Project Report on

Knight’s Tour

Artificial Intelligence

Submitted By
Akash Shrestha
067BCT502

Submitted To

Department of Electronics and Computer Engineering

AKNOWLEDGEMENT

I would like to express our deep gratitude to Dr. Sashidhar Ram Joshi and Dr. Sanjeeb Prasad Pandey for the inclusion of AI project which is expected to help us in getting practical knowledge in implementation of artificial intelligence in real life application.

I am grateful for continuous suggestions and ideas for encouraging us to do a project that would address to real world problem.

Also I would like to appreciate all the friends for their ideas and help.

Akash Shrestha
(067/BCT/502)

ABSTRACT

Knight’s Tour is a classical problem on the chess board. In this, a knight is moved in sequence of squares such that it visits every squares exactly once. This problem is most often dealt as mathematical problem and a most popular problem appearing in Artificial Intelligence. Knight’s Tour is a more general form of Hamiltonian path problem.

The program asks for the initial square to begin the search of tour. Once the user supplies with the information of the start node, then the program gradually searches all the possible nodes that are likely to complete the tour.

Further, this problem isn’t only applicable to the standard 8x8 chess board. All the boards with size mXm with m>=5 is applicable to the problem. There exists at least one knight’s tour on such board. And this project is intended to find a tour out of all the possible tours.

Contents

INTRODUCTION

Chess is a two player game played on square board. A knight's tour is a sequence of squares such that all pairs of consecutive squares are adjacent in a graph and no square appears in the sequence more than once. In general, it is a Hamiltonian path problem with nodes as squares of the chessboard and the nodes are adjacent to each other only if a knight can move to the other node in single step. In this problem, we traverse the graph such that every nodes are visited exactly once and no nodes are left unvisited.

1	16	27	22	3	18	47	56
26	23	2	17	46	57	4	19
15	28	25	62	21	48	55	58
24	35	30	45	60	63	20	5
29	14	61	34	49	44	59	54
36	31	38	41	64	53	6	9
13	40	33	50	11	8	43	52
32	37	12	39	42	51	10	7

Figure 1 Knight' tour solved for 8x8 with initial node (1, 1)

This project is intended to build a program that solves a Knight's tour in any chess boards of size mXm; m>=5, for any starting nodes that the user provides. There can exist more than one such tours possible in a chess board for provided starting node. And this project finds a single solution out of many.

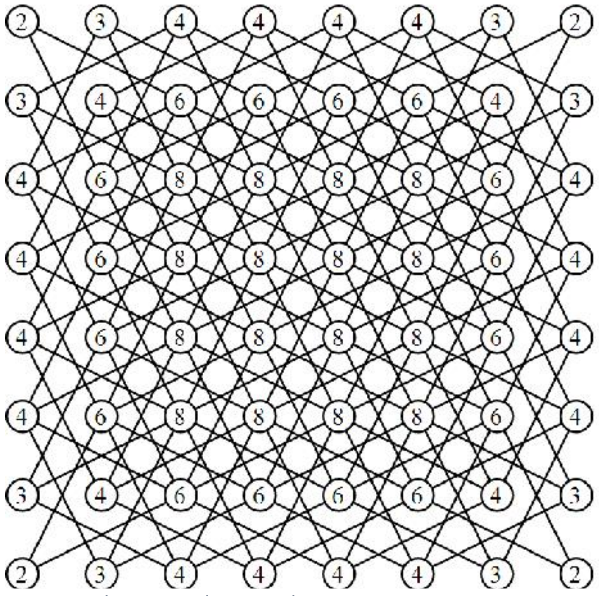


Figure 2 Chess Board as Graph

The chessboard can be viewed as a graph if we place the two nodes having knight's move adjacent to each other. For each node there can be at most 8 adjacent nodes. The chess board as a graph with corresponding degree can be shown as in above figure.

Now our problem of Knight's tour can be described as to find out a Hamiltonian path in the above graph, where the starting node is provided by user. Further, the above graph is for 8x8 chess board. The board can be of any size greater than 5.

