N-Puzzle

Team members:

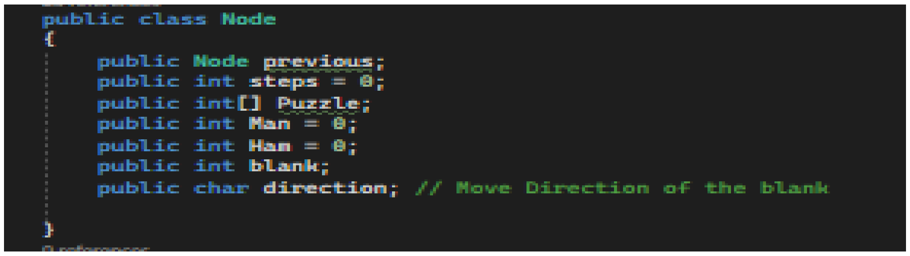
- ٢٠١٩١٧٠٠٩٠٣ محمد رفعت ابو الحمد حمدان
- ٢٠١٩١٧٠٠٧٨٥ يوسف محمد جمعه محمد
- ٢٠١٩١٧٠٠٤٩٤ محمد احمد عبد الرحمن عبد العليم

Team id: T040

Entire source code:

Class Node:

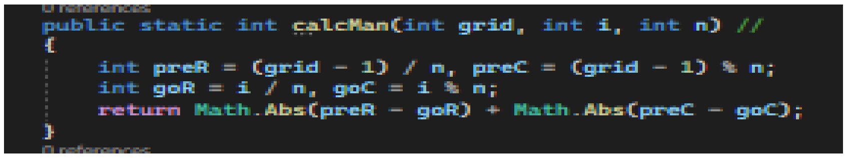
- **That include information of tree of moving of blank Puzzle**



```
public class Node
{
    public Node previous;
    public int steps = 0;
    public int[] Puzzle;
    public int Man = 0;
    public int Ham = 0;
    public int blank;
    public char direction; // Move Direction of the blank
}
```

CalcMan Function:

- **Calculate Manhattan**

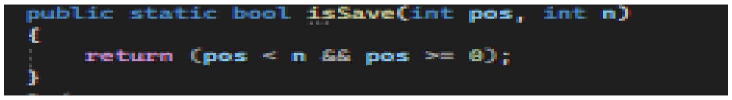


```
public static int calcMan(int grid, int i, int n) //
{
    int preR = (grid - 1) / n, preC = (grid - 1) % n;
    int goR = i / n, goC = i % n;
    return Math.Abs(preR - goR) + Math.Abs(preC - goC);
}
```

Analysis: O(1)

<u>isSave Function:</u>

- **Check validation of one movement**

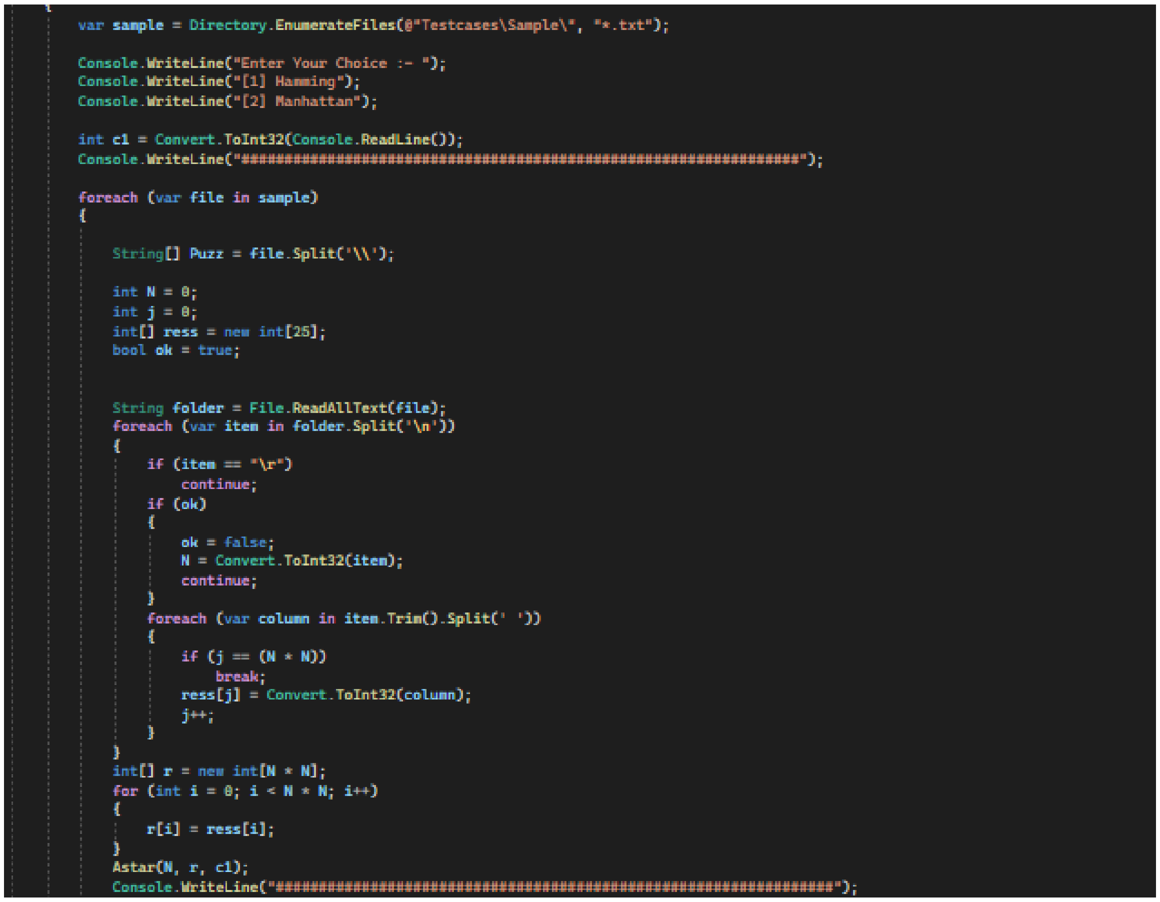


```
public static bool isSave(int pos, int n)
{
    return (pos < n && pos >= 0);
}
```

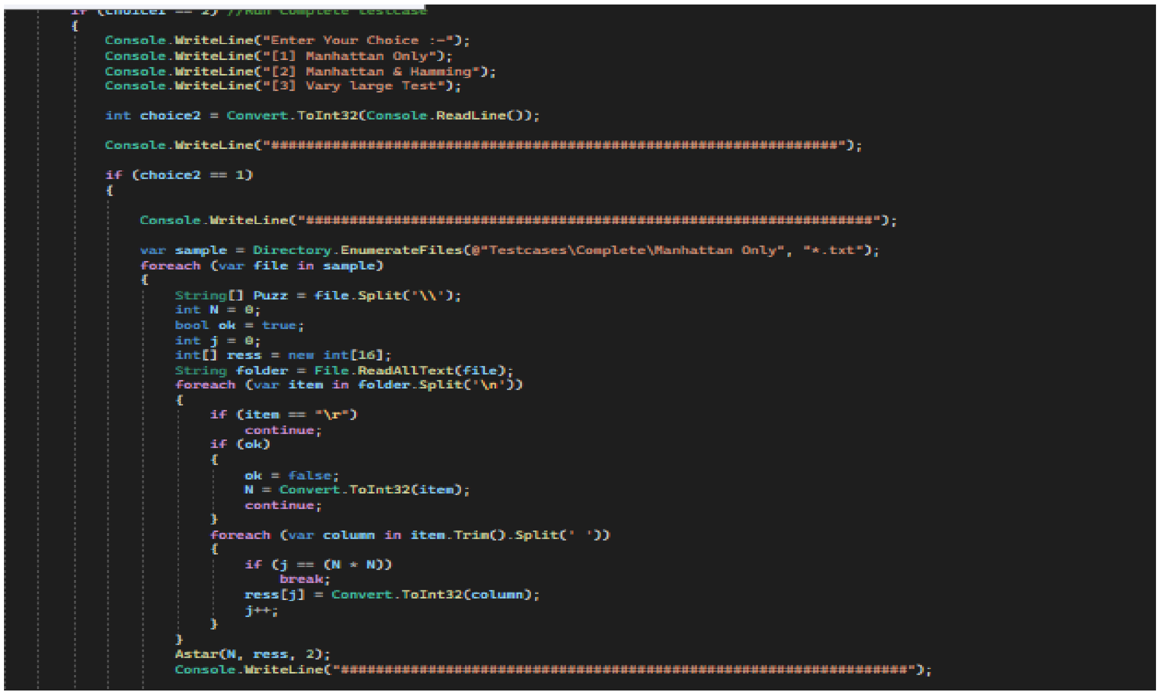
Analysis: O(1)

