CSCI 651 - Algorithm Concepts

Knight's Tour Problem

A java based solution for Knight Moves in chess game

Purushotham Tumuluri
Id: 0726386
ptumulur@nyit.edu

Chandra Lekha Chavva
Id: 1081063
cchavva@nyit.edu

Upasana Patel
Id: 1030860
upatel23@nyit.edu

Chamundeswari Nune
1080550
cnune01@nyit.edu

Abstract- The knight's tour problem is the mathematical problem of finding a way for knight to cover all the squares in chess board in a way that each square is covered only once.
In this project we gathered all information about knight's tour problem. This knight's tour problem is also a Hamiltonian path problem or Hamiltonian cycle problem, and our aim is to develop a program for legal moves of the knight.

Index Terms— knight Tour, squares, chess board

I. Introduction

Here we focused on knight legal moves, as we are familiar with knight, (also known as horse) is a piece in the game of chess. It does not move in a straight line but makes L-shaped moves to reach an empty square on a chessboard. There are only two legal moves

- Two squares in the x-direction and one square in the y-direction
- One square in the x-direction and two squares in the y-direction

There are two types of tours, Closed tour and Open tour:

Closed Tour [1]: By figure 1 we can say the square with number 64 can be joined with square number 1 using one of the knights L shaped moves. This makes one of the loop able configuration and we call this as Closed tour.

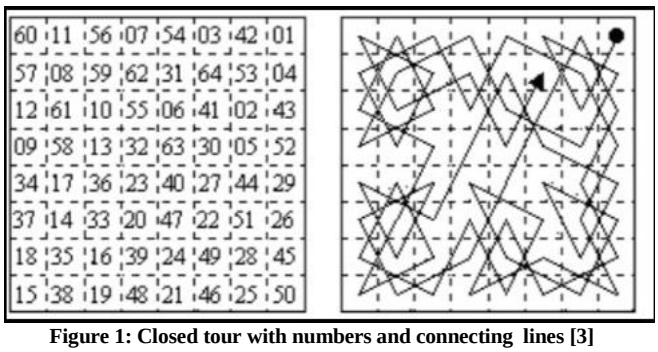

Figure 1: Closed tour with numbers and connecting lines [3]

Open Tour [2]: Considering knight legal move, Figure 2 depicts that end point square number 64 does not connect with the starting point 01. We call this as Open tour.

34	47	22	11	36	49	24	01
21	10	35	48	23	12	37	50
46	33	64	55	38	25	02	13
09	20	61	58	63	54	51	26
32	45	56	53	60	39	14	03
19	08	59	62	57	52	27	40
44	31	06	17	42	29	04	15
07	18	43	30	05	16	41	28

Figure 2: Open Tour with numbers

In this paper, we tried to find an optimal solution for the knight moves using Java by implementing Brute-Force, Divide and Conquer algorithm.

The rest of this paper is organized as follows. In section II, the related work on Methods to solve the Problem. Section III describes the tools and technologies we used. In section IV inputs the algorithm used in our project, section V and VI depicts the flow of our project and user options respectively. In section VII we inserted our execution result screen shots. In section VIII, IX, X challenges, conclusion and future work is inputted.

II. Related Work

In past years, many researchers did various studies to find the optimal solution for the knight tour problem. Few of them are,

A. Euler's Method

Euler made an attempt in 1759 to find solution for the knight tour problem, this is one of the earliest attempt. In his method [3], Euler followed a way that the Knight travels all the squares on a chess board where it visits every square only once and return to the originated square. To obtain this he defined a circuit named Euler Circuit for the knight that

traverses every edge of the graph exactly once, and ends at the same node from where it began [4].

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

Fig 3 Euler`s First example of Knight`s Tour

B. Warnsdorff's Rule:

a) Always move to an adjoining, un-visited square with minimal degree [5].

According to this rule Knight should move to an unvisited square that is adjacent with minimal degree of change. This is like a logical rule one has to follow, as lower degree squares have fewer neighbors leading to less chances of visiting them, which are not visited in the path. The rule should be implemented mandatorily as every square cannot be visited for example if there is a square with degree 0 will not be visited in order to visit then this rule should be implemented. In the similar fashion as mentioned in the above illustration if any degree 1 square is missed then there will only be one chance to visit the respective square (such a square is last square to visit). There are always chances of deviating from the rule so it is better to deviate in the starting rather than last as there will be less number of chances to avoid failure.

b) It is impossible to have three mutually adjacent squares on a chessboard [5].

