

CS-257L

Nonimperative Programming: Scheme!

Instructor:

Joel Castellanos

e-mail: joel@unm.edu

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering
Center (FEC) room 321

Light Cycle Project



5/7/2008

Light Cycle Project – Due April 30

- 30 point project.
- **In-class** Swiss Style Tournament: Best 2 out of 3.
- Each student subroutine controls a team of three light cycles.
- Each game includes 4 teams of three light cycles each.
- The Master Program will call each student subroutine with:
 1. The number 0 through 3 indicating the student's team number.
 2. A vector of 12 vectors where each of the 12 sub-vector contains the x and y coordinates of the corresponding light cycle.
- Negative coordinate values indicate that the Master Program has terminated the corresponding light cycle.
- Each student subroutine will return a vector of 3 vectors where each sub vector contains the coordinates where the subroutine's corresponding light cycle is choosing to move.

5/7/2008

2

Light Cycle – Rules of the Game

- The game is played on a 170 x 170 grid.
- Each game consists of four teams of three light cycles each.
- Each team is controlled by a single student subroutine.
- The starting positions for each team are given.
- A light cycle may move one square forward, one square left or one square right.
- A light cycle that moves beyond the grid is terminated.
- A light cycle that does not move is terminated.
- A light cycle that moves into a square that has already been occupied by a non-terminated light cycle is terminated.
- A light cycle that attempts to move illegally is terminated.
- Play continues until light cycles from a single team are all that have not been terminated.

5/7/2008

3

Light Cycle Phase 1 – Grading Rubric

- 21 Points:** The subroutine runs without errors, all moves are legal, and it only gets terminated when there are no legal options.
- 25 Points:** The subroutine plays about as well as a "dumb robot". A dumb robot always makes legal moves. If it can, the dumb robot always moves in the same direction it moved on the last turn. If it cannot continue in the same direction, then it chooses between legal turns with equal probability.
- 30 Points:** The subroutine wins 2 out of 3 games against three teams of "dumb robots".
- 5 Points Extra Credit:** Third Place
- 10 Points Extra Credit:** Second Place
- 20 Points Extra Credit:** First Place

5/7/2008

4

Light Cycle Phase 2 – Grading Rubric

Due: Friday, May 9th by 8:00 AM in WebCT.

21 Points: The subroutine wins 2 out of 3 games against three teams of "dumb robots".

25 Points: The subroutine plays about as well as the "smarter robots".

30 Points: The subroutine wins 2 out of 3 games against three teams of 3 "smarter robots".

5 Points Extra Credit: Third Place

10 Points Extra Credit: Second Place

20 Points Extra Credit: First Place

5/7/2008

5

Light Cycle – Time

- The game is turn based with all cycles considered to be moving simultaneously.
- All cycles move one square per turn.
- If two light cycles choose to move into the same square during the same turn, then the cycle that has use the least total processor time gets the square. The other light cycle is terminated.
- If a team's subroutine takes longer than an average of 50 milliseconds of processor time, then its light cycles will be terminated.

5/7/2008

6

Scheme Time

(`current-process-milliseconds`)

Returns the amount of processor time in fixnum milliseconds that has been consumed by the MzScheme process on the underlying operating system. The precision of the result is platform-specific, and seems to be about 5 ms on this system.

(`current-gc-milliseconds`)

Returns the amount of processor time in fixnum milliseconds that has been consumed by MzScheme's garbage collection so far. This time is a portion of the time reported by (`current-process-milliseconds`).

5/7/2008

7

Finding Time – Running Mean

Mean

5/7/2008

8

Scheme - runningMean

```
(define runningMean
  (lambda (a b n)
    ; runningMean=a(n-1)/n + b/n
    (+ (/ (* a (- n 1.0)) n)
       (/ b n))
  )
)
```

5/7/2008

9

The MCP - Master Control Program

- The Master Control Program is responsible for refereeing, and graphics.
- A copy of the MCP source will be given to the class.
- A subroutine that wins by hacking into the MCP still wins. For example, exploits array out of bounds vulnerabilities.
- Copies of the robot subroutines will not be distributed until after the tournament.

