Chen, Ethan
October 7, 2020
CS10, Fall 2020, Pierson

PointQuadtree Testing

In order to test each of my methods, insert(), size(), and allPoints(), I simply created a PointQuadTree and added points. Once I was sure they were inserting correctly using the debugger, I called the size() method and printed the output to see if it would give the correct value, and used a for each loop to iterate through allPoints() and ensure each point was there.

DotTreeGUI Testing

The test methods simply gave a circle to search for a dot in, and took parameters for expectedCircleRectangle, expectedInCircle, and expectedHits. These three variables were predetermined "what-should-happen" variables that are compared to the actual results. The first two methods simply provide a number of points that have certain geometric operations performed on them in findInCircle(), while expectedHits is the number of points within the circle that would be selected as hits in a real simulation. If the expected number does not match up with the number of method calls or hits there actually are, it creates an error.

In order to test DrawingGUI and ensure my findInCircle() method and algorithm were working correctly, I used both given test methods as well as one of my own. In my own test, I followed the same general concept as in the two given, but changed the points and calculated what should have been the correct values for expectedCircleRectangle, expectedInCircle, and expectedHits by hand. Each of the test cases succeeded with my code, and so I believe the code works well. As well, I have simply experimented adding points in random locations and making sure the quadTree displays properly on the screen, with each child being a different rainbow color and having bounds that are a subset of its parents.

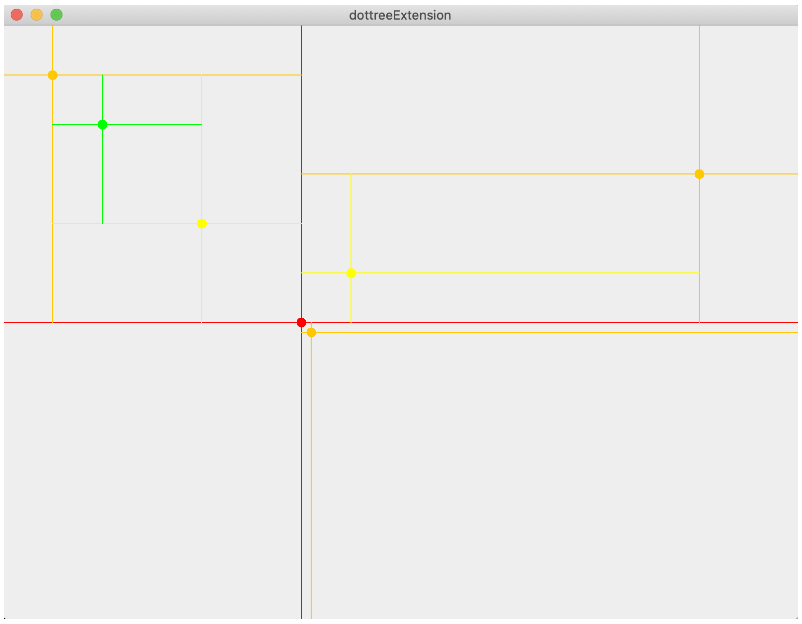


Figure 1: Ethan's Test Case

CollisionGUI Testing

Testing CollisionGUI was done mostly by eye, watching blobs as they collide with one another and turn red. However, I found a more elegant way of testing, by including in my extension an option to place a single, moveless blob. By placing a blob down and then placing another just so the edges barely touch, I could determine if the collision mechanism would work. To be even more precise, for this phase of the testing I upped the default radius of a blob to 30. Sure enough, the blobs did in fact turn red (although in some rare cases one or two would stay black, generally when clicking the same point very fast). I also would place blobs in the trajectory of a wanderer to make sure that as they pass over they turn red.

