**Node**

Private Functions:

Move() – Modifies the GameState to perform current set move

UnMove() – Removes the move we just made

WinUtil() – Determines if I win or other person has won in this state

H1Util() – Determines the H1Utility of this state given by

ConnectivityUtility() – Given by connectivity of the board for my pieces – connectivity of the board for opponent

Private Vars:

Int ID

Int numChildren

Int thisTurn

Int allocated

Int myUtil

Int\* childUtil – Must be initialized by DiscoverChildren() Function

Node\*\* myChildren

Node\* parent

GameState\* state – Not a copy, pointer to global state, modified during DFS evaluation

Public Constructors:

Node(int id, GameState\* initial, int initial\_turn)

* The first constructor for building the search tree which takes in the
  + id of the root node
  + initial board state
  + initial\_turn: 0 for X, 1 for O

Node(int id, GameState\* state, int column, int turn)

* New search tree node initialized by the parent
  + Parent must call by passing the GameState to be modified and turn
  + This new Node then calls Move() to place the token in the column assigned
  + Turn usually given by parents (turn+1)%2

Destructors:

~Node() – Deletes utility array of children and self

DeleteTree() – Fully deletes all children and then self

Public Functions:

GameState\* getState() – Returns a new state which is a copy of modified state

Int GetTurn() – Returns the turn

Int GetDepth() – Returns the dephth of this Node

Int GetNumberOfChildren() – Returns the number of children. Caller must first call DiscoverChildren(), otherwise this function returns the value -1.

Int GetUtil() – Returns my Utility Value

Int\* GetChildrenUtil() – Returns pointer to children utility value

* Must Call Number of children in order to know how many Children I have

Node\*\* GetChildren() – Returns pointer to children

Node\*\* DiscoverChildren() – Uses the current known board state to investigate empty columns and initializes new children assigning to an array.

Int ComputeUtil() – Returns the highest utility in order of Zeroth order heuristic (winning) then following order heuristics

Void Print() – Prints all details of the current Node

**Function Node::DiscoverChildren()**

**Requires:** GameState\* state of the current board state for the Node.

**Output:** An array of length *Number of Columns on board* containing the first n valid move Nodes. There is some space wastage when entire columns are not valid moves.

An array of children utility values, *Util* ,as determined by node evaluation

#Define Elements

1. Int[] childrenColumns
2. Int numberOfChildren = 0
3. Node[] myChildren
4. Int terminalPosition = 0

#Discover the Number of children

1. For each column on the board with index i
2. If (column i is a valid move)
3. childrenColumns[numberOfChildren++] = i
4. terminalPosition++
5. EndFor

#Allocate the children

1. For int i in range [0, terminalPosition]
2. myChildren[i] = new Node( …, this->state, childrenColumns[i], (this->turn+1)%2 )

**Function Node::WinUtil()**

**Requires:** GameState\* state

**Output:** The utility of the state if it is a winning state for either player. The utility is between 900 and 1000 if this player wins and -900 to -1000 if the other player wins. If nobody wins, the utility returned is 0.

#Define Elements

1. Int myUtil = 0
2. Int baseValue = 0

#Perform Check

# Only one