Reading from files:

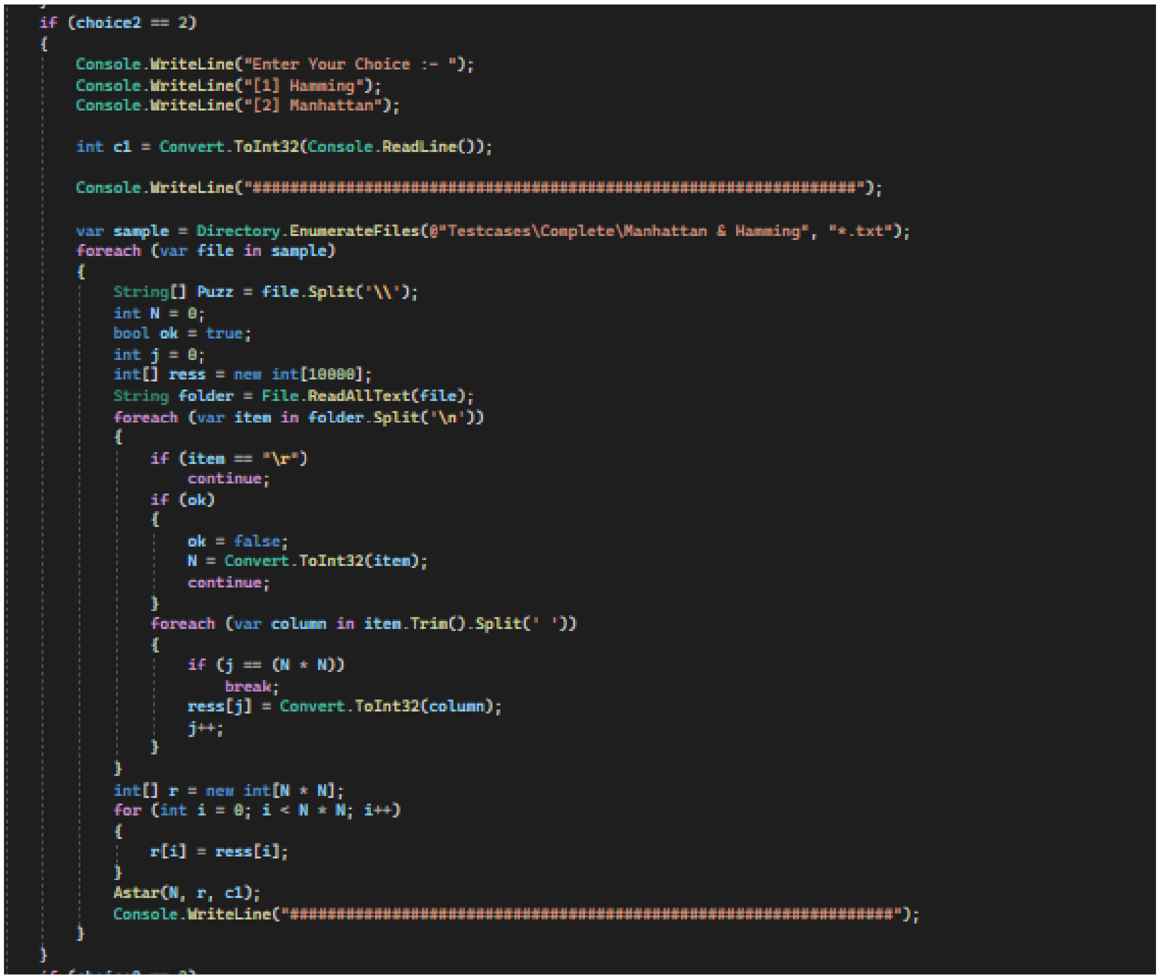
Reading test cases files



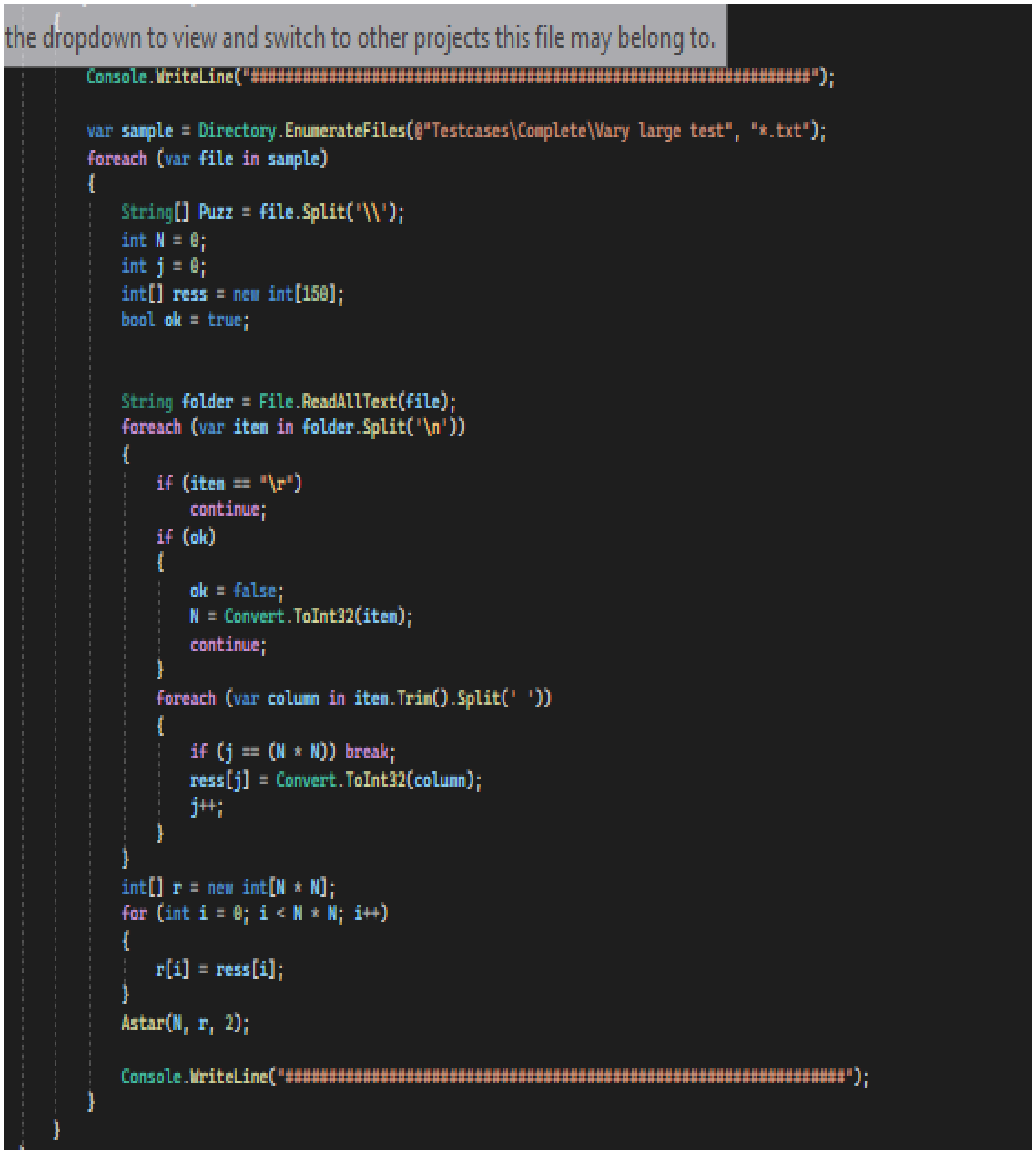
```
    {
        var sample = Directory.EnumerateFiles(@"Testcases\Sample\", "*.txt");

        Console.WriteLine("Enter Your Choice :- ");
        Console.WriteLine("[1] Hamming");
        Console.WriteLine("[2] Manhattan");

        int c1 = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("##################################################################");

        foreach (var file in sample)
        {

            String[] Puzz = file.Split('\\');

            int N = 0;
            int j = 0;
            int[] ress = new int[25];
            bool ok = true;


            String folder = File.ReadAllText(file);
            foreach (var item in folder.Split('\n'))
            {
                if (item == "\r")
                    continue;
                if (ok)
                {
                    ok = false;
                    N = Convert.ToInt32(item);
                    continue;
                }
                foreach (var column in item.Trim().Split(' '))
                {
                    if (j == (N * N))
                        break;
                    ress[j] = Convert.ToInt32(column);
                    j++;
                }
            }
            int[] r = new int[N * N];
            for (int i = 0; i < N * N; i++)
            {
                r[i] = ress[i];
            }
            Astar(N, r, c1);
            Console.WriteLine("##################################################################");
```



```
if (choice1 == 1) //Run Complete testcase
{
    Console.WriteLine("Enter Your Choice :-");
    Console.WriteLine("[1] Manhattan Only");
    Console.WriteLine("[2] Manhattan & Hamming");
    Console.WriteLine("[3] Vary large Test");

    int choice2 = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("######################################################################");

    if (choice2 == 1)
    {

        Console.WriteLine("######################################################################");

        var sample = Directory.EnumerateFiles(@"Testcases\Complete\Manhattan Only", "*.txt");
        foreach (var file in sample)
        {
            String[] Puzz = file.Split('\\');
            int N = 0;
            bool ok = true;
            int j = 0;
            int[] ress = new int[16];
            String folder = File.ReadAllText(file);
            foreach (var item in folder.Split('\n'))
            {
                if (item == "\r")
                    continue;
                if (ok)
                {
                    ok = false;
                    N = Convert.ToInt32(item);
                    continue;
                }
                foreach (var column in item.Trim().Split(' '))
                {
                    if (j == (N * N))
                        break;
                    ress[j] = Convert.ToInt32(column);
                    j++;
