ASSIGNMENT- SECOND YEAR STUDENTS

--SHIRSHAKK PURKAYASTHA

I.I.S.E.R. Bhopal

VALUE-ITERATION ALGORITHM + POLICY-ITERATION ALGORITHM

CODE(with outputs):

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from IPython.display import clear_output
from random import randint, random
from time import sleep
```

```
A_P=int(input("Enter agent Postion to start with:"))
def print_board(agent_position):
    fields = list(range(25))
    board = "----------------------\n"
    for i in range(0,25,5):
        line = fields[i:i+5]
        for field in line:
            if field == agent_position:
                board += "| A "
            elif field == fields[0] or field == fields[-11]:
                board += "| X "
            elif field == fields[-4] or field == fields[-1]:
                board += "| X "
            else:
                board += "|   "
        board += "|\n"
        board += "----------------------\n"
```

```
    print(board)
if A_P in range(0,14):
  print_board(A_P)
elif A_P in range(14,20):
  print_board(A_P+1)
elif A_P in range(20,22):
  print_board(A_P+2)
```

OUTPUT:

```
Enter agent Postion to start with:21
------------------------
| X |   |   |   |   |
------------------------
|   |   |   |   |   |
------------------------
|   |   |   |   | X |
------------------------
|   |   |   |   |   |
------------------------
|   | X |   | A | X |
------------------------
```

```
def create_state_to_state_prime_verbose_map():
    l = list(range(25))
    state_to_state_prime = {}
    for i in l:
        if i == 0 or i == 24:
            state_to_state_prime[i] = {'N': 0, 'E': 0, 'S': 0, 'W': 0}
        elif i % 5 == 0:
            state_to_state_prime[i] = {'N': i - 5 if i - 5 in l else i, 'E': i + 1
 if i + 1  in l else i, 'S': i + 5 if i + 5 in l else i, 'W': i}
        elif i % 5 == 4:
            state_to_state_prime[i] = {'N': i - 5 if i - 5 in l else i, 'E': i, 'S
': i + 5 if i + 5 in l else i, 'W': i - 1 if i - 1 in l else i}
```

```
        else:

            state_to_state_prime[i] = {'N': i - 5 if i - 5 in l else i, 'E': i + 1
 if i + 1  in l else i, 'S': i + 5 if i + 5 in l else i, 'W': i - 1 if i - 1 in l
else i}


    return state_to_state_prime
```

```
b=float(input("Enter the value of b: "))

a = 1-b-0.5

float(a)

print("a= 1-b-0.5= ",a)

def create_random_policy():

    return {i: {'N': 0.0, 'E': 0.0, 'S': 0.0, 'W': 0.0} if i == 0 or i == 24 else
{'N': 0.2, 'E': b, 'S': 0.3, 'W': a} for                                 i in rang
e(25)} # [N, E, S, W]
```

OUTPUT:

```
Enter the value of b: 0.25
a= 1-b-0.5=  0.25
```

```
def create_probability_map():

    states = list(range(25))

    state_to_state_prime = create_state_to_state_prime_verbose_map()

    probability_map = {}

    for state in states:

        for move in ["N", "E", "S", "W"]:

            for prime in states:

                probability_map[(prime, -
1, state, move)] = 0  if prime != state_to_state_prime[state][move] else 1


    return probability_map
```

```
def agent(policy, starting_position=None, verbose=False):

    l = list(range(25))

    state_to_state_prime = create_state_to_state_prime_verbose_map()

    agent_position = randint(1, 22) if starting_position is None else starting_pos
ition

    step_number = 1

    action_taken = None

    if verbose:
```

```
        print("Move: {} Position: {} Action: {}".format(step_number, agent_position, action_taken))

        print_board(agent_position)

        print("\n")

        sleep(2)


    while not (agent_position == 0 or agent_position == 14 or agent_position == 21  or agent_position == 24):

        if verbose:

            clear_output(wait=True)

            print("Move: {} Position: {} Action: {}".format(step_number, agent_position, action_taken))

            print_board(agent_position)

            print("\n")

            sleep(1)


        current_policy = policy[agent_position]

        next_move = random()

        lower_bound = 0

        for action, chance in current_policy.items():

            if chance == 0:

                continue

            if lower_bound <= next_move < lower_bound + chance:

                agent_position = state_to_state_prime[agent_position][action]

                action_taken = action

                break

            lower_bound = lower_bound + chance


        step_number += 1


    if verbose:

        clear_output(wait=True)

        print("Move: {} Position: {} Action: {}".format(step_number, agent_position, action_taken))

        print_board(agent_position)
```

```
        print("Terminal State>>>>Win")
        print("Step number:")
    return step_number
```