According to the current method three squares have to be mutually adjacent to each other respectively. For better understanding consider the following example. There is a square in column j then as per the rule the adjacent square should be mutual so it is j+1 or j-1 hence in every case the square should not be adjacent so it should be one row or column away. This is leading to conflict so many of them started to find a appropriate solution by using this rule.

c) It is impossible for a knight's tour to deviate from Warnsdorff's Rule in the last four moves [5].

When Warnsdorff's Rule is applied to knight tour then there is chance of deviation from the rule in last four moves. "An author Pohl proposed applying Warndsdorff's standard algorithm, using a sort of backtrack procedure if it fails. Pohl also considered ways of applying Warnsdorff's rule in order to find Hamiltonian paths of more general classes of graphs.

The Eulers and Warnsdorff's rule helps the user or the developer to find a better solution by using objectives followed by them.

C. Magic Square:

A Magic square is a square array of numbers such that each row and column and the two main diagonals sum to the same number [6].

For instance if we perform a knight tour with the help of numbering each square. This end up with the sum of all the columns and rows add to 260 as per below figure.

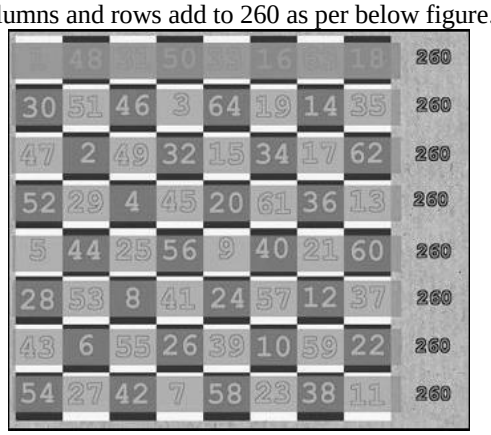

Fig: Magic square example [2]

III. Tools and Technologies

A. Tools

- Eclipse mars

B. Technologies

- Java
  Swings, AWT

IV. Algorithm Used

A. Brute-Force Algorithm:

In this algorithm we will go through all possible solutions extensively [7].

1. Start.
2. If knight at i=1, j=1 position.
3. Moves = [j+1] [i+2] or [j+2] [i+1].
4. Knight move continuous L-shaped moves.
5. Finally if the last move reach the start position closed tour.
6. Otherwise open tour.
7. Count Moves.
8. End.

Example of Brute-Force Algorithm in our project
This is the open Knight's Tour found by the algorithm in a 5×5 board, starting from position[0,0] [7]:
[0,0][1,2][2,4][4,3][3,1][1,0][2,2][0,3][1,1][3,0][4,2][3,4][1,3]
[0,1][2,0][4,1][3,3][1,4][0,2][2,1][4,0][3,2][4,4][2,3][0,4]

This is [m*n] form where m is row and n is column.

B. Divide and Conquer Algorithm

Parberry introduced a divide-and-conquer algorithm that can generate closed knight's tours on n×n or n×(n+2) boards in linear time (i.e. $O(n^2)$) for all even n and$n \geqslant 10$, and closed knight's tours missing one corner in linear time if n is odd and greater than 4 [8].

Procedure we followed in divide and conquer,

1. Start
2. Divide the chessboard into more than two halves.
3. Solve individual chessboards by Knights Legal Moves (L -shaped).
4. Combine all individual boards.
5. Final Graph is drawn by linking each move.
6. End

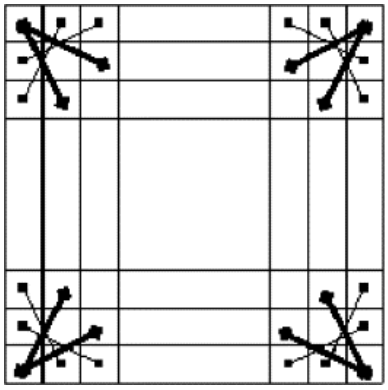

Fig 4 Divide and conquer graph[9]

In this method we break the chess board in four equal parts for instance if we take a 8*8 chess board the board was divided into four 4*4 containing 16 squares each and then we make the knight to traverse all the squares in that part. We will create links between all parts so that the knight completes the tour.
So here the knight divides the boards and conquer each cell in the board.