5/7/2008

10

Light Cycle – Student Subroutine Interface

```
(LightCycle_firstname_lastname
  TeamNumber ;0 through 3.
  cycleLocationList ;vector of 12 vectors specifying
    ;the current coordinate of each light cycle.

; Start Locations
(vector
  (vector 169 45) (vector 169 85) (vector 169 125)
  (vector 45 169) (vector 85 169) (vector 125 169)
  (vector 0 45) (vector 0 85) (vector 0 125)
  (vector 45 0) (vector 85 0) (vector 125 0)
)

; Return vector of LightCycle_firstname_lastname
(vector
  (vector x1 y1) (vector x2 y2) (vector x3 y3)
)
```

5/7/2008

11

Multiple Instances of a Procedure

```
(define make-instance-adder (lambda (startValue)
  (define addX startValue) ;Private and Persistent.
  (define adder (lambda (n)
    (set! addX (+ addX n))
    addX
  ))
  adder
))

(define myAdder1 (make-instance-adder 40))
(define myAdder2 (make-instance-adder 100))

(myAdder1 3) ;=====> ? 43
(myAdder1 4) ;=====> ? 47
(myAdder2 9) ;=====> ? 109
(myAdder2 4) ;=====> ? 113
(myAdder1 5) ;=====> ? 52
```

5/7/2008

12

MCP's Cycle Team Instances

`;The Master Control Program uses this interface to
;create an instance of each light cycle team.`

```
(define teamCallVector (vector  
  (LightCycle_firstname1_lastname1)  
  (LightCycle_firstname2_lastname2)  
  (LightCycle_firstname3_lastname3)  
  (LightCycle_firstname4_lastname4)))
```

This syntax for the student light cycle subroutines supports:

1. The capability of student subroutines having local identifiers that persistent between calls, yet reset when the Master Control Program is restarted.
2. The capability of instantiating more than one copy of the same student cycle with each copy having independent storage space.

5/7/2008

13

Call to Each Cycle Team Each Turn

`;cycleVector is a vector of 12 vectors.
; Each sub-vector is the current coordinates of
; cycles 0 through 11.`

```
(set! team0 (LightCycle_A 0  
  (apply vector (vector->list cycleVector)))  
)  
(set! team1 (LightCycle_B 1  
  (apply vector (vector->list cycleVector)))  
)
```

What is happening in the above code?

Why is it done in this way?

5/7/2008

14

Data Structure for Game Grid

```
;grid[x,y]=0 means the location is empty.
;grid[x,y]>0 means the location was entered from the
; direction given by the value of grid[x,y].
(load "Matrix.scm")
(define gridSize 170)
(define grid (matrix gridSize gridSize))
(define direction_right 1)
(define direction_down 2)
(define direction_left 4)
(define direction_up 8)
(define direction_start_code 16)
```

What capabilities does this data structure provide?

Describe details of how can it be used?

5/7/2008

15

Terminating a Light Cycle

```
(set! x2 x1) ← Why are these needed?
(set! y2 y1) ←
(do ((i 0 (+ i 1))) ((= dir dir_start_code)
  (set! dir (matrix-ref grid x1 y1))
  (matrix-set! grid x1 y1 0)
  (cond
    ;Follow trail backward by reverse directions.
    ((= dir dir_right) (set! x2 (- x1 1)))
    ((= dir dir_left) (set! x2 (+ x1 1)))
    ((= dir dir_up) (set! y2 (+ y1 1)))
    ((= dir dir_down) (set! y2 (- y1 1)))
  )
  (set! x1 x2)
  (set! y1 y2)
```

What if dir=0?

What other values can dir be?

5/7/2008

16

Master Program: updateCycleLocations

```
;Initialization
(define cycle_max 12) ;0 through 11
(define livingBits (- (expt 2 cycle_max) 1))
.
.
.
(define updateCycleLocations
  (lambda (cycleVector cycleVectorNew)
    (do ((i 0 (+ i 1))) ((>= i cycle_max))
      (cond
        ((= (bitwise-and livingBits (expt 2 i)) 0)
```

What if this were 500?

What kind of data structure is livingBits?

What is going on here?

5/7/2008

17

Master Program's Cycle Termination Test

; (x1 y1) is the current coordinate of a given cycle.
;(x2 y2) is the coordinate to which the cycle's
;controller has declared the cycle will move.

```
(cond
  ((or (> (matrix-ref grid x2 y2) 0)
       (< x2 0)
       (< y2 0)
       (>= x2 gridSize)
       (>= y2 gridSize)
       (> (abs (- x1 x2)) 1)
       (> (abs (- y1 y2)) 1)
       (and (= x1 x2) (= y1 y2))
       (and (not (= x1 x2)) (not (= y1 y2))))
```

What illegal move
does each line
test?

5/7/2008

18

Off Screen Buffering

```
(define buffer_bitmap
  (instantiate bitmap% (frameWidth frameHeight)))

; Create a drawing context for the screen and bitmap.
(define screen_dc (send canvas get-dc))
(define buffer_dc
  (instantiate bitmap-dc% (buffer_bitmap)))

(define DrawCycle (lambda (x1 y1 x2 y2 cycle)
  (define cyclePen
    (if (< cycle 0) pen-white
        (vector-ref pen-vector cycle)))
    (send screen_dc set-pen cyclePen)
    (send buffer_dc set-pen cyclePen)
    (send screen_dc draw-line x1 y1 x2 y2)
    (send buffer_dc draw-line x1 y1 x2 y2)
  ))
```

What is going on here?

Why Both?