While finding the solution, the degree of nodes gradually decreases as the tour begins since we cannot take the visited node as adjacent to any other nodes. We cannot know whether the path we've taken is correct or not, till there is a dead end in which we'll have no more nodes to travel and still some nodes remain unvisited. This condition happens at the tree depth of around 55+. Hence the space complexity will rise very much in this problem as we are dealing with tree of depth 55+ with the maximum child up to 8 for each node.

OBJECTIVE

- To practice on tree traversals, graph traversal dealing with real world problem.
- To implement possible heuristics to reduce the complexity while traversing.
- To solve general Knight's tour problem.

METHODOLOGY

Knight's Tour can be solved using Depth First Search over the graph. We can begin from the starting node as provided the user and do the following:

1. Assign move number to current node.
2. Populate adjacent unvisited nodes of the current node.
3. Push nodes to stack.
4. Pop the stack and set as current node.
5. Continue traversal from Step1 till solution is obtained. Exit after solution is obtained.

This process is very straight forward method to obtain solution. This technique will definitely find out a solution but the computational complexity will be very high. It will take much space and much time till it finds out a tour. We can make some optimization to depth first search while selecting the child out of available adjacent nodes in order to reduce the constraints. The optimization technique is named as **Warnsdorff's rule**.

Warnsdorff's rule –

For the completion of tour, what we can take into consideration is: - The child nodes of the current node with less unvisited adjacent nodes are less likely to be visited in the future. They have high chances of remaining unvisited in future, thus hindering our tour completion process. So what we can consider now is to visit those nodes having less probability of being visited in future. Thus there will less number of such hindering nodes and we can find out the tour as quick as possible (The solution obtained in a single dive into tree is much better as it will consume less time and space).

Warnsdorff's rule suggests – "***Always move to an adjacent, unvisited square with minimal degree***".

This doesn't mean that there would be no tour unless we follow Warnsdorff's rule. There would be but we'll have to expend more complexities for finding a single tour.
This also doesn't mean that there would always be a possible tour if we go on following the adjacent, unvisited square with minimal degree. This is just a thing that we can consider to optimize our search process.

A snippet that appeared on a paper published by Sam Ganzfried says –

In a 1996 REU paper Squirrel [8] performed 100 trials of this algorithm for all board sizes from 5 to 400; the conclusion was that the algorithm is quite successful for smaller boards, but the success rate drops sharply as board size increases. The algorithm produces a successful tour over 85% of the time on most boards with m less than 50, and it succeeds over 50% of the time on most boards with m less than 100. However, for m > 200 the success rate is less than 5%, and for m > 325 there were no successes at all. These observations suggest that the success rate of Warnsdorff's random algorithm rapidly goes to 0 as m increases.

But what we want is to find out at least a tour for all possible cases ie. Algorithm for 100% cases of all varieties of board size. We can actually mix up the Warnsdorff's rule and the original DFS idea to make it work on all the cases. The algorithm can be described as follows:

1. Assign move number to current node (Start node will have move number 1).
2. Populate adjacent unvisited nodes of the current node. Say the arrays of nodes thus populated be named as **childNodes,** and each node in array be refered as **childNode.**
3. Sort childNodes on the basis of adjacent, unvisited nodes each childNode have.
4. Push nodes to stack such that smallest of the above sort lies that top (so as to pop the minimal node at first).
5. Pop the stack and set as current node.
6. Continue traversal from Step1 till solution is obtained. Exit after solution is obtained.

The above algorithm will find out a tour if there exists a tour. The algorithm tries to follow Warnsdorff's rule till it can traverse down the tree. When it meets a dead end following the rule, then it pops the stack content, disobeying the rule and continues the same rule in the new sibling node in search of the solution. This algorithm is a mixture of DFS and Warnsdorff's rule. The flowchart corresponding to above description:-