V. SYSTEM DESING

A. Work Flow Diagrams

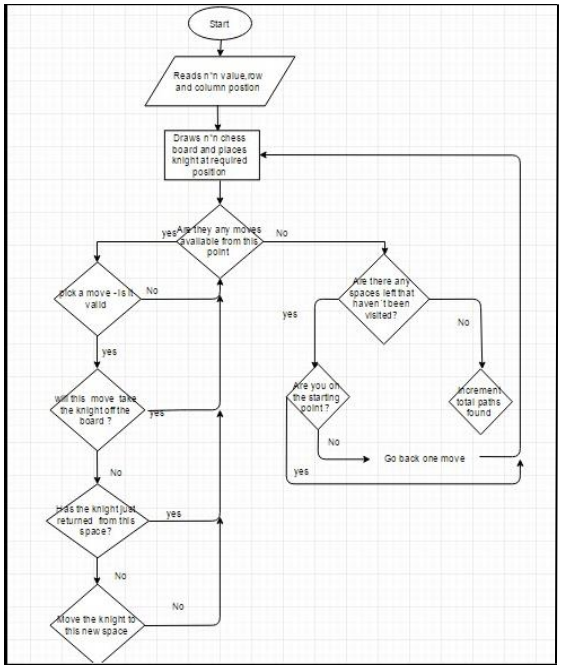

Figure 5: Flow diagram of the process [10]

Firstly we need to enter the size of the chess board; it reads the value given by the user in n*n format and draws a chess board as per user requested size. Then it check the condition whether any moves available from the point, if available the knight picks a move and checks whether it is a valid move or not? Then if that move is valid it will check the condition that this move takes knight off the board? If yes it checks for next available move as seen the figure. If no, it check the condition that the knight returned from this space? Now if it not returned from space the knight searches for the new space. If any of these condition is not true the knight moves to the starting point and checks whether any move available or not as shown in the figure.

On the other side if the knight does not find any available moves it check whether any spaces left unvisited, now it check for the condition whether it is on the starting point or not if it is not the starting point the play stops, if not on the starting point it goes one step behind.

VI. USER INTERFACE SPECIFICATIONS

A. SELECT COLUMN AND ROW

By providing this functionality of selection, user makes their own chessboard and customizes the chessboard.

B. POSITION

By using drop down option user can put their knight in any position. In other words, user decide their knight position by own.

C. MOVEMENT

Movement of the knight is all depend on how we place the knight, in which column, row user put the knight. Movement of a knight is based on the algorithm used.

D. SOLUTION

Once the user click on the play button, the backend process come to the action shows all the possible moves, here the act as shown and our real work get in to result.

E. FINAL/STOP

After the user clicks on stop button, game will be on final mode and to stop the knight moves or to stop the show. User can start new tour and play the show again.

F. Flow of the user specifications

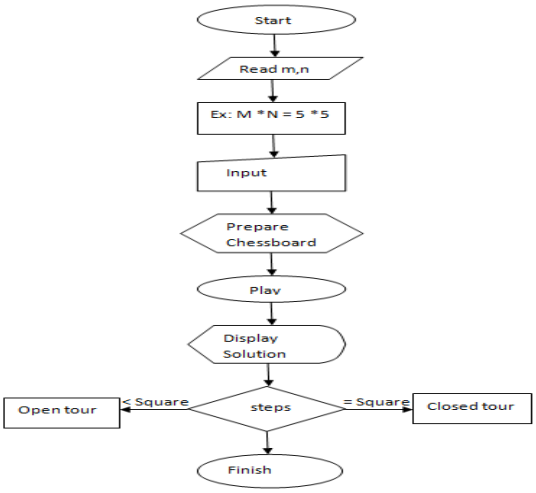

Figure 6: User flow specifications

VII. Execution Result

Below are the few screenshots that show our execution screens and output results.

The part A projects our initial window when we run our cide, part B shows our complete results where the knight covers all the squares only once. This part also shows that we achieved our goal. Part C,D shows other functionalities and part E displays our no complete tour, where the knight didn't meet our goal.

