# Safest Place in The Galaxy

## Introduction

This problem can be solved with full brute force approach and its available on GitHub <https://github.com/mlbright/bombshelter>. I could have just converted the code form java into C# but that would have 2 consequences

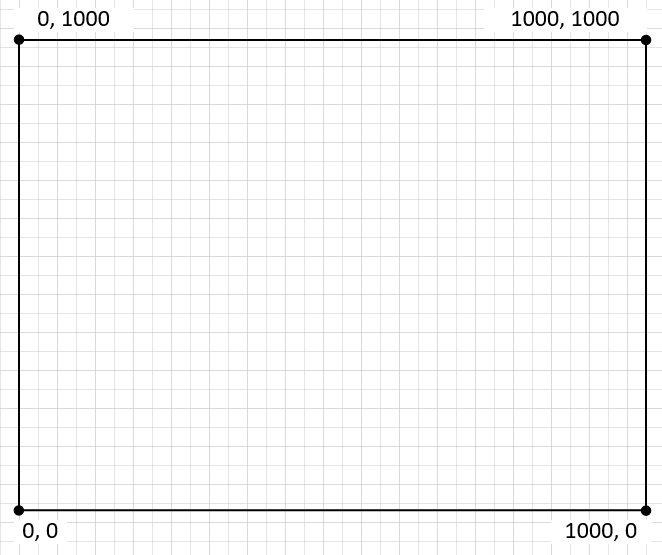
1. Its plagiarism
2. It doesn’t demonstrate my problems solving skills

Therefore, I kept thinking about it and I also discussed with my mates who are also troubled with this question.

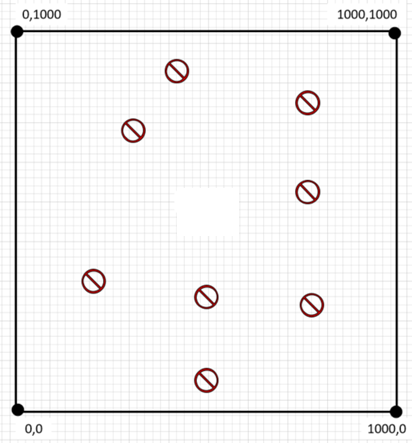
After trying some ideas on white paper, I think the best approach that I could come up with is discounting the points that are nearer to any bomb compared to the one which is the safest so far. This approach is discussed in detail in this document and implemented using C#.Net in the solution uploaded [here](https://github.com/Yawarmurtaza/AnalyticalSkills). Should you wish you run the application to see the output please scroll down to “**How to run**” section at the bottom of this document.

## The Approach

Instead of solving it via a full brute force approach, it would be far better to use a partial brute force. To demonstrate my thought process, consider we have a 2D plane (for simplicity - x & y axis) as shown below:



I have marked the reference points which im going assume are the safest points. Now when we throw some bombs at random in the x-y plane, lets say it comes up like this:

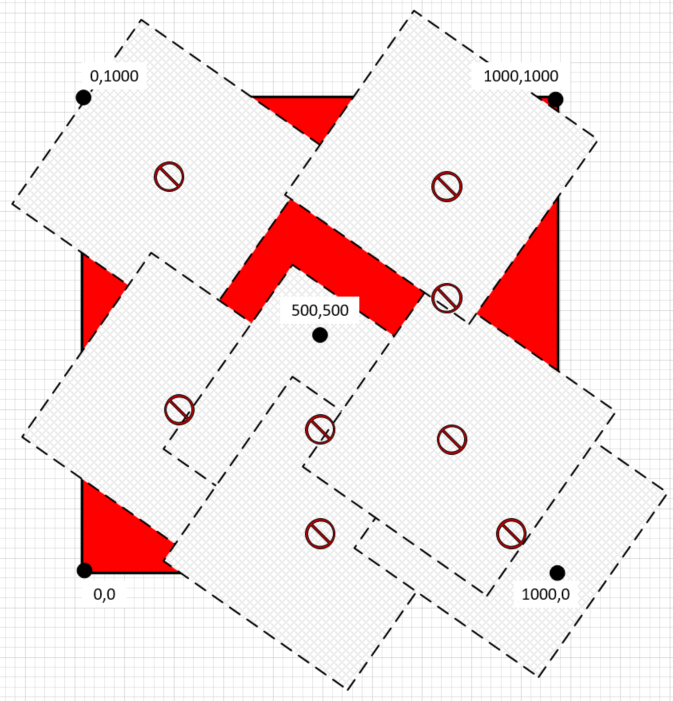


Now the idea is to loop through each bomb’s location and find the furthest bomb from any of the 5 points in above x-y plane then find the minimum distance (say radius – a rounded integer value) from that reference point to its nearest bomb. We will consider this radius is the safest distance to start with.

Next step is to draw a circle around each bomb and discount all the points occurring within each circle. It turns out that discounting all the points within a circle is an expensive task to achieve since there is no such formula that will return all the coordinates incurring within a given circle equation. The only possible way that I can think of is to check if a certain point exists in a circle of not which is as I said an expensive option – its like brute force which we want to avoid at the first place.

Drawing a square is another option which is effective and inexpensive too. However, there is a problem with squares, the corners of squares will be too long for the given radius that we have previously calculated. Consider the following diagram:

Next step is to draw a rotated square (at 45 degrees angle) around each bomb and discount all the points that occur within that diamond. This is because we already know at this time the safest distance and the bomb location and the reference point. Therefore our diagram looks like this.