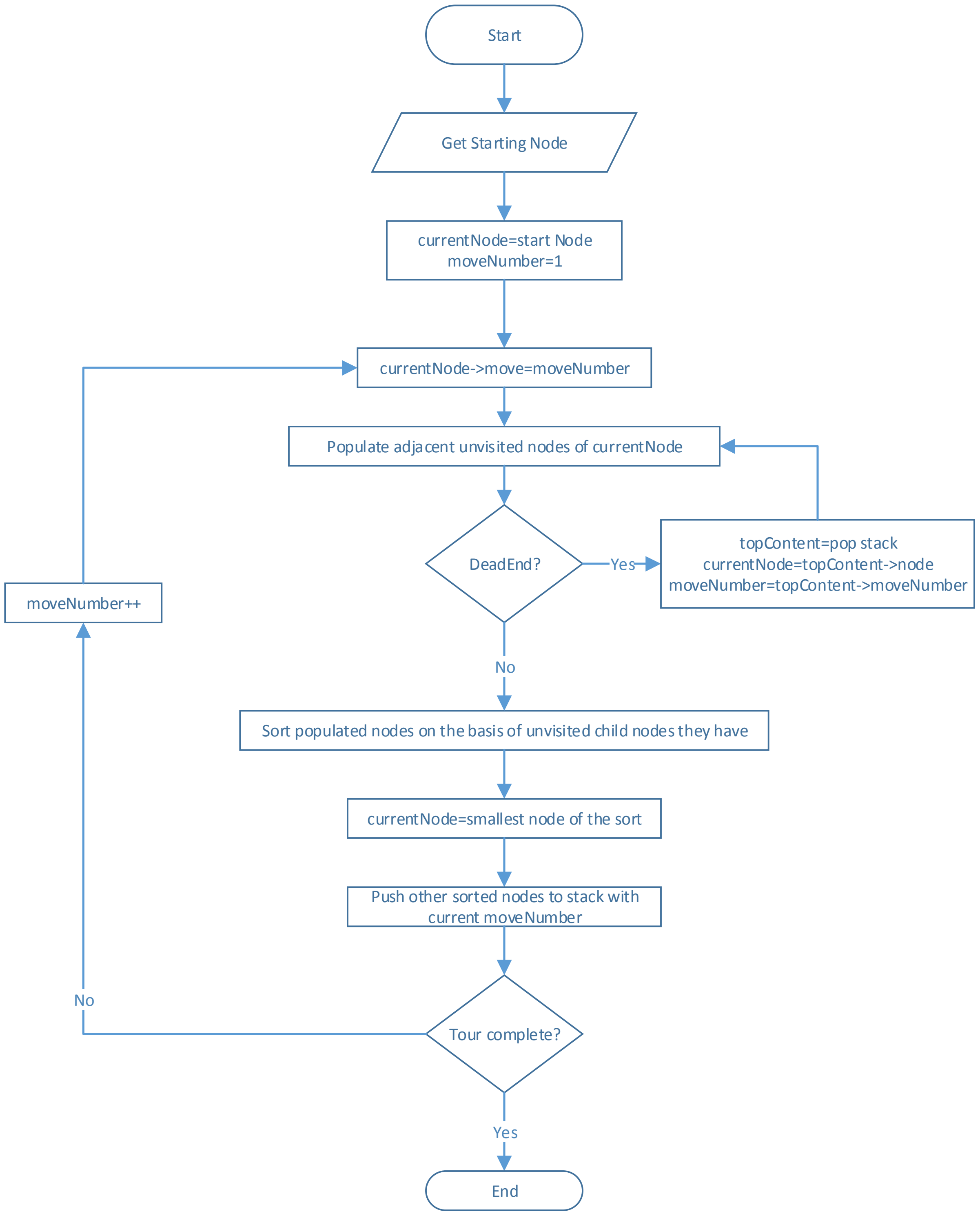