A. Initial Window

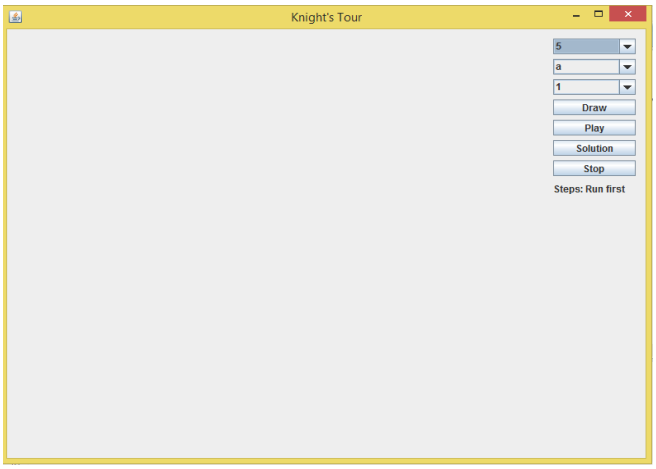

Figure 7: Screen appeared once the code is executed

This is the Window that pops when we execute the project. We see that there are three drop down Boxes and three buttons. The first drop down box allow us to select the size of the chess board from 5*5 to 8*8. The second, third dropdown boxes represents the columns and rows. We designed four buttons which allow us draw the chess board as per the selected size, play button plays the show or the tour, solution button gives us the solution for the tour and finally stop button pauses the trip.

B. Complete Tour

The squares of chess board are colored with black and white. The knight is marked with red color and the every square covered by the knight is represented with yellow color as shown in below figures.

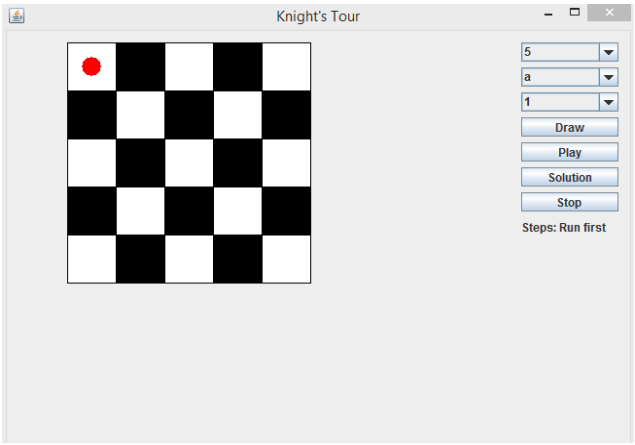

Figure 8: Screen appeared once the above option were selected and drawn

In the figure 8 we selected a chess board of size 5 and we placed the knight at column 'a' and row '1' as shown. The knight starts its trip from the placed position.

Solutions for various sizes of boards:

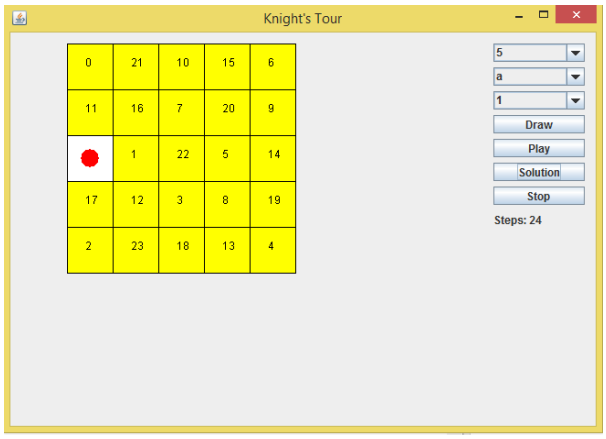

Figure 9: Solution for 5*5 board

The figure 9 shows the steps and final solution for the 5*5 size board once you click on the Solution button.

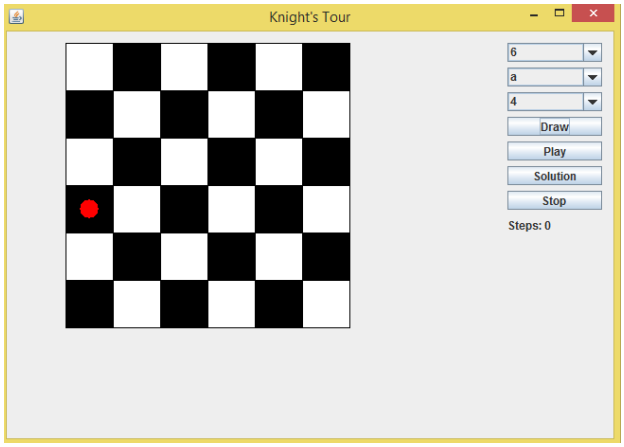

Figure 10: Checkboard of size 6*6

In the figure 10 we placed the knight at column ‘a’ and row ‘4’ in a 6*6 chess board.

