Infinite(?) Knight Tour

Bob Jansen

Terminating Knight Tour on Infinite Boards

Under what conditions does a knight tour on an infinite board end as described in this Numberphile video

The original example

Consider an infinite board where each tile is numbered from 1 to n using an anti-clockwise spiral as follows:

```
# https://github.com/bobjansen/recollections/
suppressMessages(library(recollections)) # Used later to keep track of visited squares
library(ggplot2)

board <- t(matrix(c(
    17:13,
    18L, 5:3, 12L,
    19L, 6L, 1L, 2L, 11L,
    20L, 7:9, 10L,
    21:25
  ), ncol = 5L))
board
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   17   16   15   14   13
[2,]   18    5    4    3   12
[3,]   19    6    1    2   11
[4,]   20    7    8    9   10
[5,]   21   22   23   24   25
```

The knight

1. Starts at 1
2. Never visits the same square twice
3. Moves to the square with the lowest number
4. Ends its tour when no moves are possible

Using standard knight moves the knight can jump from 1 to $14, 12, 10, 24, 22, 20, 18, 16$. The initial jump will be to 10 as that is the lowest numbered square. Continuing to follow these rules, the tour starts as follows: $1, 10, 3, 6, 9, 4, 7, 2, 16, 8, 11, ...$ and ends after 2016 steps on square 2084. The code below finds this result.

Testing

Not sure what is a good way to run tests in site a Quarto Notebook but this works, sort of:

```
testCount <- 0L
testFailures <- 0L
quartoTest <- function(test, expected) {
  testCount <<- testCount + 1L
  if (test ≠ expected) {
    testFailures <<- testFailures + 1L
    cat("Exptected: ", expected, " but got ", test, "\n", sep = "")
  }
}
quartoTestResults <- function() {
  cat(testCount - testFailures, " / ", testCount, " tests passed\n")
  testCount <<- 0L
  testFailures <<- 0L
}
```

Knight moves

The knight moves are given as offsets from the starting coordinate

```
moves <- list(
  c(2L, 1L), c(1L, 2L), c(-1L, 2L), c(-2L, 1L),
  c(-2L, -1L), c(-1L, -2L), c(1L, -2L), c(2L, -1L)
)
quartoTest(length(unique(moves)), 8L)
quartoTestResults()
```

```
1  /  1  tests passed
```

The board

The board is supposed to be infinite so I can't keep the number of each square in memory. One could try to extend the board as needed by continuing the spiral once the knight moves outside of the known board area but this seems tedious. Therefore I tried the alternative approach of translating each coordinate to its position in the sequence using the function `coordToSpiralPosition()` and some helpers as follows:

First I determine how many times the counter spiraled around the origin and the length of the side of a spiral. Then I figure how many steps the coordinate is away from the right bottom

- For the squares above the diagonal from the top left to the right bottom, I simply count the steps made forward.
- For the squares below the diagonal: mirror the coordinates and count the steps to the mirrored coordinates and add $2 \times (\text{side length} - 1)$ to the step count. I need to subtract 1 from the side to avoid double counting the corners.

```
findSpiralCount <- function(x, y) max(abs(x), abs(y)) + 1L
findSideLength <- function(spiralCount) 2L * spiralCount - 1L
aboveDiagonal <- function(x, y) x > -y
countSteps <- function(x, y, side) (side / 2) * 2L + y - x - 1L
# The function countSteps() above simplifies to
countSteps <- function(x, y, side) side + y - x - 1L

coordToSpiralPosition <- function(x, y) {
  spiralCount <- findSpiralCount(x, y)
  side <- findSideLength(spiralCount)
  start <- (side - 2L) * (side - 2L)

  if (aboveDiagonal(x, y)) {
    start + countSteps(x, y, side)
  } else {
    start + 2L * (side - 1L) + countSteps(y, x, side)
  }
}
```

Test our logic on the known inner squares

The results of `coordToSpiralPosition()` can be compared to the board matrix defined above and some values copied from the video:

```
for (i in -2:2) for (j in -2:2)
  quartoTest(
    coordToSpiralPosition(i, j),
    # Some care is needed to translate coordinates to entries in the board
    # matrix.
    board[-j + 3L, i + 3L])

quartoTest(coordToSpiralPosition(3L, -3L), 49L)
quartoTest(coordToSpiralPosition(2L, -3L), 48L)
quartoTest(coordToSpiralPosition(3L, -2L), 26L)
quartoTest(coordToSpiralPosition(3L, -2L), 26L)
quartoTest(coordToSpiralPosition(7L, -3L), 173L)
quartoTest(coordToSpiralPosition(-5L, 4L), 102L)
quartoTestResults()
```

```
31  /  31  tests passed
```

Generating moves

The candidate squares are found by adding the possible moves to the coordinate of the current square.

```
candidateMoves <- function(coord, moves) lapply(moves, \(move) coord + move)
```

The next square is found by choosing the square with the lowest spiral number that has not been visited.

```
findNextSquare <- function(coord, moves, seen) {
  best <- Inf
  bestCandidate <- NULL
  candidates <- candidateMoves(coord, moves)
  for (candidate in candidates) {
    counter <- coordToSpiralPosition(candidate[[1L]], candidate[[2L]])
    if (counter < best &&
        is.null(recollections::getValue(seen, toString(candidate)))
    ) {
      best <- counter
      bestCandidate <- candidate
    }
  }
  bestCandidate
```

```
}
```

With these functions, loop until a next square can not be found adding visited squares to the `path` list and `seen` dictionary.