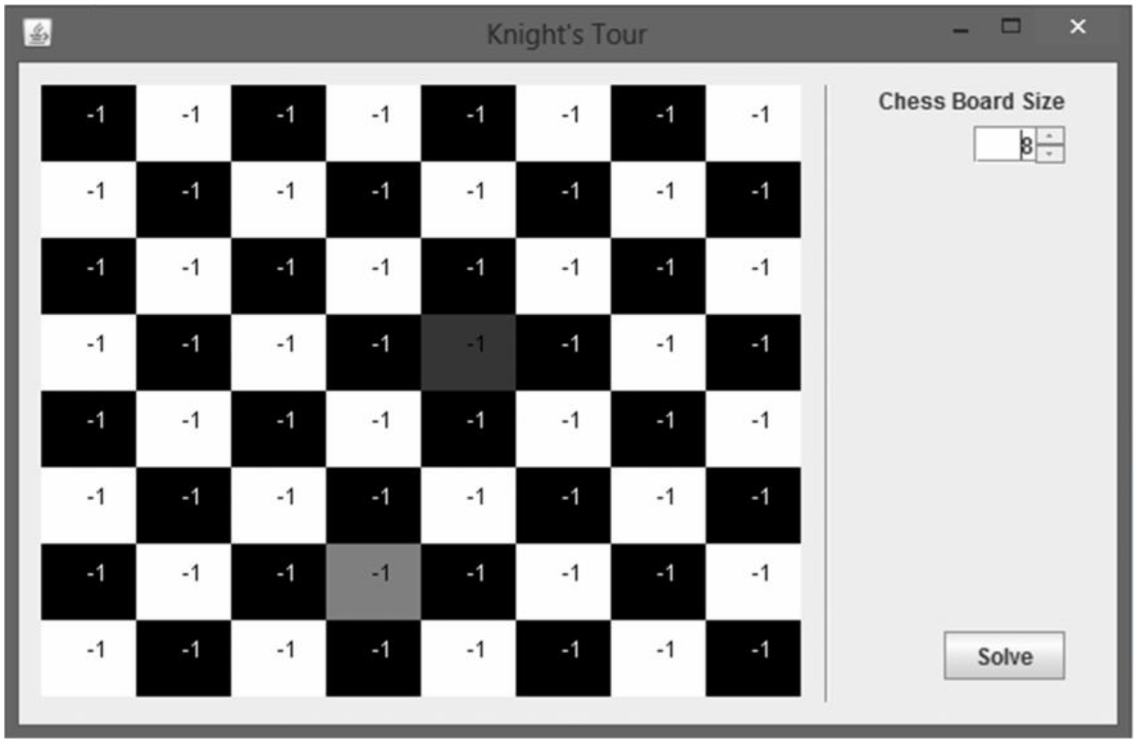

