Backtracking, Search, Heuristics

- Many problems require an approach similar to solving a maze
  - Certain mazes can be solved using the “right-hand” rule
  - Other mazes, e.g., with islands, require another approach
  - If you have “markers”, leave them at intersections, don’t explore the same place twice

- What happens if you try to search the web, using links on pages to explore other links, using those links to …
  - How many web pages are there?
  - What rules to webcrawlers/webspiders follow?
    - Who enforces the rules?
- Keep track of where you’ve been don’t go there again
  - Any problems with this approach?
Classic problem: N queens

- Can queens be placed on a chess board so that no queens attack each other?
  - Easily place two queens
  - What about 8 queens?
- Make the board N x N, this is the N queens problem
  - Place one queen/column
  - # different tries/column?
- Backtracking
  - Use “current” row in a col
  - If ok, try next col
  - If fail, back-up, next row
Backtracking idea with N queens

- Try to place a queen in each column in turn
  - Try first row in column, if ok, move onto next column
  - If solved, great, otherwise try next row, place queen, move onto the next column

- What happens when we start in a column, where to start?
  - If we fail, move back to previous column (which remembers where it is/failed)
  - When starting in a column anew, start at beginning
    - When backing up, try next location, not beginning

- Backtracking in general, record an attempt go forward
  - If going forward fails, undo the record and backup
N queens backtracking: nqueens.cpp

bool Queens::SolveAtCol(int col)
// pre: queens placed at columns 0,1,...,col-1
// post: returns true if queen can be placed in column col
//        and N queen problem solved (N is square board size)
{
    int k; int rows = myBoard.numrows();
    if (col == rows) return true;

    for(k=0; k < rows; k++)
    {   if (NoQueensAttackingAt(k,col))
        { myBoard[k][col] = true; // place a queen
            if (SolveAtCol(col+1))
            {    return true;
            }
            myBoard[k][col] = false; // unplace the queen
        }
    }
    return false;
}
Backtracking with Boggle

- Boggle™ played on 4x4 board
  - Other sizes possible
  - Form words by connecting letters horizontally, vertically, diagonally
  - Cannot re-use letters (normally)
- Two approaches
  - Build words from connections, find partial words in dictionary
  - Look up every word in the dictionary on the board
- Which is better? How is backtracking used? (see boggle.cpp)
What about crawl.cpp?

- Each student in cps100 creates a file with links
  - Links aren’t hyperlinks per se, but serve similar purpose
  - Think of a room with n doors, you must pick door k, where 0 <= k < n (one way door) and go through the door

- Is there a potential for a problem here? Why?
  - Will everyone be known by at least one person?
  - Will a chain-of-acquaintances connect any two people?

- What are some approaches to limit redundant exploration?
OO Techniques (webcrawler.h/.cpp)

- **WebCrawler** has one static method
  - What does static mean here? Why is it used?
  - What does client code look like? Why a class?

- An abstract class cannot be constructed, how is the parameter to `WebCrawler::getURL` supposed to work?
  - This is a framework, has **hooks** for methods that the client provides
  - Look at code in webcrawler.cpp, where are the hooks?

- What would client write to get a URL and the associated information?
  - Where does this code/class go?
Computer v. Human in Games

- Computers can explore a large search space of moves quickly
  - How many moves possible in chess, for example?

- Computers cannot explore every move (why) so must use heuristics
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move

- What do humans do well in these games? What about computers?
  - What about at Duke?