Report
Assignment 2 Part 3

Team 47
Koushik Viswanadha (2018101109)
Vivek Pamnani (2018111032)

Procedure for making A matrix

The following equation is the base for building matrix A,

$$\sum_i \sum_a (\delta_{ij} - p_{ij}^a) x_{ia} = \alpha_j,$$

Where,
δ : Kronecker delta function,
p^a_{ij} : Probability of reaching state j when action a is taken in state i,
In the Matrix ‘a’ each row represents a state and each column represents a transition.

Calculations:

1. NOOP:

```
# for calculating A
A = np.zeros((len(state_space), len_actions))
j_list_old = 0
acts_total = []
for i in range(len(state_space)):
    j_list = []
    p_ij = []
    act = []

    for a in actions[i]:

        if a == 1: #NOOP
            j_list.append(i)
            p_ij.append(0)
            act.append(1)
```

2. SHOOT:

```
elif a == 2: #SHOOT
    #If arrow hits
    query_success = [state_space[i][0]-1, state_space[i][1]-1, state_space[i][2]-1]
    j_list.append(get_state_index(state_space, query_success))
    p_ij.append(0.5)
    act.append(2)
    #If arrow misses
    query_failure = [state_space[i][0], state_space[i][1]-1, state_space[i][2]-1]
    j_list.append(get_state_index(state_space, query_failure))
    p_ij.append(0.5)
    act.append(2)
```

3. DODGE:

```
elif a == 3: #DODGE
    if(state_space[i][2] == 1):
        #If got arrow and 50 stamina consumed: 0.8*1
        query_arrow_50 = [state_space[i][0], state_space[i][1]+1 if state_space[i][1] < 3 else 3, 0]
        j_list.append(get_state_index(state_space, query_arrow_50))
        p_ij.append(0.8)
        act.append(3)
        #If no arrow and 50 stamina consumed: 0.2*1
        query_narrow_50 = [state_space[i][0], state_space[i][1], 0]
        j_list.append(get_state_index(state_space, query_narrow_50))
        p_ij.append(0.2)
        act.append(3)

    elif(state_space[i][2] == 2):
        #If got arrow and 50 stamina consumed: 0.8*0.8
        query_arrow_50 = [state_space[i][0], state_space[i][1]+1 if state_space[i][1] < 3 else 3, state_space[i][2]-1]
        j_list.append(get_state_index(state_space, query_arrow_50))
        p_ij.append(0.64)
        act.append(3)
        #If no arrow and 50 stamina consumed: 0.2*0.8
        query_narrow_50 = [state_space[i][0], state_space[i][1], state_space[i][2]-1]
        j_list.append(get_state_index(state_space, query_narrow_50))
        p_ij.append(0.16)
        act.append(3)
        #If got arrow and 100 stamina consumed: 0.8*0.2
        query_arrow_100 = [state_space[i][0], state_space[i][1]+1 if state_space[i][1] < 3 else 3, 0]
        j_list.append(get_state_index(state_space, query_arrow_100))
        p_ij.append(0.16)
        act.append(3)
        #If no arrow and 100 stamina consumed: 0.2*0.2
        query_narrow_100 = [state_space[i][0], state_space[i][1], 0]
        j_list.append(get_state_index(state_space, query_narrow_100))
        p_ij.append(0.04)
        act.append(3)
```

4. RECHARGE:

```
        elif a==4: #RECHARGE
            #If recharged
            query_success = [state_space[i][0], state_space[i][1], state_space[i][2]+1]
            j_list.append(get_state_index(state_space, query_success))
            p_ij.append(0.8)
            act.append(4)

            #The state does not change if not recharged, hence ignored.
            #If not recharged.
            query_failure = [state_space[i][0], state_space[i][1], state_space[i][2]]
            j_list.append(get_state_index(state_space, query_failure))
            p_ij.append(0.2)
            act.append(4)
    acts_total.append(act)
```

5. Updating the A matrix:

```
    for k in range(len(j_list)):
        #j_list[k] is the index 'j' and p_ij[k] is p^a_ij
        # print(j_list)
        row = j_list[k]
        column = j_list_old + get_element_index(actions[i], act[k])
        kronecker_delta = int(i == j_list[k])
        A[row][column] = kronecker_delta - p_ij[k]
    # print(len(actions[i]))
    # print(A)
    j_list_old += len(actions[i])
```

Procedure for finding policy.

The equation used for finding the policy:

$$\max(\mathbf{rx}) \mid \mathbf{Ax} = \boldsymbol{\alpha},\ \mathbf{x} \geq 0,$$

For Lero in a given state in the state space (MD_health, arrows, stamina), Lero may have one or more plausible actions. Out of all the actions, the best action is chosen for that state.
Thus, we iterate over the states and the actions/transitions and select the best action for each state.

Can there be multiple policies? Why? What changes can you make in your code to generate another policy?

Yes, there can be multiple policies.

An optimal solution to a Linear Programming problem may not be the most optimal solution. In other words, a Linear Programming problem may have many optimal solutions.

In finding an optimal policy for a given state, one or more actions may have the same value. Hence, the policy may differ on how the best action is chosen.

Penalty for each action may be changed which dictates policy accordingly.