Data Structures
COMP 2100
Fall 2022

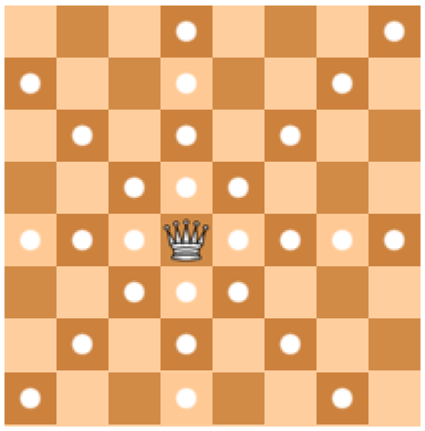
Lab 5: Queens and Knights

In this lab, based on the example of solving the Maze search using recursive backtracking, you will solve **one of the following two** programming projects: 8-Queens or Knights-Tour.

Eight Queens

Develop an application to place eight queens on a chess board such that no queen is in jeopardy from another queen.

A chess board consists of a square grid with eight rows and eight columns, in which each square can contain at most one game piece. According to the rules of chess, a queen can attack at any distance in the same row, column, or either diagonal (see diagram).



Your application should display a representation of the chess board after the placement of the eight queens.

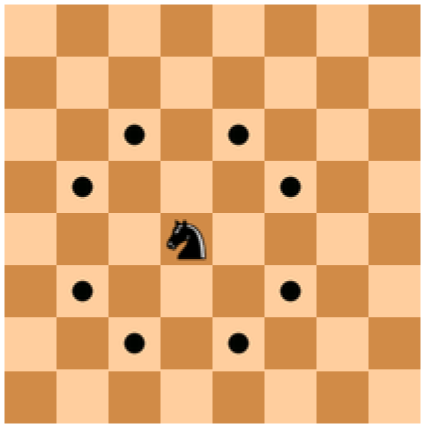
Hints:

- There must be exactly one queen in each row, and exactly one queen in each column of the chess board.
- Your application does not require any input. You can hard-code the initial placement of the first queen.
- A valid position is one that is not under attack from any currently placed queens.
- The QueensIterator constructor should begin with row 0 of the next column, and advance down that column with each call to next().

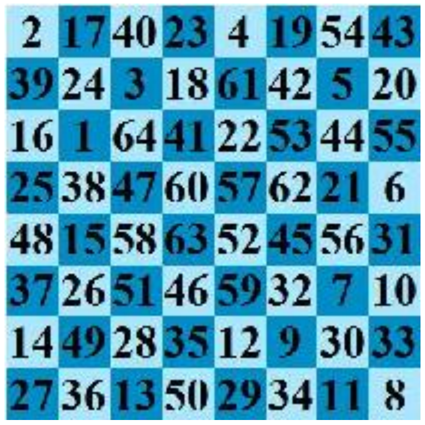
Knight's Tour

Develop an application to find a path that a knight could take on a chess board that would traverse every square exactly once.

A chess board consists of a square grid with eight rows and eight columns, in which each square can contain at most one game piece. According to the rules of chess, a knight can move in an L-shaped pattern (see diagram).



Your application should display a representation of the chess board after the path of the knight has been determined (just number the squares in the order they would be visited -- see diagram for a sample solution).



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

Hints:

- The user should be prompted for the starting location of the knight.
- Test your application only with the starting position of (0, 0).
- A valid position is one that is within the bounds of the chessboard and that is not on the current path.
- The KnightsIterator constructor should generate each of the eight positions shown in the diagram above.

To Do

You can begin by downloading the Lab05.zip archive, which contains `Application.java`, `Backtrack.java`, and `Position.java`. These files an be used as is, without modification. You will need to create two new classes, Queens.java and QueenDriver.java, or Knights.java and KnightDriver.java, depending on which problem you are solving. These classes can be modeled after the Maze program. You will also need to develop an iterator for your class (as is done in Maze.java).

Remember that individual work is expected on the lab projects!

To Submit: Email your source files as attachments to dstucki@otterbein.edu

(Board diagrams taken from Wikipedia and Wolfram Demonstrations Project.)