# Theoretical Portion of Assignment 2

1.
$$V^{\pi_D}(s) = Q^{\pi_D}(s, \pi_D(s)), \forall s \in \mathcal{N}$$
$$Q^{\pi_D}(s, \pi_D(s)) = \mathcal{R}(s, \pi_D(s)) + \sum_{s' \in \mathcal{N}} \mathcal{P}(s, \pi_D(s), s')V^{\pi_D}(s'), \forall s \in \mathcal{N}$$
$$V^{\pi_D}(s) = \mathcal{R}(s, \pi_D(s)) + \sum_{s' \in \mathcal{N}} \mathcal{P}^{\pi_D}(s, s')V^{\pi_D}(s'), \forall s \in \mathcal{N}$$
$$Q^{\pi_D}(s, \pi_D(s)) = \mathcal{R}(s, \pi_D(s)) + \sum_{s' \in \mathcal{N}} \mathcal{P}(s, \pi_D(s), s')Q^{\pi_D}(s', \pi_D(s')), \forall s \in \mathcal{N}$$

2. For each state $s \in \mathcal{S}$, we have the following:

- $\mathbb{P}[\text{move to } s+1] = a$, and corresponding reward is $1 - a$.
- $\mathbb{P}[\text{stay at } s] = 1 - a$, and corresponding reward is $1 + a$.

Therefore, neither the transition probability function nor the rewards function depend on the current state $s$. Hence, we introduce an auxiliary state space $\mathcal{S}' = \{s_1, s_2\}$ where:

- $s_1 =$ move to the next state;
- $s_2 =$ stay at the current state.

Then, we have the following, which indicates that $s_1$ and $s_2$ has exactly the same transition probability function and rewards function:

- $\mathbb{P}[s_1, a, s_1] = a$, $\mathbb{P}[s_1, a, s_2] = 1 - a$, $\mathcal{R}(s_1, a, s_1) = 1 - a$, $\mathcal{R}(s_1, a, s_2) = 1 + a$
- $\mathbb{P}[s_2, a, s_1] = a$, $\mathbb{P}[s_2, a, s_2] = 1 - a$ $\mathcal{R}(s_2, a, s_1) = 1 - a$, $\mathcal{R}(s_2, a, s_2) = 1 + a$

As a result, $s_1$ and $s_2$ has the same value function under all action $a$. They will have the same Optimal Value Function and starting from $s_1$ or $s_2$ does not matter. Hence, we have:

$$V(s_1) = a(1-a) + (1-a)(1+a) + \gamma a V(s_1) + \gamma(1-a)V(s_2), \forall a \in [0,1]$$
$$V(s_2) = a(1-a) + (1-a)(1+a) + \gamma a V(s_1) + \gamma(1-a)V(s_2), \forall a \in [0,1]$$

With $V^*(s_1) = V^*(s_2)$:

$$\begin{aligned} V^*(s_1) &= \max_{a \in [0,1]} a(1-a) + (1-a)(1+a) + \gamma a V^*(s_1) + \gamma(1-a)V^*(s_1) \\ &= \max_{a \in [0,1]} a(1-a) + (1-a)(1+a) + \gamma V^*(s_1) \end{aligned}$$

With $\gamma = 0.5$, calculating $\frac{dV^*(s_1)}{da}$ yields the requirement: $1 - 4a^* = 0$ and therefore $a* = \frac{1}{4}$. Thus, the optimal policy for all state is $a* = \frac{1}{4}$, and the Optimal Value Function is $V^*(s_1) = V^*(s_2) = 2 \times \frac{3}{4} \times \frac{6}{4} = \frac{9}{4}$. Under optimal policy, we can see that the choice of leaving or staying gives the same optimal value, so $V^*(s) = \frac{9}{4}, \pi^*(s) = \frac{1}{4} \forall s \in \mathcal{S}$.

3. (a)
   - State space: $\mathcal{S} = \{0, 1, ..., n\}, \mathcal{N} = \{1, ..., n-1\}, \mathcal{T} = \{0, n\}$, representing lilypads.
   - Action space: $\mathcal{A} = \{A, B\}$, representing sounds.
   - $$\mathcal{P}_\mathcal{R}(s,a,r,s') = \begin{cases} \frac{s}{n} & \text{if } a = A,\ r = -1,\ s' = s-1 \\ \frac{n-s}{n} & \text{if } a = A,\ r = 1,\ s' = s+1 \\ \frac{1}{n} & \text{if } a = B,\ r = s'-s,\ s' \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases}$$
   - Transitions function: $\mathbb{P}(s, A, s+1) = \frac{n-s}{n}$, $\mathbb{P}(s, A, s-1) = \frac{s}{n}$, $\mathbb{P}(s, B, i) = \frac{1}{n}, \forall i \in \mathcal{S}, \forall s \in \mathcal{N}$.
   - Rewards function:
     - $\mathcal{R}(s, A) = \frac{n-s}{n} - \frac{s}{n} = \frac{n-2s}{n}, \forall s \in \mathcal{N}$
     - $\mathcal{R}(s, B) = \frac{1}{n}(\sum_{i=0}^{n} i) = \frac{n+1}{2}, \forall s \in \mathcal{N}$

(b) Code to model this MDP as an instance of the FiniteMarkovDecisionProcess class:

```
class FrogEscape(FiniteMarkovDecisionProcess[int, int]):

    def __init__(self, n: int):
        self.n = max(min(n,100),2)
        super().__init__(self.get_action_transition_reward_map())

    def get_action_transition_reward_map(self) -> \
        StateActionMapping[int, int]:
        d: Dict[int, Dict[str, Categorical[Tuple[int, float]]]] = {}

        d[0] = None
        d[self.n] = None

        for s in range(1, self.n):
            a: Dict[str, Categorical[Tuple[int, float]]] = {}
            a[0] = Categorical({(s-1,-1.):(float(s)/float(self.n)),
                                 (s+1,1.):(1-float(s)/float(self.n))})
            a[1] = Categorical({(i, float(i-s)):(1./self.n)
                                  for i in range(self.n+1)})
            d[s] = a

        return d
```

Code to get the Optimal Value Function and the Optimal Deterministic Policy for a given $n$:

```
def optimal_deterministic(n:int) -> \
    Tuple[np.ndarray, FinitePolicy[int,int]]:
    fe: FrogEscape = FrogEscape(n = n)

    totalNode: int = np.array([2**i for i in range(1,n)]).sum()
    leaveStart: int = totalNode - 2**(n-1)

    optVF: np.ndarray = np.repeat(-np.inf,n-1)
    optPolicy: FinitePolicy[int,int] = None

    for leaves in range(leaveStart,totalNode):
        actionList: list = [None]
        if leaves % 2 == 0:
            actionList.append(0)
        else:
            actionList.append(1)
        for _ in range(n-2):
            if (math.floor(leaves/2)-1) % 2 == 0:
                actionList.append(0)
            else:
                actionList.append(1)
            leaves = math.floor(leaves/2)-1
        actionList.append(None)

        currPolicy: FinitePolicy[int,int] = \
            FinitePolicy({i:Constant(actionList[i])
                          for i in range(n+1)})

        implied_fe: FiniteMarkovRewardProcess[int] = \
            fe.apply_finite_policy(currPolicy)
        currVF: np.ndarray = \
            implied_fe.get_value_function_vec(gamma = 1.)

        if np.all(currVF > optVF):
            optVF = currVF
            optPolicy = currPolicy
    if optPolicy == None:
        return
    else:
        return optVF, optPolicy
```

For test result of the above code, see LilypadFrog.ipynb.

(c)

4.