Lecture 11: 1% Pure Luck

Make-up lab hours: 4:30-6 today

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Solving the Pegboard Puzzle

• How to represent the state of the board?
  – Which holes have pegs in them
• How can we simulate a jump?
  – board state, jump positions → board state
• How can we generate a list of all possible jumps on a given board?
• How can we find a winning sequence of jumps?

Data Abstractions

(define (make-board rows holes)
  (cons rows holes))
(define (board-holes board) (cdr board))
(define (board-rows board) (car board))
(define (make-position row col) (cons row col))
(define (get-row posn) (car posn))
(define (get-col posn) (cdr posn))
(define (same-position pos1 pos2)
  (and (= (get-row pos1) (get-row pos2))
       (= (get-col pos1) (get-col pos2))))

Removing a Peg

;;; remove-peg evaluates to the board you get by removing a peg at posn from the passed board (removing a peg adds a hole)
(define (remove-peg board posn)
  (make-board (board-rows board)
              (cons posn (board-holes board))))

Adding a Peg

;;; add-peg evaluates to the board you get by adding a peg at posn to board (adding a peg removes a hole)
(define (add-peg board posn)
  (make-board (board-rows board)
               (remove-hole (board-holes board) posn)))
Lecture 11: 1% Luck

Remove Hole

(define (remove-hole lst posn)
  (if (same-position (car lst) posn)
      (cdr lst)
      (cons (car lst) (remove-hole (cdr lst) posn)))))

Could we define remove-hole using map?
No. (length (map f lst)) is always the same as (length lst), but remove-hole needs to remove elements from the list.

What if we had a procedure (filter proc lst) that removes from lst all elements for which proc (applied to that element) is false?

Filter

(define (filter proc lst)
  (if (null? lst)
      null
      (if (proc (car lst)) ; proc is true, keep it
          (cons (car lst) (filter proc (cdr lst)))
          (filter proc (cdr lst))))) ; proc is false, drop it

> (filter (lambda (x) (> x 0)) (list 1 4 -3 2))
(1 4 2)

Filter Remove

(define (filter proc lst)
  (if (null? lst)
      null
      (if (proc (car lst)) ; proc is true, keep it
          (cons (car lst) (filter proc (cdr lst)))
          (filter proc (cdr lst))))) ; proc is false, drop it

(define (remove-hole lst posn)
  (filter (lambda (pos)
            (not (same-position pos posn)))
          lst))

Jumps

;;; move creates a list of three positions: a start (the posn that the
;;; jumping peg starts from), a jump (the posn that is being jumped
;;; over), and end (the posn that the peg will end up in)

(define (make-move start jump end) (list start jump end))
(define (get-start move) (first move))
(define (get-jump move) (second move))
(define (get-end move) (third move))

;;; execute-move evaluates to the board after making move
;;; move on board.
(define (execute-move board move)
  (add-peg (remove-peg (remove-peg board (get-start move))
                        (get-jump move))
           (get-end move)))

Solving the Peg Board Game

• Try all possible moves on the board
• Try all possible moves from the positions you get after each possible first move
• Try all possible moves from the positions you get after trying each possible move from the positions you get after each possible first move
• ...

Finding a Winning Strategy

How is winning 2-person games (e.g., chess, poker) different?
How do we find all possible jumps that land in a given target hole?

**All Possible Moves**

```
(define (all-possible-moves board)
  (append-all
    (map generate-moves (board-holes holes)))))
```

But...only legal if: start and end are positions on the board containing pegs! 
Note: could use (apply append ...) instead of append-all.

**Legal Move**

```
(define (legal-move? move)
  ;; A move is valid if:
  ;; o the start and end positions are on the board
  ;; o there is a peg at the start position
  ;; o there is a peg at the jump position
  ;; o there is not a peg at the end position
  (and (on-board? board (get-start move))
       (on-board? board (get-end move))
       (peg? board (get-start move))
       (peg? board (get-jump move))
       (not (peg? board (get-end move)))))
```
Lecture 11: 1% Luck

All Legal Moves

(define (legal-moves board)
  (filter legal-move? (all-possible-moves board)))

(define (legal-move? move)
  ;; A move is valid if:
  ;;   o the start and end positions are on the board
  ;;   o there is a peg at the start position
  ;;   o there is a peg at the jump position
  ;;   o there is not a peg at the end position
  (and (on-board? board (get-start move))
       (on-board? board (get-end move))
       (peg? board (get-start move))
       (peg? board (get-jump move))
       (not (peg? board (get-end move)))))

(define (all-possible-moves board)
  (append-all
   (map generate-moves
        (board-holes holes))))

Becoming a “Genius”!

Try all possible legal moves

Winning Position

How do we tell if a board is in a winning position?

(define (board-squares board)
  (count-squares (board-rows board)))

(define (count-squares nrows)
  (if (= nrows 1) 1
      (+ nrows (count-squares (- nrows 1)))))

(define (is-winning-position? board)
  (= (length (board-holes board))
     (- (board-squares board) 1)))

Solve Pegboard

(define (solve-pegboard board)
  (find-first-winner board (legal-moves board)))

(define (find-first-winner board moves)
  (if (null? moves)
      (if (is-winning-position? board)
          null ;; Found winning game, no moves needed
          #f) ;; A losing position, no more moves
      (let ((result (solve-pegboard
                      (execute-move board (car moves))))
            (if result ;; winner (not #f)
              (cons (car moves) result)
              ;; this move leads to winner!
              ;; (find-first-winner-board (cdr moves)))))
        try rest)

All Cracker Barrel Games

<table>
<thead>
<tr>
<th>Pegs Left</th>
<th>Number of Ways</th>
<th>Fraction of Games</th>
<th>IQ Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>0.00001</td>
<td>Leaving 10 pegs requires much more brilliance than leaving 1!</td>
</tr>
<tr>
<td>9</td>
<td>374</td>
<td>0.0027</td>
<td>“You’re Purty Smart”</td>
</tr>
<tr>
<td>8</td>
<td>82</td>
<td>0.00058</td>
<td>“Just Plain Dumb”</td>
</tr>
<tr>
<td>7</td>
<td>5688</td>
<td>0.04</td>
<td>“Just Plain Eg-no-ra-moose”</td>
</tr>
<tr>
<td>6</td>
<td>20686</td>
<td>0.15</td>
<td>“You’re Genius”</td>
</tr>
<tr>
<td>5</td>
<td>1550</td>
<td>0.01</td>
<td></td>
</tr>
</tbody>
</table>
Charge

• By luck alone, you can be a genius 1% of the time!
• By trying all possibilities, you can always be a genius
  – Next week and later: do we have time for this?
• PS3 due Monday
  – Extra Lab hours: today (4:30-6)
  – Regularly scheduled lab hours:
    Sunday (4-5:30, 8-9:30)