                }
            }
            Astar(N, ress, 2);
            Console.WriteLine("######################################################################");
```



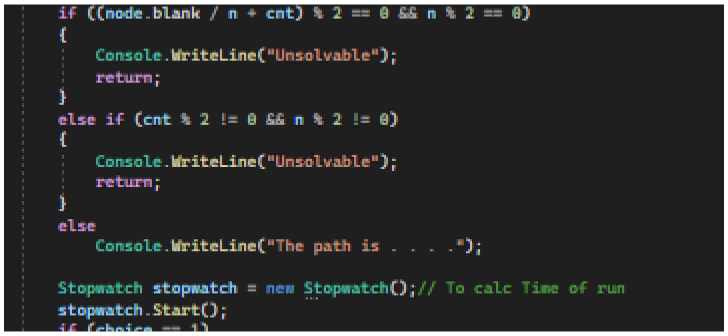
```
if (choice2 == 2)
{
    Console.WriteLine("Enter Your Choice :- ");
    Console.WriteLine("[1] Hamming");
    Console.WriteLine("[2] Manhattan");

    int c1 = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("######################################################################");

    var sample = Directory.EnumerateFiles(@"Testcases\Complete\Manhattan & Hamming", "*.txt");
    foreach (var file in sample)
    {
        String[] Puzz = file.Split('\\');
        int N = 0;
        bool ok = true;
        int j = 0;
        int[] ress = new int[10000];
        String folder = File.ReadAllText(file);
        foreach (var item in folder.Split('\n'))
        {
            if (item == "\r")
                continue;
            if (ok)
            {
                ok = false;
                N = Convert.ToInt32(item);
                continue;
            }
            foreach (var column in item.Trim().Split(' '))
            {
                if (j == (N * N))
                    break;
                ress[j] = Convert.ToInt32(column);
                j++;
            }
        }
        int[] r = new int[N * N];
        for (int i = 0; i < N * N; i++)
        {
            r[i] = ress[i];
        }
        Astar(N, r, c1);
        Console.WriteLine("######################################################################");
    }
}
```


the dropdown to view and switch to other projects this file may belong to.

```
        Console.WriteLine("######################################################################");

        var sample = Directory.EnumerateFiles(@"Testcases\Complete\Vary large test", "*.txt");
        foreach (var file in sample)
        {
            String[] Puzz = file.Split('\\');
            int N = 0;
            int j = 0;
            int[] ress = new int[150];
            bool ok = true;


            String folder = File.ReadAllText(file);
            foreach (var item in folder.Split('\n'))
            {
                if (item == "\r")
                    continue;
                if (ok)
                {
                    ok = false;
                    N = Convert.ToInt32(item);
                    continue;
                }
                foreach (var column in item.Trim().Split(' '))
                {
                    if (j == (N * N)) break;
                    ress[j] = Convert.ToInt32(column);
                    j++;
                }
            }
            int[] r = new int[N * N];
            for (int i = 0; i < N * N; i++)
            {
                r[i] = ress[i];
            }
            Astar(N, r, 2);

            Console.WriteLine("######################################################################");
        }
    }
```

<u>Astar function</u>:

