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

Pegboard Puzzle

1,1
2,1 2,2
3,1 3,2 3,3
4,1 4,2 4,3 4,4
5,1 5,2 5,3 5,4 5,5

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)))
```

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))
```

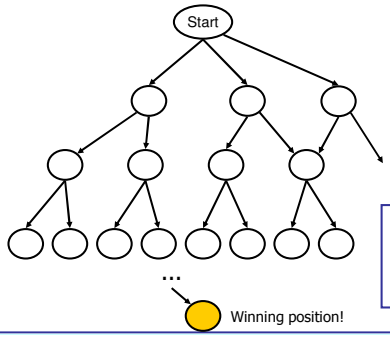

Jumps

```
;;; 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

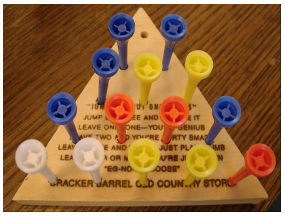
- Try all possible moves on the board
- Try all possible moves from the positions you get after each possible first move
- Try all possible moves from the positions you get after trying each possible move from the positions you get after each possible first move
- …

Finding a Winning Strategy



How is winning 2-person games (e.g., chess, poker) different?

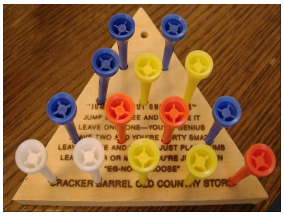
Pegboard Puzzle



1,1
2,1 2,2
3,1 3,2 3,3
4,1 4,2 4,3 4,4
5,1 5,2 5,3 5,4 5,5

How do we find all possible jumps that land in a given target hole?

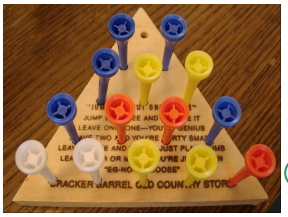
Pegboard Puzzle



1,1
2,1 2,2
3,1 3,2 3,3
4,1 4,2 4,3 4,4
5,1 5,2 5,3 5,4 5,5

How do we find all possible jumps that land in a given target hole?

Pegboard Puzzle



1,1
2,1 2,2
3,1 3,2 3,3
4,1 4,2 4,3 4,4
5,1 5,2 5,3 5,4 5,5

How do we find all possible jumps that land in a given target hole?

All Moves into Target

```
;;; generate-moves evaluates to all possible moves that move a peg into
;;; the position target, even if they are not contained on the board.
(define (generate-moves target)
   (map (lambda (hops)
          (let ((hop1 (car hops)) (hop2 (cdr hops)))
            (make-move
              (make-position (+ (get-row target) (car hop1))
                             (+ (get-col target) (cdr hop1)))
              (make-position (+ (get-row target) (car hop2))
                             (+ (get-col target) (cdr hop2)))
              target)))
        (list (cons (cons 2 0) (cons 1 0))       ;; right of target, hopping left
              (cons (cons -2 0) (cons -1 0))     ;; left of target, hopping right
              (cons (cons 0 2) (cons 0 1))       ;; below, hopping up
              (cons (cons 0 -2) (cons 0 -1))     ;; above, hopping down
              (cons (cons 2 2) (cons 1 1))       ;; above right, hopping down-left
              (cons (cons -2 2) (cons -1 1))     ;; above left, hopping down-right
              (cons (cons 2 -2) (cons 1 -1))     ;; below right, hopping up-left
              (cons (cons -2 -2) (cons -1 -1))))) ;; below left, hopping up-right
```


All Possible Moves

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

(define (append-all lst)
   (if (null? lst) null
      (append (car lst) (append-all (cdr lst)))))
```

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)))))
```

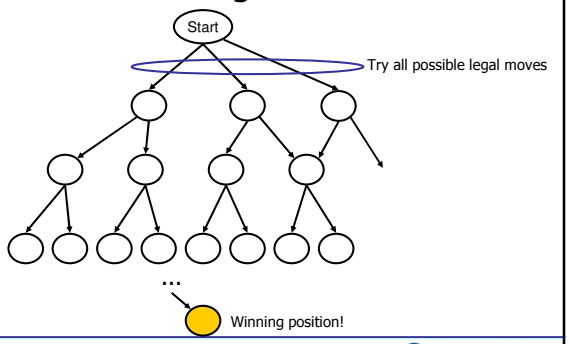
All Legal Moves

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

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


Becoming a "Genius"!



Winning Position

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

is-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

(starting with peg 2 1 missing)

Pegs Left	Number of Ways	Fraction of Games	IQ Rating
1	1550	0.01	"You're Genius"
2	20686	0.15	"You're Purty Smart"
3	62736	0.46	"Just Plain Dumb"
4	46728	0.33	
5	5688	0.04	"Just Plain Eg-no-ra-moose"
6	374	0.0027	
7	82	0.00058	
10	2	0.00001	Leaving 10 pegs requires much more brilliance than leaving 1!

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)