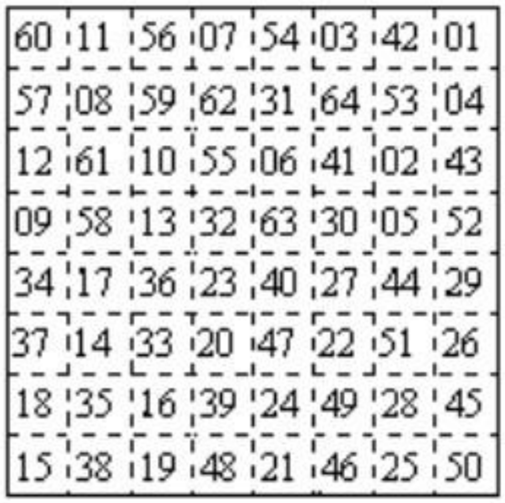
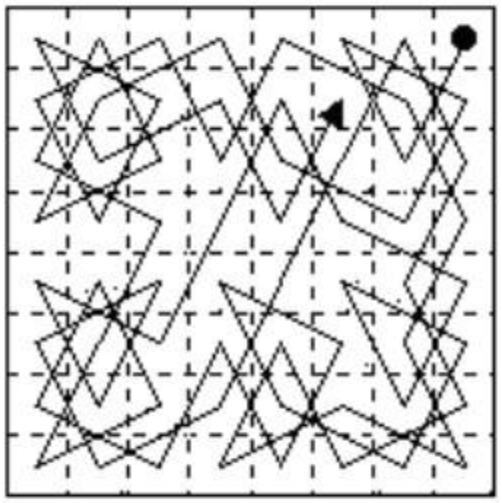
Project 13- Knight's Tour (Part 1)

We are going to be writing one of the quintessential Computer Science programs. We will begin with a typical 8x8 board (2D array of ints of course). Your program will try to find the most amount of moves that a knight can make based on a starting position. Even though I will be using an 8x8 program as an example throughout, make sure you can write code that can do a knights tour on any **square** board. Your knight should keep going until it can't anymore. Here are a couple of other guidelines:

- Each "square" in **board** is initialized to **-1**, which we'll represent using a constant called **UNVISITED**.
- Whenever the knight visits a particular square on the board, that value turns from UNVISITED to the move number on which the knight visited, meaning that at the end, the 2d array might look something like what's on the left in this image:



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



The eight possible moves are described by the chart below. For example, a move of type 0 as shown below represents moving 2 spaces horizontally followed by 1 space vertically. The eight moves may be described by a two-dimensional array **moves** as follows: {{2,-1},{1,-2},{-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1}}

Note: A move as a 1-d int array of length 2. A "move" is of the form {rowMove, colMove}, if you have multiple moves add a row with two columns: {{rowMove1, colMove1}, {{rowMove2, colMove2}}

Knight’s Tour Interface

Start by writing an interface called **KnightsTour.java**. Note: After the first-class period, I will be going around to check to make sure your interface is up to par based on the guidelines (Wednesday, October 31st).

1. In it, you should specify an int constant UNVISITED that holds the value -1, which will indicate that a square has not yet been visited. Remember, interface variables are static, final constants.
2. You should also specify an int[][] called MOVES that holds all the possible moves: {{2,-1},{1,-2},{-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1}};
3. Then, specify (again, in your interface KnightsTour) all the methods needed to:

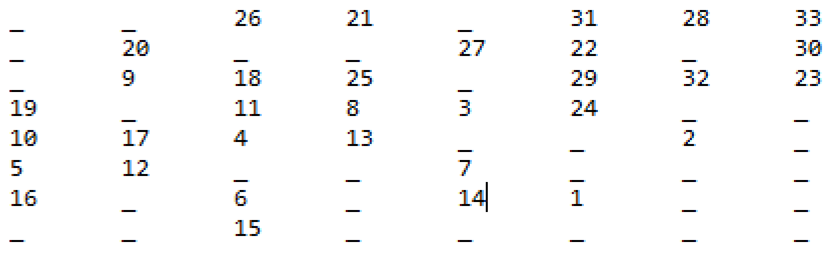
    - set up / reset the simulation
    - start the tour (either at a specified or random position)
    - make a single step (note: the direction/destination are not specified)
    - get state variables (current row/col, number of visited squares, array of possible moves from the current square).
    - get the board state as an int[row][col], where each int is either UNVISITED or the step number in which the knight visited that square
    - get the board state in a single printable String, where each row is printed on a new line

Writing Knights Tour

Now write BasicKnightsTour, a class that implements the KnightsTour interface. This will use a dumb procedure for choosing the next move -- perhaps random selection or "always choose the lowest numbered move" or something of the sort. I leave it up to you. Document your methodology clearly.

Logistics:

1. You should likely start by thinking about what instance variables are inherited through the interface and what additional instance variables need to be added to make this program work.
2. Then write your constructor, getters, and the method to churn the board's state into a String representation. If a space is unvisited, represent it in String form not with -1 but rather with an underscore. Separate squares with tabs. Example:



```
_       _       26      21      _       31      28      33
_       20      _       _       27      22      _       30
_       9       18      25      _       29      32      23
19      _       11      8       3       24      _       _
10      17      4       13      _       _       2       _
5       12      _       _       7       _       _       _
16      _       6       _       14      1       _       _
_       _       15      _       _       _       _       _
```

3. After that, you should write getPossibleMoves. Remember, this method should return a 2D array containing only valid moves. That means that if there are only 3 valid moves, the array should have **3 rows of 2 elements each (row, column)**.
4. Then write makeMove(int[]), which takes a single valid "move" and applies it to the board, returning a useful 1d array with two elements containing the resultant position afterward.
5. Then write makeMove(), which calls makeMove(int[]) but with a single valid move as its parameter. You may decide how makeMove() decides which int[] move to pass to makeMove(int[]). Perhaps it chooses a random valid move. Or it always chooses the first valid move. Or perhaps it always chooses the random move leading closer to the center. It's up to you. (I chose randomly.)
6. You should write a main method that will instantiate an object of BasicKnightsTour and calls its start (very similar to what we did in 2048).

****If you like some ideas on how to judge the next move look to the next page****

Ideas that can be used to implement this:

- Random selection
- Choose the lowest valid move number
- Choose the move closest to the center
- Choose the move closest to the edge
- Fill in clusters
- Fill in one half, then the other
- Circular pattern --> spiral inward
- Working backward
- Evenly spacing placements
- Fill in quadrants
- Backtracking a la Eight Queens
- Choose the move with the most possible moves