- **Check the puzzle is solvable or not**

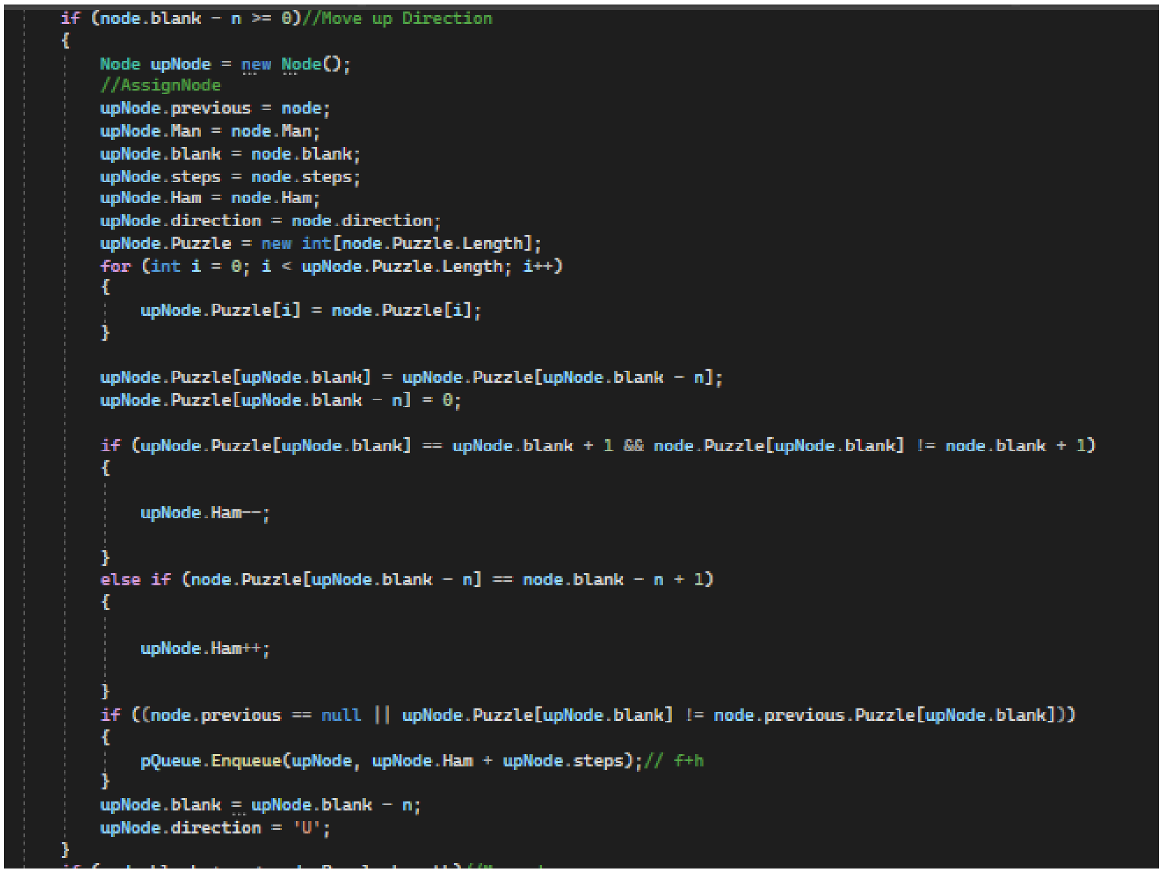


```
if ((node.blank / n + cnt) % 2 == 0 && n % 2 == 0)
{
    Console.WriteLine("Unsolvable");
    return;
}
else if (cnt % 2 != 0 && n % 2 != 0)
{
    Console.WriteLine("Unsolvable");
    return;
}
else
    Console.WriteLine("The path is . . . .");

Stopwatch stopwatch = new Stopwatch();// To calc Time of run
stopwatch.Start();
```

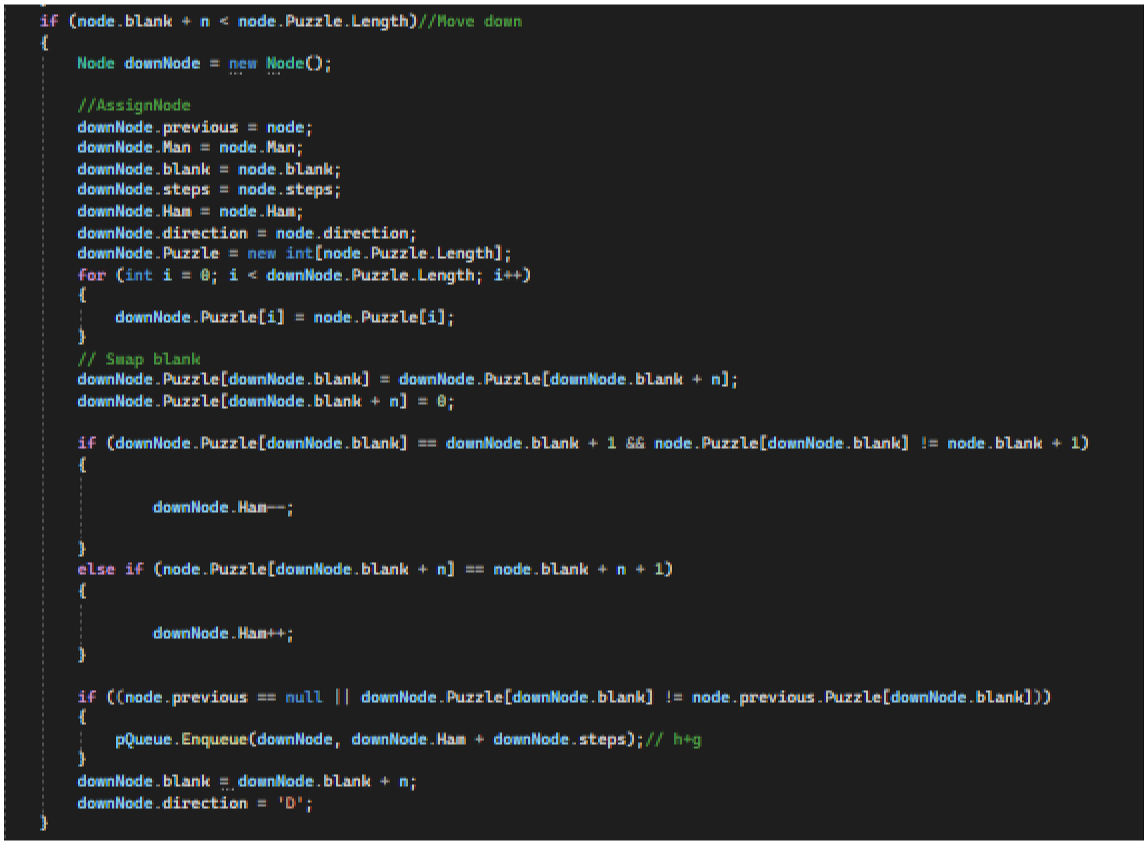
- **Solve puzzle by**
    1. **Hamming**

       Calc number of misplaced items

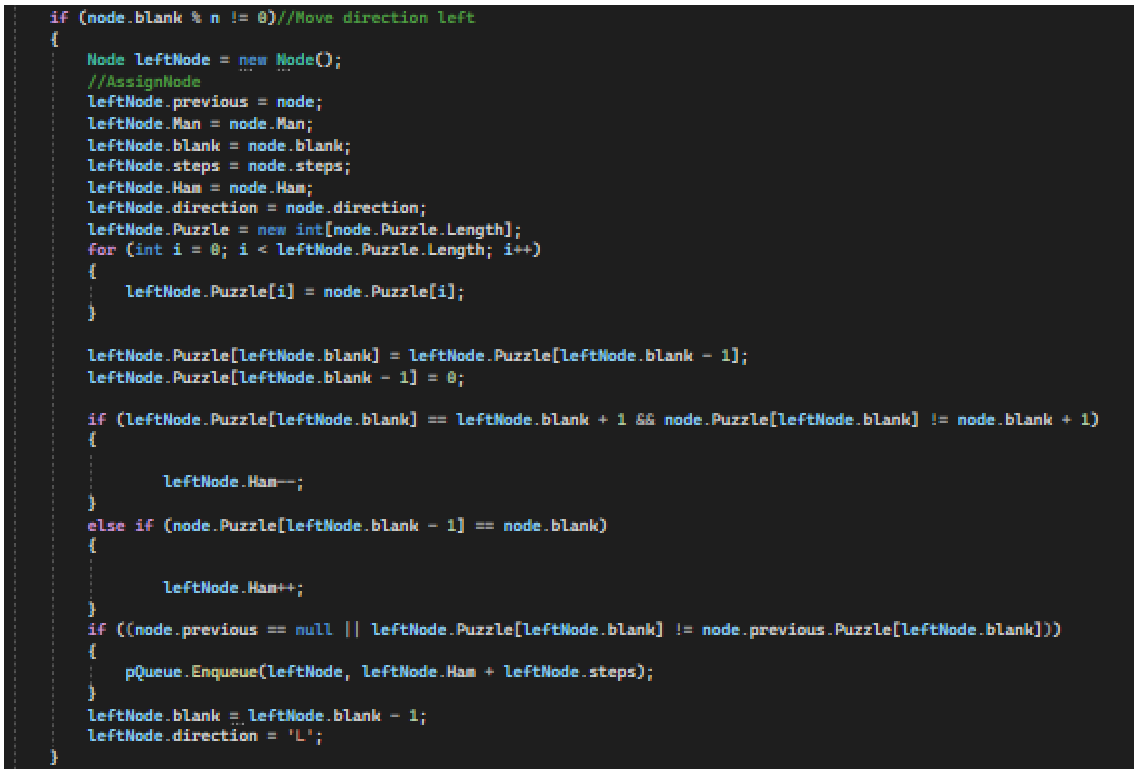
       Check availability of moving blank (UP,Down,Left,Right)



