Koushik Sahu
118CS0597
Artificial Intelligence Lab-1
Output file
Date: 1st Sept 2021(Wednesday)

1.

Code:

```
import math, statistics, collections

class Environment:
    def __init__(self, curr_location, dirt, dirt_size=1, suck_score=5, move_time=1):
        self.dirt = {'a': True if dirt[0].lower() == 'dirty' else False,
                'b': True if dirt[1].lower() == 'dirty' else False}
        self.curr_location = curr_location.lower()
        self.score = 0
        self.scoring_system = {
            'suck': suck_score,
            'dirt_size': dirt_size,
            'move_time': move_time
        }

    def move(self):
        if self.dirt[self.curr_location]:
            return self.suck()
        else:
            if self.curr_location == 'a':
                return self.right()
            else:
                return self.left()

    def performance_score(self):
        return self.score

    def suck(self):
        self.score += self.scoring_system['suck']
        self.score += self.scoring_system['dirt_size']
        self.dirt[self.curr_location] = False
        return f'Suck'

    def left(self):
        self.curr_location = 'a'
        self.score -= self.scoring_system['move_time']
```

```
        return f'Left'

    def right(self):
        self.curr_location = 'b'
        self.score -= self.scoring_system['move_time']
        return f'Right'

def valid(actions):
    if len(actions)>=2:
        return actions[-1]=='Suck' or actions[-2]=='Suck'
    return True

if __name__ == '__main__':
    print('Enter starting location(a/b): ', end='')
    loc = input()
    assert(loc=='a' or loc=='b')
    print('Is location A dirty?(y/n): ', end='')
    a_dirty = input()
    assert(a_dirty=='y' or a_dirty=='n')
    print('Is location B dirty?(y/n): ', end='')
    b_dirty = input()
    assert(b_dirty=='y' or b_dirty=='n')
    dirt = list()
    dirt.append('dirty' if a_dirty=='y' else 'clean')
    dirt.append('dirty' if b_dirty=='y' else 'clean')
    env = Environment(loc, dirt)
    action = list()
    while valid(action):
        action.append(env.move())
    print(f'Actions: {action}')
    print(f'Performance score: {env.performance_score()}')
```

Output

```
Enter starting location(a/b): a
Is location A dirty?(y/n): y
Is location B dirty?(y/n): y
Actions: ['Suck', 'Right', 'Suck', 'Left', 'Right']
Performance score: 9
```

2.
Code

```
import math, statistics, collections
```

```
class Environment:
    def __init__(self, curr_location, dirt, dirt_size=1, suck_score=5, move_time=1):
        self.dirt = {'a': True if dirt[0].lower() == 'dirty' else False,
                'b': True if dirt[1].lower() == 'dirty' else False}
        self.curr_location = curr_location.lower()
        self.score = 0
        self.scoring_system = {
            'suck': suck_score,
            'dirt_size': dirt_size,
            'move_time': move_time
        }

    def move(self):
        if self.dirt[self.curr_location]:
            return self.suck()
        else:
            if self.curr_location == 'a':
                return self.right()
            else:
                return self.left()

    def performance_score(self):
        return self.score

    def suck(self):
        self.score += self.scoring_system['suck']
        self.score += self.scoring_system['dirt_size']
        self.dirt[self.curr_location] = False
        return f'Suck'

    def left(self):
        self.curr_location = 'a'
        self.score -= self.scoring_system['move_time']
        return f'Left'

    def right(self):
        self.curr_location = 'b'
        self.score -= self.scoring_system['move_time']
        return f'Right'

def valid(actions):
    if len(actions)>=2:
        return actions[-1]=='Suck' or actions[-2]=='Suck'
    return True
```

```
if __name__ == '__main__':
    curr_location_pos = ['a', 'b']
    dirt_pos = [('clean', 'clean'),
            ('clean', 'dirty'),
            ('dirty', 'clean'),
            ('dirty', 'dirty')]
    for loc in curr_location_pos:
        for dirt in dirt_pos:
            env = Environment(loc, dirt)
            actions = list()
            while valid(actions):
                actions.append(env.move())
            print(f'Starting location: {loc}')
            print(f'Dirt position: {dirt}')
            print(f'Actions: {actions}')
            print(f'Performance measure: {env.performance_score()}')
```

Output:

```
Starting location: a
Dirt position: ('clean', 'clean')
Actions: ['Right', 'Left']
Performance measure: -2
Starting location: a
Dirt position: ('clean', 'dirty')
Actions: ['Right', 'Suck', 'Left', 'Right']
Performance measure: 3
Starting location: a
Dirt position: ('dirty', 'clean')
Actions: ['Suck', 'Right', 'Left']
Performance measure: 4
Starting location: a
Dirt position: ('dirty', 'dirty')
Actions: ['Suck', 'Right', 'Suck', 'Left', 'Right']
Performance measure: 9
Starting location: b
Dirt position: ('clean', 'clean')
Actions: ['Left', 'Right']
Performance measure: -2
Starting location: b
Dirt position: ('clean', 'dirty')
Actions: ['Suck', 'Left', 'Right']
Performance measure: 4
Starting location: b
Dirt position: ('dirty', 'clean')
Actions: ['Left', 'Suck', 'Right', 'Left']
Performance measure: 3
Starting location: b
Dirt position: ('dirty', 'dirty')
Actions: ['Suck', 'Left', 'Suck', 'Right', 'Left']
Performance measure: 9
```

