Task 7 : Hitting a Moving Target

7.1 Problem Description

A computer game has a shooter and a moving target. The shooter can hit any of n > 1 hiding spot located along a straight line in which the target can hide. The shooter can never see the target; all he knows is that the target moves to an adjacent hiding spot between every two consecutive shots.

Design a Dynamic Programing algorithm that guarantees hitting the target

7.2 Detailed Assumptions

Target Movement

- The target moves between adjacent spots in a straight line.
- There's no limit on the number of times the target can move.

Shooter

- The shooter can take one shot at a time, aiming at any of the spots.
- The shot is always successful if aimed at the spot where the target currently is.

Other Assumptions

- The number of spots (n) is greater than or equal to 1 (there must be at least one spot).

7.3 Detailed Solution

7.3.1 Description of Solution

We stimulate the target movements and we store the bullets in each hiding spot Then move forward in our hitting spot until we hit the spot n-1 then we hit the way back until we hit the spot 2 with this we guarantee hitting the target

7.3.2 Pseudo Code

```
Function getspot(i, j, n):
    If i is less than 1:
        Return j
    Else if j is greater than n:
        Return i
    Else:
        Generate a random integer between 0 and 999 (inclusive)
        If the random integer is greater than or equal to 499:
            Return i
        Else:
            Return j
    End If
End Function


Function hittingAMovingTarget(n, currentHidingSpot):
    Initialize an array dp of size n+1 with zeros

    Initialize variables:
        k = 1
        count = n-1
        shotspot = 2

    Loop i from 0 to n+1:
        If count is 1:
```

```
            Increment dp[shotspot] by 1

            Set k to -1

            Reset count to n-1

        Else:

            Update dp[shotspot] as max(dp[shotspot-1]+1, dp[shotspot+1]+1)


        Print "shot = " + shotspot + ", hiding spot = " + currentHidingSpot


        If shotspot equals currentHidingSpot:

            Break out of the loop


        Decrement count

        Update currentHidingSpot using getspot(currentHidingSpot-1,
currentHidingSpot+1, n)


        If (shotspot is not 2 and shotspot is not n-1) or count is not 1:

            Increment shotspot by k


    Return dp[currentHidingSpot]

End Function
```

7.4 Complexity Analysis for the Algorithm

Adding the time complexities of initialization and filling the DP table, we get **O(n) + O(n) = O(2n),** which can be simplified to **O(n).**

7.5 Sample Output of the Solution

```
Enter number of hiding spots: 5
Enter current Hiding Spot: 2
Maximum shots needed to guarantee hitting the target: 2
```

```
Enter number of hiding spots: 7
Enter current Hiding Spot: 5
Maximum shots needed to guarantee hitting the target: 3
```

7.6 Comparison with Another Algorithm

TABLE 1

	Dynamic Programming	Greedy
Time Complexity	O(n)	O(n)
Space Complexity	O(n)	O(1)
Optimum Solution	Always minimum number of moves to hit target	A random number of moves not always minimum

7.7 Conclusion

We use two methods to solve the problem

The first method is dynamic programming, we can say that it is an efficient method in that it always out the minimum number of moves to hit targe but it is has a high time complexity and space complexity.

The second method is greedy algorithm it's advantage is it's time complexity and space complexity But it doesn't give a very efficient output