SCHOOL OF SCIENCE & ENGINEERING

FALL 2023

CSC 4301 – 01

Term Project: Final Report

November 27th, 2023

Realized by:

Adnane Ahroum

Moncef El Kadiri

Reda Elgaf

Zakaria Amrani

Supervised by:

Professor RACHIDI, Tajjedine

Table of content:

I. Abstract:

This project aims to develop the game “Bust the Ghost”, a game looking like “Minesweeper”. However, this game is a good demonstration of the Bayesian inference principles. Using Unity, the game development engine, the objective at the end is, as the title says, bust the ghost that would be found in any of the cells of the grid.

II. Introduction:

“Bust the ghost” is a game usually represented in a grid where the number of cells would vary to your preference. In our project we will allow players to experience a seven by thirteen grid. The players would only get to experience one ghost, because sometimes, just like the minesweeper, there can be a possibility to have more than one objective to make the game more challenging and exciting. The system is managed so that the players must utilize sensor readings to locate and “bust” the ghost before their number of credits runs out, which makes them automatically lose the game. In our example time is not an impactful factor, but rather patience and decision making are the principles being illustrated.

The game starts with a uniform distribution of the ghosts’ position, and players would interact by clicking on the cells containing probabilities of where the ghost would possibly be. When clicking, the cell would illuminate with a certain color would describe the distance between that specific cell and the cell containing the ghost, the four colors are red, orange, yellow, and green. We will get back to how these cells display the accurate probabilities when we speak about Bayesian Inference.

Additionally, the players have an additional button other than the cells that they can click on, which is “PEEP”. This button is a toggle button that can be described as light switch that goes ON and OFF. This button’s role is to display the probabilities in all the cells of the grid, which provides more information and insight about the Bayesian updating process.

This report is focusing on the development of the game in the unity engine, but also on detailing the implementation of the probabilities the cells, and the Bayesian inference, as well as the overall aesthetic design of the game and its functionalities.

III. Game Environment:

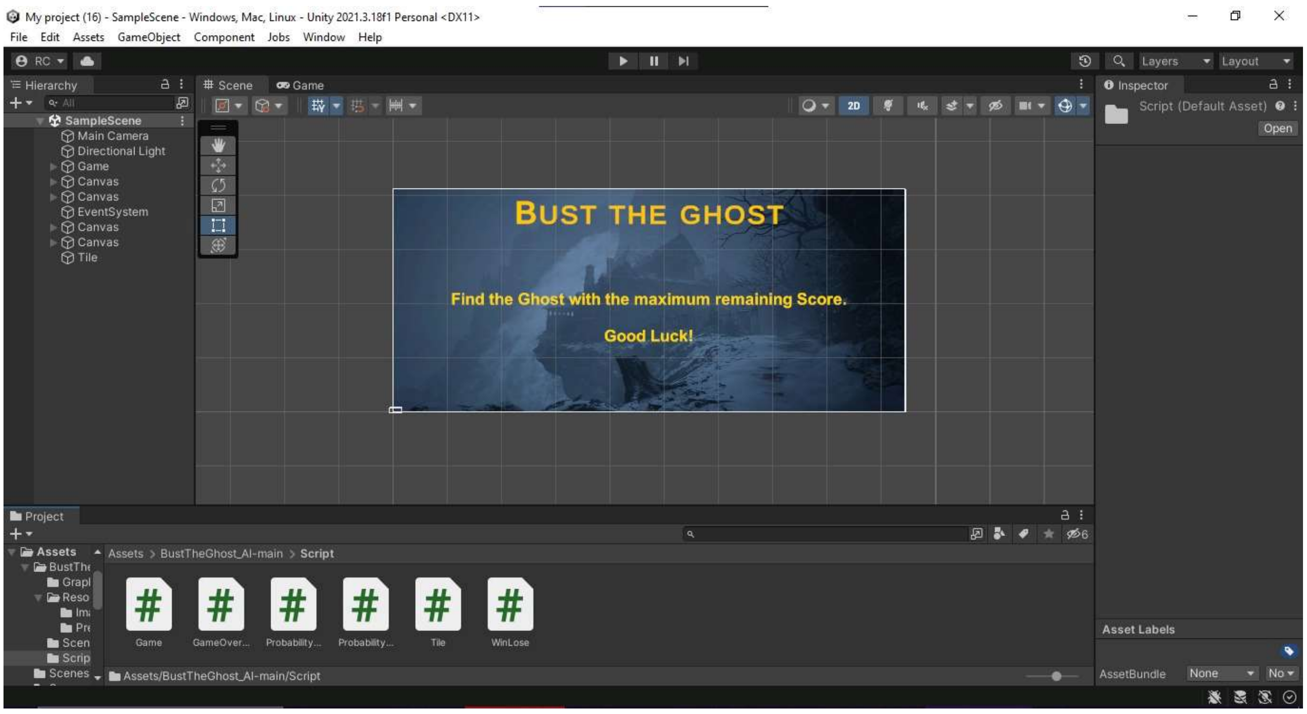
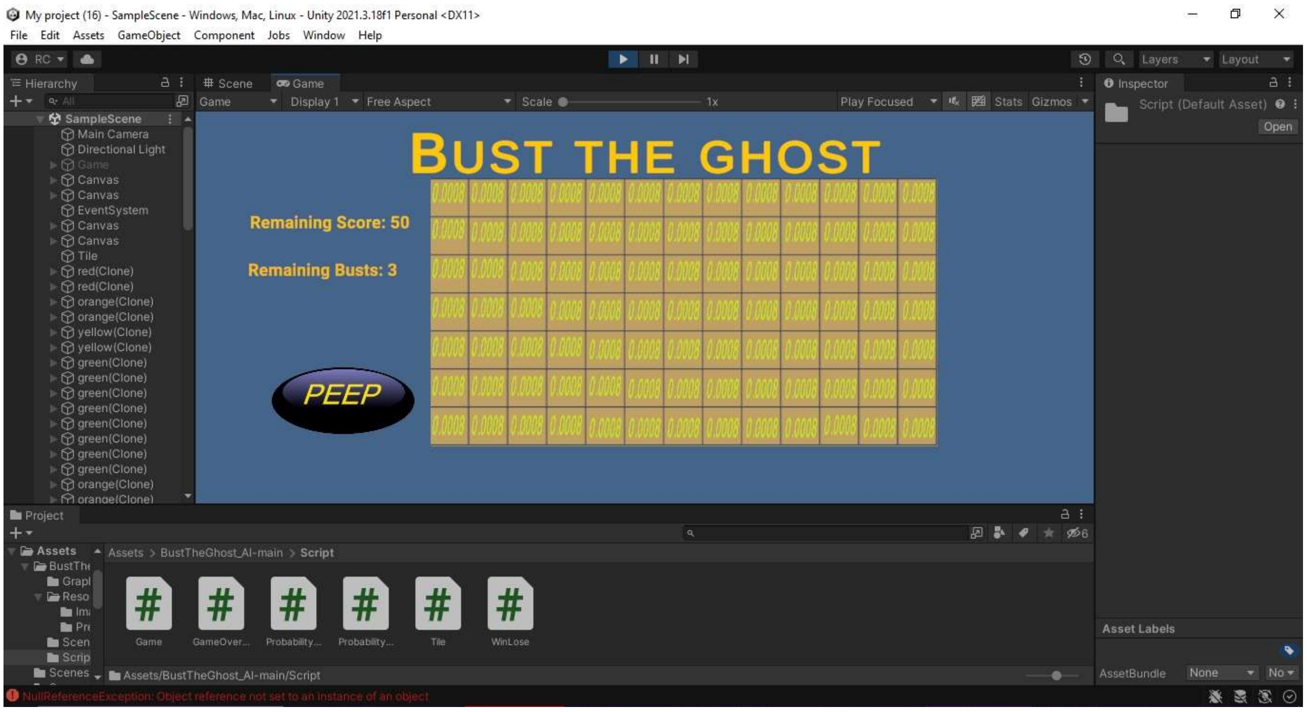

