THOUGHTS ON A4

Lecture 19
CS2110 – Fall 2014

A4 and A5

2

A4. Grading of A4 almost finished. Should make grades and feedback accessible this evening.

A5. We have begun to populate the FAQ note for A5 on the CMS. Will put more info there later today, and after that, whenever necessary. Look at it often.

Get started on A5 soon! Get it done EARLY.

Before posting a question about A5 on the Piazza:

- See whether it is already answered on the Piazza!
- Look in the Java API specs for the answer
- Google your question

About Testing

3

Example conversation after finding a bug, a typo (an x for a y), in someone's BoundingBox.getCenter. After it was fixed, everything worked.

Did you test all the BoundingBox methods before moving on to BlockTree? No.

How long do you think it takes to write a Junit testing class to test all the methods in BoundingBox? I don't know.

When I, Gries, developed the solution, it took me 20 minutes to build such a BoundingBox tester, and I found 2 errors/typos.

How many hours did you and consultants spend looking for the error when the GUI didn't work, looking mainly at BlockTree.contains, BlockTree.overlap? Probably 6-7 hours, more

About Testing

4

Fact: We all make mistakes, both simple typos and logical mistakes.

Fact: We all have the urge to move on and write more code, without proper testing of what is already written.

Fact: When we give in to that urge, we often waste time.

SO
Continually discipline yourself to write and test code incrementally. Make sure that basic methods are correct before moving on to write code that calls those methods.

Keep things simple

5

/** Return true iff this bounding box overlaps with box. */
public boolean overlaps(BoundingBox box) {

```
double n=0;
Vector2D sidepoints = new Vector2D(0,0);
//box is to the right
Vector2D topright = new Vector2D (lower.x + getWidth(), lower.y);
Vector2D bottomleft = new Vector2D (upper.x - getWidth(), upper.y);
Vector2D topmidpoint = new Vector2D(lower.x + getWidth()/2, lower.y);
Vector2D botmidpoint = new Vector2D(lower.x - getWidth()/2, lower.y);
Vector2D boxtopright = new Vector2D (box.lower.x + box.getWidth(), box.lower.y);
Vector2D boxbottomleft = new Vector2D (box.upper.x - box.getWidth(), box.upper.y);
Vector2D boxtopmidpoint = new Vector2D(box.lower.x + box.getWidth()/2, box.lower.y);
Vector2D boxbotmidpoint = new Vector2D(box.lower.x - box.getWidth()/2, box.lower.y);
if (getArea() >= box.getArea()){
  if (this.contains(box.lower) || this.contains(box.upper) || this.contains(boxbottomleft) || this.contains(boxtopright)
     || contains(boxbotmidpoint) || contains(boxtopmidpoint)){  return true;
  } else{   return false; }
}else{
 if (box.contains(lower) || box.contains(upper) || box.contains(bottomleft) || box.contains(topright) ||
  box.contains(botmidpoint) || box.contains(topmidpoint)){    return true;
  }else{
  return false; }  }
```

Keep things simple and beautiful

6

There are two ways of constructing a piece of software:
One is to make it so simple that there are obviously no errors.
The other is to make it so complicated that there are no obvious errors. — Tony Hoare

Inside every large program, there is a small program trying to get out. — Tony Hoare

When I'm working on a problem, I never think about beauty. I think only how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.
— R. Buckminster Fuller

Keep things simple and beautiful

7

When your work starts to get long and complicated,

STOP, reflect, look for different approaches.

Simplify your work by avoiding useless clutter

When do two rectangles overlap?

8

Rectangles and do not overlap if one is to the right of the other or one is below the other. Otherwise, they overlap.

```
/** Return true if this box overlaps with box. */
public boolean overlaps(BoundingBox box) {
      if (upper.x < box.lower.x) return false;
      if (box.upper.x < lower.x) return false;
      if (upper.y < box.lower.y) return false;
      if (box.upper.y < lower.y) return false;
      return true;
}
```

