Problem 1. A frog is randomly placed with equal probability at any integer from 0 to 999. There is a wall at 1000. Each second the frog jumps, on the k^{th} jump, frog jumps a distance 2^{k+1} equally likely in both directions left and right. What is the expected time for the frog to land on or cross the wall at 1000?

(Proposed by The Reverse Flash)

Proof. Define random processes X_t as the position and D_t as the direction, at time t. Now, define the position point function

$$PP(x) = \begin{cases} 1 & \text{if } x \geq 1000 \\ 0 & \text{if } x < 1000 \end{cases}.$$

Essentially, we need to find t such that $E(PP(X_t)) = 1$. Let's do that then shall we! Define a *value function*

$$v_t(x, d) = E(PP(X_t) \mid X_t = x, D_t = d).$$

Now let's define the recursive process:

$$v_1(x, d) = \frac{1}{2}v_2(x - 2^{1+1}, L) + \frac{1}{2}v_2(x + 2^{1+1}, R)$$

$$v_2(x, d) = \frac{1}{2}v_3(x - 2^{2+1}, L) + \frac{1}{2}v_3(x + 2^{2+1}, R)$$

$$...$$

and so on.

But the initial position is randomly selected between 0 and 999. To account for this, we compute the following

$$\frac{1}{1000}v_1(0, d) + \frac{1}{1000}v_1(1, d) + ... + \frac{1}{1000}v_1(999, d)$$

$$= \frac{1}{1000}\sum_{i=0}^{999} v_1(i, d).$$

Lastly, we just have to make sure that the frog never jumps to the left of 0, for that we'll include the necessary *if* conditions in the code.

I was too lazy to code it myself this time, also considering that I have a New Year's Party to attend soon, so I ended up feeding the mathematical logic to Gemini and getting a Python code out of it. Then I talked to it to make sure I understood the code and that it followed the same logic as my solution. Here is the link to the conversation (https://gemini.google.com/share/84ab0b12f8ff) and here is the final code:

```
def calculate_frog_expected_time():
    # Initial Configuration
    # We map position -> probability.
```

```
# The frog starts at any integer 0-999 with equal probability
(1/1000).
current_positions = {i: 1/1000.0 for i in range(1000)}

expected_time = 0.0
probability_finished = 0.0

# We iterate through time steps k = 1, 2, 3...
# The loop continues until the probability of remaining active is
negligible.
for k in range(1, 100):
    next_positions = {}

    # Calculate jump distance for the k-th jump: 2^(k+1)
    jump_distance = 2 ** (k + 1)

    prob_finished_this_step = 0.0

    # Process every position currently occupied by the frog
    [cite: 13]
    for x, prob in current_positions.items():
        # The recursion logic: 1/2 goes Left, 1/2 goes Right
        # Logic: v_t derived from v_{t+1} logic in Source 13

        # Left Move
        left_pos = x - jump_distance
        # Constraint: Frog never jumps to the left of 0
        if left_pos < 0:
            left_pos = 0

        # Right Move
        right_pos = x + jump_distance

        # Distribute Probability (0.5 to left, 0.5 to right)

        # Check Left Outcome
        next_positions[left_pos] =
            next_positions.get(left_pos, 0.0) + (0.5 * prob)

        # Check Right Outcome
        if right_pos >= 1000:
            prob_finished_this_step += 0.5 * prob
        else:
            next_positions[right_pos] =
                next_positions.get(right_pos, 0.0) + (0.5 * prob)
```

```
            # Update Expected Time Calculation
            # E[T] = Sum(t * Probability(finish at t))
            expected_time += k * prob_finished_this_step
            probability_finished += prob_finished_this_step

            # Update positions for the next recursive step
            current_positions = next_positions

            # Optimization: If almost all probability mass has finished,
            stop.
            if probability_finished > 0.9999999:
                break

        return expected_time

    # Run the calculation
    result = calculate_frog_expected_time()
    print(f"The Expected Time for the frog to cross the wall is: {result:.4f}")
```

The expected time is $\boxed{8.9802}$.

■