My project (16) - SampleScene - Windows, Mac, Linux - Unity 2021.3.18f1 Personal <DX11>
File Edit Assets GameObject Component Jobs Window Help
Hierarchy
SampleScene
Main Camera
Directional Light
Game
Canvas
Canvas
EventSystem
Canvas
Canvas
Tile
Scene
Game
Inspector
Script (Default Asset)
Open
BUST THE GHOST
Find the Ghost with the maximum remaining Score.
Good Luck!
Project
Assets > BustTheGhost_AI-main > Script
Game
GameOver...
Probability...
Probability...
Tile
WinLose
Assets/BustTheGhost_AI-main/Script
Asset Labels
AssetBundle None No

Figure 1: Main Scene (Title Screen of the Game)

This picture represents the title screen of the game. The only features it has is a background image which we changed its color gradient, as well as two text objects, stating the name and the objective of the game.

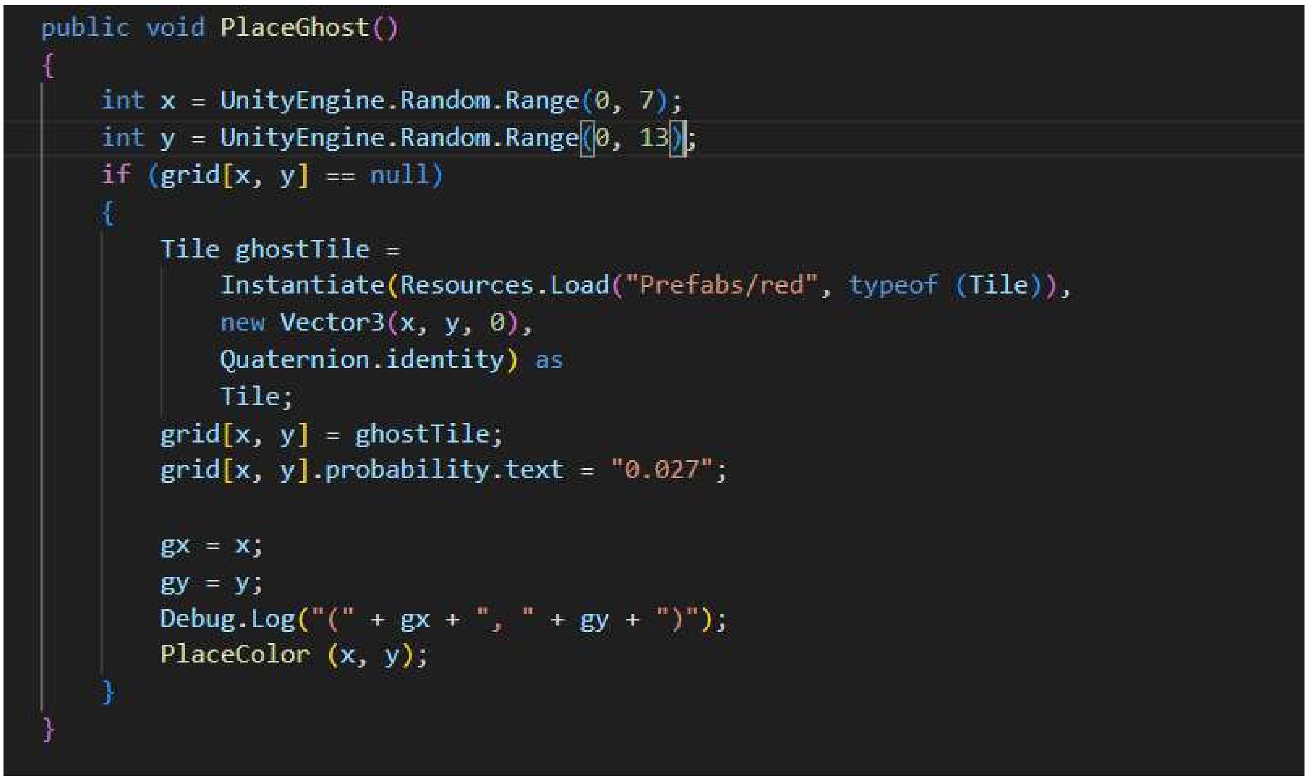


NullReferenceException: Object reference not set to an instance of an object

Figure 2: Bust the ghost interface (Main Game)