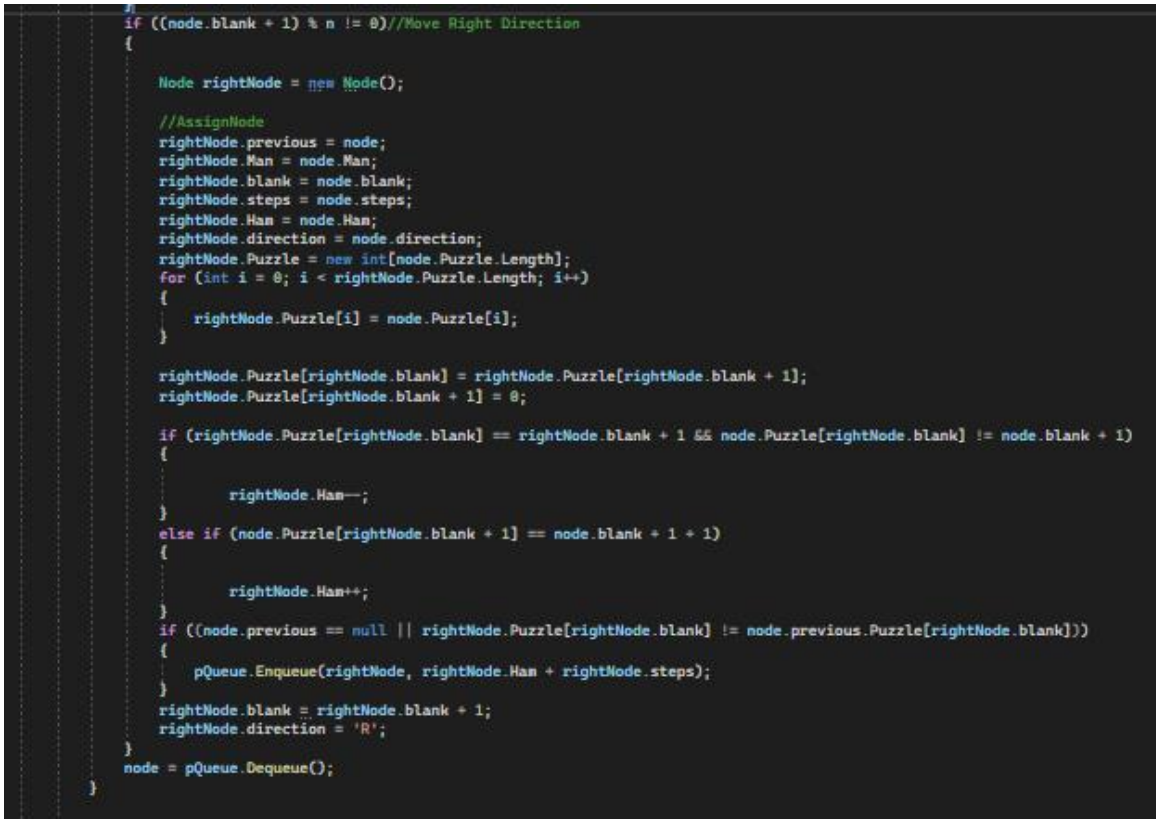
```
if (node.blank - n >= 0)//Move up Direction
{
    Node upNode = new Node();
    //AssignNode
    upNode.previous = node;
    upNode.Man = node.Man;
    upNode.blank = node.blank;
    upNode.steps = node.steps;
    upNode.Ham = node.Ham;
    upNode.direction = node.direction;
    upNode.Puzzle = new int[node.Puzzle.Length];
    for (int i = 0; i < upNode.Puzzle.Length; i++)
    {
        upNode.Puzzle[i] = node.Puzzle[i];
    }

    upNode.Puzzle[upNode.blank] = upNode.Puzzle[upNode.blank - n];
    upNode.Puzzle[upNode.blank - n] = 0;

    if (upNode.Puzzle[upNode.blank] == upNode.blank + 1 && node.Puzzle[upNode.blank] != node.blank + 1)
    {

        upNode.Ham--;

    }
    else if (node.Puzzle[upNode.blank - n] == node.blank - n + 1)
    {

        upNode.Ham++;

    }
    if ((node.previous == null || upNode.Puzzle[upNode.blank] != node.previous.Puzzle[upNode.blank]))
    {
        pQueue.Enqueue(upNode, upNode.Ham + upNode.steps);// f+h
    }
    upNode.blank = upNode.blank - n;
    upNode.direction = 'U';
}
```



```
if (node.blank + n < node.Puzzle.Length)//Move down
{
    Node downNode = new Node();

    //AssignNode
    downNode.previous = node;
    downNode.Man = node.Man;
    downNode.blank = node.blank;
    downNode.steps = node.steps;
    downNode.Ham = node.Ham;
    downNode.direction = node.direction;
    downNode.Puzzle = new int[node.Puzzle.Length];
    for (int i = 0; i < downNode.Puzzle.Length; i++)
    {
        downNode.Puzzle[i] = node.Puzzle[i];
    }
    // Swap blank
    downNode.Puzzle[downNode.blank] = downNode.Puzzle[downNode.blank + n];
    downNode.Puzzle[downNode.blank + n] = 0;

    if (downNode.Puzzle[downNode.blank] == downNode.blank + 1 && node.Puzzle[downNode.blank] != node.blank + 1)
    {

            downNode.Ham--;

    }
    else if (node.Puzzle[downNode.blank + n] == node.blank + n + 1)
    {

            downNode.Ham++;
    }

    if ((node.previous == null || downNode.Puzzle[downNode.blank] != node.previous.Puzzle[downNode.blank]))
    {
        pQueue.Enqueue(downNode, downNode.Ham + downNode.steps);// h+g
    }
    downNode.blank = downNode.blank + n;
    downNode.direction = 'D';
}
```



