```
using PlutoUI
```

```
parse_file (generic function with 1 method)
```

```
function parse_file(io::IO)
    lines = readlines(io)
    chars_on_lines = split.(lines, "")
    numbers = [parse.(Int8, col) for col in chars_on_lines]

    hcat(numbers...)'
end
```

Problem 1

Previous failed attempts

The following assumes that you can only go **right** or **down** but after solving and submitting answer that assumption was invalid which I already knew but wanted to valited

```
get_lowest_risk_path_dr (generic function with 1 method)
```

```
get_lowest_risk_path_ul (generic function with 1 method)
```

```
get_lowest_risk_path (generic function with 1 method)
```

The following uses recursion (with memoization) but that would result into deep tree but lets see if that is fine for the problem or not.

```
get_lowest_risk_path_recur (generic function with 1 method)
```

Solving through Dijkstra's Algorithm

Though I was aware that Dijkstra will certainly solve the problem but my previous attempts were to figure out some algorithm on my own without revising Dijkstra but then after few attempts I lost patience and brushed up Dijkstra to write up the solution

```
get_lowest_risk_path_dijkstra (generic function with 1 method)
```

```
function get_lowest_risk_path_dijkstra(risk_matrix)
    total_rows, total_cols = size(risk_matrix)

    get_adjacent_edges(row, col) = filter(!=(nothing), [
        (col + 1 > total_cols ? nothing : (row, col + 1)),
        (row + 1 > total_rows ? nothing : (row + 1, col)),
        (col -1 < 1 ? nothing : (row, col - 1)),
        (row - 1 < 1 ? nothing : (row - 1, col))
    ])

    # For checking if we have already calculated distance
    intree = similar(risk_matrix, Bool)
    intree .= false

    # Distance for each vertex
    distance = similar(risk_matrix, Int)
    distance .= typemax(Int)

    # For later figuring out the path, getting the parent of vertex
    parent = Dict{Tuple{Int, Int}, Tuple{Int, Int}}()

    distance[1, 1] = 0
    v = (1,1)

    while v != (total_rows, total_cols)
        r, c = v
        intree[r, c] = true
        edges = get_adjacent_edges(r, c)

        for (nr, nc) ∈ edges
            if distance[nr, nc] > distance[r, c] + risk_matrix[nr, nc]
                distance[nr, nc] = distance[r, c] + risk_matrix[nr, nc]
                parent[(nr, nc)] = (r, c)
            end
        end

        v = (1, 1)
        dist = typemax(Int)
        for row in 1:total_rows
            for col in 1:total_cols
                if intree[row, col] == false && distance[row, col] < dist
                    dist = distance[row, col]
                    v = (row, col)
                end
            end
        end
    end

    distance, parent
end
```

```
811
```

```
  0.207266 seconds (247.59 k allocations: 9.661 MiB)
```

```
with_terminal() do
    open("./Day15/prob_input.txt") do io
        risk_matrix = parse_file(io)
        @time dist, _ = get_lowest_risk_path_dijkstra(risk_matrix)
        dist[end, end]
    end
end
```

Problem 2

```
create_bigger_matrix (generic function with 2 methods)
```

```
function create_bigger_matrix(risk_matrix, multiply_factor = (5, 5))
    orig_rows, orig_cols = size(risk_matrix)
    total_rows, total_cols = (orig_rows, orig_cols) .* multiply_factor
    new_matrix = zeros(Int8, total_rows, total_cols)
    new_matrix[1:orig_rows, 1:orig_cols] .= risk_matrix

    # First doing rows
    for r in 1:total_rows
        for c in 1:orig_cols
            if new_matrix[r, c] != 0
                continue
            end

            new_matrix[r, c] = new_matrix[r-orig_rows, c] == 9 ? 1 : new_matrix[r-
orig_rows, c] + 1
        end
    end

    # Now doing cols
    for r in 1:total_rows
        for c in orig_cols+1:total_cols
            if new_matrix[r, c] != 0
                continue
            end

            new_matrix[r, c] = new_matrix[r, c-orig_cols] == 9 ? 1 : new_matrix[r, c-
orig_cols] + 1
        end
    end


    new_matrix
end
```

Though Dijkstra works great, I was curious to see if Bellman Ford algorithm will work better but it will take much more time than Dijkstra due to the huge size of array as it has to iterate $|V| - 1$ over the number of edges which in this case would be approx. $TotalRows \times TotalCols \times 4$.

```
get_lowest_risk_path_bellmanford (generic function with 1 method)
```

```
315
```

```
  0.018814 seconds (50.42 k allocations: 2.259 MiB)
```

```
with_terminal() do
    open("./Day15/prob_test_input.txt") do io
        risk_matrix = parse_file(io)
        risk_matrix = create_bigger_matrix(risk_matrix)
        @time dist, _ = get_lowest_risk_path_dijkstra(risk_matrix)
        dist[end, end]
    end
end
```