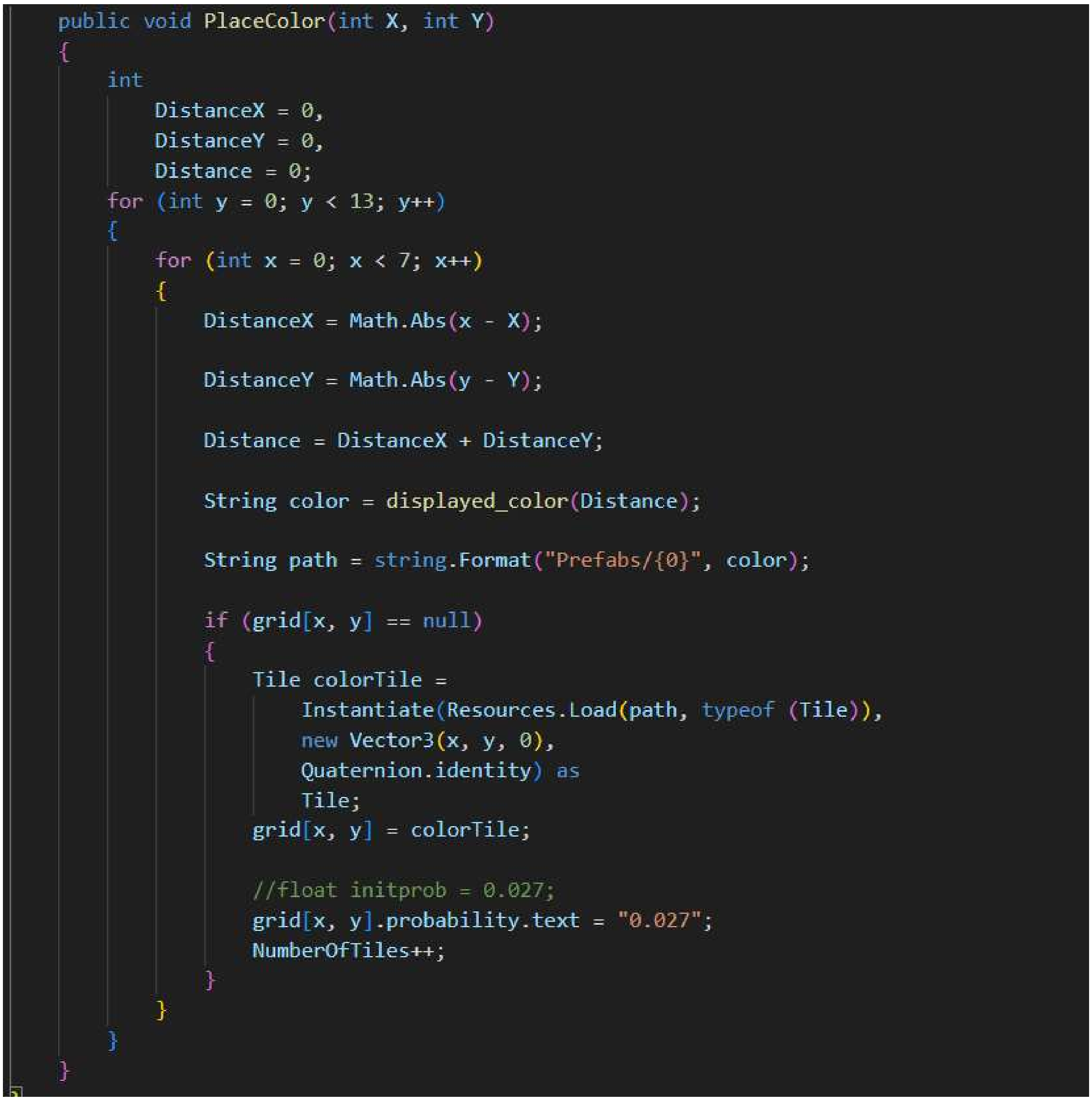
The game environment is a 7x13 grid with a ghost that is hidden in one of these cells. The first thing that the games start with is placing the ghost in a random location, then it sets prints on the board that can mislead the player about the actual ghost's position, then it finally adjusts the colors discussed earlier in each of the tiles that the user would click on. This is being done by three functions, and here is a detailed explanation of their purpose:

- The "PlaceGhost" function generates a random location for the ghost to be placed in at the beginning of each game.



```
public void PlaceGhost()
{
    int x = UnityEngine.Random.Range(0, 7);
    int y = UnityEngine.Random.Range(0, 13);
    if (grid[x, y] == null)
    {
        Tile ghostTile =
            Instantiate(Resources.Load("Prefabs/red", typeof (Tile)),
            new Vector3(x, y, 0),
            Quaternion.identity) as
            Tile;
        grid[x, y] = ghostTile;
        grid[x, y].probability.text = "0.027";

        gx = x;
        gy = y;
        Debug.Log("(" + gx + ", " + gy + ")");
        PlaceColor (x, y);
    }
}
```

- The "PlaceColor" function is used to place colored tiles on the game board based on the joint probability of the distance from the clicked cell to the ghost and the color observed. These tiles help the player make informed decisions about where to click next to find the ghost.



