Algorithm 1:

Description:

The focus of this algorithm is to form a probability distribution based on the likelihood of a square having a bomb.

- **Adjusting Squares**: First, we will adjust the current numbered squares (squares with 1 - 8) based on the number of bombs currently surrounding each numbered square. For instance, if a square has the number 3, and there are 2 bombs currently known to be nearby 3, then the 3 will be adjusted to 3 - 2 = 1.

- **Probability Distribution**: We will calculate the probability of an unopened square containing a bomb to be based on the sum of the adjusted square values near that unopened square. That is, for each unopened square, we will sum the numbers of the nearby squares (after adjustment) for each unopened square. For squares that have been opened (i.e. squares with numbers 1 - 9) will be given a value of 0. From this, we obtain a distribution for each square of the board. With this distribution, we will normalize the values such that each square will then contain a value between 0 - 1 that will represent the probability distribution of the square. In essence, all squares with a -1 value (unopened) will have some probability value between 0 and 1, while all other squares will have a guaranteed probability of 0. We will then sample from this probability distribution for a square to open.

- **Termination:** While traversing the board, if we come across enough squares with the number 9 such that the number of squares matches the number of bombs in the board, then we terminate the algorithm and all bombs will be found.

- **Random Sampling Case:** Given the way we have implemented the algorithm, there is a possibility that the initial distribution will be 0 for all squares. This is because when performing the adjusted square step, all the useful numbered squares could turn to zero because they already have their bombs accounted for. In this case, we will simply sample randomly from all squares that have not been unopened as our next square to be opened.

Correctness:

This algorithm will always find the correct number of bombs as well as the correct bomb squares. Based on the termination step mentioned above, the algorithm will check during each move of the game whether it has found the correct bombs. This algorithm will also **always eventually** find the correct bomb set because in addition to the termination step, it will always be opening unopened squares until all squares are opened. However, we know that in the worst case, if all squares are open, then all bombs are guaranteed to be found.

Runtime:

This will be based on some $x \times y$board. For each move, the adjust squares functionality will take $O(xy)$time and to create the probability distribution will also take $O(xy)$. Hence, each move the game will have an analysis runtime of $O(xy)$. Given that we have an $x \times y$board, then the worst case runtime complexity would be $O(x^2y^2)$in which case all the squares would be opened.

Algorithm 2:

Description:
The aim of this algorithm is to find bombs before they are dug, make better decisions based on high probable odds and finally to use the probability distribution in addition to better random sampling methods to choose what square to dig based on the given information.

- **Adjusting Squares**: First, we will adjust the current numbered squares (squares with 1 - 8) based on the number of bombs currently surrounding each numbered square. For instance, if a square has the number 3, and there are 2 bombs currently known to be nearby 3, then the 3 will be adjusted to 3 - 2 = 1.

- **Finding Guaranteed Bombs:** After adjusting all of the squares we will now find the squares that have guaranteed bombs inside of them. The way we do this is by checking the value inside of the adjusted square (basically after subtracting based on how many bombs near it). Then we check to see if the number of unopened squares is equivalent to the number in the adjusted square, if they are the same number we know that all the unopened squares are bombs. We then note the location of these bombs, adjust the board and repeat this algorithm until there are no guaranteed bombs to be found. It is important to note that we recalculate the squares touching the bomb whenever we recall the method. This is possible that after finding 1 bomb this can result in another bomb being found which is why we repeat this method until no new bombs are found.

- **Direct Probability:** For this function we decided to find the direct probability of the unopened squares around each individual square. The way we do this is we first make sure that the square we are checking is not a bomb. We then total all of the unopened squares around that square and then use the following equation to calculate the probability: number in the square/number of unopened squares surrounding that square. We repeat this for all squares and try to find the highest probability. It is important to note that the probability needs to be greater than or equal to 50% to even be considered. Once we have found the highest probability we return a list of all unopened squares that touch the high probability square. We then choose 1 square randomly from that list to be the next square to dig. It is important to note that we can never have a probability that is equal to 1 as the guaranteed bombs method would have already located those bombs and noted the locations.

- **Probability Distribution**: We will calculate the probability of an unopened square containing a bomb to be based on the sum of the adjusted square values near that unopened square. That is, for each unopened square, we will sum the numbers of the nearby squares (after adjustment) for each unopened square. For squares that have been opened (i.e. squares with numbers 1 - 9) will be given a value of 0. From this, we obtain a distribution for each square of the board. With this distribution, we will normalize the values such that each square will then contain a value between 0 - 1 that will represent the probability distribution of the square. In essence, all squares with a -1 value (unopened) will have some probability value between 0 and 1, while all other squares will have a guaranteed probability of 0. We will then sample from this probability distribution for a square to open. This is the same algorithm used in the other algorithm however, it is not the first algorithm that we use to find the new squares to dig, instead we use the probability function described above.

