You will find the code for this question in the `RL-book/Assignment2/assignment2_code.py` file.

We created a class to define the state. This class which we called `StateSnakeAndLadder` only has one attribute: it is the position on the grid. If the grid size is 100 the State Space is $\{1, 2, 3, ..., 100\}$.

The structure of the transition probabilities can be found in the function `get_transition_map` of the `StateSnakeAndLadderGame` class.

The graph of the probability distribution of time steps to finish the game is can be found here:

We used the code in the `RL-book/Assignment2/assignment2_code.py` file to generate it.

Let us solve the Frog Puzzle. We want to find out the expected number of steps required when there are n leaves ahead.

Let K_n be this number.

We first have that $K_0 = 0$.

Let us name the leaves. We will choose here to number them according to their distance from the destination, i.e. the last leaf (the one at which we stop counting) is going to be leaf 0, and the leaf we start at is leaf n.

Note that in this case, Markov property is respected: when the frog is on a given leaf, the expected number of jumps remaining only depends on the leaf on which the frog is and not on the leaves where the frog was before that leaf.

Hence, after the first leaf, if the frog lands on leaf i, with $i \in \{0, ..., n-1\}$, the expected value of the remaining steps will be K_i.

Besides this Markov Assumption, with the law of total probability, we get:

$$K_n = 1 + \frac{1}{n}\sum_{i=0}^{n-1} K_i = \frac{1}{n}\sum_{i=0}^{n-1}\frac{1}{n}(1 + K_i)$$

This can be rewritten:

$$n(K_n - 1) = \sum_{i=0}^{n-1} K_i$$

Similarly, we get that, as n has been chosen generically (for $n \geq 1$):

$$(n-1)(K_{n-1} - 1) = \sum_{i=0}^{n-2} K_i$$

We can thus take advantage of the fact that the differences between the two sums above will simplify. We get:

$$n(K_n - 1) - (n-1)(K_{n-1} - 1) = K_{n-1}$$

Hence:

$$nK_n - n - (n-1)K_{n-1} + (n-1) = K_{n-1}$$

$$K_n = K_{n-1} + \frac{1}{n}$$

By using the fact that $K_0 = 0$ and that $\sum_{i=1}^{n}(K_i - K_{i-1}) = K_n - K_0$, we get that:

$$K_n = \sum_{i=1}^{n}\frac{1}{i}$$

The Bellman equation gives us that the value function is such that:

$$V(s) = R(s) + \gamma \sum_{s' \in \mathtt{N}} P(s, s')V(s')$$

And the expectation of the number of dice rolls to finish the game is:

$$\mathbb{E}(N_{steps}|state = s) = 1 + \sum_{i=1}^{\text{grid\_size}} P(s, i)\mathbb{E}(N_{steps}|state = i)$$

Hence, if we let $R(s) = 1$ for any non-terminating state, and choose to have $\gamma = 1$, then we can determine the expected number of dice rolls to finish the game just by using the `get_value_function_vec` of the FiniteMarkovRewardProcess instance corresponding to the process.

This is what we did in the code.

To compute the Value Function for any discount factor $0 \leq \gamma \leq 1$, as:

$$V(s) = \mathbb{E}(R_{t+1}|S_t = s) + \gamma\mathbb{E}(R_{t+2}|S_t = s) + \gamma^2\mathbb{E}(R_{t+3}|S_t = s) + ...$$

We can use simulations and simulate `num_traces` reward traces for a given starting state and for each trace compute the discounted sum of rewards.

We then get an estimate of the value function by taking the mean of the discounted sum of rewards across the different traces we generated.

This is what we did in the code for this problem.