Start
Get Starting Node
currentNode=start Node
moveNumber=1
currentNode->move=moveNumber
Populate adjacent unvisited nodes of currentNode
DeadEnd?
Yes
topContent=pop stack
currentNode=topContent->node
moveNumber=topContent->moveNumber
No
Sort populated nodes on the basis of unvisited child nodes they have
currentNode=smallest node of the sort
Push other sorted nodes to stack with current moveNumber
Tour complete?
No
moveNumber++
Yes
End

Figure 3 Flowchart of solving technique

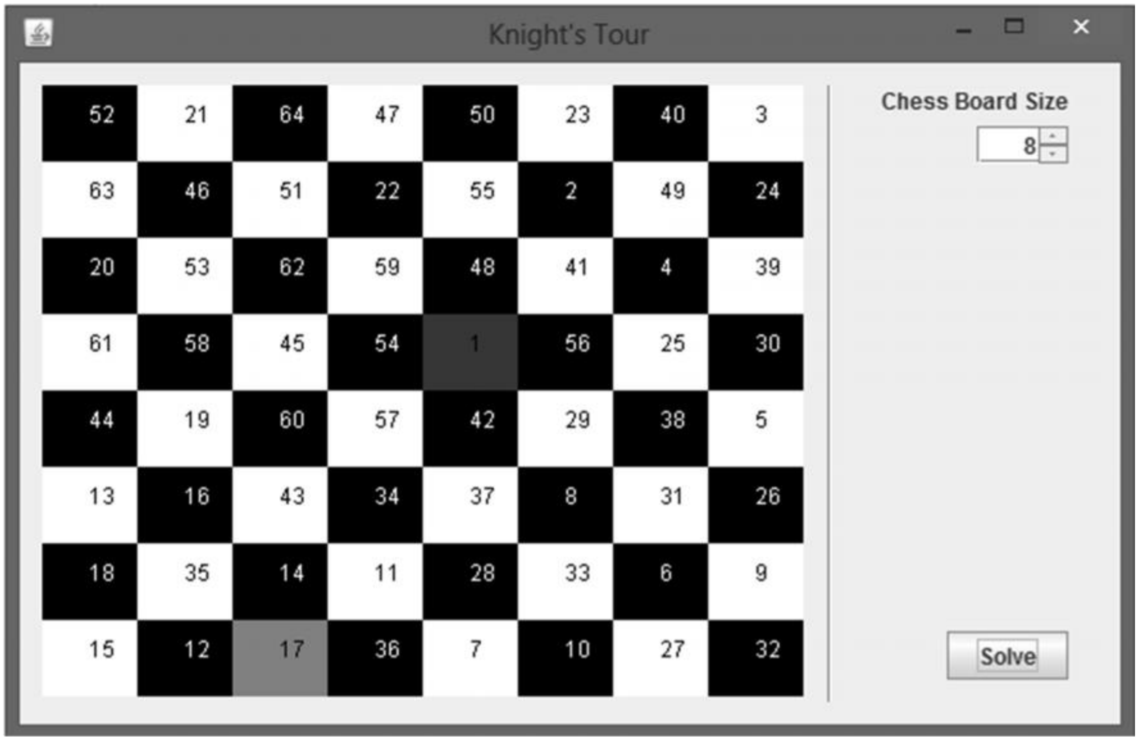
OUTPUT

1. Initial board with starting square selected (8x8)



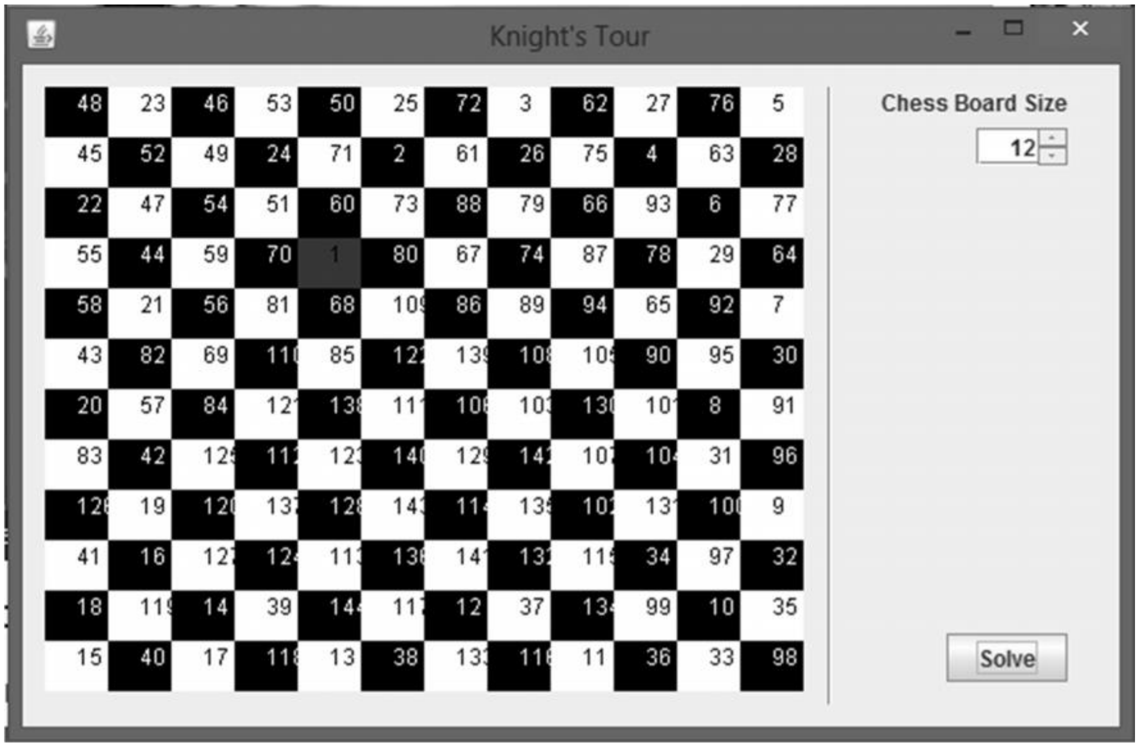
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

