Program #4 Helper

Prof Bill - May 2018

This is my helper for P4, Rect210. thanks… yow, bill

TODO List
P4 isn't really difficult, but it has a lot of moving pieces. Therefore, it's critical to stay organized. Step #1 - keep a TODO list. Here's mine.

1. **Foundational work** - getting basic rect and rect factory working
    a. Copy P4 classes from k: drive
    b. Create MyRect is-a Rect210 and MyRectFactory is-a Rect210Factory.
    c. Code em up; try them out in a tiny test bed

2. **P4Generate** - create console program to generate random rects
    a. Setup class and main()
    b. Add read/write rect file to RectIO class
    c. Create rects with factory and write to file
    d. Add user queries for all the random rect parameters

3. **P4Intersect** - create console program to find rect intersections
    a. Setup class and main()
    b. Brute force
    c. Sweep algo
    d. Call algos in main() and time their execution
    e. Write RectIntersect files

4. **Program4 GUI** -
    a. Hello, World
    b. VBOX + initButtons() + initRects() + scene show = GO!
    c. Connect to Sweep guru to do step-by-step algo

See? There are a lot of steps. But none of them are earth-shattering.
More help(er) coming soon. Email me!
thanks… yow, bill

Following my TODO list…

Step 1 - Foundation work

Getting basic rects up and running… my classes are:

- ➢ MyRect is-a Rect210
- ➢ MyRectFactory is-a Rect210Factory
- ➢ Get a main() up so you can create rects and print them.

In MyRect, the intersects() method is a toughie. How can we tell if two rects intersect? I like this approach:
www.hackerearth.com/practice/notes/how-to-check-if-two-rectangles-intersect-or-not/

New, May 25 => My brute force and sweep line results are now identical. (huzzah!) How? Change your intersects() method, so that rects that share a border do **not** intersect. This means that intersection is borders crossing, not just adjacent. This gets rid of the problem where sweep line events with the same x coordinate sometimes under-count rect intersections. (huzzah again!)

Step 2 - P4Generate

Generate a file of random rects

- ➔ My main() class is P4Generate
- ➔ I created a class call RectIO to read and write rect files
- ➔ I added some code to query the user, P4Helper.java

This is meatball code, with nothing really difficult here.

New May 24 => I have filled in the code for RectIO to use our Rect factory. I did this to help you and also to show you how cool factories are.

Step 3 - P4Intersect

Determine rect intersections and write them to a file.
Run brute force, save.
Run Sweep line, save.

New, May 25 => I have rect test files and results for you on the k: drive. Start with the README_RESULTS pdf in that folder. Email me with any questions.

K:\18SP\CSC_210_1\common_area\program4\files_and_results

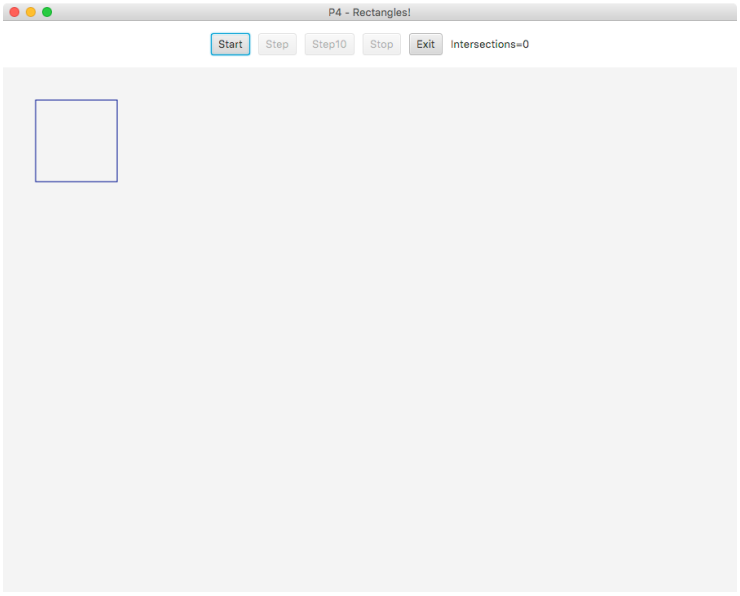
Step 4 - Program4 gui

Two recommendations:

- Keep your start() method small and organized. Break your gui up into sensible methods: initButtons(), initRectPane(), etc.
- I recommend a very simple MVC (Model View Controller) approach. Assign each button to a method in your gui: do_start(), doStep(), etc.

I used a JavaFX Pane to draw my Rectangles.

Here's mine. Optional! Yours may differ.



Sweep Line Algorithm

From our Princeton notes…

Sweep line algorithm pseudo code:

```
// Sweep Line algorithm for determining rect intersections
sweepLine( rectList)
   sweepList = new rect list (empty)
   intersectList = new rect intersect list (empty)
   pq = new PriorityQueue()    // use JFC; it's a heap
   initPQ( rectList)

// init priority queue for sweep line
initPQ( rectList)
   for each rect r: rectList {
      SweepEvent swe = create entry event for r.x   // upper left x
      pq.add( swe)
      swe = create exit event for r.xMax    // lower right x
      pq.add( swe)
   }

// removeMin() returns the next sweep event (rect) to process
SweepEvent removeMin()
   if pq.isEmpty() the return null    // no events, so sweep is done

   swe = pq.poll()
   if swe is entry
      sweepList.add( swe.rect)   // add this rect to sweep
   else    // exit event
      sweepList.remove( swe.rect)    // rm this rect from sweep
      for rect r: sweepList {    // upon exit, check for intersections
         if swe.rect intersects r
            intersectList.add( swe.rect, r)
      }
```

Notes:

- ★ In GUI, each step is a call to removeMin()
- ★ In p4Intersect, just call put removeMin() in a loop until entry
- ★ sweepList is an ArrayList of rects currently being swept; in Princeton algorithm it's a (fancy) intersect BST; we'll talk.

P4 Classes

We have lots going on and lots of classes. Here's a snapshot of all of them.

Shared code

Grab this shared code from the k: drive…

- P4Helper - query methods I used on P4Generate
- Rect210 - our rect interface
- Rect210Factory - our rect factory interface
- RectIntersect - simple rect intersect class: r1 and r2
- RectIO - read and write rect files; write rect intersect file; factory =cool!

My classes

Soon to be coded by you…

Foundation

- MyRect is-a Rect210 - intersects() is the toughie; see my notes
- MyRectFactory is-a Rect210Factory

Programs

- P4Generate - generate program main(); it's small
- P4Intersect - intersect program main(); it's small
- Program4 - P4 GUI; not small (ha); plug in SweepLineGuru

Intersect Algorithms

- BruteForceAlgorithm - run brute force on a list of rects, return list of rect intersects
- SweepLineGuru - sweep line algorithm; can be run step-by-step (removeMin method) or all in one (run method)
- SweepEvent - holds rect events during sweep algorithm; these are placed on the PQ

thanks… yow, bill

PS - I used Netbeans (run/generate javadoc) to create Javadoc pages for my solution. They're here: wtkrieger.faculty.noctrl.edu/csc210-spring2018/p4/javadoc/.