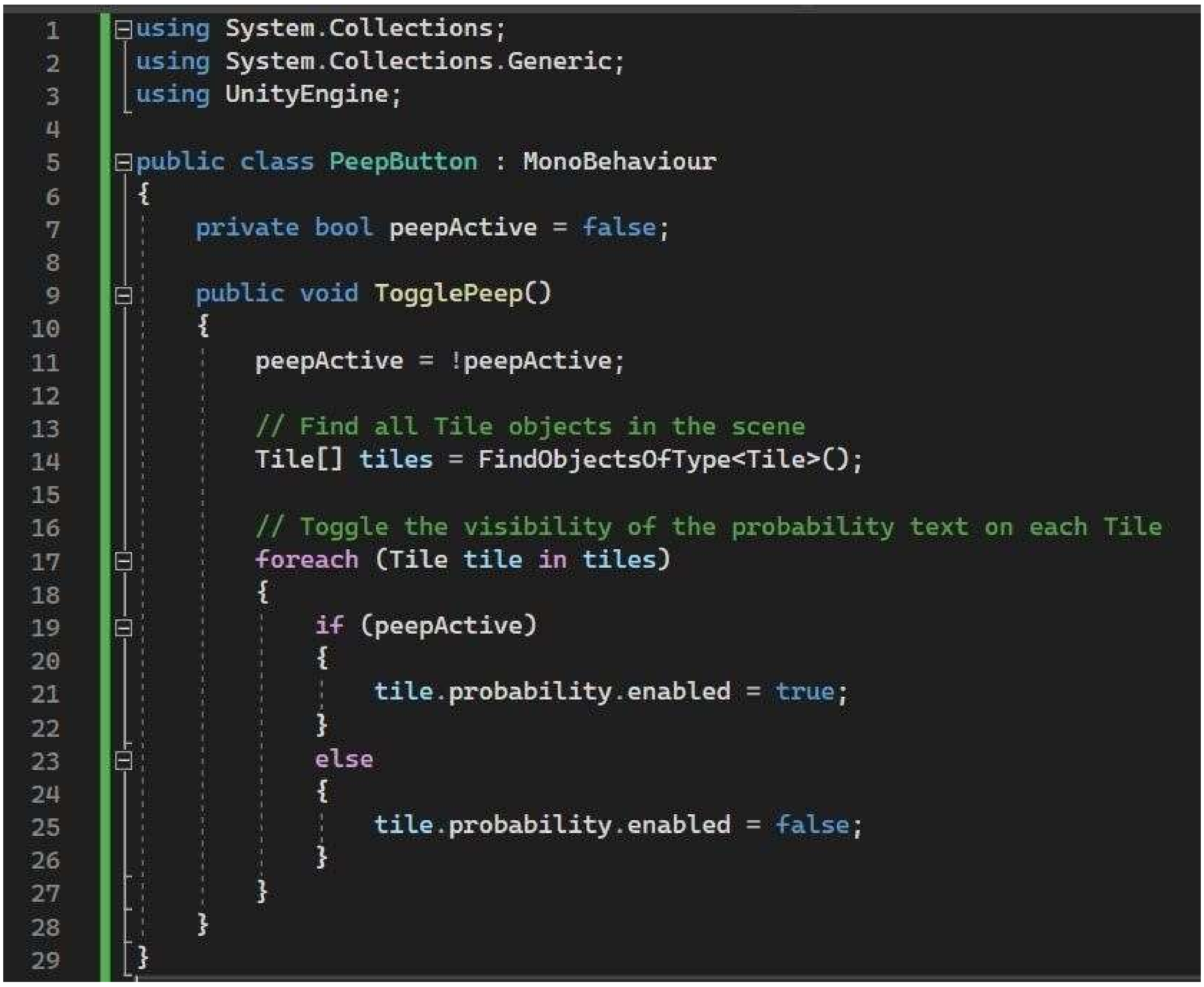
```csharp
    public void PlaceColor(int X, int Y)
    {
        int
            DistanceX = 0,
            DistanceY = 0,
            Distance = 0;
        for (int y = 0; y < 13; y++)
        {
            for (int x = 0; x < 7; x++)
            {
                DistanceX = Math.Abs(x - X);

                DistanceY = Math.Abs(y - Y);

                Distance = DistanceX + DistanceY;

                String color = displayed_color(Distance);

                String path = string.Format("Prefabs/{0}", color);

                if (grid[x, y] == null)
                {
                    Tile colorTile =
                        Instantiate(Resources.Load(path, typeof (Tile)),
                        new Vector3(x, y, 0),
                        Quaternion.identity) as
                        Tile;
                    grid[x, y] = colorTile;

                    //float initprob = 0.027;
                    grid[x, y].probability.text = "0.027";
                    NumberOfTiles++;
                }
            }
        }
    }
```

IV. Game Features:

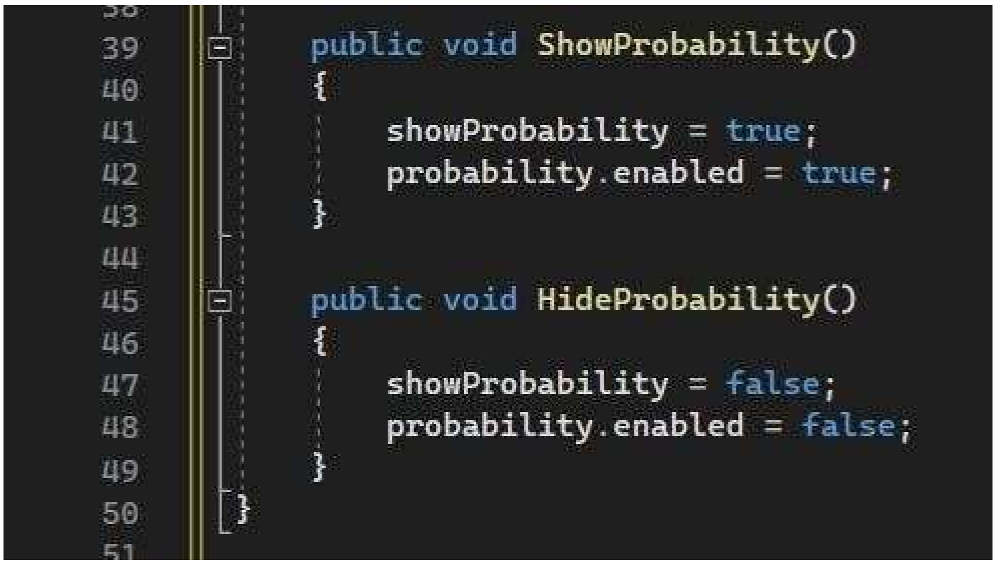
a. Peep Button;

