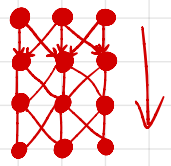
- least nodes
- least turns

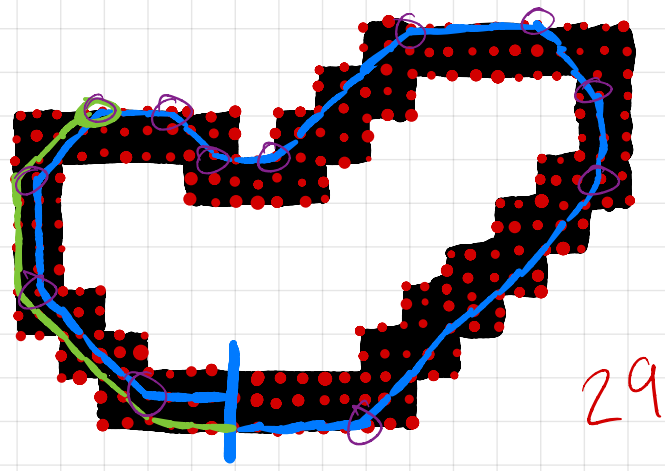


- 3 straight faster than 2 turn
- favor more straights

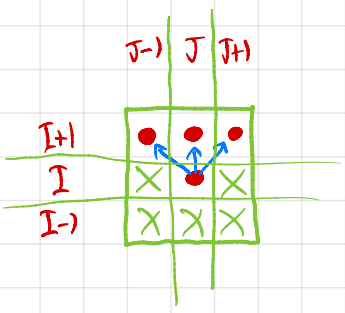
$(14 \cdot 2)+1 = 29$ 29

1	2	3	4	5	6	7	8	9	10	11	12	13	14

$(9 \cdot 2)+1$
19



$29 \times 19 = 551^2 = 303{,}601$



Need to make it dependent on previous move

focus on slope of movement

(adding up what is being added to the variables)

[I+1][J+1] = 0101

[I+1][J] = 0100

[I+1][J-1] = 0111

favor straights

one row/column will always have a 1 in it

Store previous slope
if new slope == old slope:
    distance discount on that move
else:
    distance stays the same for all

00 = no move
01 = positive move
11 = negative move
I = Up and down
J = left and right

0111 7	0100 4	0101 5
0011 3	0000 0	0001 1
1111 15	1100 12	1101 13

0s and 1s are the string format of the move.
The number is the integer version of the move

```
def getMoves(prevSlope):
    if prevSlope == 0111:
        moves = [0011, 0111, 0100]
    elif prevSlope == 0100:
        moves = [0111, 0100, 0101]
    elif prevSlope == 0101:
        moves = [0100, 0101, 0001]
    elif prevSlope == 0011:
        moves = [1111, 0011, 0111]
    elif prevSlope == 0001:
        moves = [0101, 0001, 1101]
    elif prevSlope == 1111:
        moves = [0011, 1111, 1100]
    elif prevSlope == 1100:
        moves = [1111, 1100, 1101]
    else:
        moves = [1100, 1101, 0001]
    return moves
```

Possibly make into a dictionary

```
def chooseNextPath(moves, currPosX, currPosY, xCoords, yCoords, startX, startY, currPathX, currPathY, visited)
    nextMove = []
    nextDistance = inf
    index = 0
    for move in moves:
        i = int(move[1])
        j = int(move[3])
        iNeg = int(move[0])
        jNeg = int(move[2])
        if iNeg:
            i *= -1
        if jNeg:
            j *= -1
        nextY = currPosY + i
        nextX = currPosX + j
        if nextX in xCoords and nextY in yCoords and (nextX, nextY) not in visited
            xDist = |startX - nextX|
            yDist = |startY - nextY|
            dist = xDist + yDist
            if index == 1
                dist -= 1
            if dist < nextDistance:
                nextDistance = dist
                nextMove = (nextX, nextY)
    else:
        nextMove = (None, None)
```

```
def findStart(startX, startY, moves, currPosX, currPosY, xCoords, yCoords):
    currPathX = []
    currPathY = []
    visited = [(currPosX, currPosY)]

    while (currPosX, currPosY) != (startX, startY)
        moves = getMoves(moves[1])
        currPosX, currPosY = choosePath(moves, currPosX, currPosY, xCoords, yCoords, startX, startY, currPathX, currPathY
                                                                                                                  visited)
        visited.append((currPosX, currPosY))
        if currPosX == None:
            currPosX = currPathX.pop()
            currPosY = currPathY.pop()
        else:
            currPathX.append(currPosX)
            currPathY.append(currPosY)
```

This gets general best path with holes

get best path with 2 paths
- get first path going one direction
- get second path going opposite direction

loop through second path
if node in first path:
append to final path

---

first [last: max(dist)] (coords) = half

second [last: max(dist)] = half

final path = half + half

This gets several best path with some minor hiccups