Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2024

Project 1 - The Trapped Knight

Due Friday, February 9

Description

In this project, you will write a computer code to generate a particular sequence of numbers described in the following YouTube video: The Trapped Knight

Begin by watching the video and make sure you understand exactly how the sequence is generated. Then continue to implement the code in the 3 parts described below.

Part 1 - Initialize the board

We will store the chess board in a 2d-array of integers. The size of the board is $(2n+1)$-by-$(2n+1)$, for a given integer n. This means the board extends from the center square by n steps in all directions.

The first step is to initialize the board by filling it with the integers described in the video. Finish the implementation of the function definition in the cell below such that it returns this "spiral pattern" for any given input parameter n.

An example is given below: for the following input

```
board = initialize_board(3)
```

the correct output is

```
7×7 Matrix{Int64}:
 37  36  35  34  33  32  31
 38  17  16  15  14  13  30
 39  18   5   4   3  12  29
 40  19   6   1   2  11  28
 41  20   7   8   9  10  27
 42  21  22  23  24  25  26
 43  44  45  46  47  48  49
```

Test your function for various values of n to make sure it is correct before you continue.

Hints:

- Note that since Julia uses 1-based indexing, the center square of the array `board` is given by element `board[n+1,n+1]`.
- After the center 1 has been placed, there are exactly n "circles" of numbers of increasing radius. This is naturally implemented using a for-loop.
- In each "circle", there are 4 segments going up, left, down, and right. These are also naturally implemented using a sequence of 4 for-loops.

In [2]:
```
# Initializes the board on a [-n:n]x[-n:n] domain with spiral numbers
#
# Example: initialize_board(2) returns
# 17 16 15 14 13
# 18  5  4  3 12
# 19  6  1  2 11
# 20  7  8  9 10
# 21 22 23 24 25
#
# Inputs:
#  n     = integer size of board to allocate
# Outputs:
#  board = 2n+1 x 2n+1 integer array filled with spiral numbers
function initialize_board(n)
    board = zeros(Int64, 2*n+1, 2*n+1)
    board[n+1, n+1] = 1
    curr = 2
    for i in 1:n
        for j in 1:2*i
            board[n+i+1-j,n+i+1] = curr
            curr += 1
        end
        for j in 1:2*i
            board[n-i+1,n+i+1-j] = curr
            curr += 1
        end
        for j in 1:2*i
            board[n-i+1+j,n-i+1] = curr
            curr += 1
        end
        for j in 1:2*i
            board[n+i+1,n-i+j+1] = curr
            curr += 1
        end
    end
    return board
end
board = initialize_board(3)
```

Out[2]:
```
7×7 Matrix{Int64}:
 37  36  35  34  33  32  31
 38  17  16  15  14  13  30
 39  18   5   4   3  12  29
 40  19   6   1   2  11  28
 41  20   7   8   9  10  27
 42  21  22  23  24  25  26
 43  44  45  46  47  48  49
```

Part 2 - Simulate the walk

Next we will write the function to simulate the walk and produce the sequence. This function will take an initialized board as input, and produce a list of numbers as well as the corresponding x- and y-coordinates.

For example, the following input:

```
board = initialize_board(2)
display(board)
seq, xs, ys = simulate_walk(board);
println("Sequence = ", seq)
println("x-coordinates = ", xs)
println("y-coordinates = ", ys)
```

should produce the following correct output:

```
5×5 Matrix{Int64}:
 17  16  15  14  13
 18   5   4   3  12
 19   6   1   2  11
 20   7   8   9  10
 21  22  23  24  25
Sequence = [1, 10, 3, 6, 9, 4, 7, 2, 5, 8, 11, 14]
x-coordinates = [0, 2, 1, -1, 1, 0, -1, 1, -1, 0, 2, 1]
y-coordinates = [0, 1, -1, 0, 1, -1, 1, 0, -1, 1, 0, -2]
```

Again test your code, first using small values of n as shown above, which makes it easier to look at the results and find errors.

Hints:

- It is convenient to create another 2d-array of booleans, indicating if a square has been visited or not.
- Make sure you never allow the knight to jump outside the board. That is, the only valid positions are n steps from the center square in either direction

In [3]:
```
# Simulates the trapped knight walk on a pre-initialized board and return
#
# Inputs:
#  board    = 2n+1 x 2n+1 integer array filled with spiral numbers
# Outputs:
#  sequence = integer array containing the sequence of spiral numbers the
#  x_path   = integer array containing the x coordinates of each step of
#  y_path   = integer array containing the y coordinates of each step of

function can_visit(x, y, n, visited)
    arr = []
    for i in [(2, -1), (2, 1), (-1, -2), (1, 2), (-1, 2), (1, -2), (-2, -
        if 1 <= x+i[1] <= 2*n+1 && 1 <= y+i[2] <= 2*n+1 && visited[x+i[1]
            push!(arr, (x+i[1], y+i[2]))
        end
    end
    return arr
end

function simulate_walk(board)
    get_val(pair) = board[pair[1], pair[2]]
    n = div(size(board, 1), 2)
```

```
    visited = zeros(Int8, size(board))
    x = n+1
    y = n+1
    visited[x, y] = 1
    y_path = [x - n - 1]
    x_path = [y - n - 1]
    sequence = [get_val((x, y))]
    possible_moves = can_visit(x, y, n, visited)
    while length(possible_moves) > 0
        move = argmin(get_val, possible_moves)
        push!(sequence, get_val(move))
        x, y = move[1], move[2]
        visited[x, y] = 1
        push!(y_path, x - n - 1)
        push!(x_path, y - n - 1)
        possible_moves = can_visit(x, y, n, visited)
    end
    return sequence, x_path, y_path
end
```

Out[3]: simulate_walk (generic function with 1 method)

In [4]:
```
board = initialize_board(2)
display(board)
seq, xs, ys = simulate_walk(board);
println("Sequence = ", seq)
println("x-coordinates = ", xs)
println("y-coordinates = ", ys)
```

```
5×5 Matrix{Int64}:
 17  16  15  14  13
 18   5   4   3  12
 19   6   1   2  11
 20   7   8   9  10
 21  22  23  24  25
Sequence = [1, 10, 3, 6, 9, 4, 7, 2, 5, 8, 11, 14]
x-coordinates = [0, 2, 1, -1, 1, 0, -1, 1, -1, 0, 2, 1]
y-coordinates = [0, 1, -1, 0, 1, -1, 1, 0, -1, 1, 0, -2]
```

Part 3 - Generate the full sequence and plot the path

Finally, use your code to generate the full sequence ($n = 100$ is sufficient), output the *last* number, and plot the path by straight lines between all the visited x,y-coordinates.

In [5]:
```
board = initialize_board(100)
seq, xs, ys = simulate_walk(board);
println("Last number: ", seq[length(seq)])
```

```
Last number: 2084
```

In [8]:
```
using Plots
plot(xs, ys, line = :arrow)
```

```
[ Info: Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80]
[ Info: Precompiling IJuliaExt [2f4121a4-3b3a-5ce6-9c5e-1f2673ce168a]
```

Out[8]: