Code Jam 2008 - Qualification Round

Analysis: Fly Swatter

This was the hardest problem of the set, as we can see from the submission statistics, but 652 people succeeded in solving it correctly.

The first step in solving this problem involves a standard trick. Instead of looking for the region that will hit the fly's circular body, we look for the region where the center of the fly must avoid. The problem is thus transformed into an equivalent one where we thicken the size of the racquet to **t** + **f** and the radius of the strings from **r** to **r** + **f**. And we consider the fly to be simply a single point, only focusing on its center.

Now the main problem is to find the area of the holes (the areas between the strings, or between the strings and the racquet), and divide it by the area of the disc corresponding to the original racquet. It is easy to see this gives the probability that the racquet *does not* hit the fly.

A simplifying observation is that we can just look at the first quadrant of the circle, because the racquet is symmetrical.

Now we can go through each square gap in the racquet and check a few cases. Each case consists in computing the area of intersection between the square and the circle representing the inside of the racquet. The case where the square is totally inside or totally outside the circle is easy. The cases left when one corner, two or three corners of the square are inside the circle and the other corners are outside. There are no other cases because we just look at the intersection of a quarter of the circle with small squares.

Each of these cases can be solved by splitting the intersection area into triangles and a circular segment. Here's some code that does this:

```
double circle_segment(double rad, double th) {
  return rad*rad*(th - sin(th))/2;
}

double rad = R-t-f;
double ar = 0.0;
for (double x1 = r+f; x1 < R-t-f; x1 += g+2*r)
for (double y1 = r+f; y1 < R-t-f; y1 += g+2*r) {
  double x2 = x1 + g - 2*f;
  double y2 = y1 + g - 2*f;
  if (x2 <= x1 || y2 <= y1) continue;
  if (x1*x1 + y1*y1 >= rad*rad) continue;
  if (x2*x2 + y2*y2 <= rad*rad) {
    // All points are inside circle.
    ar += (x2-x1)*(y2-y1);
  } else if (x1*x1 + y2*y2 >= rad*rad &&
             x2*x2 + y1*y1 >= rad*rad) {
    // Only (x1,y1) inside circle.
    ar += circle_segment(rad, acos(x1/rad) - asin(y1/rad)) +
          (sqrt(rad*rad - x1*x1)-y1) *
          (sqrt(rad*rad - y1*y1)-x1) / 2;
  } else if (x1*x1 + y2*y2 >= rad*rad) {
    // (x1,y1) and (x2,y1) inside circle.
    ar += circle_segment(rad, acos(x1/rad) - acos(x2/rad)) +
          (x2-x1) * (sqrt(rad*rad - x1*x1)-y1 +
```

```
                    sqrt(rad*rad - x2*x2)-y1) / 2;
  } else if (x2*x2 + y1*y1 >= rad*rad) {
    // (x1,y1) and (x1,y2) inside circle.
    ar += circle_segment(rad, asin(y2/rad) - asin(y1/rad)) +
          (y2-y1) * (sqrt(rad*rad - y1*y1)-x1 +
                     sqrt(rad*rad - y2*y2)-x1) / 2;
  } else {
    // All except (x2,y2) inside circle.
    ar += circle_segment(rad, asin(y2/rad) - acos(x2/rad)) +
          (x2-x1)*(y2-y1) -
          (y2-sqrt(rad*rad - x2*x2)) *
          (x2-sqrt(rad*rad - y2*y2)) / 2;
  }
}
printf("Case #%d: %.6lf\n", prob++, 1.0 - ar / (PI*R*R/4));
```

This solution takes O(**S**^{2}) time, where **S** is the number of vertical strings of the racquet. It's not hard to come up with an O(**S**) solution because there are at most 4**S** border squares which can be found efficiently, but the previous solution was fast enough.

Instead of solving the problem exactly, an iterative solution which approximates the area to the needed precision would have also worked. One such solution uses divide and conquer by splitting the square into four smaller squares and then checking the simple cases where the squares are totally inside or totally outside the square. In the cases where the circle and square intersect just recurse if the current square is larger than some chosen precision. An observation is that we can divide every length by the radius of the racquet because it gets canceled in the division between the area of the gaps in the racquet and the disc area. This observation helps the iterative solution since we can make the number of iterations smaller. Here's some sample code:

```
double intersection(double x1, double y1,
                    double x2, double y2) {
  // the normalized radius is 1
  if (x1*x1 + y1*y1 > 1) {
    return 0;
  }
  if (x2*x2 + y2*y2 < 1) {
    return (x2-x1) * (y2-y1);
  }
  // EPS2 = 1e-6 * 1e-6
  if ((x2-x1)*(y2-y1) < EPS2) {
    // this trick helps in doing 10 or 100 times less
    // iterations than we would need to get the same
    // precision if we just return 0;
    return (x2-x1) * (y2-y1) / 2;
  }

  double mx = (x1 + x2) / 2;
  double my = (y1 + y2) / 2;

  return intersection(x1, y1, mx, my) +
    intersection(mx, y1, x2, my) +
    intersection(x1, my, mx, y2) +
    intersection(mx, my, x2, y2);
}
```