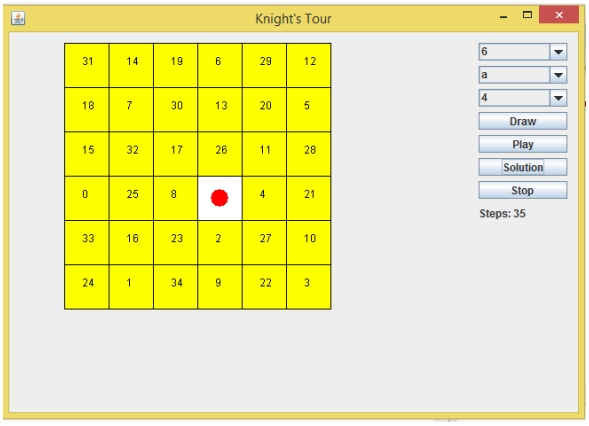

Figure 11: Knight Tour for 6*6 board

The figure 11 shows solution for a knight trip on 6*6 chess board.

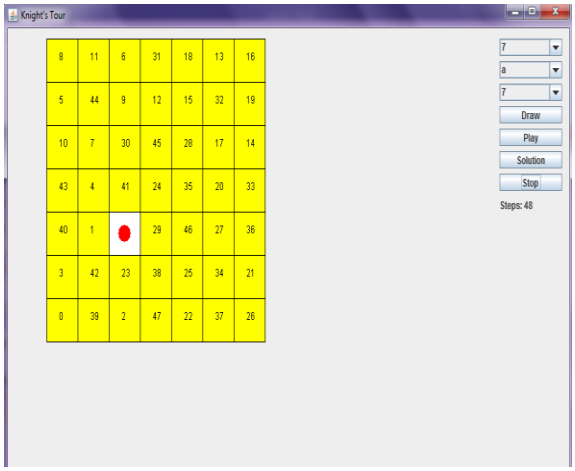

Figure 12: Solution for a knight trip of 7*7 chess board

The figure 12 demonstrates the output of a knight trip on 7*7 chess board

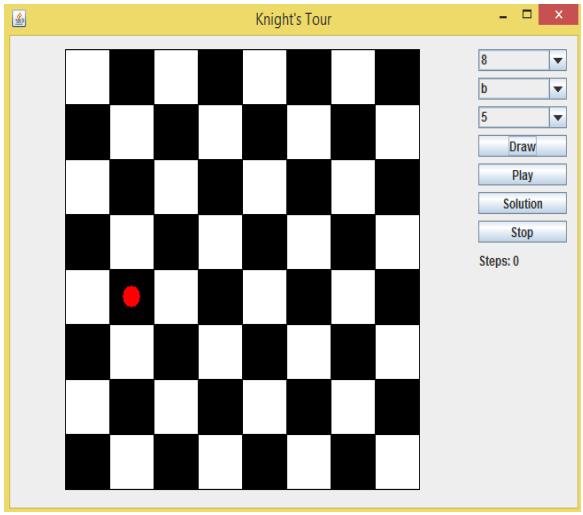

Figure 13: Knight placed on board of size 8*8

We draw a chess board of size 8*8 and places the knight on column b at row 5 as shown in figure 13.

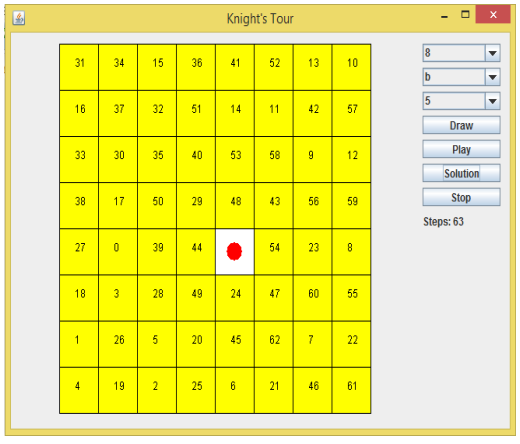

Figure 14: Knight Tour solution for the 8*8 board

The figure 14 shows the solution for the 8*8 chess board.

