Kuncheng Feng
CSC 466 Presentation

Location Object

Abstract

Due to the induced demand of RandomPlayerPlus to keep track of locations on a board,it needs to constantly move locations to different lists (explored, unexplored, preferred), this class is created to make it easier.

Demo

The reason I feel I need this new class is because the member and remove function cannot correctly match lists with same elements:

```
[5]> (equal '(x y) '(x y))
T
[6]> (equal '(x y) (car '((x y) (a b))))
T
[7]> (member '(x y) '((x y) (a b)))
NIL
[8]> (member '(x . y) '((x . y) (a . b)))
NIL
[9]> (remove '(x y) '((x y) (a b)))
((X Y) (A B))
```

However, they can recognize the same instance in a list:

```
[10]> (setf l1 (newLocation 0 0))
#<LOCATION #x1AADA96D>
[11]> (setf l2 (newLocation 1 0))
#<LOCATION #x1AADBAA1>
[12]> (setf ls (list l1 l2))
(#<LOCATION #x1AADA96D> #<LOCATION #x1AADBAA1>)
[13]> (member l1 ls)
(#<LOCATION #x1AADA96D> #<LOCATION #x1AADBAA1>)
[14]> (remove l1 ls)
(#<LOCATION #x1AADBAA1>)
```

The class also include some functions for easier manipulation:

```
[15]> (setf allL (generateAllLocations 10 10))
(#<LOCATION #x1AAE2ACD> #<LOCATION #x1AAE2EAD> #<LOCATION #x1AAE328D>
#<LOCATION #x1AAE366D>
 #<LOCATION #x1AAE3A4D> #<LOCATION #x1AAE3E2D> #<LOCATION #x1AAE420D>
#<LOCATION #x1AAE45ED>
 #<LOCATION #x1AAE49CD> #<LOCATION #x1AAE4DAD> #<LOCATION #x1AAE518D>
#<LOCATION #x1AAE556D>
 #<LOCATION #x1AAE594D> #<LOCATION #x1AAE5D2D> #<LOCATION #x1AAE610D>
#<LOCATION #x1AAE64ED>
 #<LOCATION #x1AAE68CD> #<LOCATION #x1AAE6CAD> #<LOCATION #x1AAE708D>
#<LOCATION #x1AAE746D>
 #<LOCATION #x1AAE784D> #<LOCATION #x1AAE7C2D> #<LOCATION #x1AAE800D>
#<LOCATION #x1AAE83ED>
 #<LOCATION #x1AAE87CD> #<LOCATION #x1AAE8BAD> #<LOCATION #x1AAE8F8D>
#<LOCATION #x1AAE936D>
 #<LOCATION #x1AAE974D> #<LOCATION #x1AAE9B2D> #<LOCATION #x1AAE9F0D>
#<LOCATION #x1AAEA2ED>
 #<LOCATION #x1AAEA6CD> #<LOCATION #x1AAEAAAD> #<LOCATION #x1AAEAE8D>
#<LOCATION #x1AAEB26D>
 #<LOCATION #x1AAEB64D> #<LOCATION #x1AAEBA2D> #<LOCATION #x1AAEBE0D>
#<LOCATION #x1AAEC1ED>
 #<LOCATION #x1AAEC5CD> #<LOCATION #x1AAEC9AD> #<LOCATION #x1AAECD8D>
#<LOCATION #x1AAED16D>
 #<LOCATION #x1AAED54D> #<LOCATION #x1AAED92D> #<LOCATION #x1AAEDD0D>
#<LOCATION #x1AAEE0ED>
 #<LOCATION #x1AAEE4CD> #<LOCATION #x1AAEE8AD> #<LOCATION #x1AAEEC8D>
#<LOCATION #x1AAEF06D>
 #<LOCATION #x1AAEF44D> #<LOCATION #x1AAEF82D> #<LOCATION #x1AAEFC0D>
#<LOCATION #x1AAEFFED>
 #<LOCATION #x1AAF03CD> #<LOCATION #x1AAF07AD> #<LOCATION #x1AAF0B8D>
#<LOCATION #x1AAF0F6D>
 #<LOCATION #x1AAF134D> #<LOCATION #x1AAF172D> #<LOCATION #x1AAF1B0D>
#<LOCATION #x1AAF1EED>
 #<LOCATION #x1AAF22CD> #<LOCATION #x1AAF26AD> #<LOCATION #x1AAF2A8D>
#<LOCATION #x1AAF2E6D>
 #<LOCATION #x1AAF324D> #<LOCATION #x1AAF362D> #<LOCATION #x1AAF3A0D>
#<LOCATION #x1AAF3DED>
 #<LOCATION #x1AAF41CD> #<LOCATION #x1AAF45AD> #<LOCATION #x1AAF498D>
#<LOCATION #x1AAF4D6D>
 #<LOCATION #x1AAF514D> #<LOCATION #x1AAF552D> #<LOCATION #x1AAF590D>
#<LOCATION #x1AAF5CED>
 #<LOCATION #x1AAF60CD> #<LOCATION #x1AAF64AD> #<LOCATION #x1AAF688D>
#<LOCATION #x1AAF6C6D>
 #<LOCATION #x1AAF704D> #<LOCATION #x1AAF742D> #<LOCATION #x1AAF780D>
#<LOCATION #x1AAF7BED>
 #<LOCATION #x1AAF7FCD> #<LOCATION #x1AAF83AD> #<LOCATION #x1AAF878D>
#<LOCATION #x1AAF8B6D>
 #<LOCATION #x1AAF8F4D> #<LOCATION #x1AAF932D> #<LOCATION #x1AAF970D>
#<LOCATION #x1AAF9AED>
 #<LOCATION #x1AAF9ECD> #<LOCATION #x1AAFA2AD> #<LOCATION #x1AAFA68D>
#<LOCATION #x1AAFAA6D>)
[16]> (setf 5-5 (getFromList 5 5 allL))
#<LOCATION #x1AAC7CDD>
[17]> (location-x 5-5)
5
[18]> (location-y 5-5)
5
[19]> (setf adjacents (getAdjacents 5-5 allL))
(#<LOCATION #x1AAC7CF1> #<LOCATION #x1AAC7CC9> #<LOCATION #x1AAC7DA5>
#<LOCATION #x1AAC7C15>)
[20]> (loop for l in adjacents do (format t "Location X Y: ~A ~A~%"
(location-x l) (location-y l)))
Location X Y: 5 6
Location X Y: 5 4
Location X Y: 6 5
Location X Y: 4 5
NIL
[21]> (getAdjacents (getFromList 0 4 allL) allL)
(#<LOCATION #x1AAC76ED> #<LOCATION #x1AAC76C5> #<LOCATION
#x1AAC77A1>)
[22]> (getAdjacents (getFromList 9 9 allL) allL)
(#<LOCATION #x1AAC7E31> #<LOCATION #x1AAC7D7D>)
[23]> (getLeftAdjacent (getFromList 2 2 allL) allL)
#<LOCATION #x1AAC7779>
```