```
data = []
for i in range(100):
    clear_output(wait=True)
    print("{}%\n".format((i + 1) / 10))
    data.append(agent(create_random_policy()))
print("Average steps to finish: {}".format(sum(data)/len(data)))
```

OUTPUT:

```
10.0%

Average steps to finish: 9.94
```

```
agent(create_random_policy(), verbose=True)
```

OUTPUT:

```
Move: 13 Position: 21 Action: S
----------------------
| X |   |   |   |   |
----------------------
|   |   |   |   |   |
----------------------
|   |   |   |   | X |
----------------------
|   |   |   |   |   |
----------------------
|   | A |   |   | X |
----------------------

Terminal State>>>>Win
Step number:
13
```

```
def create_greedy_policy(V_s):
    s_to_sprime = create_state_to_state_prime_verbose_map()
    policy = {}

    for state in range(25):

        state_values = {a: V_s[s_to_sprime[state][a]] for a in ['N', 'S', 'E', 'W'
]}

        if state == 0 or state == 14 or state== 21 or state==24:
```

```
            policy[state] = {'N': 0.0, 'E': 0.0, 'S': 0.0, 'W': 0.0}
#Terminal State>>NO movement req.


        else:

            max_actions = [k for k, v in state_values.items() if v == max(state_va
lues.values())]

            policy[state] = {a: 1 / len(max_actions) if a in max_actions else 0.0
for a in ['N', 'S', 'E', 'W']}

    return policy
```

```
def iterative_policy_evaluation(policy, theta=0.01, discount_rate=0.5):

    V_s = {i: 0 for i in range(25)}

    probablitiy_map = create_probability_map()


    delta = 100

    while not delta < theta:

        delta = 0

        for state in range(25):

            v = V_s[state]


            total = 0

            for action in ["N", "E", "S", "W"]:

                action_total = 0

                for state_prime in range(25):

                    action_total += probablitiy_map[(state_prime, -
1, state, action)] * (-1 + discount_rate * V_s[state_prime])

                total += policy[state][action] * action_total


            V_s[state] = round(total, 1)

            delta = max(delta, abs(v - V_s[state]))

    return V_s
```

```
print("Random Policy-Value Iteration Algorithm:")

policy = create_random_policy()

V_s = iterative_policy_evaluation(policy)

print(V_s)
```

```python
print("\nPolicy Iteration Algorithm:")

V_s = iterative_policy_evaluation(policy)

policy = create_greedy_policy(V_s)

print(V_s)
```

OUTPUT:

```
Random Policy-Value Iteration Algorithm:
{0: 0.0, 1: -1.7, 2: -1.9, 3: -1.9, 4: -1.9, 5: -1.7, 6: -1.9, 7: -1.9, 8: -1.9,
9: -1.9, 10: -1.9, 11: -1.9, 12: -1.9, 13: -1.9, 14: -1.9, 15: -1.9, 16: -1.9, 17:
-1.9, 18: -1.9, 19: -1.6, 20: -1.9, 21: -1.9, 22: -1.9, 23: -1.7, 24: 0.0}

Policy Iteration Algorithm:
{0: 0.0, 1: -1.7, 2: -1.9, 3: -1.9, 4: -1.9, 5: -1.7, 6: -1.9, 7: -1.9, 8: -1.9,
9: -1.9, 10: -1.9, 11: -1.9, 12: -1.9, 13: -1.9, 14: -1.9, 15: -1.9, 16: -1.9, 17:
-1.9, 18: -1.9, 19: -1.6, 20: -1.9, 21: -1.9, 22: -1.9, 23: -1.7, 24: 0.0}
```