Players have a "PEEP" button, a toggle switch, which displays the probabilities of all grid cells. This feature enhances the game by providing players with valuable insights into the Bayesian updating process.

Figure 3: Peepbutton.cs



```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PeepButton : MonoBehaviour
{
    private bool peepActive = false;

    public void TogglePeep()
    {
        peepActive = !peepActive;

        // Find all Tile objects in the scene
        Tile[] tiles = FindObjectsOfType<Tile>();

        // Toggle the visibility of the probability text on each Tile
        foreach (Tile tile in tiles)
        {
            if (peepActive)
            {
                tile.probability.enabled = true;
            }
            else
            {
                tile.probability.enabled = false;
            }
        }
    }
}
```

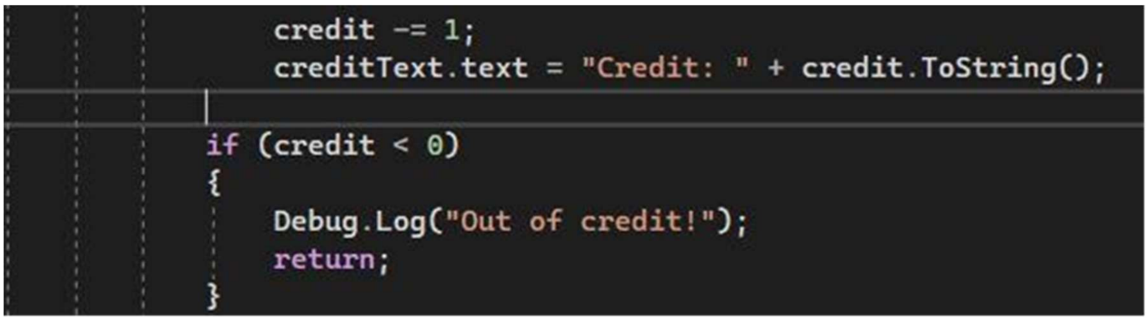


```
public void ShowProbability()
{
    showProbability = true;
    probability.enabled = true;
}

public void HideProbability()
{
    showProbability = false;
    probability.enabled = false;
}
}
```

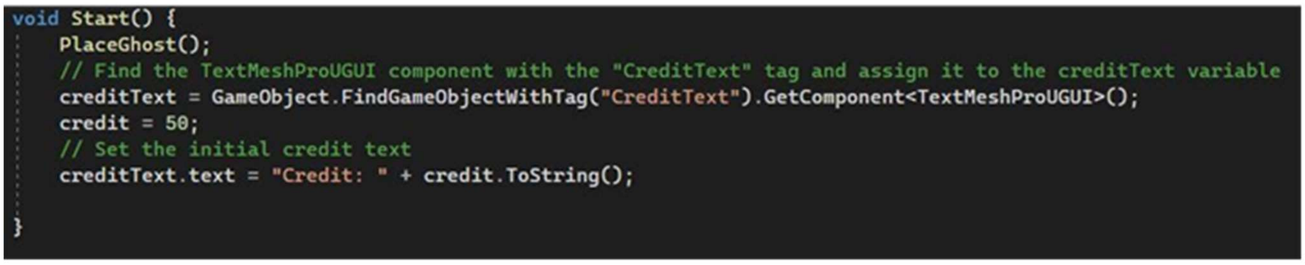
Figure 4: Added features related to PEEP in tile.cs file

To incorporate the Peep functionality into the game, a new script called "PeepButton" was created and associated with a Peep button on the graphical user interface (GUI). This script consists of two functions, Hide and ShowProbability, which manipulate the "tile.probability" text element. These functions control the visibility of the probability information displayed on the game tiles.

b. Credit System:



```
        credit -= 1;
        creditText.text = "Credit: " + credit.ToString();

    if (credit < 0)
    {
        Debug.Log("Out of credit!");
        return;
    }
```