3.
Code

```
import math, statistics, collections, copy

class ConnectFour:
    def __init__(self, row=6, column=7):
        assert(row>=4 and column>=4)
        self.r = row
        self.c = column
        self.turn = 1
        self.board = [[0 for i in range(self.c)] for i in range(self.r)]

    def display(self):
        print('Current board configuration')
        for i in range(0, self.c):
            print(i, end='')
        print()
        for i in range(0, self.c):
            print('-', end='')
        print()
        for i in range(0, self.r):
            for j in range(0, self.c):
                if self.board[i][j] == 0:
                    print('*', end='')
                else:
                    print(self.board[i][j], end='')
            print('')

    def result(self):
        for i in range(0, self.r):
            for j in range(0, self.c-3):
                if self.board[i][j] != 0 and \
                    self.board[i][j] == self.board[i][j+1] and \
                    self.board[i][j+1] == self.board[i][j+2] and \
                    self.board[i][j+2] == self.board[i][j+3]:
                        return self.board[i][j]

        for j in range(0, self.c):
            for i in range(0, self.r-3):
                if self.board[i][j] != 0 and \
                    self.board[i][j] == self.board[i+1][j] and \
                    self.board[i+1][j] == self.board[i+2][j] and \
                    self.board[i+2][j] == self.board[i+3][j]:
                        return self.board[i][j]
```

```
        for i in range(0, self.r-3):
            for j in range(0, self.c-3):
                if self.board[i][j] != 0 and \
                    self.board[i][j] == self.board[i+1][j+1] and \
                    self.board[i+1][j+1] == self.board[i+2][j+2] and \
                    self.board[i+2][j+2] == self.board[i+3][j+3]:
                        return self.board[i][j]


        for i in range(3, self.r):
            for j in range(0, self.c-3):
                if self.board[i][j] != 0 and \
                    self.board[i][j] == self.board[i-1][j+1] and \
                    self.board[i-1][j+1] == self.board[i-2][j+2] and \
                    self.board[i-2][j+2] == self.board[i-3][j+3]:
                        return self.board[i][j]

        return 0

    def move(self, col):
        assert(col>=0 and col<self.c)
        move_made = False
        for i in range(self.r-1, -1, -1):
            if self.board[i][col] == 0:
                move_made = True
                self.board[i][col] = self.turn
                self.turn = 1 if self.turn == 2 else 2
                break
        return move_made

def play(first_turn):
    cf = ConnectFour()
    cf.turn = first_turn
    while cf.result() == 0:
        cf.display()
        if cf.turn == 1:
            move_made = False
            while not move_made:
                print('Enter column number to drop you disc into: ', end='')
                col = int(input())
                valid_move = cf.move(col)
                move_made = valid_move
                if not valid_move:
```

```
                print('Invalid move as desired column is filled already! Enter again')
        else:
            possible_col = list()
            winning_col = list()
            for i in range(0, cf.c):
                cf_copy = copy.deepcopy(cf)
                valid_move = cf_copy.move(i)
                if valid_move:
                    if cf_copy.result() == 2:
                        winning_col.append(i)
                    else:
                        possible = True
                        for j in range(0, cf.c):
                            cf_copy1 = copy.deepcopy(cf_copy)
                            valid_move = cf_copy1.move(j)
                            if valid_move and cf_copy1.result() == 1:
                                possible = False
                        if possible: possible_col.append(i)
            move_col = -1
            if len(winning_col) != 0:
                move_col = winning_col[0]
            elif len(possible_col) != 0:
                move_col = possible_col[0]
            else:
                move_col = 0
            cf.move(move_col)
            print(f'Agent inserts disc to column {move_col}')
    cf.display()
    return cf.result()

if __name__ == '__main__':
    print('Do you want to play first?(y|n): ', end='')
    response = input()
    turn = 1 if response.lower() == 'y' else 0
    res = play(turn)
    if res == 1:
        print('Human wins     ')
    else:
        print('Agent wins     ')
```

Output:

```
Press ENTER or type command to continue
Do you want to play first?(y|n): y
Current board configuration
0123456
-------
*******
*******
*******
*******
*******
*******
Enter column number to drop you disc into: 0
Current board configuration
0123456
-------
*******
*******
*******
*******
*******
1******
Agent inserts disc to column 0
Current board configuration
0123456
-------
*******
*******
*******
*******
2******
1******
Enter column number to drop you disc into: 1
Current board configuration
0123456
-------
*******
*******
*******
*******
2******
11*****
Agent inserts disc to column 0
Current board configuration
0123456
-------
*******
*******
*******
2******
```

```
*******
*******
*******
2******
2******
1112***
Enter column number to drop you disc into: 1
Current board configuration
0123456
-------
*******
*******
*******
2******
21*****
1112***
Agent inserts disc to column 0
Current board configuration
0123456
-------
*******
*******
2******
2******
21*****
1112***
Enter column number to drop you disc into: 1
Current board configuration
0123456
-------
*******
*******
2******
21*****
21*****
1112***
Agent inserts disc to column 0
Current board configuration
0123456
-------
*******
2******
2******
21*****
21*****
1112***
Agent wins🤖
```