- **Termination:** While traversing the board, if we come across enough squares with the number 9 such that the number of squares matches the number of bombs in the board, then we terminate the algorithm and all bombs will be found. It is important to note that we set the found guaranteed bombs to 9 on the board so that this check takes into account the bomb we have already found

- **Not near 0 Random Sampling:** If the probability distribution, and the direct probability both don't produce anything we will consider the following. We will first try to open an unopened square that is completely surrounded by all other unopened squares. This can provide us with the most information about the current square being opened and all of its nearby squares. If there are no such squares, then we proceed with the following: Out of all unopened squares, we remove some squares that we deem not worth digging. For this we will remove all squares that touch a square that has a 0. This is because any unopened square near a 0 won't have a bomb and is more likely to be a 0 and provide no information that would help us find all the bombs. Instead we will randomly choose a square that is not touching a 0 so that we have a chance to dig a bomb or a higher chance to find a square with a number that is not 0, therefore giving us more information. It is important to note, due to the way that values are adjusted on our board a square that has found all bombs around it will be considered a 0 square and therefore all unopened squares around it will not be considered for digging. For example if a square that has the number 1, and has a bomb next to it, our subtracting method will adjust this square to a 0 therefore all unopened squares around this square will no be considered for digging

Correctness:
This algorithm will always terminate and always find the correct number of bombs. The termination step will look at both the number of guaranteed bombs and number of dug bombs. Since the guaranteed bombs are 100% likely to be bombs we can use the combination of both in the termination step. Additionally, we know that this algorithm will eventually reach this state; in the worst case it will dig in a square that is not near a 0 therefore making sure that in the worst case it will dig every possible location a bomb could be. This step in combination with the first one means that we will always arrive at a solution every single time.

Runtime:
The running time of this algorithm in the worst case comes out to be $O(x^2y^2)$. This is because each method discussed in this algorithm takes $O(xy)$ time to complete. In the worst case we will have to open all squares over all the iterations which would mean opening xy squares over the whole algorithm making the running time to be $O(x^2y^2)$. Best case for this algorithm would be the guaranteed method finding all the bombs which would then take $O(nxy)$ time, where n is the number of bombs.

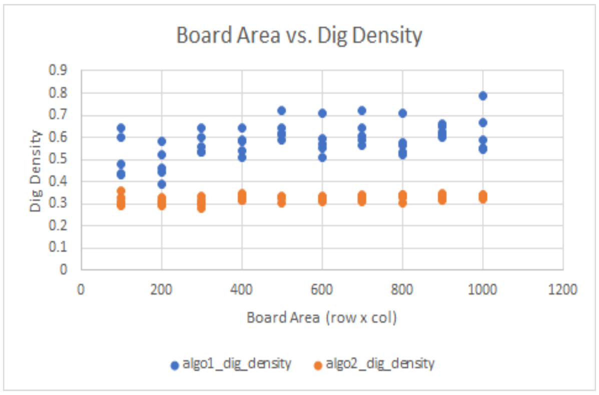
Plots and Analysis:

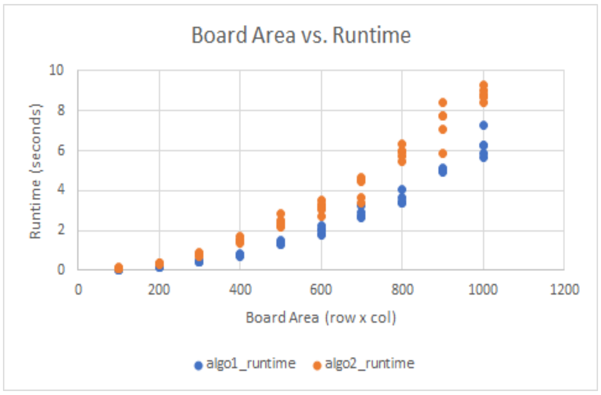
We define dig density as the following:

$$dig\ density\ = \frac{\#\ of\ Digs}{Board\ Area}$$

Hence it is clear that the lower the dig density, the higher performance the algorithm had. In the following plots, we will be analyzing how board area and bomb density affect the runtime and dig density (performance) of an algorithm. The plots show the different test cases on the boards provided in terms of their dig density and runtime.

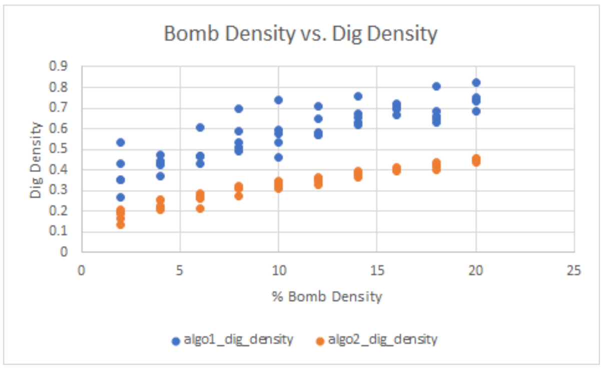
Board Area Analysis:

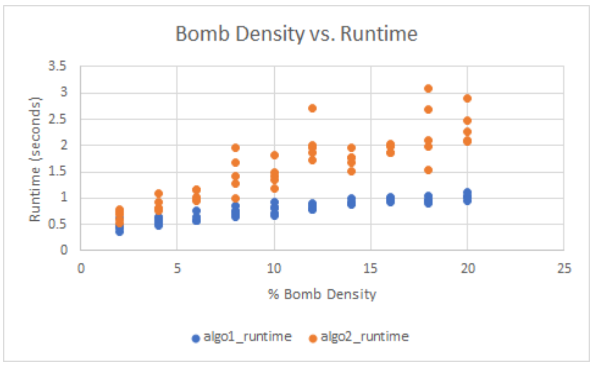




As seen in the first plot, algorithm 1 has a much higher variance due to its sampling based on a probability distribution. algorithm 2, however, had a very consistently low dig density which goes to show that even as the board area increases, the number of digs required increases proportionally. Due to the extra checks/cases and direct probabilities used in algo 2, it was able to detect the bombs more quickly. However, in the plot on the right, we will see a downside of algorithm 2 in that it does exponentially grow faster than algorithm 1, due to the extra computations needed.

Bomb Density Analysis:





In terms of analyzing bomb density, algorithm 2 has a higher performance but also has a high runtime compared to algorithm 1. Both algorithms also experience an increase in dig density as well as runtime as the % bomb density increases.