PFTN2Wks

- Making progress on Boggle assignment
  - Many, many classes, each designed to do one thing
  - Some related by inheritance, object-oriented concept
  - Many single purpose classes — can be hard to grok
  - Recursive backtracking, also hard to grok

- Toward trees and recurrences
  - More self-referential structures
  - Recursive algorithms
  - Self-referential run-time analysis (kind of)

OO Aside: Interfaces, Abstract Classes

- Interface is a contract
  - X implements Y: X provides method implementations
  - No code sharing among implementing classes
  - Characteristics of interface: signatures only
  - Why create an interface?

- Abstract class leverages common code
  - Often implements interface with common code
  - Consider Map/AbstractMap, List/AbstractList, ...
  - One method labeled abstract, requires implementation!

Object-oriented [programming|design]

- Classes encapsulate state and behavior
  - Methods are behavior, ivars/fields are state
  - Ideally class models/deals with one concept, not many
  - State is typically 'hidden' from client code, methods used

- In bigger programs inheritance very useful
  - Avoid code duplication
  - Allow for open/close: classes open to extension, closed to modification
  - Hollywood principle: don't call us, we'll call you
    - You can make your own List class, sort it! How?

Interfaces and Abstraction in Boggle

- General idea: create IThing for interfaces
  - IPlayer, IAutoPlayer, ILexicon, IBoardMaker
  - Facilitates new implementations if I Thing doesn't change
    - Hollywood principle for rest of code/Boggle classes
    - We've designed the interfaces, they don’t change

- AbstractPlayer and AbstractAutoPlayer
  - Factor out common code, differentiate in subclasses
  - Game play by Gui/Controller/View, back-and-forth plays
  - Be careful, be wary of ignoring/not knowing parent code
    - How do you update score in a BoardFirstAutoPlayer

- How can you create different boards?
Inheritance concepts

- **Parent/super class can be extended by subclassing**
  - Possible to use methods from parent class, subs have them!
  - Possible to override parent methods, change behavior
  - Possible to do both! Call `super.doThis();`

- **Often you don't have access to parent .java file**
  - Still can subclass, use methods, extend/override them
  - Do NOT have access to private data fields
  - DO have access to protected data fields

- **Hard to do OO design, leave for later courses**
  - But get an idea now as to power and utility

Searching with no guarantees

- **Search for best move in automated game play**
  - Can we explore every move?
  - Are there candidate moves ranked by “goodness”?
  - Can we explore entire tree of possible moves?

- **Search with partial information**
  - Predictive texting with T9 or iTap or ...
  - Finding words on a Boggle board
  - What numbers fit in Sudoku suare

- **Try something, if at first you don’t succeed ....**

Backtracking by image search

Search, Backtracking, Heuristics

- **How do you find a needle in a haystack?**
  - How does a computer play chess?
  - Why would you write that program?

- **How does Bing/Googlemap find routes from one place to another?**
  - Shortest path algorithms
  - Longest path algorithms

- **Optimal algorithms and heuristic algorithms**
  - When is close good enough? How do measure “closeness”?
  - When is optimality important, how much does it cost?
Exhaustive Search/Heuristics

- We can probably explore entire game tree for tic-tac-toe, but not for chess
  - How many tic-tac-toe boards are there?
  - How many chess boards are there?

- What do we do when the search space is huge?
  - Brute-force/exhaustive won't work, need heuristics?
  - What about google-maps/Garmin finding routes?

- Backtracking can use both concepts
  - Game tree pruning a good idea most of the time

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 NxN, this is the N queens problem
  - Place one queen/column
  - Horiz/Vert/DirDiag attacks

- Backtracking
  - Tentative placement
  - Recurse, if ok done!
  - If fail, undo tentative, retry

- wikipedia-n-queens

Backtracking idea with N queens

- For each column C, tentatively place a queen
  - Try first row in column C, if ok, move onto next column
    - Typically “move on” is recursive
  - If solved, done, otherwise try next row in column C
    - Must unplace queen when failing/unwind recursion

- Each column C “knows” what row R it’s on
  - If first time, that’s row zero, but might be an attack
  - Unwind recursion/backtrack, try “next” location

- Backtracking: record an attempt go forward
  - Move must be “undoable” on backtracking/unwinding

N queens backtracking: Queens.java

```java
public boolean solve(int col){
    if (col == mySize) return true;
    // try each row until all are tried
    for(int r=0; r < mySize; r++){
        if (myBoard.safeToPlace(r,col)){
            myBoard.setQueen(r,col,true);
            if (solve(col+1)){
                return true;
            }
            myBoard.setQueen(r,col,false);
        }
    }
    return false;
}
```
Basic ideas in backtracking search

- **Enumerate all possible choices/moves**
  - We try these choices in order, committing to a choice
  - If the choice doesn’t pan out we must undo the choice
    - Backtracking step, choices must be undoable

- Inherently recursive, when to stop searching?
  - When all columns tried in N queens
  - When we have found the exit in a maze
  - When every possible moved tried in Tic-tac-toe or chess?
    - Is there a difference between these games?