```python
data = []

for i in range(100):

    clear_output(wait=True)

    print("{}%\n".format((i + 1) / 10))

    data.append(agent(policy))

print("Average steps to finish: {}".format(sum(data)/len(data)))
```

OUTPUT:

```
0.2%
```

```python
agent(policy, verbose=True)
```

OUTPUT:

```
Move: 1 Position: 6 Action: None
----------------------
| X |   |   |   |   |
----------------------
|   | A |   |   |   |
----------------------
|   |   |   |   | X |
----------------------
|   |   |   |   |   |
----------------------
|   | X |   |   | X |
----------------------
```

```python
create_state_to_state_prime_verbose_map()
```

OUTPUT:

```
{0: {'E': 0, 'N': 0, 'S': 0, 'W': 0},
 1: {'E': 2, 'N': 1, 'S': 6, 'W': 0},
 2: {'E': 3, 'N': 2, 'S': 7, 'W': 1},
 3: {'E': 4, 'N': 3, 'S': 8, 'W': 2},
```

```
 4: {'E': 4, 'N': 4, 'S': 9, 'W': 3},
 5: {'E': 6, 'N': 0, 'S': 10, 'W': 5},
 6: {'E': 7, 'N': 1, 'S': 11, 'W': 5},
 7: {'E': 8, 'N': 2, 'S': 12, 'W': 6},
 8: {'E': 9, 'N': 3, 'S': 13, 'W': 7},
 9: {'E': 9, 'N': 4, 'S': 14, 'W': 8},
 10: {'E': 11, 'N': 5, 'S': 15, 'W': 10},
 11: {'E': 12, 'N': 6, 'S': 16, 'W': 10},
 12: {'E': 13, 'N': 7, 'S': 17, 'W': 11},
 13: {'E': 14, 'N': 8, 'S': 18, 'W': 12},
 14: {'E': 14, 'N': 9, 'S': 19, 'W': 13},
 15: {'E': 16, 'N': 10, 'S': 20, 'W': 15},
 16: {'E': 17, 'N': 11, 'S': 21, 'W': 15},
 17: {'E': 18, 'N': 12, 'S': 22, 'W': 16},
 18: {'E': 19, 'N': 13, 'S': 23, 'W': 17},
 19: {'E': 19, 'N': 14, 'S': 24, 'W': 18},
 20: {'E': 21, 'N': 15, 'S': 20, 'W': 20},
 21: {'E': 22, 'N': 16, 'S': 21, 'W': 20},
 22: {'E': 23, 'N': 17, 'S': 22, 'W': 21},
 23: {'E': 24, 'N': 18, 'S': 23, 'W': 22},
 24: {'E': 0, 'N': 0, 'S': 0, 'W': 0}}
```

```
create_random_policy()
```

OUTPUT:

```
{0: {'E': 0.0, 'N': 0.0, 'S': 0.0, 'W': 0.0},
 1: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 2: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 3: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 4: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 5: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 6: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 7: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 8: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 9: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 10: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 11: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 12: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 13: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 14: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 15: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 16: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 17: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 18: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 19: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 20: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 21: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 22: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 23: {'E': 0.25, 'N': 0.2, 'S': 0.3, 'W': 0.25},
 24: {'E': 0.0, 'N': 0.0, 'S': 0.0, 'W': 0.0}}
```

- **QUESTION 3/4:** Calculate the optimal policy by implementing the value iteration algorithm. Discuss the impact of change in a and b on policy.

ANSWER: With an increase in the value of b(which means a decrease in the value of a); we see that the average steps taken decreases, i.e. the model takes lesser time to reach the closest terminal state.

- **QUESTION 5:**

Conceptual Question:

Difference between Value-Iteration Algorithm and Policy-Iteration Algorithm

POLICY ITERATION ALGORITHM:

- This algorithm manipulates the given random policy, instead of finding the optimal policy using the Optimal Value Function. Starting with the random policy, we find the value function of the given policy and then keep on finding a new optimised policy based on the previous value.
- Optimal Policy from given random Policy.

VALUE ITERATION ALGORTIHM:

- This algorithm manipulates the given random value function and then iterate over and over until we find a new, better optimal Value Function. We find the optimal Policy from the optimal Value Function.
- Optimal Policy from the Optimal Value Function.

****