CSC466 Project Task 2: Modeling, Generating, and Displaying the Game Board

Abstract

In this task I established a method to model a board with the use of CLOS. I then created a function that could generate a board, by default in a random configuration, or with the optional input of a list to place the mines manually. To generate the board first a list of 0s and 1s are generated, then this list is turned into a list of tile objects. After the objects are created I then establish the links between each tile and the values of each tile. I then created a function to display the board and a demo for this document.

Demo

```
[1]> ( load "ms.l" )

;; Loading file ms.l ...

;; Loaded file ms.l

T

[2]> ( demo--board )

 >>> Testing board generation and display

Generating a 10x10 board

Mine location list:

(1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
1 0 1 0)

Tile object list:

(#<TILE #x1AA33DB1> #<TILE #x1AA33E91> #<TILE #x1AA33ED1> #<TILE
#x1AA33F11> #<TILE #x1AA33F51> #<TILE #x1AA33F91> #<TILE
#x1AA33FD1> #<TILE #x1AA34011> #<TILE #x1AA34051> #<TILE
```

#x1AA34091> #<TILE #x1AA340D1> #<TILE #x1AA34111> #<TILE
#x1AA34151> #<TILE #x1AA34191> #<TILE #x1AA341D1> #<TILE
#x1AA34211> #<TILE #x1AA34251> #<TILE #x1AA34291> #<TILE
#x1AA342D1> #<TILE #x1AA34311> #<TILE #x1AA34351> #<TILE
#x1AA34391> #<TILE #x1AA343D1> #<TILE #x1AA34411> #<TILE
#x1AA34451> #<TILE #x1AA34491> #<TILE #x1AA344D1> #<TILE
#x1AA34511> #<TILE #x1AA34551> #<TILE #x1AA34591> #<TILE
#x1AA345D1> #<TILE #x1AA34611> #<TILE #x1AA34651> #<TILE
#x1AA34691> #<TILE #x1AA346D1> #<TILE #x1AA34711> #<TILE
#x1AA34751> #<TILE #x1AA34791> #<TILE #x1AA347D1> #<TILE
#x1AA34811> #<TILE #x1AA34851> #<TILE #x1AA34891> #<TILE
#x1AA348D1> #<TILE #x1AA34911> #<TILE #x1AA34951> #<TILE
#x1AA34991> #<TILE #x1AA349D1> #<TILE #x1AA34A11> #<TILE
#x1AA34A51> #<TILE #x1AA34A91> #<TILE #x1AA34AD1> #<TILE
#x1AA34B11> #<TILE #x1AA34B51> #<TILE #x1AA34B91> #<TILE
#x1AA34BD1> #<TILE #x1AA34C11> #<TILE #x1AA34C51> #<TILE
#x1AA34C91> #<TILE #x1AA34CD1> #<TILE #x1AA34D11> #<TILE
#x1AA34D51> #<TILE #x1AA34D91> #<TILE #x1AA34DD1> #<TILE
#x1AA34E11> #<TILE #x1AA34E51> #<TILE #x1AA34E91> #<TILE
#x1AA34ED1> #<TILE #x1AA34F11> #<TILE #x1AA34F51> #<TILE
#x1AA34F91> #<TILE #x1AA34FD1> #<TILE #x1AA35011> #<TILE
#x1AA35051> #<TILE #x1AA35091> #<TILE #x1AA350D1> #<TILE
#x1AA35111> #<TILE #x1AA35151> #<TILE #x1AA35191> #<TILE
#x1AA351D1> #<TILE #x1AA35211> #<TILE #x1AA35251> #<TILE
#x1AA35291> #<TILE #x1AA352D1> #<TILE #x1AA35311> #<TILE
#x1AA35351> #<TILE #x1AA35391> #<TILE #x1AA353D1> #<TILE
#x1AA35411> #<TILE #x1AA35451> #<TILE #x1AA35491> #<TILE
#x1AA354D1> #<TILE #x1AA35511> #<TILE #x1AA35551> #<TILE
#x1AA35591> #<TILE #x1AA355D1> #<TILE #x1AA35611> #<TILE
#x1AA35651> #<TILE #x1AA35691> #<TILE #x1AA356D1> #<TILE
#x1AA35711>)

