```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KnightTour
{
   abstract class Chess
   {
      public int[,] board = new int[8,8]; //64 = 8*8
      public int[] Position { get; set; }
      //public abstract int[][] getOptions();
   }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace KnightTour
{
```

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    public void setOutputBox(String s)
    {
        OutputBox.Text = s;
    }

    private void button1_Click(object sender, EventArgs e)
    {
    }

    private void OutputBox_TextChanged(object sender, EventArgs e)
    {

    }

    private void Form1_Load(object sender, EventArgs e)
    {
        update();
    }
    private void update()
    {
        string output = "";
        using (StreamReader sr = new StreamReader("JaiquonNonIntelligentMethod.txt"))
```

```
        {
            while (!sr.EndOfStream)
                output += sr.ReadLine();
        }
        setOutputBox(output);
    }

    private void start_Click(object sender, EventArgs e)
    {
        bool s = smart.Checked, nonS = nonSmart.Checked;
        int trial = Convert.ToInt32(trials.Text), x = Convert.ToInt32(col.Value), y =
Convert.ToInt32(row.Value);
        if (s)
        {
            for (int t = 0; t < trial; t++)
            {
                Intelligent i = new Intelligent(x, y);
            }
        }
        else if (nonS)
        {
            Non_Intelligent nonI = new Non_Intelligent(trial, x, y);
            sd.Text = nonI.standDev+"";
            avg.Text = nonI.avg + "";
            update();
        }
        else
            MessageBox.Show("Select Computer Method", "Computer Method", MessageBoxButtons.OK,
MessageBoxIcon.Error);
```

```
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace ConsoleApp2
{
    class hauristic
    {

        public void hauristicBoard()
        {
            int[,] board = new int[8, 8];
            int[] horizontal = new int[8] { 2, 1, -1, -2, -2, -1, 1, 2 };
            int[] vertical = new int[8] { -1, -2, -2, -1, 1, 2, 2, 1 };
            bool[] legal = new bool[8] { false };
            int startRow = 0;   //will be user defined
            int startCol = 0;   //will be user defined
            int currentRow, currentCol;

            //to count the number of blocks so that it equals 64
            for (int block = 1; block <= 64; block++)
            {
```

```
   int[,] hBoard = new int[8, 8]{ //Heuristic board.
      {2,3,4,4,4,4,3,2},
      {3,4,6,6,6,6,4,3},
      {4,6,8,8,8,8,6,4},
      {4,6,8,8,8,8,6,4},
      {4,6,8,8,8,8,6,4},
      {4,6,8,8,8,8,6,4},
      {3,4,6,6,6,6,4,3},
      {2,3,4,4,4,4,3,2}
   };

   moves = moves(currentRow,currentCol, vertical, horizontal,hBoard, legal)
}//end of for

currentCol = startCol;
currentRow = startRow;

//populating the board with 0s
for (int i = 0; i < 8; i++)
{
   for (int j = 0; j < 8; j++)
   {
      board[i, j] = 0;
   }
}

//occupied tile
board[currentCol, currentRow] = 1;
```

```
}

public void options(int cRow, int cCol, int[] vert, int[] horiz, int[,] hboard, bool[] legal)
{// finds the moves available
   int move = 0;
   int tempRow = cRow + vert[move];
   int tempCol = cCol + horiz[move];

   if (hboard[tempRow, tempCol] == 0)
   {
      legal[move] = true;
   }
   else
   {
      legal[move] = false;
   }
   ++move;
}

public int moves(int cRow, int cCol, int[] vert, int[] horiz, int[,] hboard, bool[] legal)
{// finds the best possible moves
   int minAccess = 2;
   int bestMove = 0;

   for (int options = 0; options < 8; options++)
   {
      if (legal[options] == false)
      {
```

```
            continue;
        }
        //store current legal move in temp var
        int tempR = cRow + vert[options];
        int tempC = cCol + horiz[options];

        //choose the lowest number avaible and make that the best move.
        if (hboard[tempR, tempC] < minAccess)
        {
            minAccess = hboard[tempR, tempC];
            bestMove = options;
            break;
        }




    }
    return bestMove;
}

public void Write(String s)
{//writing to the text file

    using (StreamWriter sw = new StreamWriter(Path.Combine(Directory.GetCurrentDirectory(),
"AsherHauristicMethod.txt"), true))
    {
        sw.Write(s);
    }
```