2. Solved chess board (8x8)



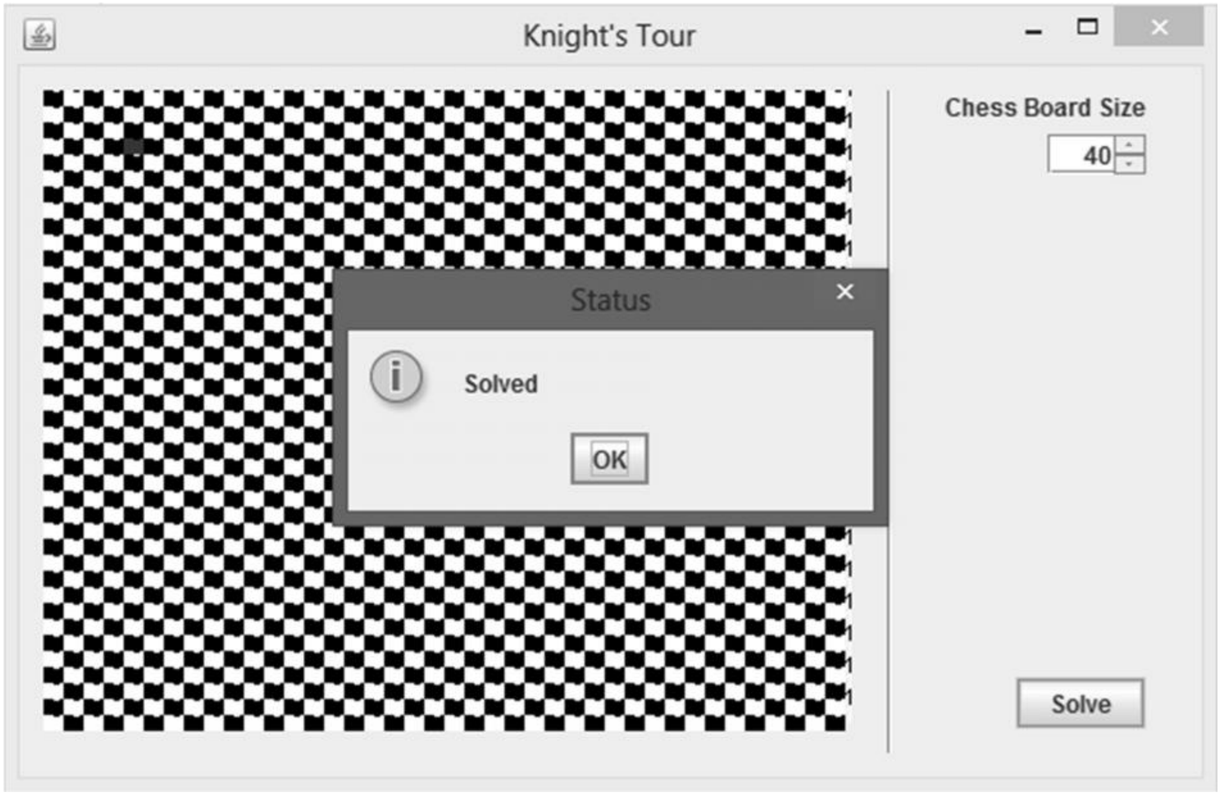
52	21	64	47	50	23	40	3
63	46	51	22	55	2	49	24
20	53	62	59	48	41	4	39
61	58	45	54	1	56	25	30
44	19	60	57	42	29	38	5
13	16	43	34	37	8	31	26
18	35	14	11	28	33	6	9
15	12	17	36	7	10	27	32

3. Different sized solved board (12x12)



48	23	46	53	50	25	72	3	62	27	76	5
45	52	49	24	71	2	61	26	75	4	63	28
22	47	54	51	60	73	88	79	66	93	6	77
55	44	59	70	1	80	67	74	87	78	29	64
58	21	56	81	68	109	86	89	94	65	92	7
43	82	69	110	85	122	139	108	105	90	95	30
20	57	84	121	138	111	106	103	130	101	8	91
83	42	125	112	123	140	129	142	107	104	31	96
126	19	120	137	128	143	114	135	102	131	100	9
41	16	127	124	113	136	141	132	115	34	97	32
18	119	14	39	144	117	12	37	134	99	10	35
15	40	17	118	13	38	133	116	11	36	33	98

4. Higher sized solved board (40x40)



Solved

Refrences

1. http://www.cs.cmu.edu/~sganzfri/Knights_REU04.pdf
   Paper by Sam Ganzfried