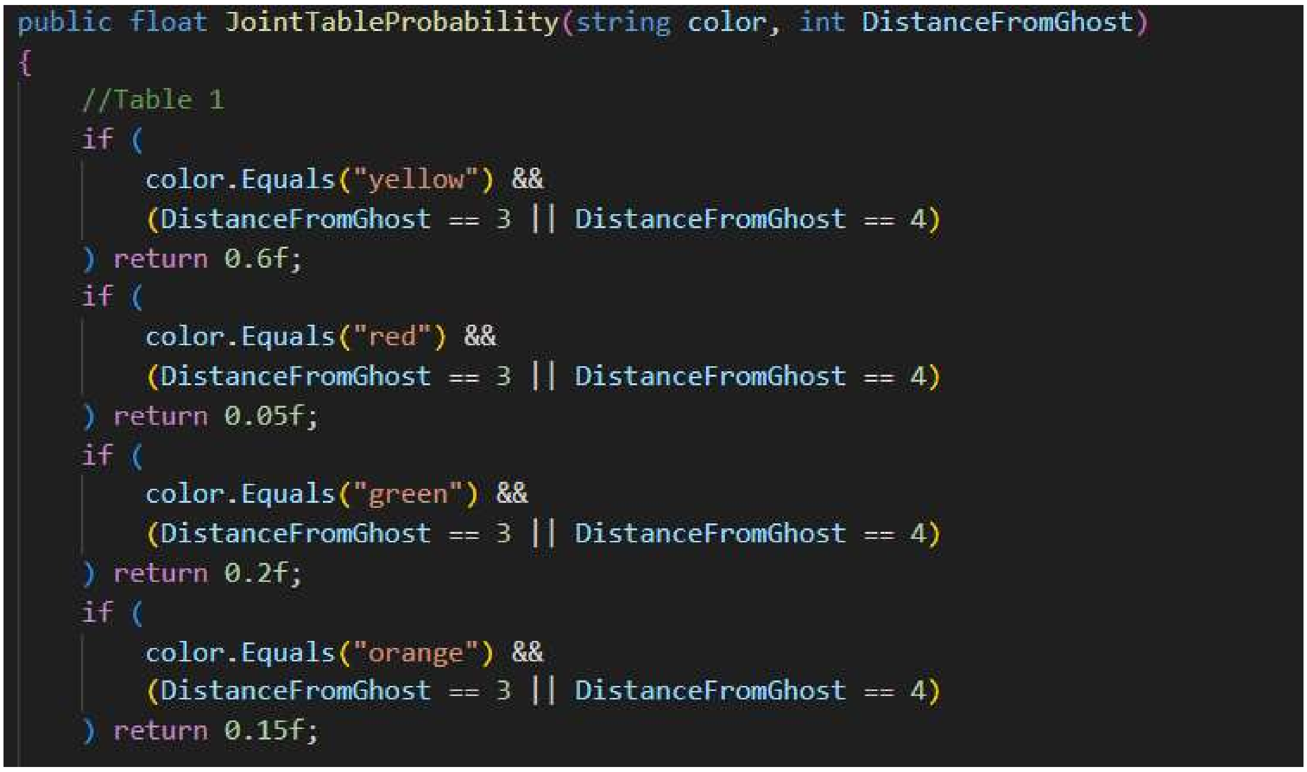


```
void Start() {
    PlaceGhost();
    // Find the TextMeshProUGUI component with the "CreditText" tag and assign it to the creditText variable
    creditText = GameObject.FindGameObjectWithTag("CreditText").GetComponent<TextMeshProUGUI>();
    credit = 50;
    // Set the initial credit text
    creditText.text = "Credit: " + credit.ToString();

}
```

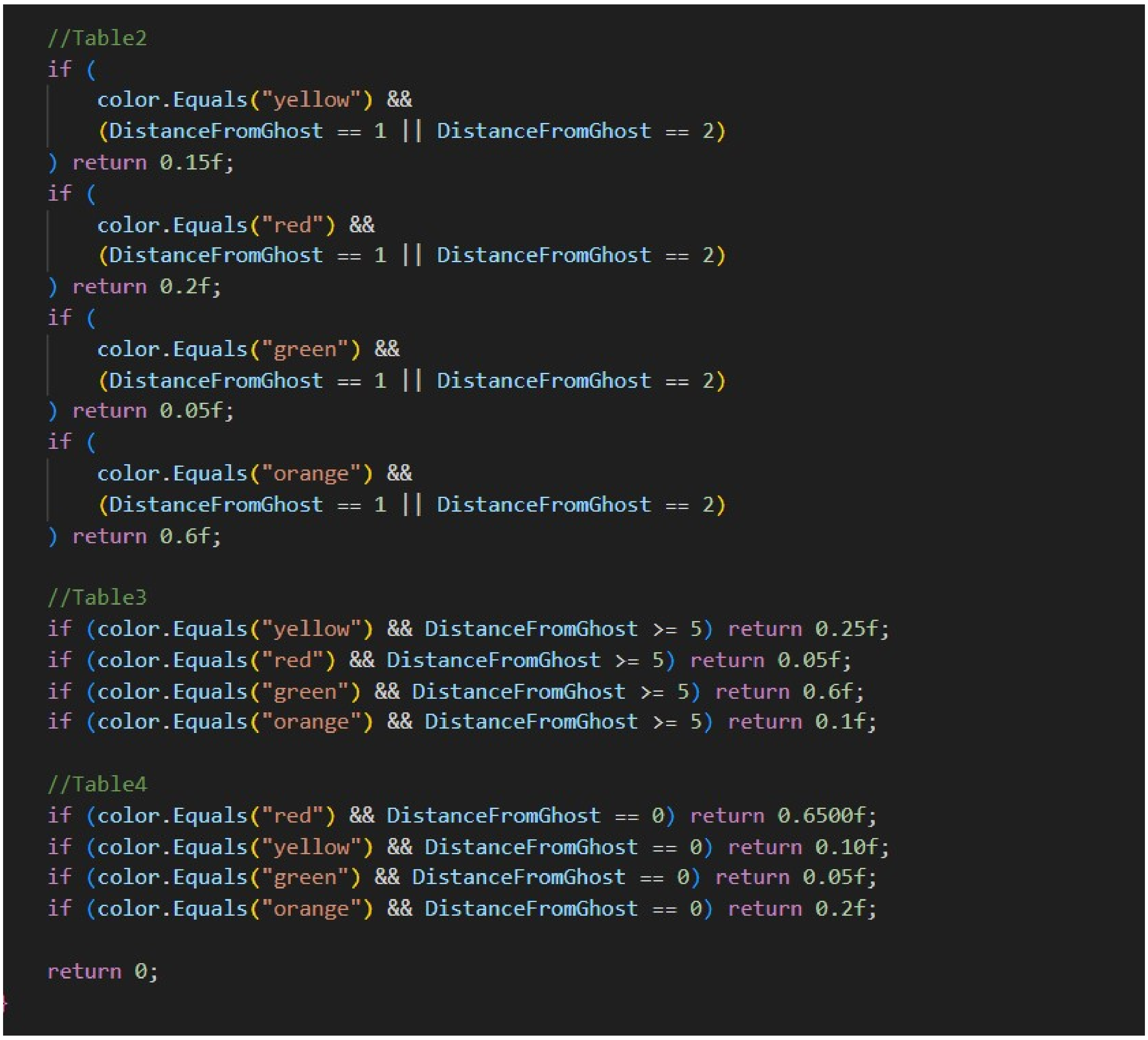
The code above represents the credit system implemented inside the game interface. The credit system is implemented within the “Game.cs” script, which serves as the main game logic. Within this script, a new variable called "credit" has been defined. This credit variable is linked to a “TextMesh” element in the graphical user interface (GUI). The credit value is dynamically updated throughout the game based on player actions and events. This credit system adds an additional layer of gameplay mechanics and allows players to track and manage their in-game progress.

