo Tree Search C++** from MyLearningSpace. It should be in the Practical tab on MyLearningSpace.
2. Open the Project in Visual Studio.
3. Run the program, DEBUG takes longer but will allow you to step through the code.

# What is Monte Carlo Tree Search

Monte Carlo Tree Search algorithms (MCTS) is a method of A.I. where a system generates and simulates every possible move and decides which the best. The algorithm has four actions it runs repeatedly until the tree is complete.

1. Selection - Select an outlying node in the tree
2. Expansion - Create the next follow on action
3. Simulation - Simulate the next action to determine how good it is
4. Backpropagation - Pass the simulation result back to the root of the tree



If you want to read more into the technical aspects the algorithm, there is a good post [here](http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/) that covers the topic very well.

# Understanding the Code

The base code is split into three main classes:

* BaseAction - Describes the action
* BaseNode - Describes the branch of the tree
* BaseWorldState - Describes the world state

Each of these classes needs to be inherited into a custom set of classes that can solve the function. For example, the BaseNode does not know how to Simulate a particular branch as it needs to be given tailored instructions depending on the problem to solve. This is why there is a TicTacToeBaseNode that inherits from BaseNode to provide additional information and override some existing functions.

## Steps

Initially we create a single instance of the SearchTreeEngine. This is given a starting node to act as the root of the tree; all branches come from this node. The starting node is added as the first node in the open set (named ‘openBrances’). The open set is a queue, so that new nodes are added to the back and they are processed in the order they are created.

Once setup is complete the engine can be ran. During runtime the engine follows three sets:

1. Extract the front node from the open set
2. Expand the node, creating new branches for each available move. These are pushed into the back of the open set.
3. Simulate each of these new nodes.
4. Repeat

# Exercises

The code for this lab already runs and works but there is a major problem with how it simulates and back-propagates data. Each win returns a +1, whereas a loss returns a -1. This means that the AI will prioritise moves that keep the game going longest as they have the most ‘win’ conditions at the end.

1. Open the TicTacToeNode.cpp file and find the SimulateChildBranch function. This function calculates a new ranking and back-propagates it back to the root of the tree.
2. Change the Simulate function so the AI focuses only on winning the game. Remove the negative ranking if it loses. How does this affect the A.I.’ behaviour?
   1. Play around with how the ranking is awarded. See if you can get the algorithm to focus on particular plays.
3. Edit the TicTacToeNode class so that it applies a multiplier to the ranking so that it provides a lower rank the deep the node is in the tree.
   1. Feel free to add new functions to the class. For example, one that uses recursion that finds the depth of the tree from the node calling the function.
4. Plan out how you would expand and simulate a game of Checkers using the algorithm.
   1. If you feel adventurous, implement this into a program.