COMP 50 Lab: Mapping the Nearest Hospital

November 13–14, 2013

Today you will make maps of hospitals, and you will have a chance to build an interactive graphics application that finds the hospital closest to the mouse.

How to install more geographic names

The USGS names database has over 2 million names. I've created a medium-size database for our use. You'll need to install another package:

1. Launch DrRacket
2. From the File menu choose Install Package
3. In the "Package Source" box, type

   ```
   http://www.cs.tufts.edu/comp/50/packages/medium-geonames.zip
   ```

You will get a spray of ugly error messages. **Ignore them.** As soon as the Close button lights up (black text instead of gray text), press it.

Increasing the memory limit

Go the Racket menu, choose the "Limit Memory…" option, and set the memory limit to 512MB.

Testing the installation

Unfortunately it is not possible to install new teachpacks on the teachpack menu. Instead you have to put

```
(require medium-geo)
```

into your source code.[1]

If your installation has worked correctly, you should be able to do this in DrRacket's Interactions window:

```
> (require medium-geo)
> (length geonames)
38166
> (third geonames)
(make-usgs "Ariel Point" "Cliff" "Coconino" "AZ" 36.1535944 -112.0010001)
> (fourth geonames)
(make-usgs "Arizona State Hospital" "Hospital" "Maricopa" "AZ" 33.4528158 -112.0243292)
```

[1] If this looks like it is slowing down your computer a lot, you can try `(require geo)` instead and see if it is any faster.

Start a source file for your lab with the line

```
(require medium-geo)
```

This line will give you access to `geonames`, which is a `(listof point-of-interest)`. It will also give you access to the `usgs` structure used to define points of interest.

Data definitions

For the lab you will use the same data definitions you used last week:

A. *Degrees* is a number measuring an angle in degrees.

B. A *feature-class* is one of the following strings: `"Airport"`, `"Arch"`, `"Area"`, `"Arroyo"`, `"Bar"`, `"Basin"`, `"Bay"`, `"Beach"`, `"Bench"`, `"Bend"`, `"Bridge"`, `"Building"`, `"Canal"`, `"Cape"`, `"Cemetery"`, `"Census"`, `"Channel"`, `"Church"`, `"Civil"`, `"Cliff"`, `"Crater"`, `"Crossing"`, `"Dam"`, `"Falls"`, `"Flat"`, `"Forest"`, `"Gap"`, `"Glacier"`, `"Gut"`, `"Harbor"`, `"Hospital"`, `"Island"`, `"Isthmus"`, `"Lake"`, `"Lava"`, `"Levee"`, `"Locale"`, `"Military"`, `"Mine"`, `"Oilfield"`, `"Park"`, `"Pillar"`, `"Plain"`, `"Populated Place"`, `"Post Office"`, `"Range"`, `"Rapids"`, `"Reserve"`, `"Reservoir"`, `"Ridge"`, `"School"`, `"Sea"`, `"Slope"`, `"Spring"`, `"Stream"`, `"Summit"`, `"Swamp"`, `"Tower"`, `"Trail"`, `"Tunnel"`, `"Unknown"`, `"Valley"`, `"Well"`, or `"Woods"`.

C. A *point-of-interest* is a structure:

```
(make-usgs name class county state latd lond)
```

where

- `name` is a string
- `class` is a *feature-class*
- `county` is a string representing a county name
- `state` is string containing the two-letter postal abbreviation for an American state
- `latd` is the point's latitude in degrees
- `lond` is the point's longitude in degrees

D. As on the homework, A `(2Dpoint X)` is a structure

```
(make-point x y value)
```

where `x` and `y` are numbers and `value` is an *X*.

E. A `(bbox X)`, also known as a *bounding box measured with X*, is a structure:

```
(make-bbox south north west east)
```

where `south`, `north`, `west`, and `east` are X's.