V. Probabilistic Inferencing:

a. Joint Probabilistic Inferencing:



```
public float JointTableProbability(string color, int DistanceFromGhost)
{
    //Table 1
    if (
        color.Equals("yellow") &&
        (DistanceFromGhost == 3 || DistanceFromGhost == 4)
    ) return 0.6f;
    if (
        color.Equals("red") &&
        (DistanceFromGhost == 3 || DistanceFromGhost == 4)
    ) return 0.05f;
    if (
        color.Equals("green") &&
        (DistanceFromGhost == 3 || DistanceFromGhost == 4)
    ) return 0.2f;
    if (
        color.Equals("orange") &&
        (DistanceFromGhost == 3 || DistanceFromGhost == 4)
    ) return 0.15f;
```



```
//Table2
if (
    color.Equals("yellow") &&
    (DistanceFromGhost == 1 || DistanceFromGhost == 2)
) return 0.15f;
if (
    color.Equals("red") &&
    (DistanceFromGhost == 1 || DistanceFromGhost == 2)
) return 0.2f;
if (
    color.Equals("green") &&
    (DistanceFromGhost == 1 || DistanceFromGhost == 2)
) return 0.05f;
if (
    color.Equals("orange") &&
    (DistanceFromGhost == 1 || DistanceFromGhost == 2)
) return 0.6f;

//Table3
if (color.Equals("yellow") && DistanceFromGhost >= 5) return 0.25f;
if (color.Equals("red") && DistanceFromGhost >= 5) return 0.05f;
if (color.Equals("green") && DistanceFromGhost >= 5) return 0.6f;
if (color.Equals("orange") && DistanceFromGhost >= 5) return 0.1f;

//Table4
if (color.Equals("red") && DistanceFromGhost == 0) return 0.6500f;
if (color.Equals("yellow") && DistanceFromGhost == 0) return 0.10f;
if (color.Equals("green") && DistanceFromGhost == 0) return 0.05f;
if (color.Equals("orange") && DistanceFromGhost == 0) return 0.2f;

return 0;
}
```

Figure 5: JointTableProbability function in Game.cs

Distance from Ghost / Color	Yellow	Red	Green	Orange
0	0.10	0.65	0.05	0.2
1-2	0.15	0.2	0.05	0.6
3-4	0.6	0.05	0.2	0.15
>= 5	0.25	0.05	0.6	0.1

Figure 6: Joint Probability full table implementation

The joint probability table displays conditional probabilities for two variables: "Distance from Ghost" and "Color." The "Distance from Ghost" variable is categorized into "0," "1-2," "3-4," and ">= 5," representing the number of squares between the ghost and the player. The "Color" variable is divided into "Yellow," "Red," "Green," and "Orange," indicating the color of the ghost.

By dividng the whole table into four rows, we can see the following assignments:

In The first row, when the color is "red" and the DistanceFromGhost is 0, the probability is 0.65. For "yellow," "green," and "orange" colors with a distance of 0, the probabilities are 0.1, 0.05, and 0.2 respectively.

The second row represents a different combination. When the color is "yellow" and the DistanceFromGhost is 1 or 2, the probability is 0.15. For "red," "green," and "orange" colors with the same distance, the probabilities are 0.2, 0.05, and 0.6 respectively.

In the third row, when the color is "yellow" and the DistanceFromGhost is either 3 or 4, the probability is 0.6. For "red" and "green" colors with the same distance, the probabilities are 0.05 and 0.2 respectively. The probability for the "orange" color in this case is 0.15.

The last row considers the scenario when the color is "yellow," and the DistanceFromGhost is greater than or equal to 5. The corresponding probability is 0.25. For "red," "green," and "orange" colors in this case, the probabilities are 0.05, 0.6, and 0.1 respectively.

If none of the conditions in the function match, it returns a probability value of 0.

By following the equation:

$$P(Ghost) = P\left(\frac{Ghost}{Color}\right)$$

$$= P\big(Ghost(t-1)\big) * P\left(\frac{Color}{Distance\ from\ Ghost}\right)$$

We can represent the application of Bayesian inferencing to update the probability of the ghost being in a specific cell. The prior probability $P(Ghost)$ indicates the initial chance of the ghost being in each cell before any sensor readings are taken. When a player selects a cell, a sensor reading yields a color, determined by the conditional probability distribution $P\left(\frac{Color}{Distance\ from\ Gho}\right)$ This shows how likely it is to observe a specific color depending on the ghost's distance. This distribution is then used to update the posterior probability of the ghost's presence in that cell after the sensor reading.

The updated posterior probability $P\big(Ghost(t)\big)$ is calculated by multiplying the prior probability $P\big(Ghost(t-1)\big)$ with the likelihood of the observed color given the ghost's distance, represented by $P\left(\frac{Color}{Distance\ from\ Ghost}\right)$. This process revises the probability for the ghost being in that cell, considering the latest sensor input.

