Agenda 16 Nov 2011

Logistics

Java intensive – Thursday noon, Deschutes 200
Optional midterm redo Wed of dead week?

Designing a class

Linked lists
Designing a Class

The unvarnished truth
Question on Monday: How do you decide what methods are in a class, and what they do?

These are the notes I used to start coding the project that became assignment 8, the Boggle Solver, but they aren’t the beginning or the end.

So here’s an honest example ... Tuesday afternoon and evening, sketching and refining a design for a Sudoku solver.
I’m not completely starting from scratch. I have a strategy in mind: Depth-first search with constraint propagation at each level.
What my design really looks like, at first ...

Design exercise Tuesday: Sudoku solver

Lots of diagrams, bits of pseudo-code, NO consideration (yet) of how the code is divided into classes and methods.
What operations will I need?

Operations:
- Starting with full set of possibilities, mark off the symbols that are used.
- Store allowed possibilities.
- Possibilities = Possibilities \( \cap \) new constraint.
- Is null? Is singleton?

Rep choices:
- Java tree set, other set rep.
- Array of boolean (bitset).
- Array of symbol.
- Always small, selectable (bitset)?
Roughing out an interface

Plenty of alternatives and false starts ...

to find good ideas, generate lots of bad and good ideas.
Simple or general? Simple, this time.

class DigitSet {
    static int digitIndex (char dig) {
        assert in range '0'...'9'
        return ord (dig) - ord ('0')
    }
    public DigitSet () {
        elements = new boolean [9] {
            true, true, ...false
        }
    }
    delete set index to false
    intersect: loop through digits
    is singleton: loop, count
Designing other parts ...

Propagate: \[ \text{board} \rightarrow \{ \text{fail, complete, incomplete} \} + \text{board} \]

\[
\begin{align*}
\left\{ \\
\text{Apply row constraints (for each row)} \\
\text{Apply col constraints (for each col)} \\
\text{Apply sqr constraints} \\
\text{Determine set of possibilities for whole row/col/sq} \\
\text{Intersect into each element} \\
\end{align*}
\]

As iterator? Or duplicate code?
Designing the search and the board class

- Sudoku board rep -
  - operations -
    - Get file value at row, col
    - Get set of candidates at row, col
    - Set remaining candidates at row, col
    - Set choice at row, col
    - Save board } for back-tracking
    - Restore board } in dfs

Options for save/restore:
- Copy of board, but recompute constraints
- Stack of changes w/ mark
Eventually some parts look solid ...

Sudden dfs:

status ← propagate() // Succeed, fail, or unfinished
  if failed already, return fail
  if succeed, return succeed

make a list of open squares,
  sorted by # choices (computed by propagate)

for each open square,
  for each choice in that square
    save board
    set choice
dfs (board)
  if success then return success
  // otherwise it has failed —
  // no third choice
  restore board

Lend loop
  Lend loop
    fail?
Ready to flesh out (pseudo)code

I have enough of the Sudoku program design to (mostly) know what methods I’ll need for a DigitSet class.

Coding or pseudo-coding it will reveal other choices, and possibly alter my design
Following the notes ...

class DigitSet {
    private boolean[ ] digits;

    static final NUMDIGITS = 9;

    public DigitSet() {
        digits = new boolean[NUMDIGITS];
        this.clearConstraints();
    }

    /**
     * Clear constraints means all choices are possible.
     */
    public clearConstraints() {
        for (int i=0; i < NUMDIGITS; ++i) {
            digits[i] = true;
        }
    }
Noticing issues as I code ...

/**
 * Mark one of the digits as "already used" by
 * removing it from the set of choices.
 * @param digit The digit (char) to be removed.
 */
public remove(char digit) {
    assert (digit >= '1' && digit <= '9');
    int index = (int) digit - (int) '1';
    Magic numbers ... I don't want these scattered in my code. I can isolate them better in a method.
***/
* Map digits to elements (indexes) of boolean array.
* @param digit The character to map, '1' ... '9'
* @return index 0..8 to index array "digits"
*/
private int index(char digit) {
    assert (digit >= '1' && digit <= '9');
    int i = (int) digit - (int) '1';
    return i;
}
... and the “remove” logic gets simpler

/**
 * Mark one of the digits as "already used" by removing it from the set of choices.
 * @param digit  The digit (char) to be removed.
 */
public void remove(char digit) {
    digits[ index(digit) ] = false;
}
Is this the right signature for “intersect”? 

/**
 * Intersect two sets (apply both sets of constraints).
 * @param other The other set to be intersected with this one.
 */
public void intersect(DigitSet other) {
    for (int i=0; i < NUMDIGITS; ++i) {
        this.digits[i] = this.digits[i] && other.digits[i];
    }
}

Or should intersect return a DigitSet? I think this is ok because of the way I plan to use it.
Of course there are bugs ...

$ javac DigitSet.java
DigitSet.java:12: <identifier> expected
    static final NUMDIGITS = 9;
          ^
DigitSet.java:24: invalid method declaration; return type required
    public clearConstraints() {
            ^
DigitSet.java:64: invalid method declaration; return type required
    public Result() {
            ^
DigitSet.java:73: ';' expected
    } elif (count == 1) {
            ^
DigitSet.java:75: 'else' without 'if'
    } else {
            ^
Not done ...

Needs to be tested

More changes will be suggested as I code the rest of the Sudoku solver

over and over ... lots of revision and refinement