Tile value and link example: Tile: 50 nw: NIL n: #<TILE
#x1AA2A4F5> ne: #<TILE #x1AA2A535> e: #<TILE #x1AA2A7B5> se:
#<TILE #x1AA2AA35> s: #<TILE #x1AA2A9F5> sw: NIL w: NIL
revealed: NIL mine: NIL flag: NIL value: 1

```
Displaying board with around 75% of tiles revealed
       0 1 2 3 4 5 6 7 8 9
     +----------------------+
   0 | > # 1 # 1 1 # 1 0 0
   1 | 2 3 # 2 2 2 > 1 0 0
   2 | 2 > > 1 1 > 2 1 0 0
   3 | > # 3 2 # # # 1 1 1
   4 | 2 4 # # 0 # 0 # > 1
   5 | 1 > > # 1 # 1 # # 1
   6 | 2 4 # 4 > 3 > # 0 0
   7 | 1 > 2 # # 3 # 2 1 0
   8 | # 2 # 1 1 2 2 > 2 1
   9 | > 1 0 # 0 1 > # > 1
```

```
Displaying a 20x15 board
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
     +----------------------------------------+
   0 | # # # # # # # # # # # # # # # # # # # # |
   1 | # # # # # # # # # # # # # # # # # # # # |
   2 | # # # # # # # # # # # # # # # # # # # # |
   3 | # # # # # # # # # # # # # # # # # # # # |
   4 | # # # # # # # # # # # # # # # # # # # # |
   5 | # # # # # # # # # # # # # # # # # # # # |
   6 | # # # # # # # # # # # # # # # # # # # # |
   7 | # # # # # # # # # # # # # # # # # # # # |
   8 | # # # # # # # # # # # # # # # # # # # # |
   9 | # # # # # # # # # # # # # # # # # # # # |
  10 | # # # # # # # # # # # # # # # # # # # # |
  11 | # # # # # # # # # # # # # # # # # # # # |
  12 | # # # # # # # # # # # # # # # # # # # # |
  13 | # # # # # # # # # # # # # # # # # # # # |
  14 | # # # # # # # # # # # # # # # # # # # # |
NIL
[3]>
```