Data examples

Here are some helpful points of interest:

```
(make-usgs "Ash Creek Ranch" "Locale" "Graham" "AZ" 33.364221 -110.0239777)
(make-usgs "MGH" "Hospital" "Suffolk" "MA" 42.3625989 -71.0693009)
(make-usgs "USF Central Utility" "Building" "Pinellas" "FL" 27.7636 -82.6362)
(make-usgs "New England Montessori Children's House" "School" "Worcester" "MA" 42.417128 -
```

Functions defined for you

Standard functions on lists

At http://docs.racket-lang.org/htdp-langs/intermediate.html#%28part._htdp-intermediate._.Higher-.Order_.Functions%29 you will find a list of all the predefined abstract list functions. Here are some of the important ones, most of which we've discussed in class:

- `andmap` and `ormap`
- `filter`
- `map`
- `argmin` and `argmax`
- `foldl` and `foldr`

Functions specific to this lab

This function tells if a point of interest is a hospital:

```
; hospital? : point-of-interest -> boolean
; tell if the given point of interest is a hospital
(define (hospital? p)
  (string=? (usgs-class p) "Hospital"))

(check-expect
   (hospital?
    (make-usgs "Avery Hospital" "Hospital" "Middlesex" "CT" 41.7292668 -72.7056497))
   true)
(check-expect
   (hospital?
    (make-usgs "Bailey Creek" "Stream" "New Haven" "CT" 41.2737092 -72.6450939))
   false)
```

Here's a function that can tell you if a point of interest lies in a bounding box. The function returns a function!

```
; within-bbox : (bbox degrees) -> (point-of-interest -> boolean)
; to return a function that tells if a given point-of-interest is
;                                    within the given bounding box
(define (within-bbox bb)
```

```
  (local [(define (within? pt)
            (and (<= (bbox-west bb) (usgs-lond pt) (bbox-east bb))
                 (<= (bbox-south bb) (usgs-latd pt) (bbox-north bb))))]
         within?))

(define bb1 (make-bbox 42 43 -72 -70))
(check-expect ((within-bbox bb1)
               (make-usgs "Boston" "test pt" "Middlesex" "MA" 42.33 -71)) true)
(check-expect ((within-bbox bb1)
               (make-usgs "North" "test pt" "Middlesex" "MA" 43.33 -71)) false)
(check-expect ((within-bbox bb1)
               (make-usgs "STL" "test pt" "?" "MO" 38.6  -90.2)) false)
```

