```julia
begin
    using PlutoUI
    using DataStructures
end
```

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

```julia
function parse_line(line)
    return split(line, "") .|> x -> parse(Int8, x)
end
```

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

```julia
function parse_file(io::IO)
    return hcat([parse_line(line) for line in eachline(io)]...)'
end
```

```
simulate_and_find_flash_count! (generic function with 1 method)
```

```
function simulate_and_find_flash_count!(energy_levels)
    flashed_state = similar(energy_levels, Bool)
    flashed_state .= false

    tr, tc = size(energy_levels)
    Ifirst, Ilast, Iunit = CartesianIndex(1,1), CartesianIndex(tr, tc),
CartesianIndex(1, 1)

    # Step 1: Increase everything by unit 1
    energy_levels .+= 1
    R = findall(n -> n >= 10, energy_levels)
    queue = Queue{CartesianIndex}()
    map(I -> enqueue!(queue, I), R)
    energy_levels[R] .= 0
    flashed_state[R] .= true
    flash_count = length(R)
    while length(queue) > 0
        I = dequeue!(queue)
        # Step 2: For the flashing ones increase adjacent by 1
        for J in max(Ifirst, I-Iunit):min(Ilast, I+Iunit)
            if J == I || flashed_state[J] == true
                continue
            end

            energy_levels[J] += 1

            # Step 3: If the adjacent also needs to flash then add them to queue
            if energy_levels[J] >= 10
                flash_count+=1
                energy_levels[J] = 0
                flashed_state[J] = true
                enqueue!(queue, J)
            end
        end
    end

    flash_count
end
```

Problem 1

```
simulate_and_find_flash_counts! (generic function with 1 method)
```

```
function simulate_and_find_flash_counts!(energy_levels; steps = 100)
    sum([simulate_and_find_flash_count!(energy_levels) for _ in 1:steps])
end
```

```
(1655, 10×10 adjoint(::Matrix{Int8}) with eltype Int8:)
         0  0  0  0  8  6  6  8  3  3
         0  0  0  5  3  9  9  6  8  3
         0  0  5  3  2  2  2  6  7  3
         0  6  3  2  2  2  2  9  9  3
         2  5  2  2  2  2  2  2  5  3
         1  5  2  2  2  2  2  2  5  3
         2  5  2  2  2  2  2  6  4  3
         0  5  2  2  2  2  6  4  3  3
         0  6  4  4  3  6  4  3  3  3
         2  5  0  0  6  4  3  3  3  3
```

```
0.264899 seconds (587.83 k allocations: 32.989 MiB, 99.83% compilation time)
```

```julia
with_terminal() do
    open("./Day11/prob_input.txt") do io
        energy_levels = parse_file(io)
        @time simulate_and_find_flash_counts!(energy_levels; steps=100), energy_levels
    end
end
```

Problem 2

```
simulate_and_find_when_all_flash! (generic function with 1 method)
```

```julia
function simulate_and_find_when_all_flash!(energy_levels; max_steps = 10_000)

    has_all_flashed = false
    current_step = 0

    while has_all_flashed == false && current_step <= max_steps
        current_step += 1
        simulate_and_find_flash_count!(energy_levels)

        has_all_flashed = all(energy_levels .== 0)
    end

    if (current_step > max_steps)
        error("Reached max number of steps :(")
    end

    return current_step
end
```

```
(337, 10×10 adjoint(::Matrix{Int8}) with eltype Int8:)
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  0  0
```

```
0.001371 seconds (9.10 k allocations: 3.117 MiB)
```

```
with_terminal() do
    open("./Day11/prob_input.txt") do io
        energy_levels = parse_file(io)
        @time simulate_and_find_when_all_flash!(energy_levels), energy_levels
    end
end
```

Though above solved the problem bu I want to to see if I can remove the extra step of checking all zeros after a step. As we get the flash_count afterwards so we can simply check if $FlashCount = TotalRows \times TotalColumns$.

```
simulate_and_find_when_all_flash_optim! (generic function with 1 method)
```

```
function simulate_and_find_when_all_flash_optim!(energy_levels; max_steps = 10_000)

    has_all_flashed = false
    current_step = 0
    tr, tc = size(energy_levels)

    while has_all_flashed == false && current_step <= max_steps
        current_step += 1
        has_all_flashed = simulate_and_find_flash_count!(energy_levels) == tr*tc
    end

    if (current_step > max_steps)
        error("Reached max number of steps :(")
    end

    return current_step
end
```

```
(337, 10×10 adjoint(::Matrix{Int8}) with eltype Int8:)
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
```

```
  0.000938 seconds (8.42 k allocations: 3.071 MiB)
```

```
with_terminal() do
    open("./Day11/prob_input.txt") do io
        energy_levels = parse_file(io)
        @time simulate_and_find_when_all_flash_optim!(energy_levels), energy_levels
    end
end
```

The perf gain isn't hugely difference but it did remove approx. **0.7k** memory usage