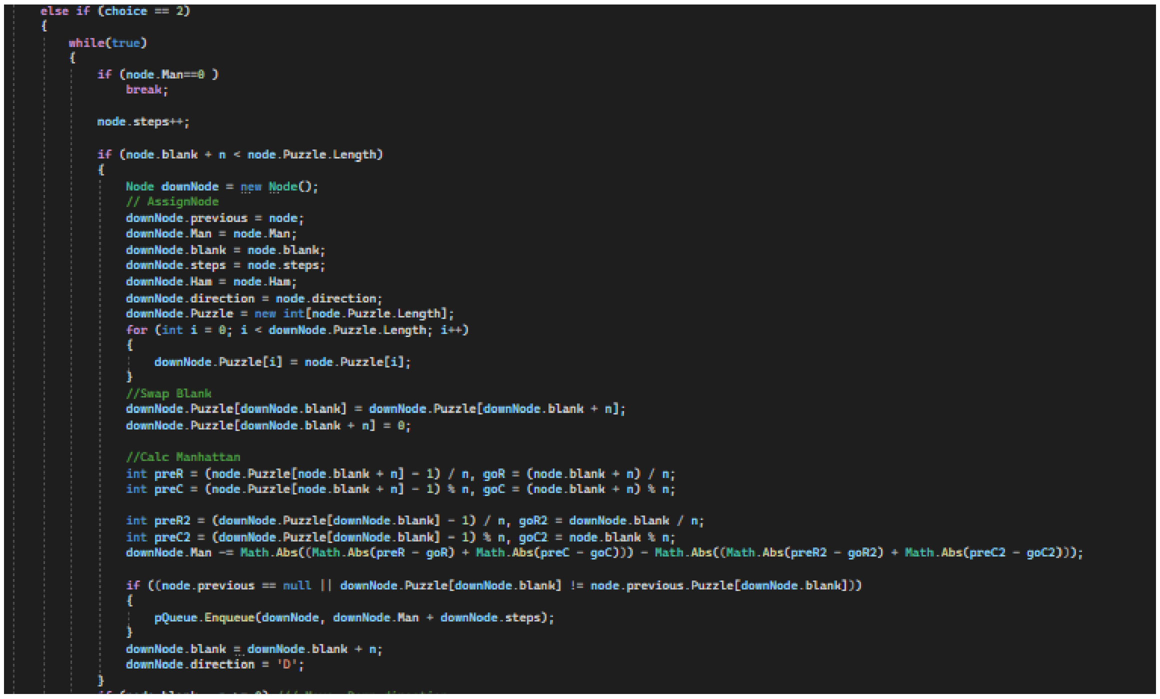
```
if (node.blank % n != 0)//Move direction left
{
    Node leftNode = new Node();
    //AssignNode
    leftNode.previous = node;
    leftNode.Man = node.Man;
    leftNode.blank = node.blank;
    leftNode.steps = node.steps;
    leftNode.Ham = node.Ham;
    leftNode.direction = node.direction;
    leftNode.Puzzle = new int[node.Puzzle.Length];
    for (int i = 0; i < leftNode.Puzzle.Length; i++)
    {
        leftNode.Puzzle[i] = node.Puzzle[i];
    }

    leftNode.Puzzle[leftNode.blank] = leftNode.Puzzle[leftNode.blank - 1];
    leftNode.Puzzle[leftNode.blank - 1] = 0;

    if (leftNode.Puzzle[leftNode.blank] == leftNode.blank + 1 && node.Puzzle[leftNode.blank] != node.blank + 1)
    {

            leftNode.Ham--;
    }
    else if (node.Puzzle[leftNode.blank - 1] == node.blank)
    {

            leftNode.Ham++;
    }
    if ((node.previous == null || leftNode.Puzzle[leftNode.blank] != node.previous.Puzzle[leftNode.blank]))
    {
        pQueue.Enqueue(leftNode, leftNode.Ham + leftNode.steps);
    }
    leftNode.blank = leftNode.blank - 1;
    leftNode.direction = 'L';
}
```



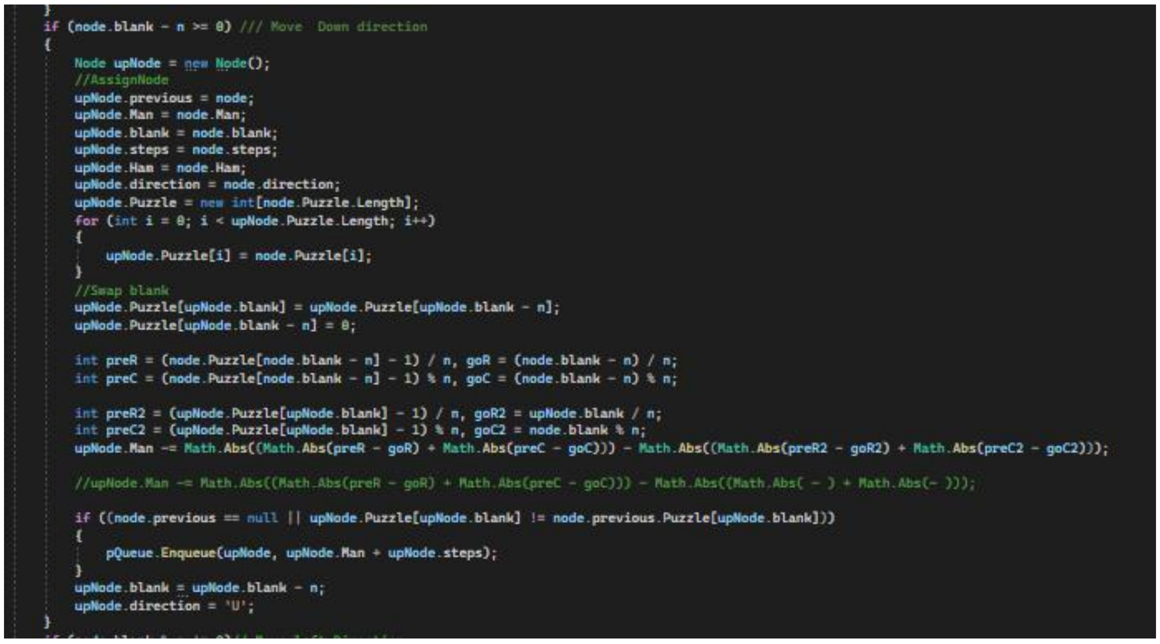
```
            if ((node.blank + 1) % n != 0)//Move Right Direction
            {

                Node rightNode = new Node();

                //AssignNode
                rightNode.previous = node;
                rightNode.Man = node.Man;
                rightNode.blank = node.blank;
                rightNode.steps = node.steps;
                rightNode.Ham = node.Ham;
                rightNode.direction = node.direction;
                rightNode.Puzzle = new int[node.Puzzle.Length];
                for (int i = 0; i < rightNode.Puzzle.Length; i++)
                {
                    rightNode.Puzzle[i] = node.Puzzle[i];
                }

                rightNode.Puzzle[rightNode.blank] = rightNode.Puzzle[rightNode.blank + 1];
                rightNode.Puzzle[rightNode.blank + 1] = 0;

                if (rightNode.Puzzle[rightNode.blank] == rightNode.blank + 1 && node.Puzzle[rightNode.blank] != node.blank + 1)
                {

                        rightNode.Ham--;
                }
                else if (node.Puzzle[rightNode.blank + 1] == node.blank + 1 + 1)
                {

                        rightNode.Ham++;
                }
                if ((node.previous == null || rightNode.Puzzle[rightNode.blank] != node.previous.Puzzle[rightNode.blank]))
                {
                    pQueue.Enqueue(rightNode, rightNode.Ham + rightNode.steps);
                }
                rightNode.blank = rightNode.blank + 1;
                rightNode.direction = 'R';
            }
            node = pQueue.Dequeue();
        }
```