Code

```
( defclass board ()
    (
        ( width :accessor board-width :initarg :width )
        ( height :accessor board-height :initarg :height )
        ( mines :accessor board-mines :initarg :mines )
        ( tiles :accessor board-tiles :initarg :tiles )
    )
)
( defun generate-board ( width height mines &key ( tiles nil ) ( demo nil ) )
    ( if ( null tiles ) ( setf tiles ( random-tiles ( * width height ) mines '()
) ) )
    ( if demo
        ( format t "Mine location list: ~a~%~%" tiles )
    )
    ( setf tiles ( init-tiles tiles ) )
    ( if demo
        ( format t "Tile object list: ~a~%~%" tiles )
    )
    ( link-tiles tiles width height )
    ( eval-tiles tiles )
    ( cond
        ( demo ( format t "Tile value and link example: " ) ( tile-info ( nth 50
tiles ) ) )
    )
    ( setf *board* ( make-instance 'board :width width :height height :mines
mines :tiles tiles ) )
    nil
)

; Generate a random list of 0s and 1s where each 1 represents a mine
; Takes the number of tiles, number of mines, and a list as inputs
( defun random-tiles ( n m l &aux v)
    ( cond
        ( ( = 0 n ) l )
        ( t
            ( setf v ( random n ) )
            ( if ( < v m )
                ( random-tiles ( - n 1 ) ( - m 1 ) ( cons 1 l ) )
                ( random-tiles ( - n 1 ) m ( cons 0 l ) )
            )
        )
    )
)
```

```
; Generate a list of tile objects
; Takes a list of 0s and 1s as input where 1s are mines
( defun init-tiles ( tiles &aux tiles-c )
    ( setf tiles-c '() )
    ( dotimes ( n ( length tiles ) )
        ( if ( = 1 ( nth n tiles ) )
            ( setf tiles-c ( append tiles-c ( list ( make-instance 'tile :mine t
:name n ) ) ) )
            ( setf tiles-c ( append tiles-c ( list ( make-instance 'tile :name n
) ) ) )
        )
    )
    tiles-c
)

; Link a list of tile objects together
; Takes a list of tile objects, the width of the board, and the height of the
board as inputs
( defun link-tiles ( tiles width height &aux r c ti )
    ( dotimes ( n ( length tiles ) )
        ( setf ti ( nth n tiles ) )
        ( setf ( values r c ) ( floor n width ) )
        ( if ( and ( /= 0 r ) ( /= 0 c ) ) ( setf ( tile-nw ti ) ( nth ( - n
width 1 ) tiles ) ) )
        ( if ( /= 0 r ) ( setf ( tile-n ti ) ( nth ( - n width ) tiles ) ) )
        ( if ( and ( /= 0 r) ( /= ( - width 1 ) c ) ) ( setf ( tile-ne ti ) ( nth
( - n ( - width 1 ) ) tiles ) ) )
        ( if ( /= 0 c ) ( setf ( tile-w ti ) ( nth ( - n 1 ) tiles ) ) )
        ( if ( /= ( - width 1 ) c ) ( setf ( tile-e ti ) ( nth ( + n 1 ) tiles )
) )
        ( if ( and ( /= ( - height 1 ) r) ( /= 0 c ) ) ( setf ( tile-sw ti ) (
nth ( + n ( - width 1 ) ) tiles ) ) )
        ( if ( /= ( - height 1 ) r ) ( setf ( tile-s ti ) ( nth ( + n width )
tiles ) ) )
        ( if ( and ( /= ( - height 1 ) r ) ( /= ( - width 1 ) c ) ) ( setf (
tile-se ti ) ( nth ( + n width 1 ) tiles ) ) )
    )
)

; Calculate the number of mines adjacent to each tile and set that as the value
for the tile
; Takes a list of tile objects as input
( defun eval-tiles ( tiles &aux ti v )
    ( dotimes ( n ( length tiles ) )
        ( setf v 0 )
```

```
        ( setf ti ( nth n tiles ) )
        ( if ( and ( not ( null ( tile-nw ti ) ) ) ( tile-mine ( tile-nw ti ) ) )
( setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-n ti ) ) ) ( tile-mine ( tile-n ti ) ) ) (
setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-ne ti ) ) ) ( tile-mine ( tile-ne ti ) ) )
( setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-w ti ) ) ) ( tile-mine ( tile-w ti ) ) ) (
setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-e ti ) ) ) ( tile-mine ( tile-e ti ) ) ) (
setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-sw ti ) ) ) ( tile-mine ( tile-sw ti ) ) )
( setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-s ti ) ) ) ( tile-mine ( tile-s ti ) ) ) (
setf v ( + v 1 ) ) )
        ( if ( and ( not ( null ( tile-se ti ) ) ) ( tile-mine ( tile-se ti ) ) )
( setf v ( + v 1 ) ) )
        ( setf ( tile-value ti ) v )
    )
)

; Displays the current board bound to *board*
( defun display-board ( &aux a b ti )
    ( if ( null *board* ) ( return-from display-board ) )
    ( format t "~%       ")
    ( setf a 0 )
    ( dotimes ( n ( board-width *board* ) )
        ( if ( = a 10 ) ( setf a 0 ) )
        ( format t "~a " a )
        ( setf a ( + a 1 ) ) )
    )
    ( format t "~%     +")
    ( dotimes ( n ( board-width *board* ) )
        ( format t "--" )
    )
    ( format t "-+~%" )
    ( setf a 0 )
    ( setf b 0 )
    ( dotimes ( n ( length ( board-tiles *board* ) ) )
        ( setf ti ( nth n ( board-tiles *board* ) ) )
        ( if ( = b 0 )
            ( cond
                ( ( < a 10 ) ( format t  "   ~a | " a) )
                ( ( < a 100 ) ( format t "  ~a | " a) )
                ( t ( format t " ~a | " a ) )
```

```
            )
        )
        ( cond
            ( ( tile-flag ti ) ( format t "> " ) )
            ( ( tile-revealed ti )
                ( if ( tile-mine ti ) ( format t "X ") ( format t "~a " ( tile-
value ti ) ) )
            )
            ( t ( format t "# ") )
        )
        ( setf b ( + b 1 ) )
        ( if ( = b ( board-width *board* ) ) ( setf b 0 ) )
        ( if ( = 0 b ) ( setf a ( + a 1 ) ) )
        ( if ( = 0 b ) ( terpri ) )
    )
    ( terpri )
)


( defun demo--board ( &aux a)
    ( format t "~% >>> Testing board generation and display~%~%")
    ( format t "Generating a 10x10 board~%" )
    ( generate-board 10 10 20 :demo t )
    ( dotimes ( n 100 )
        ( setf a ( random 4 ) )
        ( if ( /= a 1 )
            ( cond
                ( ( tile-mine ( nth n ( board-tiles *board* )  ) )
                    ( setf ( tile-flag ( nth n ( board-tiles *board* ) ) ) t )
                )
                ( t ( setf ( tile-revealed ( nth n ( board-tiles *board* ) ) ) t
) )
            )
        )
    )
    ( format t "~%Displaying board with around 75% of tiles revealed~%" )
    ( display-board )

    ( format t "Displaying a 20x15 board~%")
    ( generate-board 20 15 40 )
    ( display-board )
)
```