The next function wraps a point of interest inside a `2Dpoint`. It converts the GPS coordinate system to (x, y) coordinates suitable for use in making flat maps. There is some distortion; points at higher latitudes appear further apart than they actually are on the surface of the Earth. You can download both functions from `http://www.cs.tufts.edu/comp/50/labs/hospitals.rkt`.

```
; xy-converter : bbox number -> (point-of-interest -> (2Dpoint point-of-interest))
; to return a function that attaches (x, y) coordinates to the given point-of-interest,
; relative to the given bbox.  Height is the height of the bbox in pixels
(define (xy-converter bb height)
  (local
   [(define pixels-per-latitude-degree ; # of pixels representing 1 degree of latitude
      (/ height (- (bbox-north bb) (bbox-south bb))))

    (define pixels-per-longitude-degree ; # of pixels representing 1 degree of longitude
      (* (cosd (average (bbox-north bb) (bbox-south bb)))
         pixels-per-latitude-degree))

    ; x-in-pixels : degrees -> number
    ; convert the given longitude to an x distance from the west boundary
    (define (x-in-pixels x-degrees)
      (* pixels-per-longitude-degree (- x-degrees (bbox-west bb))))

    ; y-in-pixels : degrees -> number
    ; convert the given latitude to a y distance down from the north boundary
    (define (y-in-pixels y-degrees)
      (* pixels-per-latitude-degree (- (bbox-north bb) y-degrees)))

    ; convert : point-of-interest -> (2Dpoint point-of-interest)
    ; wrap `pt` in its (x, y) coordinates within `bb`
    (define (convert pt)
      (make-point (x-in-pixels (usgs-lond pt))
                  (y-in-pixels (usgs-latd pt))
                  pt))]
   convert))

; cosd : degrees -> number
; to return the cosine of the given number of degrees
(define (cosd degrees)
  (cos (* pi (/ degrees 180))))

(check-within (cosd 45) (/ (sqrt 2) 2) EPSILON)
```

```
(check-within (cosd 90) 0 EPSILON)

; average : number number -> number
; to return the average of the two given numbers
(define (average x y)
  (/ (+ x y) 2))

(check-expect (average 6 7) 6.5)


(define c1 (xy-converter (make-bbox 42 43 -72 -70) 100))

(define west (make-usgs "West" "test point" "?" "MA" 42.5 -72))
(define wnw  (make-usgs "West Northwest" "test point" "?" "MA" 42.75 -72 ))
(define nw   (make-usgs "Northwest" "test point" "?" "MA" 43 -72))
(define ne   (make-usgs "Northeast" "test point" "?" "MA" 43 -70))
(define sw   (make-usgs "Southwest" "test point" "?" "MA" 42 -72))
(define sse  (make-usgs "South Southeast" "test pt" "?" "MA" 42 -70.5))
(define mid  (make-usgs "Middle" "test pt" "?" "MA" 42.5 -71))

(define shrink (cosd 42.5))

(check-within (c1 west) (make-point 0 50 west) EPSILON)
(check-within (c1 wnw) (make-point 0 25 wnw) EPSILON)
(check-within (c1 nw) (make-point 0 0 nw) EPSILON)
(check-within (c1 sw) (make-point 0 100 sw) EPSILON)
(check-within (c1 ne) (make-point (* 200 shrink) 0 ne) EPSILON)
(check-within (c1 sse) (make-point (* 2 shrink 75) 100 sse) EPSILON)
(check-within (c1 mid) (make-point (* 2 shrink 50) 50  mid) EPSILON)


(define EPSILON 0.0001)
```

Part One: Static maps

All of these problems are to be solved **without defining any *recursive* functions**. You may recognize a couple of functions from last week; if you or your partner have already written these functions, you don't have to write them again. Just use the ones from the last lab.

1. Define a function `place-2Dpoints` which is given an image and a list of `(2Dpoint X)` and returns a copy of the original image in which each point is marked by a solid black circle 2.5 pixels in diameter.

   Teachpack knowledge: You will find `place-image` helpful here.

2. Define a value `northeast` which is a bounding box bounded on the south by 38 degrees, on the north by 48 degrees, on the west by -91 degrees, and on the east by -67 degrees.

3. Using function `within-bbox` and value `northeast`, the following code defines a function `northeast?` that tells if a given point of interest lies within the `northeast` bounding box:

   ```
   (define northeast? (within-bbox northeast))
   ```

   Using the `northeast?` function, define a value `northeast-points` that contains all the points of interest that lie within the `northeast` bounding box.

4. Define a value `northeast-hospitals` that contains all the *hospitals* that lie within the `northeast` bounding box. It should be a `(listof point-of-interest)`.

5. Using the function `xy-converter` that is provided for you, define a function `as-northeast-point` that converts a given point of interest into a `(2Dpoint point-of-interest)` with `x` and `y` coordinates suitable for a map of the `northeast` bounding box. The map is planned to be 720 pixels high.

   As a test case, MGH should be located within 1 pixel of coordinates $(1050, 406)$.

6. Define a value `northeast-map` which is an in image of a map 1300 pixels wide and 720 pixels high. The map should contain a marker for every point of interest in the `northeast` bounding box.

   Teachpack knowledge: You will find `empty-scene` helpful here.

7. Define a value `northeast-hospital-map` which is an image of a map 1300 pixels wide and 720 pixels high. The map should containin a marker for every *hospital* in the `northeast` bounding box.

Part Two: Animated maps

If you are in a severe automobile crash in a rural part of the United States, chances are that you will be flown in a helicopter to the nearest hospital.[2] In this part of the lab you simulate a flying helicopter that needs to know where the nearest hospital is.

None of your code may use recursion.

