Program Overview:

PointQuadTree is a data structure that supports a tree that can have up to four children. In the PointQuadTree class, there are also various methods that support the functionality of the quadtree. For example, the insert method can take a specific generic point and insert it into the tree by recursively looping through the quadrants and children of the tree in order to either insert the point into the already existing child in the quadrant or to create a completely new sub quadtree and assign it to the child. There are also methods like size and allPoints that can traverse through the entire tree in order to return the size of the tree or a list containing all the points in the quadtree. Finally, there is the findInCircle method that searches the area of a circle and looks for all of the points of the quadtree that are inside it, and returns a list of these points which are marked as “hit”. These methods are further used in the tests of DotTreeGUI and CollisionGUI.

DotQuadTreeGUI is a GUI that supports the implementation of graphics and user inputs of key and mouse presses to visualize the methods in PointQuadTree. For example, by running the various tests test0, test1, and test2, we can see the different quadrants and quadtrees being formed. Using various key presses, we can also add more dots to existing trees or create new trees, and modify a circle and its parameters in order to highlight the dots found in the circle’s radius.

CollisionGUI is a GUI class that uses PointQuadTree in order to visualize collisions through the use of blobs. It can take various key presses to modify how the blobs

interact with each other (if blobs collide they can be destroyed, or just turn red), add more blobs, change the type of blob, and change the speed at which the blobs move. The blobs use find to look around the blob to check for collisions.

Test Case Overview:

In our test case, we are utilizing the Geometry function and its getNumCircleRectableTest() and getNumInCircleTest() to find out how many times Geomtery.circleIntersectsRectangle and Geometry.pointInCircle are called and how many points are expected to be found for the given query coordinate and radius.
If the actual results are different from the expected values, the test does not pass.

If test2() is called, testFind is called on the different points of the tree. For Point B at (300, 300), with a query circle radius of 10, the circle is expected to intersect a rectangle 5 times. 3 points are expected to be found in the circle, while 1 point, Point B, is expected to be found as a hit.
For Point C at (50, 50), with a query circle radius of 10, the circle is expected to intersect a rectangle 4 times. 2 points are expected to be found in the circle, while 1 point, Point C, is expected to be found as a hit.
For Point D at (400, 400), with a query circle radius of 10, the circle is expected to intersect a rectangle 5 times. 3 points are expected to be found in the circle, while 1 point, Point D, is expected to be found as a hit.

For Point E at (400, 50), with a query circle radius of 10, the circle is expected to intersect a rectangle 4 times. 2 points are expected to be found in the circle, while 1 point, Point E, is expected to be found as a hit.