CSE 326 Spring 1995
Data Structures
Project 3
April 26, 1995

DUE: Tuesday, May 16, 1995 at 12:01am

1 Objectives

- To become familiar with depth-first search.
- To demonstrate how to specialize data structures to make them more efficient for their intended purpose.
- To introduce the concept of a game tree.

2 Project Materials

All of the materials you will need are in `/cse/courses/cse326/project3`. The only files you should need to modify are `GameDAG.C` and possibly `GameDAG.h`.

3 The Basic Problem

The goal of this assignment is to use the abstract concepts of a graph and a depth-first search to help us design an unbeatable tic-tac-toe program. To see how a graphs relate to tic-tac-toe, consider the following:

1. There are a finite number of "states" in which a tic-tac-toe board can exist. More specifically, there are 3 possibilities for each square, "X", "O", or "Empty", and 9 squares, for a total of 3^9 possible states. We let these be the vertices of our graph.
2. We can define a relationship "$\rightarrow$" by $V_1 \rightarrow V_2$ if and only if we can take the board represented by the vertex V_1, make exactly one move (assumed to be legal) on it, and end up with the board represented by V_2. In our graph, there will be an edge from V_1 to V_2 if and only if $V_1 \rightarrow V_2$.

The above rules define some digraph D. Observe that D must be acyclic because it is impossible to start at some state, make legal moves (which always involve adding symbols to the board), and arrive back at the same state. However, D is not a tree because it is possible to arrive at some state by more than one distinct sequence of moves. Therefore D must be a directed, acyclic graph (DAG) whose root is the empty board.

The digraph D is useful because it allows the computer to enumerate all possible games of tic-tac-toe simply by starting at the root and finding all possible paths through the DAG. A

more general way of putting this is that given any board configuration, the computer can find its corresponding vertex in the DAG and from there enumerate all possible outcomes of the game.

Given this ability, we can design scheme which conceptually allows the computer to look at the current board, judge all possible outcomes, and make moves which lead to the most favorable outcomes. We can make this process efficient in two ways. First, rather than searching for all possible outcomes and selecting the best every time a query is made, we can do it once for each valid state and store the result in a lookup table. This optimization turns our problem from one which involves finding all possible paths in a graph (exponential time) to one which involves doing a depth-first search (linear time). Second, we can avoid ever building a concrete representation of the graph. We can always calculate all of its successors of any given node when we need them so there is no need to store them in an adjaceny list or array.

Your job in this assignment is to implement the scheme described above. There are two issues involved: dealing with the data structure and, given information about all possible outcomes from a given state, picking the best move to make.

4 What To Do

- As usual, you should make a copy of all of the materials in the project directory.
- Also as usual, I have had to write far more code for this assignment than you will. You are being given all of the materials for a complete tic-tac-toe game; if you `make` the code I have provided what you will get is a program called `ttt` which allows you to play a game with two human players. It will also give you the option of having one or more computer players, but choosing this option before modifying the program will cause it to do something nasty.
- Study the files I have provided for you. You will be making extensive use of the `Board` class defined in `Board.h`, but you will not need to modify or even understand its implementation. Your job will be to implement the class defined in `GameDAG.h` by filling in the empty method implementations in `GameDAG.C`. Finally, if you're interested in seeing how this class is used, you can look at `Player.h` and `Game.C`.
- *Do not write any code yet!* Think carefully about the problem and plan out your solution. Draw lots of pictures of game graphs and think about how your algorithm would deal with them. There are lots of possible variations on how to choose the optimal move for each position in the graph. Any reasonable approach should have these properties:

  1. The computer should never lose. At worst it should force a draw.
  2. If there is a move from a given configuration which guarantees a win, it should be taken.

- Once you have carefully considered the problem, fill in the implementations for the routines in `GameDAG.h`. Although the focus of the assignment is on performing the graph traversal, you must also figure out how to map every possible `Board` to a unique vertex number.
- You should submit your program for grading by doing a "`make submit`".