BlockTree.overlap: too much case analysis

9

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
    if (block != null) {
        if (t.block != null) {
            return Block.overlaps(block, thisD, t.block, d);
        } else {
            if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
                return false;
            else  return overlaps(thisD, t.left, d) || overlaps(thisD, t.right, d);
        }
    } else {
        if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
            return false;
        else  return left.overlaps(thisD, t, d) || right.overlaps(thisD, t, d)
    }
}
```

BlockTree.overlaps: too much case analysis

10

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  if(this==null || t==null){return false;}
  if(!this.box.overlaps(thisD, t.box, d)){
   return false;
  }
  if(this.isLeaf()&&t.isLeaf())  return true;
  if(this.isLeaf()) return t.overlaps(d, this, thisD);
  if(t.isLeaf()){
     if(this.left.box.getArea()>this.right.box.getArea
          return this.left.overlaps(thisD,t,d) ||
                 this.right.overlaps(thisD, t, d);
  }
   return this.right.overlaps(thisD, t, d)||
        this.left.overlaps(thisD, t, d);
  }
  if(this.left.box.getArea()>this.right.box.getArea()){
```

What else is wrong with this?

Unnecessary clutter: "this." No spaces around operators, after if, before {

BlockTree.overlaps: too much case analysis

11

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  if (this==null || t==null) return false}
  if (!this.box.overlaps(thisD, t.box, d)) return false;
  if (isLeaf()) return t.overlaps(d, this, thisD);
  if (t.isLeaf()){
     if (left.box.getArea() > right.box.getArea()){
        return left.overlaps(thisD,t,d) ||  right.overla
     }
    return right.overlaps(thisD, t, d)  ||  left.overlap
  }
  if (left.box.getArea() > right.box.getArea())
    return left.overlaps(thisD, t.left, d)  ||  left.overl
           right.overlaps(thisD, t.left, d) || right.ove
  return right.overlaps(thisD, t.left, d) || right.overlaps(thisD,t.right,d) ||
         left.overlaps(thisD, t.left, d) || left.overlaps(thisD,t.right,d);
  }
```

Leafs not handled consistently

The clutter is removed. Much better! Can read it all

Still too much case analysis. Shouldn't be looking down into the left/right so much

A beautiful overlaps

12

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  // If the blocks don't overlap, return false.
  if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
        return false;
  // the blocks overlap
  // If the trees are both leafs, return true
  if (isLeaf()  &&  t.isLeaf()) return true;

  // Recurse on the longer of this and t
  if (box.getLength() > t.box.getLength())
    return left.overlaps(thisD, t, d) || right.overlaps(thisD, t, d);
  else
    return t.left.overlaps(d, this, thisD) || t.right.overlaps(d, this, thisD);
}
```

A beautiful overlaps

13

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
        return false;
  if (isLeaf()  &&  t.isLeaf()) return true;

  if (box.getLength() > t.box.getLength())
    return left.overlaps(thisD, t, d) || right.overlaps(thisD, t, d);
  else
    return t.left.overlaps(d, this, thisD) || t.right.overlaps(d, this, thisD);
}
```

Why is recursing on longer better? We provide intution

14

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
        return false;
  if (isLeaf()  &&  t.isLeaf()) return true;

  if (box.getLength() > t.box.getLength())
    return left.overlaps(thisD, t, d) || right.overlaps(thisD, t, d);
  else
    return t.left.overlaps(d, this, thisD) || t.right.overlaps(d, this, thisD);
}
```

Suppose t contains 2 blocks and depth of this tree is d.

Worst case: total of d recursive calls

Recurse on shorter? Need more case analysis

15

```
public boolean overlaps(Vector2D thisD, BlockTree t, Vector2D d) {
  if (!box.displaced(thisD).overlaps(t.box.displaced(d)))
        return false;
  if (isLeaf()  &&  t.isLeaf()) return true;
  if (one of the trees is a leaf) take care of this case
      // Takes up to d recursive calls

  if (box.getLength() < t.box.getLength())
    return left.overlaps(thisD, t, d) || right.overlaps(thisD, t, d);
  else
}   return t.left.overlaps(d, this, thisD) || t.right.overlaps(d, this, thisD);
```

Suppose t contains 2 blocks and depth of this tree is d.
2d recursive calls: d for left.overlap and d for right.overlap.

Summary

16

1. Code and test incrementally. Don't write a call on a method unless that method has been checked thoroughly.
2. Use already written methods –don't reinvent the wheel.
3. Strive for clarity, simplicity, brevity.
4. Avoid unnecessary clutter and case analysis.
5. Use returns in functions to avoid case analysis. See Code Style Guidelines cs.cornell.edu/courses/CS2110/2014fa/style_guidelines.html#returns
6. Don't accept your first "correct" method as the final one. Like an essay in English, it may need reorganizing, rethinking, reworking.