CS 7320
HW1 Vacuum Cleaner Submission Template

Name: Charles Bryan
ID: 49350684

**Program Output:**

```
/Users/charlesbryan/anaconda3/bin/python
/Users/charlesbryan/Desktop/AI/401_1242/hw1.vacuum/VacWorld.py


Part1. Start in room R

Start State: ['R', [1, 1]]

cycle:0    action=SUCK           state=['R', [1, 0]]

cycle:1    action=MOVELEFT       state=['L', [1, 0]]

cycle:2    action=SUCK           state=['L', [0, 0]]

All rooms clean!


Part2. Start in room L

Start State: ['L', [1, 1]]

cycle:0    action=SUCK           state=['L', [0, 1]]

cycle:1    action=MOVERIGHT      state=['R', [0, 1]]

cycle:2    action=SUCK           state=['R', [0, 0]]

All rooms clean!


Process finished with exit code 0
```

## Program Code:

<paste text of your code here. Courier font. Follow code guidelines for spacing>

```
'''
VacWorld.py


This is code to get you started.
You need to write code for 3 functions:


    def getAction(state):
        given the current state, return either:
          'SUCK', 'MOVELEFT' or 'MOVERIGHT'
     - in R with dirt -> SUCK
     - in L with dirt -> SUCK
     - in R no dirt   -> MOVELEFT
     - in L no dirt   -> MOVERIGHT


    def updateState(state, action)
        return the next state given current state and action
            action:SUCK vacRoom:R -> ['R', [.., 0]]
            action:SUCK vacRoom:L -> ['L', [0, ..]]
            action:MOVELEFT  vacRoom:R -> ['L', .. ]
            action:MOVERIGHT vacRoom:L -> ['R', .. ]
            other state action pairs: -> no change to state
            note: in above, ... means no change


    def bool_all_rooms_clean(state):
        #returns True or False  if all room are clean or not
'''
```

```
# add your name and ID here    --------------

Name = 'Charles Bryan'

ID = '49350684'



def state_validation(state):

    '''

    Function to validate the state

    '''

    if not isinstance(state, list):

        raise ValueError("Invalid state format: State must be a list")

    elif not len(state) == 2:

        raise ValueError("Invalid state format: The length of state
must be 2.")

    elif not state[0] == 'R' and not state[0] == 'L':

        raise ValueError("Invalid state format: The position must be
'R' or 'L'.")

    elif len(state[1]) != 2:

        raise ValueError("Invalid state format: The dirt info must
have a length of 2.")

    elif not (state[1][0] == 0 or state[1][0] == 1):

        raise ValueError("Invalid state format: The left dirt info
must be '0' or '1'.")

    elif not (state[1][1] == 0 or state[1][1] == 1):

        raise ValueError("Invalid state format: The right dirt info
must be '0' or '1'.")



def getAction(state):
```

```
    '''state: current state
       return  the action based on state
    '''
    state_validation(state)
    position, [dirt_in_L, dirt_in_R] = state

    if position == 'L' and dirt_in_L:
        return 'SUCK'
    elif position == 'R' and dirt_in_R:
        return 'SUCK'

    if position == 'L':
        return 'MOVERIGHT'
    elif position == 'R':
        return 'MOVELEFT'


def updateState(state, action):
    ''' state : current state
        action : current action
        return : next state
    '''
    new_state = state
    position, [dirt_in_L, dirt_in_R] = state

    if action == 'SUCK':
        if position == 'L':
            new_state = [position, [0, dirt_in_R]]
```

```
        elif position == 'R':
            new_state = [position, [dirt_in_L, 0]]
    else:
        if action == 'MOVERIGHT':
            new_state = ['R', [dirt_in_L, dirt_in_R]]
        elif action == 'MOVELEFT':
            new_state = ['L', [dirt_in_L, dirt_in_R]]
    return new_state


def bool_all_rooms_clean(state):
    '''returns True or False if all room are clean or not'''
    state_validation(state)
    position, [dirt_in_L, dirt_in_R] = state
    return not (dirt_in_L or dirt_in_R)

def run_vacuum(start_state):
    state = start_state
    loop_count = 0  # counter to stop program when it goes wrong

    # run the simulation until all rooms clean
    while not bool_all_rooms_clean(state):
        action = getAction(state)
        state = updateState(state, action)

        # show state and stop if too many loops
        print(f"cycle:{loop_count}\taction={action}\t\tstate={state}")
        loop_count += 1
```

```
        if loop_count > 20:
            print(
                "Uh oh ! Code stuck in a state ... program
terminated")
            return

    print("All rooms clean!")
    return


def main():
    # when you run this code without implementing above functions
    # the program will stop after 20 cycles
    # Part 1: Start in room R
    start_state = ['R', [1, 1]]
    print(f"\nPart1. Start in room R\nStart State: {start_state}")
    run_vacuum(start_state)


    # Part 2: Start in room L
    start_state = ['L', [1, 1]]
    print(f"\nPart2. Start in room L\nStart State: {start_state}")
    run_vacuum(start_state)


if __name__ == '__main__':
    main()
```