YouTube Link : https://www.youtube.com/watch?v=cbkws2IFAvo
GitHub Repository : https://github.com/N-ADA/Ghost-Buster.git

Produced by Nada Bounajma
&
Nour El Houda Touti

credit goes to Tarodev - used his algorithm to build the grid
https://www.youtube.com/watch?v=kkAjpQAM-jE

In the Game Grid there is one Ghost hiding and we don't know where the ghost is. Therefore, to find it, we used measurements which are probabilities. Each box/tile is assigned a probability, and once clicked, we get a color back (red, yellow, green, or orange)

The further away we get from the ghost, the probability of Red goes down, orange goes up, yellow goes higher than orange, and Green gets the highest probability :
P(Red) < P(Orange) < P(Yellow) < P(Green)

In the game, we keep clicking on boxes and once we think we hit the ghost(meaning once the box turn RED) we press the Button that is seen on top of the Grid (check the Youtube video). We either receive a 'Success' Message, meaning we busted the ghost, or 'Fail' Message, meaning we missed the ghost.

Green color ==> means we are at least 5 blocks away from the ghost
Red color ==> means we are more likely on the ghost, but we still don't know that for sure, it could be just a noisy measurement. That is why it's better to keep clicking on the boxes around it to get assured then proceed to click on the Button on top of the Grid to bust it(check the YouTube video)

We first create our Game Grid and give it a size of 8x20. We randomly place the ghost as the following code shows

```
public static void Random_Ghost_Position() {
    var ghost = new Vector2(Random.Range(0, w), Random.Range(0, l));
    this.ghost = ghost;
    Debug.Log("The ghost is at position : " + ghost);
}
```

And distribute the equal probabilities on all nodes

```
public static void init_proba() {
    double InitProba = (double) 1 / ( w * l );
    for(int x = 0; x < tiles.Count; x++) {
        Tile tile = tiles.ElementAt(x).Value;
        tile.give_proba(InitProba);
    }
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < l; y++) probabilities[i,j] = init_proba;
    }
}
```

The following code runs on each click == > it changes the probability of the tile that we clicked so the tile reflects and matches with the corresponding color that it shows. After that, we normalize the rest of the tiles to represent a logical probability distribution.

```
public static void update_proba(Tile tile, Vector3 tile_clicked) {
    int x = (int)tile_clicked.x;
    int y = (int)tile_clicked.y;
    double prev_prob = probabilities[x, y];
    int distance = ghost_distance(x, y);
    color color_at_current_cell = color(distance);
    double color_prob = conditional_proba(color_at_current_cell, distance);
    double updated_proba = prev_prob * color_prob;
    probabilities[x, y] = updated_proba;
    tile.give_color(color_at_current_cell);
    tile.give_proba(updated_proba);
    normalization();
}
```

The following code distribute probabilities on the tiles based on their distance from the ghost

```
public static Dictionary<color, double> distribution(int distance_from_ghost) {
    if (distance_from_ghost == 0) return R_Dict;
    if (distance_from_ghost == 1 || distance_from_ghost == 2) return O_Dict;
    if (distance_from_ghost == 3 || distance_from_ghost == 4) return Y_Dict;
    if (distance_from_ghost >= 5) return G_Dict;
    return G_Dict;
}
```