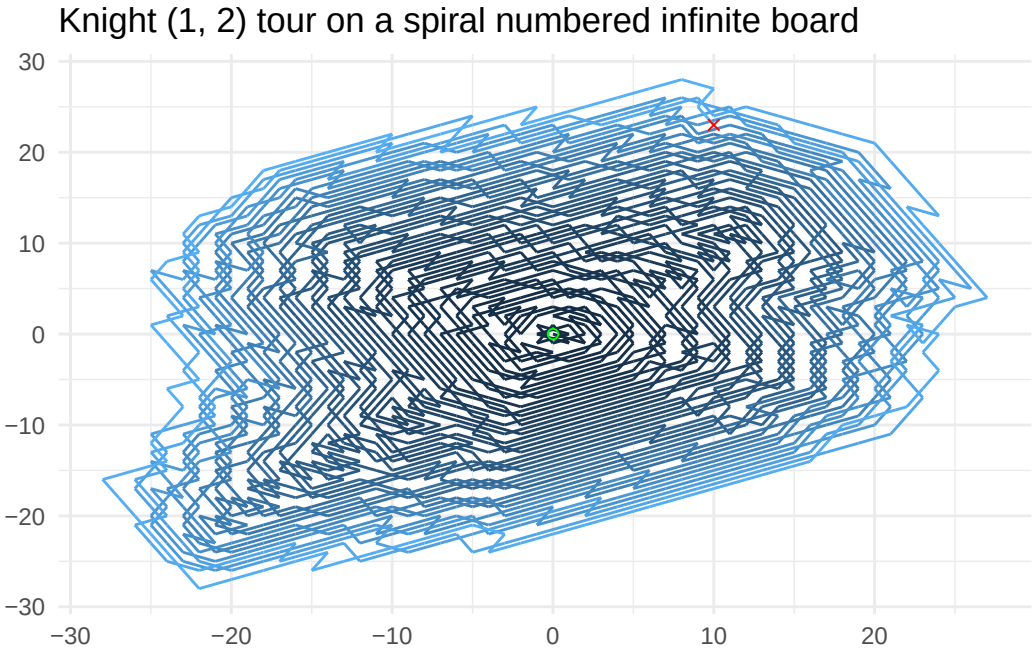
```
simulateTour <- function(moves) {
  seen <- recollections::dictionary()
  path <- list()
  nextSquare <- c(0L, 0L)
  while (!is.null(nextSquare)) {
    path[[length(path) + 1L]] <- nextSquare
    recollections::setValue(seen, toString(nextSquare), TRUE)
    nextSquare <- findNextSquare(nextSquare, moves, seen)
  }

  steps <- length(path)
  final_coord <- path[[steps]]
  cat(
    "The path has ", steps, " steps\n",
    "Final square has counter: ",
    coordToSpiralPosition(final_coord[[1L]], final_coord[[2L]]), "\n",
    "Final square is: (", final_coord[[1L]], ", ", final_coord[[2L]], ")\n",
    sep = ""
  )
  path
}
path <- simulateTour(moves)
```

```
The path has 2016 steps
Final square has counter: 2084
Final square is: (10, 23)
```

The tour can be plotted as well:

```
plotTour <- function(path, knightType = "(1, 2)") {
  df <- as.data.frame(do.call(rbind, path))
  colnames(df) <- c("x", "y")
  df$index <- 1:nrow(df)
  ggplot(df) +
    geom_path(aes(x = x, y = y, colour = index)) +
    geom_point(
```

```
      data = df[1L, ], shape = 1L, colour = "green",
      aes(x = x, y = y)) +
    geom_point(
      data = df[nrow(df), ], shape = 4L, colour = "red",
      aes(x = x, y = y)) +
    theme_minimal() +
    theme(legend.position = "none", axis.title = element_blank()) +
    ggtitle(sprintf(
      "Knight %s tour on a spiral numbered infinite board", knightType))
}
plotTour(path)
```



I'm also interested in moves of super knights: knight likes that jump further. A super knight makes moves of a given length that are symmetric around the x-axis and y-axis.

```
generateMoves <- function(xDelta, yDelta) {
  moves <- list()
  for (x in c(-xDelta, xDelta)) {
    for (y in c(-yDelta, yDelta)) {
      moves[[length(moves) + 1L]] <- c(x, y)
      moves[[length(moves) + 1L]] <- c(y, x)
    }
```

```
  }
  moves
}
quartoTest(length(unique(generateMoves(1L, 2L))), 8L)
quartoTestResults()
```

```
1  /  1  tests passed
```

Combining move generation, simulation and plotting it is easy to quickly gather the results and visualize the tours of arbitrary super knights.

```
deltas <- c(1L, 4L)
plotTour(
  simulateTour(generateMoves(deltas[[1L]], deltas[[2L]])),
  knightType = sprintf("(%s, %s)", deltas[[1L]], deltas[[2L]]))
```

```
The path has 13103 steps
Final square has counter: 10847
Final square is: (-52, 22)
```