



# **Saving Shipwrecked Sailors with Bayes' Rule**

Python Programming

Clara García Latorre  
Noemí Serna Martín  
Jimena Arnau Martínez

# INDEX

Abstract.....	2
Introduction.....	2
Aim.....	2
Methodology .....	2
Scope.....	2
Theoretical Background and Game Design.....	3
Bayesian Inference .....	3
Search success probabilities .....	3
Practical Implementation .....	3
Technologies used .....	3
Game Mechanics .....	3
User Interaction.....	3
Search Algorithm .....	4
Data initialization.....	4
Search simulation .....	4
Bayesian probability update .....	4
Cavern selection and step.....	4
Movement Constraints.....	5
Robot visual movement .....	5
Stopping condition .....	5
Step limit.....	5
Results .....	6
Summary .....	6
Bibliography .....	6

## ABSTRACT

This project presents a cave exploration game where a robot searches for lost sailors hidden in one of three caves (A, B, or C), using Bayesian inference to update its belief based on search results. The implementation is done in Python with Tkinter for the GUI. The specific objectives are:

- Develop an interactive simulation that applies Bayesian inference in a game context.
- Visualize probability updates and robot behavior dynamically.
- Integrate logic, statistical models, and GUI design to create an educational experience.

## INTRODUCTION

### AIM

The main goal of this project is to:

- Create an interactive simulation that demonstrates how Bayesian reasoning can be applied in a game context.
- Develop a GUI-based application that visualizes probability updates and robot behavior dynamically.
- Integrate logic, visuals, and statistical models to create an engaging and educational experience.

### METHODOLOGY

1. Design of game logic with Bayesian updating.
2. GUI development using Tkinter.
3. Creation and integration of visual assets (cave and robot images).
4. Randomized assignment of sailor location at game start.

### SCOPE

The project includes game logic based on probabilistic reasoning, an interactive graphical user interface, a simulation of an intelligent search algorithm, and numerical benchmarking using LINPACK in C. Additionally, it serves as an educational tool for learning artificial intelligence, Bayesian logic, and GUI programming.

On the other hand, real-time 3D graphics, multiplayer or network features, and a full-featured game engine with object systems are not included.

## THEORETICAL BACKGROUND AND GAME DESIGN

### BAYESIAN INFERENCE

The robot selects the cave with the highest current probability using Bayes' theorem:

$$P(H|E) = (P(E|H) * P(H)) / P(E)$$

In this simulation, H represents the hypothesis, which refers to the possible locations of the lost sailors—specifically, that they are in cave A, B, or C. E stands for the evidence, which is the outcome of each search attempt. After each search, the robot uses the result (evidence) to update the probabilities of each hypothesis using Bayesian inference, gradually narrowing down the most likely cave where the sailors are hidden.

### SEARCH SUCCESS PROBABILITIES

The search success probabilities reflect the inherent uncertainty in the search process. When searching the correct cave, there is a high 90% chance of successfully finding the target, indicating that the search is very likely to yield a positive result if conducted in the right location. However, to model realistic imperfections, there is also a 10% false positive chance when searching incorrect caves, meaning that the search might incorrectly suggest the target is present even when it is not. This introduces a level of uncertainty that complicates decision-making, requiring probabilistic reasoning and Bayesian updating to interpret search outcomes accurately.

## PRACTICAL IMPLEMENTATION

### TECHNOLOGIES USED

- Python 3.x
- Pillow (install via `pip install pillow`)
- Tkinter (for GUI)
- Visual studio (IDE)

### GAME MECHANICS

- Objective: Find the hidden sailors using intelligent search.
- Maximum of 10 search steps.
- Game ends when: Probability of the correct cave exceeds 0.999, or 10 searches have been performed.

### USER INTERACTION

The player interacts only by clicking the "Next" button, triggering each search step.

## SEARCH ALGORITHM

### DATA INITIALIZATION

At the beginning, three possible caverns (A, B, and C) are defined. The true location of the sailors is randomly chosen using `random.choice()`. Initial probabilities are evenly distributed ( $1/3$  each), representing complete uncertainty.

```
caverns = ['A', 'B', 'C']
true_location = random.choice(caverns)
probabilities = {'A': 1/3, 'B': 1/3, 'C': 1/3}
step = 0
```

### SEARCH SIMULATION

This function simulates the robot searching a cavern. If the cavern is the correct one, there is a 90% chance of detecting the sailors. If it's incorrect, there's still a 10% chance of a false positive (noise or misleading evidence).

```
def search(cavern):
    if cavern == true_location:
        return random.random() < 0.9
    else:
        return random.random() < 0.1
```