C. When Knight is placed outside the board.

If we draw a board of size m*m and places the knight outside the board, the trip will not start because the knight cannot detect the rows the and column required for the movement. Please refer below figure 15.

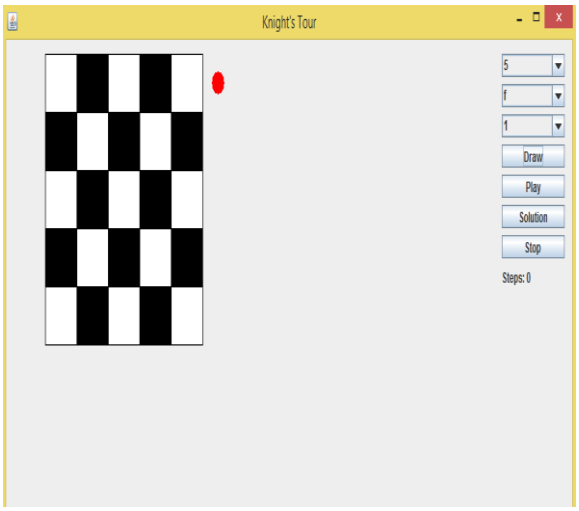

Figure 15: Knight is placed outside the chess board

D. Stop Functionality

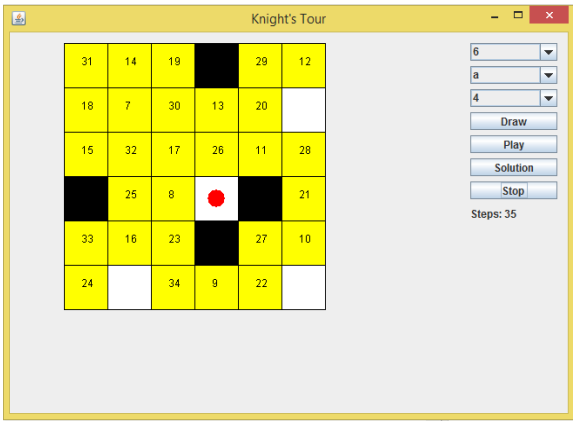

Figure 15: The tour of the knight is stopped in between the tour.

By clicking on stop button we can make the knight to stop the operation. To enable stop functionality first we need to first play the show before clicking on solution as shown in the figure 15.

E. No Complete tour

In few cases the knight does not satisfy our goal.
Below are the few cases we found that our knight does not complete the tour by covering all the squares. For instance on a 5*5 chess board when a knight is placed at column c and row 2 the knight starts it tour and reaches the maximum legal moves and stops at the corner of the chess board as seen in the figure 16. In this figure 16, the knight was not able to make any legal moves further and stops.

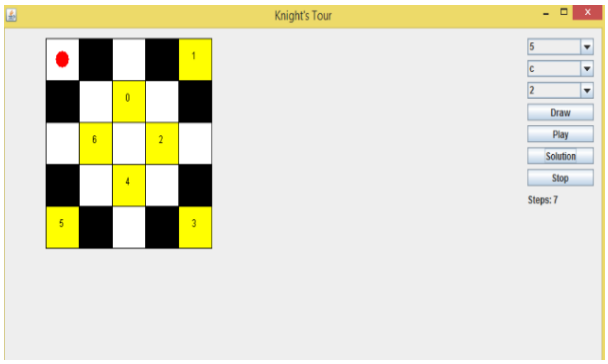

Figure 16: No complete tour on 5*5 chess board

Below is the one more example for the incomplete tour when knight is placed on 7*7 chess board at column d and row 7

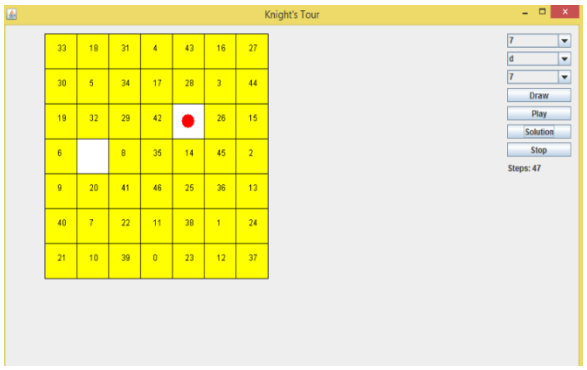

Figure 17: No complete tour on 7*7 board.

