Please write a Python script to solve the problem below. It should read the input from a file input.txt, which has the same format as the example:

The Historians use their fancy device again, this time to whisk you all away to the North Pole prototype suit manufacturing lab... in the year 1518! It turns out that having direct access to history is very convenient for a group of historians.

You still have to be careful of time paradoxes, and so it will be important to avoid anyone from 1518 while The Historians search for the Chief. Unfortunately, a single guard is patrolling this part of the lab.

Maybe you can work out where the guard will go ahead of time so that The Historians can search safely?

You start by making a map (your puzzle input) of the situation. For example:

```
....#.....
.........#
..........
..#.......
.......#..
..........
.#..^.....
........#.
#.........
......#...
```

The map shows the current position of the guard with ^ (to indicate the guard is currently facing up from the perspective of the map). Any obstructions - crates, desks, alchemical reactors, etc. - are shown as #.

Lab guards in 1518 follow a very strict patrol protocol which involves repeatedly following these steps:

If there is something directly in front of you, turn right 90 degrees.
Otherwise, take a step forward.
Following the above protocol, the guard moves up several times until she reaches an obstacle (in this case, a pile of failed suit prototypes):

```
....#.....
....^....#
..........
..#.......
.......#..
..........
.#........
```

.....#.
#.....
.....#...

Because there is now an obstacle in front of the guard, she turns right before continuing straight in her new facing direction:

....#.....
.....>#
.....
..#.....
.....#..
.....
.#.....
.....#.
#.....
.....#...

Reaching another obstacle (a spool of several very long polymers), she turns right again and continues downward:

....#.....
.....#
.....
..#.....
.....#..
.....
.#.....v.
.....#.
#.....
.....#...

This process continues for a while, but the guard eventually leaves the mapped area (after walking past a tank of universal solvent):

....#.....
.....#
.....
..#.....
.....#..
.....
.#.....
.....#.
#.....
.....#v..

By predicting the guard's route, you can determine which specific positions in the lab will be in the patrol path. Including the guard's starting position, the positions visited by the guard before leaving the area are marked with an X:

....#.....
....XXXXX#
....X...X.
..#.X...X.

..XXXXX#X.
..X.X.X.X.
.#XXXXXXX.
.XXXXXXX#.
#XXXXXXX..
.....#X..
In this example, the guard will visit 41 distinct positions on your map.

Predict the path of the guard. How many distinct positions will the guard visit before leaving the mapped area?

Here's a Python script to solve the problem. It simulates the guard's movements based on the given rules and tracks the distinct positions visited before the guard leaves the mapped area.

Failed to generate

Here is the script:

python Copy code

```python
def read_input(file_name):
    with open(file_name, "r") as file:
        return [list(line.strip()) for line in file]

def find_guard_position_and_direction(grid):
    directions = {'^': (-1, 0), '>': (0, 1), 'v': (1, 0), '<': (0, -1)}
    for row in range(len(grid)):
        for col in range(len(grid[0])):
            if grid[row][col] in directions:
                return (row, col, grid[row][col])
    return None

def simulate_guard(grid):
    directions = ['^', '>', 'v', '<']  # Clockwise order of directions
    dir_map = {'^': (-1, 0), '>': (0, 1), 'v': (1, 0), '<': (0, -1)}
    visited = set()

    # Find the initial position and direction of the guard
    row, col, current_dir = find_guard_position_and_direction(grid)
    visited.add((row, col))

    while True:
        # Calculate the next position based on the current direction
        dr, dc = dir_map[current_dir]
        next_row, next_col = row + dr, col + dc

        if 0 <= next_row < len(grid) and 0 <= next_col < len(grid[0]) and
grid[next_row][next_col] == '#':
```

```
            # Obstacle ahead, turn right
            current_dir = directions[(directions.index(current_dir) + 1) % 4]
        else:
            # Move forward
            if 0 <= next_row < len(grid) and 0 <= next_col < len(grid[0]):
                row, col = next_row, next_col
                visited.add((row, col))
            else:
                # Guard has moved out of bounds
                break

    return len(visited)

if __name__ == "__main__":
    grid = read_input("input.txt")
    result = simulate_guard(grid)
    print("Distinct positions visited:", result)
```

How It Works:

1. **Input Parsing**: The `read_input` function reads the map from `input.txt` into a 2D list.
2. **Guard Initialization**: The `find_guard_position_and_direction` function locates the guard's initial position and direction.
3. **Simulation**: The `simulate_guard` function follows the patrol protocol:
   - Turns right if an obstacle is in front.
   - Moves forward otherwise.
   - Tracks visited positions.
4. **Output**: The number of distinct positions visited is printed.

Example Usage:

Save the example map in a file named `input.txt` and run the script. It will calculate and output the number of distinct positions the guard visits.

Please update the script for part 2 of the problem:

While The Historians begin working around the guard's patrol route, you borrow their fancy device and step outside the lab. From the safety of a supply closet, you time travel through the last few months and record the nightly status of the lab's guard post on the walls of the closet.

Returning after what seems like only a few seconds to The Historians, they explain that the guard's patrol area is simply too large for them to safely search the lab without getting caught.