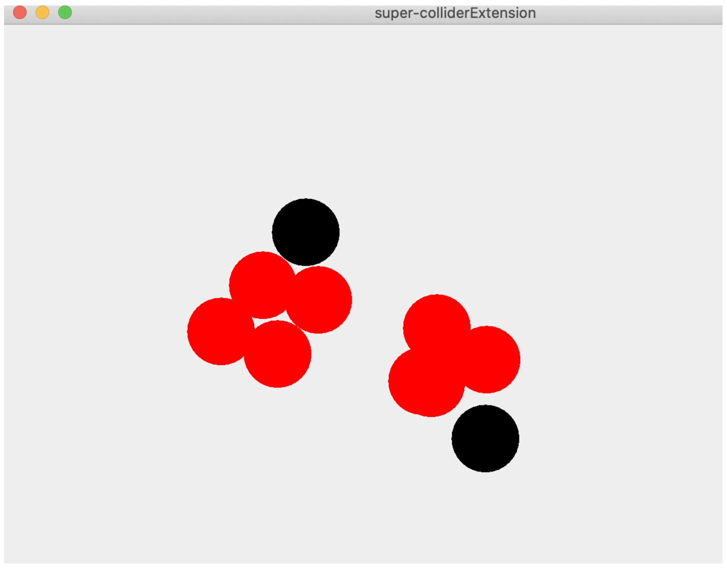
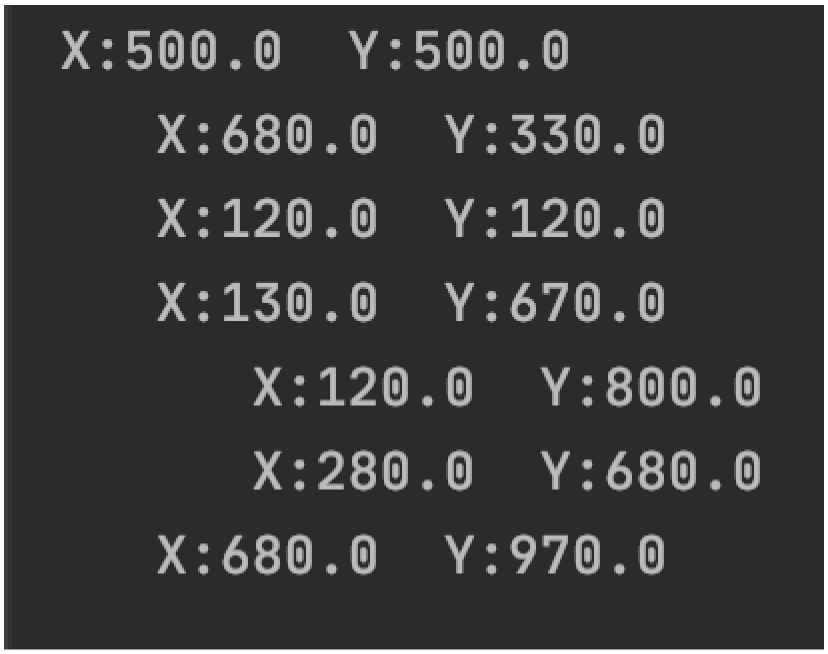


