Feature: Collisions Between Characters, Enemies, Projectiles, and StaticEntities

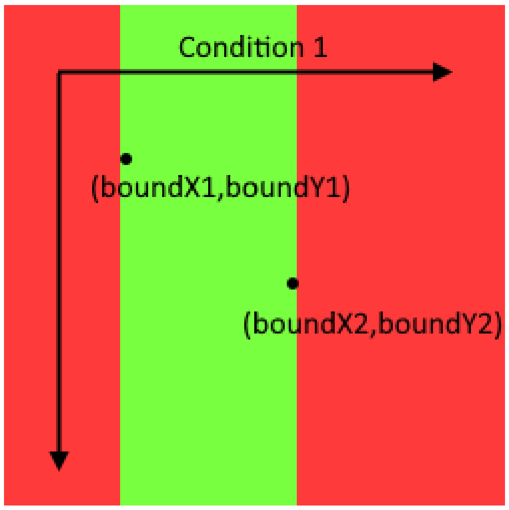
One of the features that we've chosen to include in our test report is collisions between objects, as they are a pretty integral part to the functionality of the game. Collisions between objects are handled by a class of methods in one singular static class named Collision.java. Although this may have been an oversight, and collisions would more appropriately be handled in the entity class extended by all the above classes (omitting staticEntity.java).

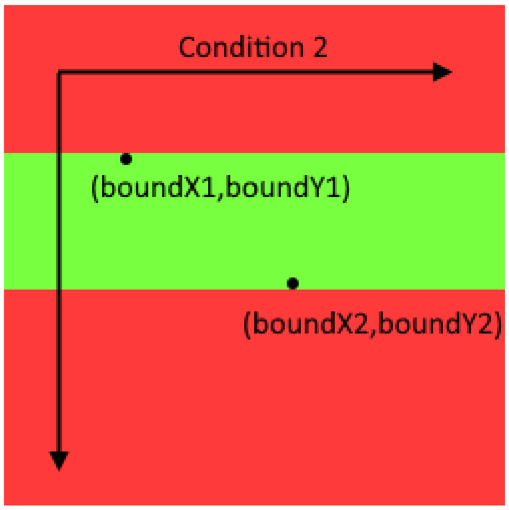
Unit Testing

In the collision class there is a helper method named isPointInSquare (it should be rectangle), that is utilised by all the other methods in the class. Although this method was initially private, it's so essential to the functionality of the other methods, that verifying that it works correctly, significantly reduces the amount of work we must do in the integration test of the other methods in this class. Thus, we've made the decision to change it to public so it could be tested appropriately. Essentially, what this method does is, given two integer coordinates pX and pY, it returns true if this point is in a rectangle given by the integer bounds boundX1, boundY1, boundX2 and boundY2.

To appropriately explain testing for this method let me briefly provide some context to how coordinates are represented in our game. If we consider the point (x,y) in the plane, as we move to the right, the value of x will increase, and as we move down the value of y will increase.

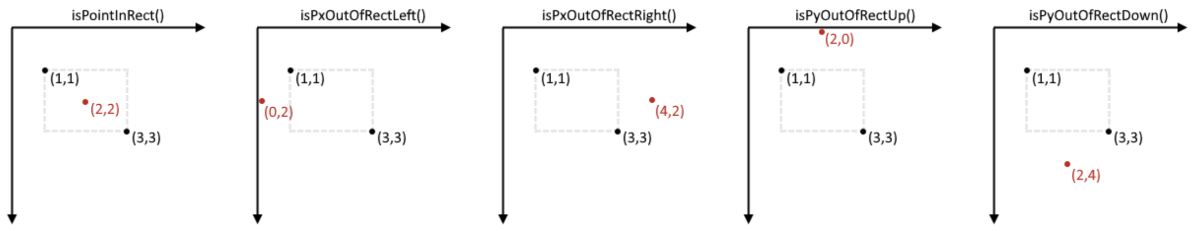
In order to determine this, the method checks two conditions, first it checks if the point pX, is within the bounds boundX1 and boundX2. If it is, then it checks if the point pY is within the bounds boundY1 and boundY2. If both conditions (which we will refer to as condition 1 and condition 2 respectively) are met, then the method returns true. We can illustrate each of these conditions for two arbitrary points as such,





Where the green area represents the range of values of pX and pY for which condition 1 and condition 2 respectively will evaluate to true. We can easily see, that there are 3 cases each condition. Testing each combination of each of these cases would result in 9 separate test cases, which is a little excessive. So, instead of testing each nine, we divided the testing of this method into five general separate unit tests, isPointInRect(), isPxOutOfRectLeft(), isPxOutOfRectRight(), isPyOutOfRectUp(), isPyOutOfRectDown().

Each of these methods uses the same bounds to test pX and pY, boundX1 = 1, boundY1 = 1, boundX2 = 3, boundY2 = 3. The values used for pX and pY tested with each of these methods are shown in the diagrams below where the red point represents the point being tested.



Running mvn test, we see all unit test are successful.

Integration Testing Between Objects

Assume that the other method is true. All mentioned methods are essentially the same, so I will delegate my explanation to just one of the four.