To ensure these probabilities are valid, they are normalized to sum up to 1 across all cells. This normalization is crucial because the sum of prior and posterior probabilities might not initially equal 1, and we need to adjust them to maintain their relative proportions.

Overall, this Bayesian inference method enables players to make more informed decisions about their next moves to locate the ghost. It involves continually updating posterior probabilities based on observed colors. In this game, probabilistic inference is a key strategy for estimating the likelihood of the ghost's location, using principles of probability theory and data from the joint probability table. This approach involves:

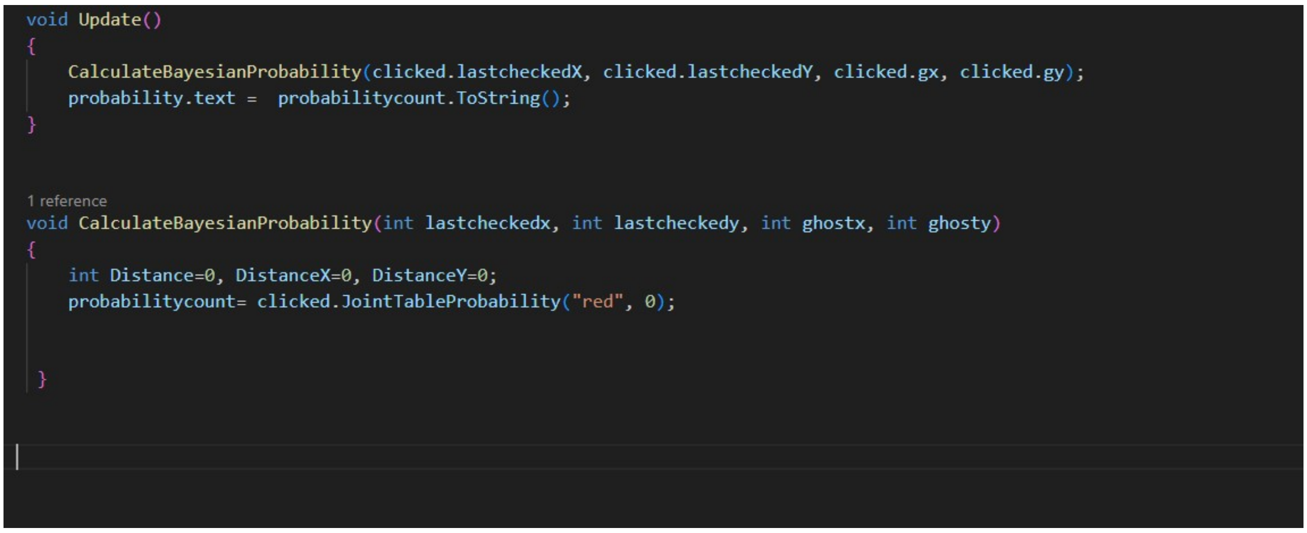
1) **Observation**: Players click on a tile, observing its color, which provides clues about the ghost's location, as different colors correlate with varying distances from the ghost.

2) **Joint probability table**: This table includes joint probability values for each color and distance combination, indicating the chance of seeing a particular color at a certain distance from the ghost. The JointTableProbability function retrieves these values.

3) **Inference:** The code computes the ghost's location probability by applying probability theory rules. It multiplies the joint probability by the inverse of the distance and divides by a normalization factor (90), giving more importance to tiles nearer to the ghost and scaling the probabilities appropriately.

The advantage of probabilistic inference in this game is its ability to guide players in making educated guesses about the ghost's probable location based on the tile colors observed and their distances from the ghost. Displaying these probabilities on the grid allows players to better strategize their search for the ghost.

b. Bayesian Probabilistic Inferencing:



```
void Update()
{
    CalculateBayesianProbability(clicked.lastcheckedX, clicked.lastcheckedY, clicked.gx, clicked.gy);
    probability.text =  probabilitycount.ToString();
}


1 reference
void CalculateBayesianProbability(int lastcheckedx, int lastcheckedy, int ghostx, int ghosty)
{
    int Distance=0, DistanceX=0, DistanceY=0;
    probabilitycount= clicked.JointTableProbability("red", 0);


 }
```

Figure 7: “CalculateBayesianProbability” function in Probabilitytext.cs

The "ProbabilityText" script in Unity is essential for displaying the likelihood of the ghost's presence in each recently checked cell. This script helps players make better decisions by providing updated probability information. It uses the "CalculateBayesianProbability" function to assess the ghost's probability in the last inspected cell, considering the cell's x and y coordinates and the ghost's location. The probability is calculated based on the cell's color and distance, using the "JointProbabilityTable" method from the "Game" object. The calculated probability is then stored in the "probabilitycount" variable.

To keep the displayed probability current, the "Update" function consistently refreshes this value on the screen. It does this by invoking the "CalculateBayesianProbability" function and then updating the "TextMeshPro" component with the new probability value in each frame update.

Overall, the "ProbabilityText" Script is a key tool in the game, offering players vital insights into where the ghost might be located. This information is crucial for strategizing and improving their chances of finding the ghost and winning the game.

VI. Conclusion:

In conclusion, the Unity-based game project "Bust the Ghost" effectively demonstrates the application of Bayesian inferencing to enhance gameplay. By incorporating probabilistic inferencing, the game provides a dynamic and interactive experience, where players use sensor readings to infer the ghost's location. Key elements such as the "ProbabilityText" script and the "CalculateBayesianProbability" function are crucial in updating the probability of the ghost's presence in each cell based on sensor readings. The use of a joint probability table further aids in determining the likelihood of observing specific colors at varying distances from the ghost. This Bayesian approach allows for continuously updated posterior probabilities, offering players strategic insights based on the observed colors and distances from the ghost. Consequently, "Bust the Ghost" not only entertains but also educates players about Bayesian probability theory, making it an innovative and instructive gaming experience.

Demonstration video link: https://youtu.be/ufYvbSCznzs