Artificial Intelligence Project 3 Report

Bust the Ghost Project

Chifaa Bouzid

82723

CSC 4301 01

Dr. Tajjeddine Rachidi

April 20, 2022

Teammate: Ayoub Mabkhout

Video Link

Please find the link of the demo below:

https://www.youtube.com/watch?v=ZSmXIHrsoPY

Probabilistic Inferencing Description and Analysis

The class Proba has all the code related to probabilities. We get the sensed distance between a clicked tile and the ghost tile using a normal distribution through the nextGaussian in the Proba class. Please note that the code of the nextGaussian method was inspired by a contribution in StackOverflow. (https://stackoverflow.com/questions/218060/random-gaussian-variables).

The sensed distance is given as a random value which is generated following the normal distribution given a mean and a standard deviation. The mean is the real distance between the clicked tile and the ghost's. As it could be seen from figure 1, most (approximately 70%) of the generated sensed distances, would be very close to the mean which is the real distance. Then, we generate the color based on that sensed distance through the getColorFromDistance method in the class of GridManager. Of course, the color generated corresponds to the values that were mentioned in the instructions. Meaning, green for a sensed distance of 5+ and so on.

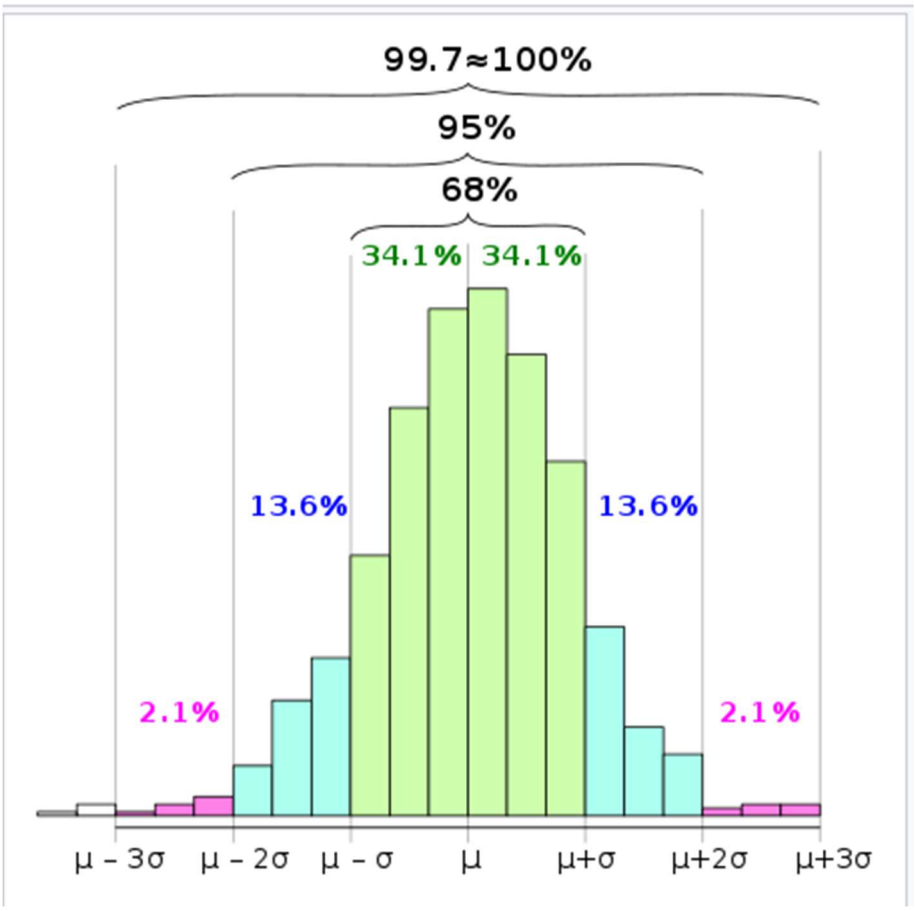

Figure 1: Normal Distribution Representation

The following is a graph that describes the normal distribution approach that we have used:

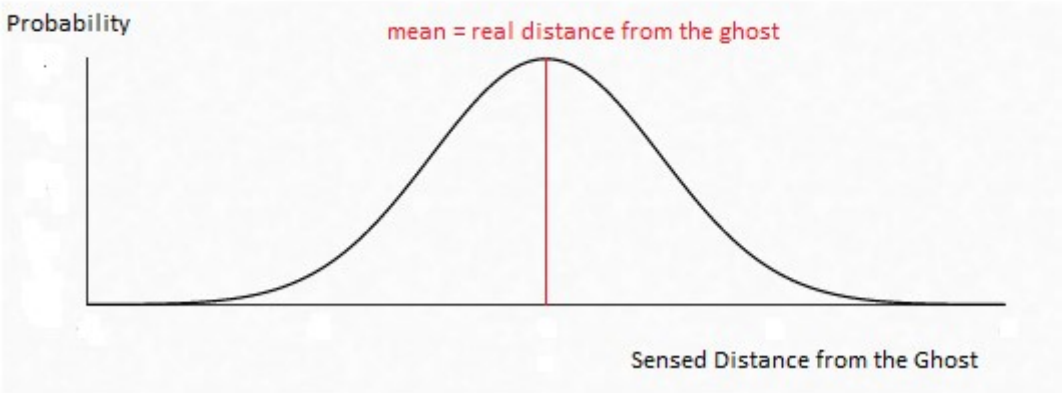

Figure 2: Normal Distribution Graph

After clicking on a tile and displaying the color generated, we update the grid using the method UpdateGrid(). As it was indicated in the instructions, we get the probability which is set for each tile which is going to serve as a prior probability in our further calculations. Then, the distance between each tile and the clicked one is calculated using the Manhattan approach. Afterwards, we try to approximate the distance between each other tile in the grid and the ghost's tile using the color of the clicked tile by generating a range of possible distances. That is when the method MonteCarloProbability() becomes handy to compute the probability of the ghost being in that cell. Please refer to the screenshot below in figure 3 to see what the MonteCarloProbability method looks like. Generally, it generates random values following again the normal distribution and then they are averaged out to provide the probability. As it could be seen we used a normal function. Please refer to figure 4 to see the function that we have used. In MonteCarloProbability method, we have used 100 iterations as when we use greater values such as 1000 or 10000, the game becomes very slow if not crashes. Also, to picture better what our approach looks like, please refer to a descriptive representation in figure 5.

```
public static double MonteCarloProbability(double start, double end, double mean){
    int n_iterations = 100;
    double sum = 0;
    for(int i = 0; i < n_iterations ; i++){
        sum+= NormalFunction(RandomDouble(start,end),mean);
    }
    return  (sum/n_iterations)*Math.Abs(start-end);
}
```

Figure 3: MonteCarloProbability Method Screenshot

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

Figure 4: Normal Function

Figure 5: Approach Description

After getting the probability, we update the probability of the cell by multiplying the prior probability by the obtained one from the MonteCarloProbability method. Next, we add the probability to the sum which is going to be used to normalize. When it comes to the normalization, it is simply dividing the probability of each cell by the sum of probabilities that was already calculated.