### BAYESIAN PROBABILITY UPDATE

This module applies Bayes' Theorem to update the belief of each cavern containing the sailors. If a clue is found in a cavern, its probability increases; otherwise, it drops and the others increase.

```
def update_probabilities(probs, searched_cavern, found):
    for cavern in probs:
        if cavern == searched_cavern:
            probs[cavern] *= 0.9 if found else 0.1
        else:
            probs[cavern] *= 0.05 if found else 0.9
    total = sum(probs.values())
    for cavern in probs:
        probs[cavern] /= total
```

### CAVERN SELECTION AND STEP

At every step, the robot automatically selects the cavern with the highest current probability. This is a greedy decision based on updated beliefs.

```
cavern_to_search = max(probabilities, key=probabilities.get)
```

## MOVEMENT CONSTRAINTS

### ROBOT VISUAL MOVEMENT

The robot moves visually to the selected cavern by showing its image above it. It appears only in one place at a time — the one with the highest probability.

```
for cavern_id in caverns:
    if cavern_id == cavern_to_search:
        robot_labels[cavern_id].config(image=robot_photo)
    else:
        robot_labels[cavern_id].config(image="") # Clear others
```

### STOPPING CONDITION

If any cavern reaches 99.9% certainty, the simulation is stopped, and the robot is considered to have found the sailors.

```
if probabilities[cavern_to_search] >= 0.999:
    avanzar_btn.config(state="disabled")
    result_label.config(
        text=f"Sailors found in cavern {cavern_to_search}!\nSimulation stopped at step {step}."
    )
    return
```

### STEP LIMIT

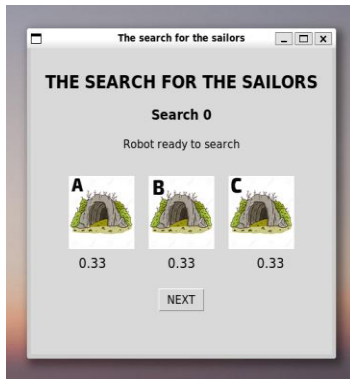
If the robot hasn't reached a conclusion after 10 steps, the simulation stops and displays the final probability distribution for each cavern.

```
if step >= 10:
    avanzar_btn.config(state="disabled")
    final_msg = "\nFinal result:\n" + "\n".join(
        [f"Cavern {c}: {probabilities[c]:.2f}" for c in caverns]
    )
    result_label.config(text=final_msg)
```

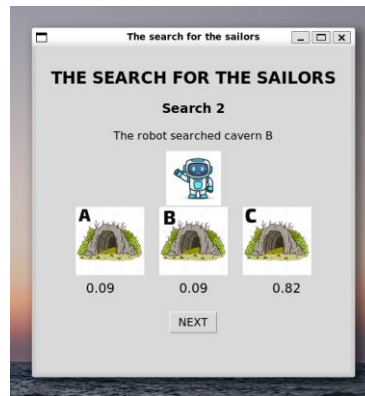
## RESULTS

The robot updates its beliefs correctly after each search step. The GUI displays the caves, robot, and probability values clearly, allowing users to observe how Bayesian updating works in practice. The simulation performs reliably, consistently guiding the robot toward the correct cave.

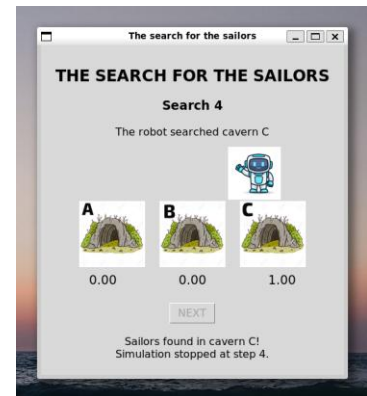
Initial Setup



Midway Point



End Point



## SUMMARY

The goal of the project was successfully achieved. The game correctly simulates the search for the sailors using intelligent Bayesian inference to update probabilities after each search step.

The game runs as expected, with the robot choosing the most probable cavern to search each time. The GUI displays the caverns, the robot, and the probabilities, providing clear feedback about the robot's progress. The player can observe how the probabilities change with each step, which shows how the Bayesian updating algorithm works in practice.

Although the game works as intended, there were some challenges during the development, such as fine-tuning the image handling and ensuring smooth updates to the probabilities. However, these issues were resolved and didn't hinder the overall functionality.

Reflecting on the project, we learned how to implement Bayesian inference in a real-world simulation, which helped us understand its practical application in decision-making. Additionally, it was a great exercise in integrating logic with a user interface, especially using Tkinter for the GUI.

## BIBLIOGRAPHY

- Python Software Foundation. (2023). Tkinter — Python interface to Tcl/Tk. Retrieved from <https://docs.python.org/3/library/tkinter.html>
- Python Software Foundation. (2023). random — Generate pseudo-random numbers. Retrieved from <https://docs.python.org/3/library/random.html>