Relevant Code

About half of the code is shown here, as I felt they are the most important, plus they all do what their function/method name suggests.

The getAdjacents function takes in a location, and returns all adjacent location instances in a list, if they are present in the list passed in.
The idea is to pass in a list of unexplored locations, and find all neighboring locations that are relative to the current location.

```
(defmethod getAdjacents((l location) (ls list) &aux adjacent result)
    (setf result (list))

    ; Left
    (setf adjacent (getLeftAdjacent l ls))
    (if (not (equal adjacent nil))
        (setf result (cons adjacent result))
    )

    ; Right
    (setf adjacent (getRightAdjacent l ls))
    (if (not (equal adjacent nil))
        (setf result (cons adjacent result))
    )

    ; Up
    (setf adjacent (getAboveAdjacent l ls))
    (if (not (equal adjacent nil))
        (setf result (cons adjacent result))
    )

    ; Down
    (setf adjacent (getBelowAdjacent l ls))
    (if (not (equal adjacent nil))
        (setf result (cons adjacent result))
    )

    result
)
```

The get(Left/Right/Above/Below)Adjacent method all acquire the x and y coordinates of a location, and try to acquire the one next to it from the list.

```
(defmethod getAboveAdjacent((l location) (ls list) &aux x y)
    (setf x (location-x l))
    (setf y (location-y l))
    (getFromList x (- y 1) ls)
)

(defmethod getBelowAdjacent((l location) (ls list) &aux x y)
    (setf x (location-x l))
    (setf y (location-y l))
    (getFromList x (+ y 1) ls)
)

(defmethod getLeftAdjacent((l location) (ls list) &aux x y)
    (setf x (location-x l))
    (setf y (location-y l))
    (getFromList (- x 1) y ls)
)

(defmethod getRightAdjacent((l location) (ls list) &aux x y)
    (setf x (location-x l))
    (setf y (location-y l))
    (getFromList (+ x 1) y ls)
)
```

The getFromList method will dig through a list of locations, and return an instance with matching x and y.

```lisp
(defmethod getFromList(x y (ls list) &aux location)
    (setf location (car ls))
    (cond
        ; End of the list
        ((equal location nil)
            nil
        )
        ; Location's X and Y matches
        ((and (= x (location-x location)) (= y (location-y location)))
            location
        )
        ; No match, continue the search
        (t
            (getFromList x y (cdr ls))
        )
    )
)
```