```
        }


    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KnightTour
{
    class Knight:Chess
    {
        private int[][] move = { new int[]{2,1}, new int[]{1,2} }; // Allowed moves {Horizontal, Vertical}


        public int[,] getOptions()
        {
            int[] pos = Position; //[x,y]
            int x=pos[0],y=pos[1];
            int[][] availMoves = new int[][]{
                                    new int[4]{x+move[0][0],x-move[0][0], x+move[1][0], x-move[1][0]},
                                    new int[4]{y+move[0][1],y-move[0][1], y+move[1][1], y-move[1][1]}
                                };
            int[,] r = new int[8,2]; //Returning Variable with the options


            /*
```

```
   * Moves allowed to make (Max:8)
* X, Y
* x+2,y+1 - Left Up L Horizontal
* x-2,y+1 - Right Up L Horizontal
* x+2,y-1 - Left Down L Horizontal
* x-2,y-1 - Right Down L Horizontal
* x+1,y+2 - Left Up Vertical
* x-1,y+2 - Right Up L Vertical
* x+1,y-2 - Left Down Vertical
* x-1,y-2 - Right Down Vertical
*/
  for (int i = 0; i < 8; i++)
  {
     if (i < 4)
     {
        if (!Bound(availMoves[0][i], availMoves[1][i]))
           continue;
        r[i, 0] = availMoves[0][i];
        r[i, 1] = availMoves[1][i];
     }
     else
     {
        if (!Bound(availMoves[0][i-4], availMoves[1][i-4]))
           continue;
        r[i, 0] = availMoves[0][i-4];
        r[i, 1] = availMoves[1][i-4];
     }
  }
  return r;
```

```
        }
        private bool Bound(int x, int y) //Check if values are off the board
        {
            if (x > 0 && x < 8)
                if (y > 0 && y < 8)
                    return true;
            return false;
        }
        public void Move(int x, int y) //Move the knight
        {
            if(x>0 && y>0)
                Position = new int[]{x,y};
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace KnightTour
{
    class Non_Intelligent:Chess
    {
        /*
         * Randomly select Knight's move until there's no more choices/options
```

```
 */
const string fn = "JaiquonNonIntelligentMethod.txt";
private static Knight k = new Knight();
private static Task task = new Task();
public int trials, squares; //Trials = Game runs, Squares = movements,
double avg, standDev;// Standard Deviation =
public int[] sqs = {};

public Non_Intelligent(int trials, int x, int y)
{

   this.trials = trials; //Set Trials
   k.Position = new int[]{x,y}; //Initialize default position
   for (int t = 0; t < trials; t++)
   {
      Start();
      if (trials > 1)
      {
         sqs[t] = squares;
      }
      OutputTrial(t + 1);
      task.Wait(2000);
   }
   for (int s=0; s < sqs.Length; s++)
   {
      avg += sqs[s];
   }
   avg /= trials;
}
```

```
private void Start() {
   int[,] moves = k.getOptions(); //Get first options
   Random r = new Random();
   int bc=0; //Board Counter
   board[k.Position[0],k.Position[1]] = bc++;
   while (moves.Length > 0)
   {
      int oc = 0; //Option Counter (If no moves are left)
      while (board[k.Position[0], k.Position[1]] > 0)//Find untouched square based on options
      {
         int i = r.Next(8); //Get random move
         //Check if the random move is not set
         while (moves[i, 0] == 0 || moves[i, 1] == 0)
            i = r.Next(8); //Set new move
         //Set Knight to position
         k.Position[0] = moves[i, 0];
         k.Position[1] = moves[i, 1];
         oc++;
         if (oc >= 8) break; //If options are already out
      }
      if (oc >= 8) break; //End of game
      board[k.Position[0],k.Position[1]] = bc++;//Update the board movement
      moves = k.getOptions(); //Get new options
      squares++;
   }
   OutputBoard();
}
private void OutputBoard()
{
```

```
        for (int x = 0; x < 8; x++)
        {
            for (int y = 0; y < 8; y++)
            {
                if(y<7)
                    Write(board[x, y] + "\t");
                else
                    Write(board[x,y]+"");
            }
            Write("\n");
        }
    }
    private void OutputTrial(int trial)
    {
        Write("Trial " + trial + ": The knight was able to successfully touch " + squares + " squares.\n");
    }


    public void Write(String s)
    {
        using (StreamWriter sw = new
StreamWriter(Path.Combine(Directory.GetCurrentDirectory(),fn),true))
        {
            sw.Write(s);
        }
    }
  }
}
```