Figure 2: Manual Testing of CollisionGUI

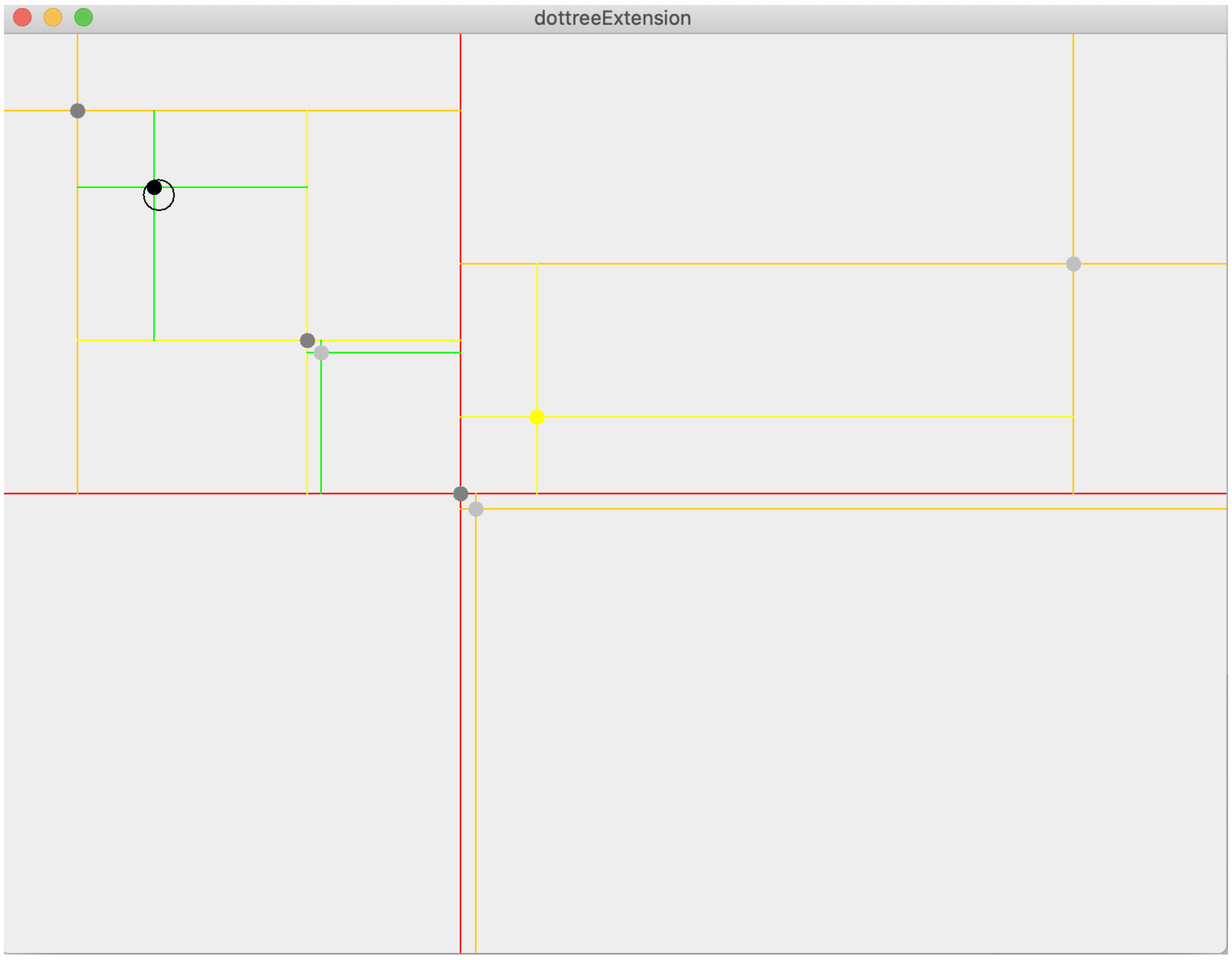
Extensions

PointQuadtreeExtension includes a toString method that prints out the list of points and its children using indents as shown:



```
X:500.0  Y:500.0
    X:680.0  Y:330.0
    X:120.0  Y:120.0
    X:130.0  Y:670.0
        X:120.0  Y:800.0
        X:280.0  Y:680.0
    X:680.0  Y:970.0
```

DotTreeGUIExtension has two added modes: user input mode (‘u’) and test mode (‘t’). User input mode creates an entirely new tree by prompting the user for x and y coordinates in IntelliJ’s console. On the other hand, test mode, when activated, also changes the colors of any points that have been tested with the geometric methods from findInCircle(). If a point only calls Geometry.circleIntersectsRectangle, it turns light grey. If it has called both that method and Geometry.pointInCircle, it turns dark grey. And, as before, if it hits, it turns black. Each color is effectively the number of points that the test methods count.



CollisionGUIExtension adds more options for blobs, namely a WanderingPulsar (‘w’) and Teleporter (‘t’) (as well as just the generic Blob for testing purposes under ‘.’) . Additionally, the radius is no longer fixed to the blob default of 5. Instead the program generates a random number between 0 and 20 as the radius (with the exception of Blob which maintains its fixed value). The findCollisions method has also been changed as a result, from checking within a radius of 2*blobRadius to checking within a radius blobRadius + otherBlobRadius. This should allow collisions to be found even when the radii are not consistent across all blobs.