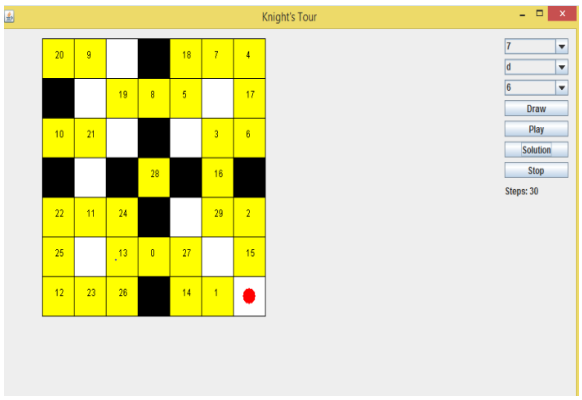

Figure 18: No complete tour on 7*7 when knight is placed at column d and row 6

VIII. Challenges

Solving the Knight's Tour math problem involves moving the knight in the chessboard such that each square is visited exactly once.

Diving the chess board and linking the all the boards in such a way that knight covers the entire square in divide and conquer algorithm.

Number ways a knight can traverse in a closed are in millions and in open tour we cannot even say.

IX. Conclusion

In our work we mainly concentrated on finding a solution for the knight mathematical tour problem using various methods. It was helped us in understanding various algorithms and researches made on this problem, at the end we conclude that Warnsdorffs rule helped us to find the optimal solution.

X. Future Work

We are planning to use more simple and effective algorithms to find a best solution for the problems.

There are few other cases that the knight not able to perform the complete tour we are planning to find the solution even for it.

XI. Acknowledgment

This paper is made possible through the help and support from everyone, including: professor, family, and friends. Especially, please allow me to dedicate my acknowledgment of gratitude toward the following significant advisors and contributors. The product of this paper would not be possible without all of them.

XII. References

[1] J. Erde, "The closed knight tour problem in higher dimensions,"25-Oct-2012.[Online]. Available at: http://www.combinatorics.org/ojs/index.php/eljc/article/viewFile/v19i4p9/pdf

[2] C. N., "Can you Solve the Knight's Tour Math Problem?," YouTube, May-2012. [Online]. Available at: https://www.youtube.com/watch?v=9fSFC00ZKPg [Accessed: 08-May-2016].

[3] "Introducing Knight's Tours," Introducing Knight's Tours.[Online].Availableat:http://www.mayhematics.com/t/1n.htm

[4] The Knight's Tour by Colleen Raimondi http://academics.smcvt.edu/jellis-monaghan/combo2/Archive/Combo%20s03/special%20topics%2003/The%20Knights%20Tour.ppt

[5] A NEW ALGORITHM FOR KNIGHT'S TOURS by SAM GANZFRIED [Online]. Available at : https://www.cs.cmu.edu/~sganzfri/Knights_REU04.pdf

[6] "DISTRIBUTED PROBLEM SOLVING IN MOZART (MAGIC KNIGHT'S TOUR) by Vivek Singh. . [Online]. Available: https://www.comp.nus.edu.sg/~henz/students/singh_thesis.pdf

[7] "Knight's Tour brute force algorithm example," SIGQUIT, Dec-2010. [Online]. Available at: https://sigquit.wordpress.com/2010/01/13/knights-tour-brute-force-algorithm-example/

[8] "An efficient algorithm for the Knight's tour problem" by Ian Parberry. [Online]. Available at: https://pdfs.semanticscholar.org/927e/dbaa256301f500e8b3fcd52eaa6f0cc2c768.pdf.

[9] "The Knight's Tour," Gordon Tumilty, 2010. [Online]. Available at: https://gtumilty.wordpress.com/2010/02/25/the-knights-tour/

[10] "Optimal algorithms for constructing knight's tours on arbitrary n×mn×m chessboards," Optimal algorithms for constructing knight's tours on arbitrary n×m chessboards. http://www.sciencedirect.com/science/article/pii/S0166218X04003488

[11] "Knight's tour," Wikipedia. Available at: https://en.wikipedia.org/wiki/Knight%27s_tour