- Summary: enumerate choices, try a choice, undo a choice, this is brute force search: try everything

Queens Details

- **How do we know when it’s safe to place a queen?**
  - No queen in same row, or diagonal
  - For each column, store the row that a queen is in
  - See QBoard.java for details

- Advanced OO: For GUI version, we use a decorator
  - The QBoardGUI is an IQueenState class and it has an IQueenState object in it
  - Appears as an IQueenState to client, but uses an existing one to help do its work
  - One of many object oriented design patterns, seen in Huff in the BitInputStream class

Boggle Program

- **GridGame APT: how many winning moves?**
  - It’s my turn and there are no places to go: return 0, why?
  - Try to place an ‘X’, count opponent wins. If 0 then …
    - After placing ‘X’ and counting, unplace the ‘X’, why?

- **What about looking for a word on Boggle Board?**
  - If we’ve already made ‘EA’ and we look at b[3][5]
    - Can we make ‘EAT’? Can we make ‘EACH’?
    - Trying for ‘EAT’, if b[3][5] is ‘T’ then …
    - Trying for ‘EACH’, if b[3][5] is ‘A’ then …
    - Trying for ‘EACH’, if b[3][5] is ‘C’ then …
Boggle Search for Word

- Starting at board location (row, col): find a string S
  - We want to keep track of where we are in the string
  - Also track what board locations used for S search

- How do we know when we're done?
  - Base case of recursive, backtracking call
  - Where we are in the string?

- How do we keep track of used locations?
  - Store in array list: tentatively use current one, recurse
  - If we don’t succeed, take off the last one stored!

Using Howto Hints

- Helper method in GoodWordOnBoardFinder
  - Needed: board, row, col, word, word-index, list
  - Search for "skunk" at (0,0) at (0,1) at ...
  - Call help(board, row, col, "skunk", 0, list)
  - If we find the 's', where do we look next and for what?
  - If we don't find the 's' what does that mean?
  - If (row, col) isn't legal position then ...
  - If we've found the last 'k' what does that mean?

- Do we first check for ...
  - Does order of checking make a difference?

Lots and lots of classes – sensible?

- Javadoc can help understand methods/classes
  - We are using an 'un-named' package in Boggle
  - Generate javadoc from within Eclipse, Project menu
  - Useful for browsing classes/files

- Java-isms
  - Class Foo is in Foo.java
  - We use IThing for an Interface named Thing
  - You can document private stuff too, useful in development

- Experience is a good teacher

Daphne Koller

- 2004, Macarthur
- 2008, first ACM/Infosys

“The world is noisy and messy …You need to deal with the noise and uncertainty.”

“I find it distressing that the view of the field is that you sit in your office by yourself surrounded by old pizza boxes and cans of Coke, hacking away at the bowels of the Windows operating system,” she said. “I spend most of my time thinking about things like how does a cell work or how do we understand images in the world around us?”
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?

Games at Duke

- Alan Biermann
  - Natural language processing
  - Compsci 1: Great Ideas
  - Duchess, checkers, chess
- Tom Truscott
  - Duke undergraduate working with/for Biermann
  - Usenet: online community
  - Second EFF Pioneer Award (with Vint Cerf!)

Heuristics

- A heuristic is a rule of thumb, doesn’t always work, isn’t guaranteed to work, but useful in many/most cases
  - Search problems that are “big” often can be approximated or solved with the right heuristics
- What heuristic is good for Sudoku?
  - Is there always a no-reasoning move, e.g., 5 goes here?
  - What about “if I put a 5 here, then…”?
  - Do something else?
- What other optimizations/improvements can we make?
  - For chess, checkers: good heuristics, good data structures

Barbara Liskov

- First woman to earn PhD from compsci dept
  - Stanford
- Turing award in 2008
  - OO, SE, PL

“IT's much better to go for the thing that's exciting. But the question of how you know what’s worth working on and what's not separates someone who's going to be really good at research and someone who’s not. There's no prescription. It comes from your own intuition and judgment.”