2. Manhattan

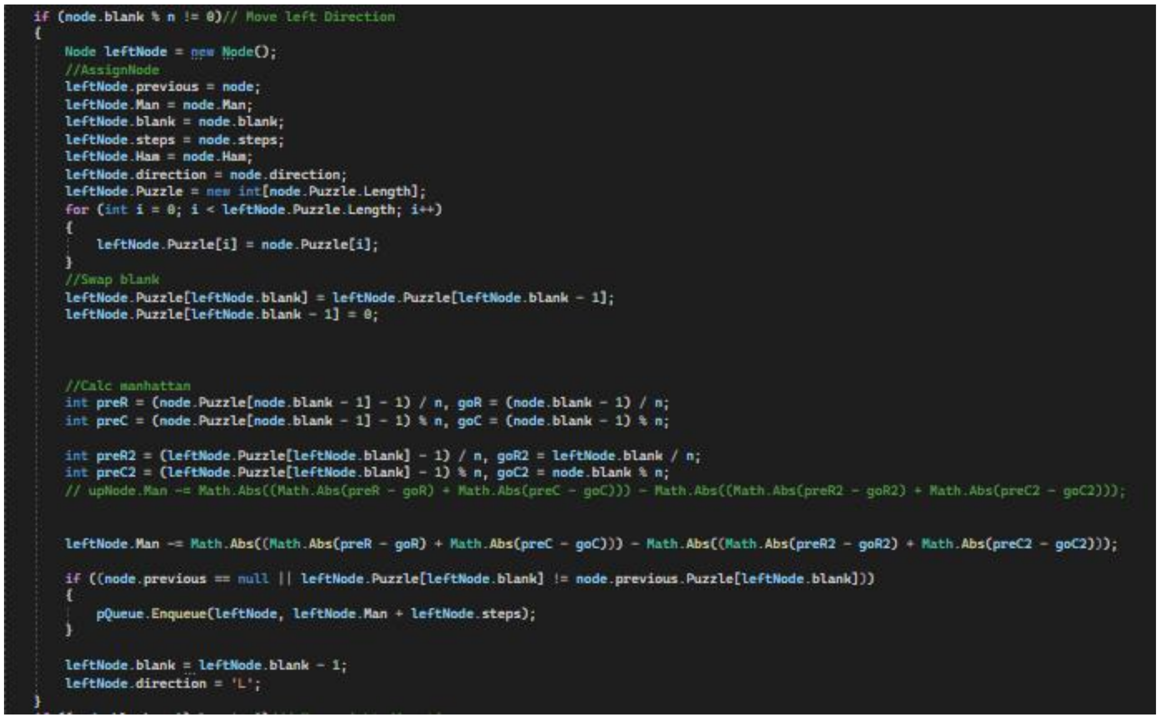
- **Calc distance of goal position and state position items**
- **Check availability of moving blank (UP,Down,Left,Right)**



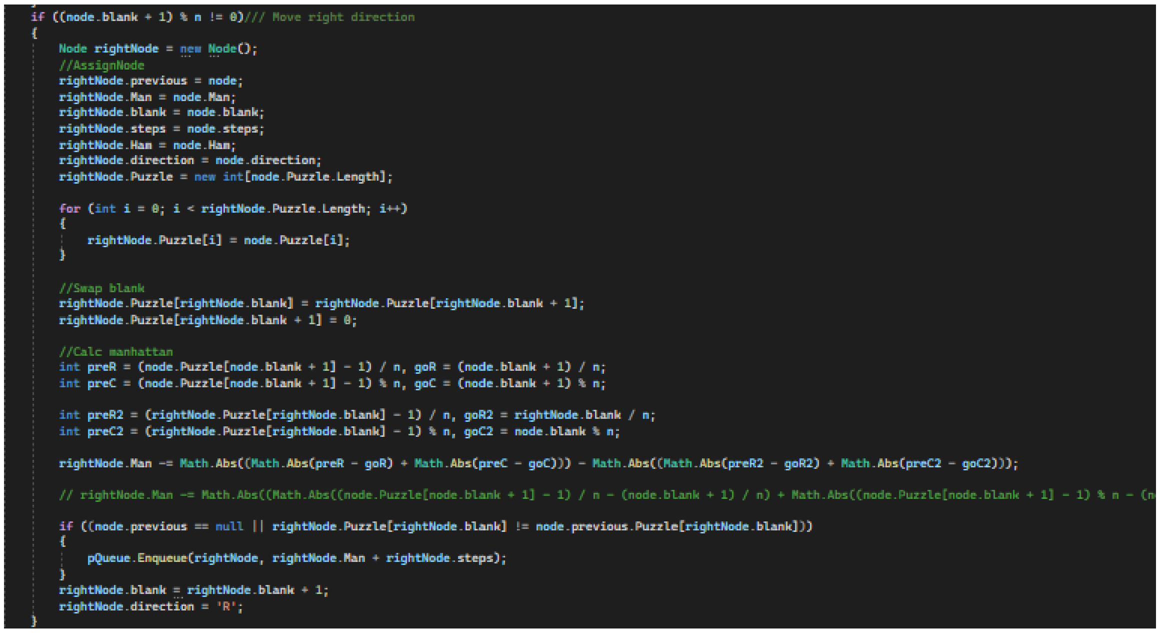
```
    else if (choice == 2)
    {
        while(true)
        {
            if (node.Man==0 )
                break;

            node.steps++;

            if (node.blank + n < node.Puzzle.Length)
            {
                Node downNode = new Node();
                // AssignNode
                downNode.previous = node;
                downNode.Man = node.Man;
                downNode.blank = node.blank;
                downNode.steps = node.steps;
                downNode.Ham = node.Ham;
                downNode.direction = node.direction;
                downNode.Puzzle = new int[node.Puzzle.Length];
                for (int i = 0; i < downNode.Puzzle.Length; i++)
                {
                    downNode.Puzzle[i] = node.Puzzle[i];
                }
                //Swap Blank
                downNode.Puzzle[downNode.blank] = downNode.Puzzle[downNode.blank + n];
                downNode.Puzzle[downNode.blank + n] = 0;

                //Calc Manhattan
                int preR = (node.Puzzle[node.blank + n] - 1) / n, goR = (node.blank + n) / n;
                int preC = (node.Puzzle[node.blank + n] - 1) % n, goC = (node.blank + n) % n;

                int preR2 = (downNode.Puzzle[downNode.blank] - 1) / n, goR2 = downNode.blank / n;
                int preC2 = (downNode.Puzzle[downNode.blank] - 1) % n, goC2 = node.blank % n;
                downNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs(preR2 - goR2) + Math.Abs(preC2 - goC2)));

                if ((node.previous == null || downNode.Puzzle[downNode.blank] != node.previous.Puzzle[downNode.blank]))
                {
                    pQueue.Enqueue(downNode, downNode.Man + downNode.steps);
                }
                downNode.blank = downNode.blank + n;
                downNode.direction = 'D';
            }
```



```
}
if (node.blank - n >= 0) /// Move  Down direction
{
    Node upNode = new Node();
    //AssignNode
    upNode.previous = node;
    upNode.Man = node.Man;
    upNode.blank = node.blank;
    upNode.steps = node.steps;
    upNode.Ham = node.Ham;
    upNode.direction = node.direction;
    upNode.Puzzle = new int[node.Puzzle.Length];
    for (int i = 0; i < upNode.Puzzle.Length; i++)
    {
        upNode.Puzzle[i] = node.Puzzle[i];
    }
    //Swap blank
    upNode.Puzzle[upNode.blank] = upNode.Puzzle[upNode.blank - n];
    upNode.Puzzle[upNode.blank - n] = 0;

    int preR = (node.Puzzle[node.blank - n] - 1) / n, goR = (node.blank - n) / n;
    int preC = (node.Puzzle[node.blank - n] - 1) % n, goC = (node.blank - n) % n;

    int preR2 = (upNode.Puzzle[upNode.blank] - 1) / n, goR2 = upNode.blank / n;
    int preC2 = (upNode.Puzzle[upNode.blank] - 1) % n, goC2 = node.blank % n;
    upNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs(preR2 - goR2) + Math.Abs(preC2 - goC2)));

    //upNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs( - ) + Math.Abs(- )));

    if ((node.previous == null || upNode.Puzzle[upNode.blank] != node.previous.Puzzle[upNode.blank]))
    {
        pQueue.Enqueue(upNode, upNode.Man + upNode.steps);
    }
    upNode.blank = upNode.blank - n;
    upNode.direction = 'U';
}
```