8. Define a function that is given (x, y) pixel coordinates and a *nonempty* list of `(2Dpoint point-of-interest)` and returns the point of interest that is *closest* to the given point. (If a particular 2D-point is located at (x_p, y_p), its distance from (x, y) is approximately the square root of $(x_p - x)^2 + (y_p - y)^2$.[3]

9. Located at

   ```
   (make-usgs "nowhere" "Unknown" "?" "IN" 40.232364 -86.618958)
   ```

   Convert this location to a 2D-point and find the nearest hospital. Verify that the hospital is named for a Catholic saint canonized in the 1730s.

   Hint: Try using `map` with `as-northeast-point`.

10. *Write a data definition* for a world state with the following choices:

    - The world state contains a list of hospitals and the (x, y) coordinates of the helicopter
    - The world state consists of a single hospital

    A hospital should be represented as a `(2Dpoint point-of-interest)`.

11. Design a `big-bang` program with the following properties:

    A. The world is rendered as a map on which every hospital is marked by a solid black circle with a radius of 2.5 pixels. But the hospital that is closest to the helicopter should be marked by a solid red circle with a radius of 4 pixels.

    B. Every time a mouse-event occurs, the world state is updated to assume that the flying helicopter is located at the current mouse coordinates.

    C. When the `q` key is pressed, the simulation ends, and it returns the `point-of-interest` that represents the nearest hospital to the helicopter.

[2] Actually, you will probably be flown to the nearest tertiary care center, but the USGS doesn't track those.

[3] Hint: the closest point is the one with the smallest square distance—you don't need to compute the square root.

The lab machines are too slow. Therefore I recommend that your main function take a list of hospitals as its argument. To shorten the list, try

```
(half-list (half-list (half-list northeast-hospitals)))
```

Where `half-list` is defined as follows:

```
; half-list : (listof X) -> (listof X)
; to return every other element of the given list
(define (half-list xs)
  (cond [(empty? xs) '()]
        [(empty? (rest xs)) '()]
        [(cons? (rest xs)) (cons (first xs) (half-list (rest (rest xs))))]))

(check-expect (half-list '()) '())
(check-expect (half-list '(1)) '())
(check-expect (half-list '(1 2 3)) '(1))
(check-expect (half-list '(1 2 3 4)) '(1 3))
```

Alert: Function `as-northeast-point` uses *inexact* numbers. Test cases for several functions will have to use `check-within`.

Challenge Problems

These are problems that might amuse you, or you might conceivably decide that you want to do one so you can show it in your learning portfolio.

12. Your animation is probably having trouble keeping up with the mouse. But the map never changes! Try storing an image of the map in the world state, so that you don't have to draw all the hospitals each time.
13. Change the world state and your program so that you show *all* the points of interest on the map, but that only the nearest *hospital* is illuminated in red.
14. Write a function that is given a bounding box and produces a map of all points of interest within that bounding box. If you try from 32 to 48 degrees north and from -118 degrees to -67 degrees west, you will be able to see most of the continental United States. Or zoom in on your home state.
15. Extend your animation so that the bounding box becomes part of the world state. Respond to keyboard events `"wheel-up"` and `"wheel-down"` by zooming in and out. Respond to `"left"`, `"right"`, `"up"`, and `"down"` by panning the map in the given direction.
16. Even if you save a pre-drawn map, you will find that with 4,000 hospitals, the computer is hard pressed to keep up with the mouse. Use a 2D-tree to find the nearest hospital quickly.

Submitting the lab

Ten minutes before the end of the lab, put the following text at the beginning of a DrRacket file. You may use an empty file or the source code you have been working on:

```
#|
What I did during this lab:
   (you fill in this part)

What I learned during this lab:
   (you fill in this part)

|#
```

Finally, *submit this file through the handin server* as `lab-map-hospitals`. **Submit it as lab, not as homework.** You will need to submit it using *two* usernames connected by a + sign, as in

```
Jane.Doe+Richard.Roe
```

You submit using Jane Doe's password.