

C

I gave a grid with a given amount of columns and rows. Given a tile position I want to get a list of all tile coordinates in clockwise order around the given center tile

To generate a list of all tile coordinates surrounding a given center tile in a grid in a clockwise order, we can follow these steps:

1. **Identify the neighbors:** First, identify the positions of all neighboring tiles around the center tile.
2. **Order them clockwise:** Then, order these neighboring tiles in a clockwise direction, starting from a specific direction (usually top, or top-left).

Given a grid with `rows` and `columns`, and a center tile at position `(center_row, center_column)`, here's how to find the neighboring tiles in clockwise order:

Steps:

1. **Define the neighbors:** The eight possible neighbors of a tile at `(r, c)` are:
 - Top: `(r-1, c)`
 - Top-right: `(r-1, c+1)`
 - Right: `(r, c+1)`
 - Bottom-right: `(r+1, c+1)`
 - Bottom: `(r+1, c)`
 - Bottom-left: `(r+1, c-1)`
 - Left: `(r, c-1)`
 - Top-left: `(r-1, c-1)`
2. **Filter out invalid neighbors:** Some of these neighbors might be outside the grid bounds, so they need to be filtered out.
3. **Return the ordered list:** The valid neighboring positions are then returned in the order specified above.

Python Implementation:

```
python

def get_clockwise_neighbors(center_row, center_column, rows, columns):
    # Define all possible neighbors
    neighbors = [
        (center_row-1, center_column),
        (center_row-1, center_column+1),
        (center_row, center_column+1),
        (center_row+1, center_column+1),
        (center_row+1, center_column),
        (center_row+1, center_column-1),
        (center_row, center_column-1),
        (center_row-1, center_column-1)
    ]

    # Filter out neighbors that are outside the grid
    valid_neighbors = [
        (r, c) for r, c in neighbors
        if 0 <= r < rows and 0 <= c < columns
    ]

    return valid_neighbors

# Example usage
center_row, center_column = 2, 2 # Center tile
rows, columns = 5, 5 # Grid dimensions
neighbors = get_clockwise_neighbors(center_row, center_column, rows, columns)
print(neighbors)
```

Explanation:

- **center_row and center_column:** Coordinates of the center tile.
- **rows and columns:** Dimensions of the grid.
- **neighbors:** List of potential neighbors in clockwise order.
- **valid_neighbors:** Filtered list of valid neighbors within grid bounds.

Example:

If the grid is 5x5 and the center tile is at (2, 2):

- Clockwise neighbors would be: [(1, 2), (1, 3), (2, 3), (3, 3), (3, 2), (3, 1), (2, 1), (1, 1)]

1), (2, 1), (1, 1)].

This code snippet will return these neighbors in the correct order while filtering out any that are out of bounds.



Please use swift for code generation, and the tile coordinates should be generated in a clockwise spiral form from the given tile at (r, c)



Certainly! To generate the coordinates in a clockwise spiral form from the given tile (r, c) in a grid using Swift, you'll want to iterate outward from the center tile, progressively increasing the radius of the spiral.

Here is a Swift implementation:

```
swift

import Foundation

func getSpiralNeighbors(centerRow: Int, centerCol: Int, rows: Int, cols: Int) -> [(Int, Int)] {
    var result: [(Int, Int)] = []

    // Add the center tile as the starting point
    result.append((centerRow, centerCol))

    var distance = 1
    while result.count < rows * cols {
        // Spiral loop: up, right, down, left
        // Up
        for i in 0..
```

```

        let newRow = centerRow +
        let newCol = centerColumn
        if newRow >= 0 && newRow
            result.append((newRow, newCol))
        }
    }
    // Right
    for i in 0..

```

Explanation:

- 1. Start with the center tile:** The algorithm starts by adding the center tile to the result list.
- 2. Spiral expansion:**
 - The **distance** variable tracks how far from the center you are expanding in each spiral iteration.
 - In each step, the algorithm moves in the order: Up, Left, Down, Right, while

checking the bounds of the grid to

ensure valid coordinates.

- The distance is incremented in each spiral iteration, moving further from the center tile.
3. **Check bounds:** For each step, the algorithm checks if the new coordinates are within the grid bounds (i.e., $0 \leq \text{row} < \text{rows}$ and $0 \leq \text{col} < \text{columns}$).

Example:

If the grid is 5x5, and the center tile is (2, 2), the output will be the coordinates in a spiral order, starting from (2, 2) and expanding outward.

This method ensures that all tiles in the grid are covered in a spiral pattern around the given center tile.