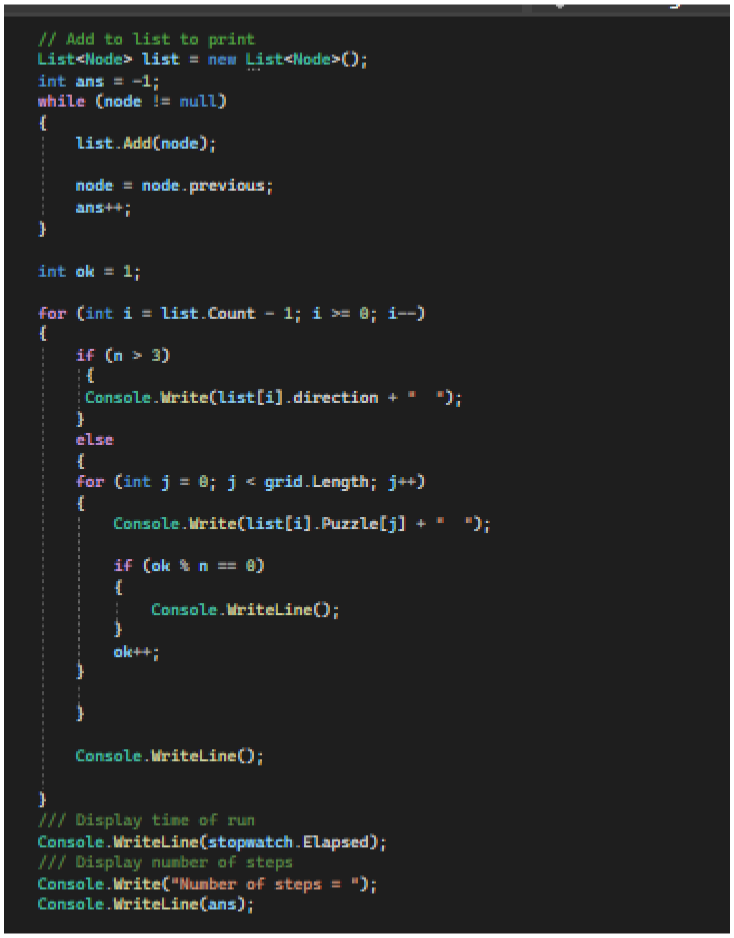


```
if (node.blank % n != 0)// Move left Direction
{
    Node leftNode = new Node();
    //AssignNode
    leftNode.previous = node;
    leftNode.Man = node.Man;
    leftNode.blank = node.blank;
    leftNode.steps = node.steps;
    leftNode.Ham = node.Ham;
    leftNode.direction = node.direction;
    leftNode.Puzzle = new int[node.Puzzle.Length];
    for (int i = 0; i < leftNode.Puzzle.Length; i++)
    {
        leftNode.Puzzle[i] = node.Puzzle[i];
    }
    //Swap blank
    leftNode.Puzzle[leftNode.blank] = leftNode.Puzzle[leftNode.blank - 1];
    leftNode.Puzzle[leftNode.blank - 1] = 0;



    //Calc manhattan
    int preR = (node.Puzzle[node.blank - 1] - 1) / n, goR = (node.blank - 1) / n;
    int preC = (node.Puzzle[node.blank - 1] - 1) % n, goC = (node.blank - 1) % n;

    int preR2 = (leftNode.Puzzle[leftNode.blank] - 1) / n, goR2 = leftNode.blank / n;
    int preC2 = (leftNode.Puzzle[leftNode.blank] - 1) % n, goC2 = node.blank % n;
    // upNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs(preR2 - goR2) + Math.Abs(preC2 - goC2)));


    leftNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs(preR2 - goR2) + Math.Abs(preC2 - goC2)));

    if ((node.previous == null || leftNode.Puzzle[leftNode.blank] != node.previous.Puzzle[leftNode.blank]))
    {
        pQueue.Enqueue(leftNode, leftNode.Man + leftNode.steps);
    }

    leftNode.blank = leftNode.blank - 1;
    leftNode.direction = 'L';
}
```



```
if ((node.blank + 1) % n != 0)/// Move right direction
{
    Node rightNode = new Node();
    //AssignNode
    rightNode.previous = node;
    rightNode.Man = node.Man;
    rightNode.blank = node.blank;
    rightNode.steps = node.steps;
    rightNode.Ham = node.Ham;
    rightNode.direction = node.direction;
    rightNode.Puzzle = new int[node.Puzzle.Length];

    for (int i = 0; i < rightNode.Puzzle.Length; i++)
    {
        rightNode.Puzzle[i] = node.Puzzle[i];
    }

    //Swap blank
    rightNode.Puzzle[rightNode.blank] = rightNode.Puzzle[rightNode.blank + 1];
    rightNode.Puzzle[rightNode.blank + 1] = 0;

    //Calc manhattan
    int preR = (node.Puzzle[node.blank + 1] - 1) / n, goR = (node.blank + 1) / n;
    int preC = (node.Puzzle[node.blank + 1] - 1) % n, goC = (node.blank + 1) % n;

    int preR2 = (rightNode.Puzzle[rightNode.blank] - 1) / n, goR2 = rightNode.blank / n;
    int preC2 = (rightNode.Puzzle[rightNode.blank] - 1) % n, goC2 = node.blank % n;

    rightNode.Man -= Math.Abs((Math.Abs(preR - goR) + Math.Abs(preC - goC))) - Math.Abs((Math.Abs(preR2 - goR2) + Math.Abs(preC2 - goC2)));

    // rightNode.Man -= Math.Abs((Math.Abs((node.Puzzle[node.blank + 1] - 1) / n - (node.blank + 1) / n) + Math.Abs((node.Puzzle[node.blank + 1] - 1) % n - (n

    if ((node.previous == null || rightNode.Puzzle[rightNode.blank] != node.previous.Puzzle[rightNode.blank]))
    {
        pQueue.Enqueue(rightNode, rightNode.Man + rightNode.steps);
    }
    rightNode.blank = rightNode.blank + 1;
    rightNode.direction = 'R';
}
```

- **Add Node To LIST**
  Add information of class new node to list to display it



```
// Add to list to print
List<Node> list = new List<Node>();
int ans = -1;
while (node != null)
{
    list.Add(node);

    node = node.previous;
    ans++;
}

int ok = 1;

for (int i = list.Count - 1; i >= 0; i--)
{
    if (n > 3)
     {
     Console.Write(list[i].direction + "  ");
    }
    else
    {
    for (int j = 0; j < grid.Length; j++)
    {
        Console.Write(list[i].Puzzle[j] + "  ");

        if (ok % n == 0)
        {
            Console.WriteLine();
        }
        ok++;
    }

    }

    Console.WriteLine();

}
/// Display time of run
Console.WriteLine(stopwatch.Elapsed);
/// Display number of steps
Console.Write("Number of steps = ");
Console.WriteLine(ans);
```

<u>Analysis:</u>

Solvable or not :O(N^2)

Astar : O(E log(V))

Test Cases time:

Sample

Test Case	Hamming	Manhattan
8 Puzzle (1)	0.001	0.005
8 Puzzle (2)	0.006	0.005
8 Puzzle (3)	0.032	0.024
15 Puzzle 1	0.087	0.061
24 puzzle 1	0.221	0.148
24 puzzle 2	0.15	0.103

Complete

Manhattan Only

Test Case	Manhattan
15 Puzzle 1	4.086
15 Puzzle 2	0.832
15 Puzzle 3	0.816
15 Puzzle 4	24.17

Manhattan & Hamming

Test Case	Hamming	Manhattan
50 Puzzle	0.227	0.019
99 Puzzle 1	0.173	0.028
99 Puzzle 2	0.035	0.083
999 Puzzle	0.005	0.005

V large : 17.18 sec

GitHub:

https://github.com/Mohammed-Refat

References:

A* Pseudocodo **https://en.wikipedia.org/wiki/Iterative_deepening_A***

Solvability **https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/?fbclid=IwAR1XzWKH3fxPFj-q7Yh5NjWyXVfh2j9OZPzOUnqzEaBgYsUd4lQ--SE1IM8**