Exercise 8 by Phil Szalay, Florian Schneider

January 30, 2020

```
In [1]: import numpy as np

In [2]: actions = ['north', 'east', 'south', 'west']

In [3]: def pi(s, a):
            return 0.25

        def r_a_s_s(s, s_next, a):
            # in case of grid world we don't need parameter s and a, since reward is always -1
            if (s in (0, 15) and s_next in (0, 15)):
                return 0
            else:
                return -1

        def p_a_s_s(s, s_next, a):
            # the probability of going from state s to s_next
            return 0.25

        def get_possible_next_states(state):
            if (state == 0):
                return (0, 0, 0, 0)
            if (state == 1):
                return (1, 2, 5, 0)
            if (state == 2):
                return (2, 3, 6, 1)
            if (state == 3):
                return (3, 3, 7, 2)
            if (state == 4):
                return (0, 5, 8, 4)
            if (state == 5):
                return (1, 6, 9, 4)
            if (state == 6):
                return (2, 5, 7, 10)
            if (state == 7):
                return (3, 7, 11, 6)
            if (state == 8):
                return (4, 9, 12, 8)
```

```
    if (state == 9):
        return (5, 8, 10, 13)
    if (state == 10):
        return (6, 9, 11, 14)
    if (state == 11):
        return (7, 11, 15, 10)
    if (state == 12):
        return (8, 13, 12, 12)
    if (state == 13):
        return (9, 12, 14, 13)
    if (state == 14):
        return (10, 13, 14, 15)
    if (state == 15):
        return (15, 15, 15, 15)

def calc_bellman_equation(current_state, V, gamma):
    V_s = 0

    for action in actions:
        reward = 0
        possible_next_states = get_possible_next_states(current_state)

        for possible_next_state in possible_next_states:
            reward += p_a_s_s(current_state, possible_next_state, action) * (r_a_s_s(cu

        V_s += pi(current_state, action) * reward

    return V_s


def iterative_policy_evaluation():
    V = np.zeros(16)

    theta = 0.001
    run = 1

    while (run):
        delta = 0
        V_temp = V.copy()
        for i, v_state in enumerate(V):
            if(i in (0, 15)):
                gamma = 0
            else:
                gamma = 1

            v = v_state
            V[i] = calc_bellman_equation(i, V_temp, gamma)
            delta = max(delta, abs(v - V[i]))
```

```
                # stop loop if delta is small enough
                if (delta <= theta):
                    run = 0

            return V
```

In [4]:
```
# Run iterative policy evaluation with function pi
res = iterative_policy_evaluation()
print(res)
```

```
[  0.         -13.98945772 -19.98437823 -21.98251832 -13.98945772
 -17.98623815 -19.98448273 -19.98437823 -19.98437823 -19.98448273
 -17.98623815 -13.98945772 -21.98251832 -19.98437823 -13.98945772
   0.        ]
```

In [5]:
```
print(res.shape)
```

```
(16,)
```