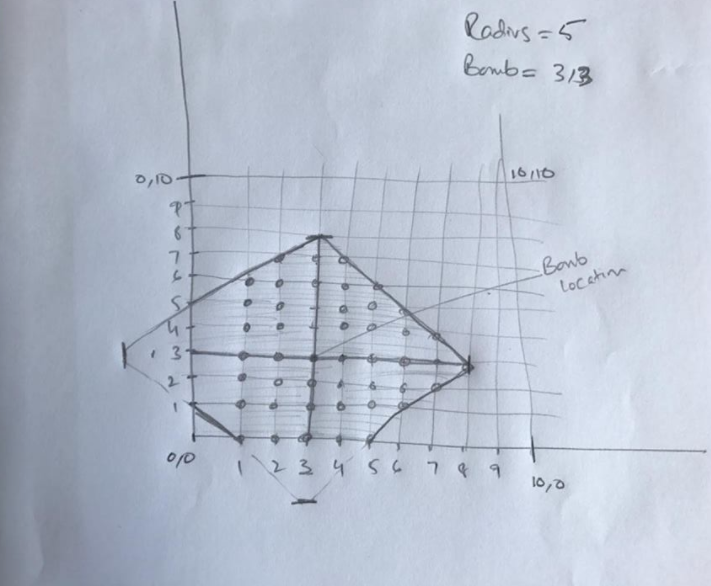
By discounting all the points within these diamonds, we are significantly reducing search.

Now we will find another reference point within the red area of the x-y plane and see if the radius is bigger than previous one. If it does then we will draw a bigger diamond on all bombs to discount even more points.

In a 3D model, this diamond will have a z-axis which will make a 3D diamond around all the bomb. You might be wondering why I am using diamonds over circles or spheres (in 3D plane)

## Diamonds over Spheres

Constructing a diamond around the bombs turns out much cheaper than constructing spheres. With diamonds, we deal with straight lines, from the centre of the bomb we take the length of the radius in either sides of y and x axis. This is shown in the picture below:



With the bomb places at 3,3 and radius of 5, we have identified 54 points within that diamond that can be discounted. We can than calculate the number of points occurring within this diamond by looping through x and y axis.

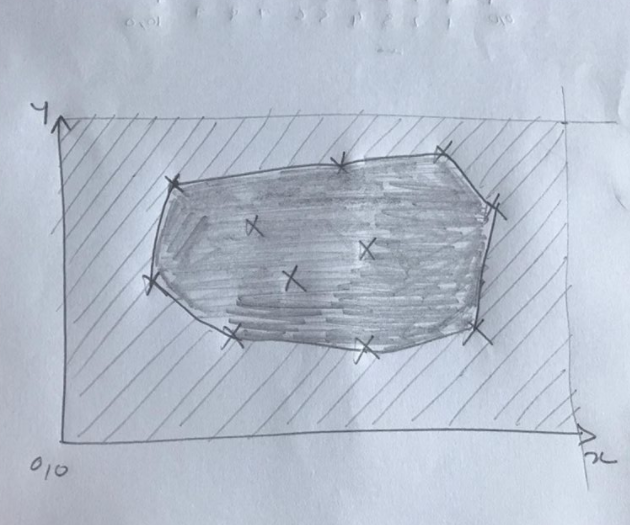
Spheres is a complete different story, the equation of sphere of radius R cantered at the origin is given in Cartesian coordinates by , we can potentially use this to find if a certain point exists in a sphere or not which is an expensive way to discount the points in a given sphere.

## Range of Points

Whilst thinking over ‘discounting points’, we can probably use the range of x, y and z coordinates to discount the points occurring inside the diamond cantered around the bomb. The idea is that since we know the radius, we know the centre of the bomb (bomb location) we should be able to get the range of points on x, y and z axis.

## Another approach

If we draw a shape that will join all the outer bombs and calculate the area of that shape, and we already know the area of our cube, we can determine the possibility of the location of safest place.



In the picture above, I have shaded the area within the shape created by joining the outer bombs together. Now the idea is to calculate the area within that shape and find if that is greater than the area outside the shape. If its not then probably the safest point is outside the shape.

We then can discount all the points within that shape. I know this approach wont work for all the scenarios, I just wanted to show you the approaches that I was thinking in order to get to the solution.

I can discuss on this with couple of more ideas that I have in mind should you wish to take this any further.

## How to run

In the appSettings.config file under AnalyticalSkills.SafestPlace.Runner\config folder you will see an entry for the location of the test data file

<add key="TestDataFilePath" value="E:\Development\AnalyticalSkills\SourceTestFiles\TestData.txt"/>

Change its value to point to the test data file, you can keep any name but the extension has to be “**.txt**” because there is a text data provider that handles the access to the test data from the .txt files. The idea is that in future if we the data to come from a web service or a json file, we can just add another data provider and the rest of the application would just work.

# Adding This Adding That

There are 2 version of recursive methods that add 2 byte arrays and return the sum in 3rd byte array. AddRecursive calls a private method RecursiveAdd that which recursively add the arrays together. AddRecursiveV2 calls itself and returns the result array back to the client. I have written 2 version just to show it can be done with recursion (which is not really beneficial in this particular case).

**End of The Document**