Fortunately, they are pretty sure that adding a single new obstruction won't cause a time paradox. They'd like to place the new obstruction in such a way that the guard will get stuck in a loop, making the rest of the lab safe to search.

To have the lowest chance of creating a time paradox, The Historians would like to know all of the possible positions for such an obstruction. The new obstruction can't be placed at the guard's starting position - the guard is there right now and would notice.

In the above example, there are only 6 different positions where a new obstruction would cause the guard to get stuck in a loop. The diagrams of these six situations use O to mark the new obstruction, | to show a position where the guard moves up/down, - to show a position where the guard moves left/right, and + to show a position where the guard moves both up/down and left/right.

Option one, put a printing press next to the guard's starting position:

```
....#.....
....+---+#
....|...|.
..#.|...|.
....|..#|.
....|...|.
.#.O^---+.
........#.
#.........
......#...
```

Option two, put a stack of failed suit prototypes in the bottom right quadrant of the mapped area:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
......O.#.
#.........
......#...
```

Option three, put a crate of chimney-squeeze prototype fabric next to the standing desk in the bottom right quadrant:

```
....#.....
....+---+#
....|...|.
..#.|...|.
```

```
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
.+----+O#.
#+----+...
......#...
```

Option four, put an alchemical retroencabulator near the bottom left corner:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
..|...|.#.
#O+---+...
......#...
```

Option five, put the alchemical retroencabulator a bit to the right instead:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
....|.|.#.
#..O+-+...
......#...
```

Option six, put a tank of sovereign glue right next to the tank of universal solvent:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
.+----++#.
#+----++..
......#O..
```

It doesn't really matter what you choose to use as an obstacle so long as you and The Historians can put it into position without the guard noticing. The important thing is having enough options that you can find one that minimizes time paradoxes, and in this example, there are 6 different positions you could choose.

You need to get the guard stuck in a loop by adding a single new

obstruction. How many different positions could you choose for this obstruction?

Here's an updated Python script for part 2 of the problem. This version simulates the guard's movement while checking for potential positions where a new obstruction could cause the guard to get stuck in a loop.

Script

python Copy code

```python
def read_input(file_name):
    with open(file_name, "r") as file:
        return [list(line.strip()) for line in file]

def find_guard_position_and_direction(grid):
    directions = {'^': (-1, 0), '>': (0, 1), 'v': (1, 0), '<': (0, -1)}
    for row in range(len(grid)):
        for col in range(len(grid[0])):
            if grid[row][col] in directions:
                return (row, col, grid[row][col])
    return None

def simulate_guard_with_obstruction(grid, obstruction_pos=None):
    directions = ['^', '>', 'v', '<']  # Clockwise order of directions
    dir_map = {'^': (-1, 0), '>': (0, 1), 'v': (1, 0), '<': (0, -1)}
    visited = set()
    path = set()

    # Place the obstruction if specified
    if obstruction_pos:
        grid[obstruction_pos[0]][obstruction_pos[1]] = '#'

    # Find the initial position and direction of the guard
    row, col, current_dir = find_guard_position_and_direction(grid)
    visited.add((row, col))

    while True:
        # Add current position and direction to path
        path.add((row, col, current_dir))

        # Calculate the next position based on the current direction
        dr, dc = dir_map[current_dir]
        next_row, next_col = row + dr, col + dc

        if 0 <= next_row < len(grid) and 0 <= next_col < len(grid[0]) and
grid[next_row][next_col] == '#':
            # Obstacle ahead, turn right
            current_dir = directions[(directions.index(current_dir) + 1) % 4]
        else:
            # Move forward
```

```
            if 0 <= next_row < len(grid) and 0 <= next_col < len(grid[0]):
                row, col = next_row, next_col
                if (row, col, current_dir) in path:
                    return True  # Loop detected
                visited.add((row, col))
            else:
                return False  # Guard moved out of bounds

def find_loop_positions(grid):
    loop_positions = set()
    for row in range(len(grid)):
        for col in range(len(grid[0])):
            if grid[row][col] == '.' and (row, col) !=
find_guard_position_and_direction(grid)[:2]:
                # Simulate guard with the potential new obstruction
                grid_copy = [row[:] for row in grid]
                if simulate_guard_with_obstruction(grid_copy, (row, col)):
                    loop_positions.add((row, col))
    return loop_positions

if __name__ == "__main__":
    grid = read_input("input.txt")
    loop_positions = find_loop_positions(grid)
    print("Number of positions causing a loop:", len(loop_positions))
```

Explanation

1. **Input Parsing**: The `read_input` function loads the map into a 2D list.
2. **Guard Initialization**: The `find_guard_position_and_direction` function locates the guard's starting position and direction.
3. **Simulation with Obstruction**:
   - `simulate_guard_with_obstruction` simulates the guard's patrol after placing an obstruction at a given position.
   - It checks if the guard enters a loop by recording positions and directions visited.
4. **Find Loop Positions**:
   - `find_loop_positions` iterates through all potential positions for placing an obstruction (excluding obstacles and the guard's starting position).
   - It checks if adding an obstruction at each position causes a loop.
5. **Output**: The script prints the count of positions that can cause a loop.

Example Usage

Save the map in `input.txt` and run the script. It will output the number of positions where placing a new obstruction causes the guard to get stuck in a loop.

Let me know if you have more questions!