# Artificail Intelligence assignment3 Report *2016025514 서현아*

# "Q-Learning 기법을 이용하여 Frozen Lake 목표 지점까지 이동하기"

## 1. 문제 해결 방식

1. Learning

   주어지는 Training Data Set 없이, 프로그램 자체적으로 취하는 action을 통해서 reward를 받아가며 학습하는 방식.

   구덩이에 빠지는 경우는 -1, 목표 지점에 도착하는 경우에는 +1의 reward를 주면서 Q-table을 제작함.

   Q value를 정하는 식은 Q(current state, current action) = reward + 0.5 * maxarg(Q(next state, next action))으로함.

## 2. Source Code (추가적인 설명은 함수 설명 아래의 코드 주석 참고)

1. openFile(파일명) : 인자로 받은 파일 (Frozen Lake) 을 열어서 첫번째 줄은 해당 Frozen lake의 version, 세로 길이, 가로 길이의 정보를 담은 lakeInfo 변수에 저장하고, 그 이후의 Frozen Lake자체는 개행 문자를 제거하여 frozen_lake라는 변수에 넣는 함수. frozen_lake와 lakeInfo를 반환한다.

```
def openFile(filename): #인자 : 열어야 할 파일이름
    frozen_lake = []

    #파일을 임시로 저장하는 변수 : tmp_lake
    tmp_lake = list(open(filename, 'r', encoding="UTF-8"))

    #tmp_lake의 첫번째 줄에서 개행문자를 제거하고 정보를 받는 변수 : lakeInfo
    lakeInfo = tmp_lake[0].rstrip('\n')
    #tmp_lake를 frozen_lake로 만들기 위해서
    #lakeInfo의 내용은 tmp_lake에서 제거함
    tmp_lake.pop(0)

    #tmp_lake의 각 줄을 frozen_lake라는 리스트 변수에 저장함
    for line in tmp_lake:
        frozen_lake.append(line.rstrip('\n'))

    #frozen_lake와 lakeInfo를 반환
    return frozen_lake, lakeInfo
```

2. initQtable(lakeInfo) : 위 함수에서 반환한 lakeInfo를 이용하여, frozen lake 파일의 버전 정보, 가로 길이, 세로 길이를 변수에 저장하고, 총 state의 개수를 구한다.

   version 정보는 최종적으로 path를 기록한 frozen lake를 txt 파일로 저장할 때, 파일 명을 위한 정보이다.

또한 Qtable 이라는 2차원 배열을 선언하며 0으로 전체를 초기화 시켜준다. 이때 Qtable의 세로 길이는 state의 개수이며, 각 state 당 상/하/좌/우를 의미하는 4가지 요소를 갖고 있다.

```
def initQtable(lakeInfo):
    #lakeInfo를 띄어쓰기 기준으로 잘라서 Info에 넣는다.
    Info = lakeInfo.split();

    #첫번째 요소 : 버전 정보
    version = int(Info[0])
    #두번째 요소 : frozen lake의 세로 길이
    height = int(Info[1])
    #세번째 요소 : frozen lake의 가로 길이
    width = int(Info[2])

    #state 개수 : 가로길이 x 세로길이
    state_num = int(Info[1]) * int(Info[2])

    #Qtable : state_num by 4 의 2차원 배열
    #내부를 모두 0으로 초기화
    Qtable = [[0 for col in range(4)] for row in range(state_num)]

    #버전 정보, state 개수, 초기화한 Qtable, 세로 길이와 가로 길이를 반환
    return version, state_num, Qtable, height, width
```

3. findStart(frozen_lake, height, width) : frozen_lake를 이중 for문으로 탐색하면서 시작점을 반환하는 함수.

```
def findStart(frozen_lake, height, width):
    for i in range(height):
        for j in range(width):
            #'S' 이면 시작점
            if frozen_lake[i][j] == 'S':
                #state의 번호는 (가로길이 -1) + (세로길이 -1)*가로길이
                return i*width + j
```

4. fidnGoal(frozen_lake, height, width) : frozen_lake를 이중 for문으로 탐색하면서 도착점을 반환하는 함수

```
def findGoal(frozen_lake, height, width):
    for i in range(height):
        for j in range(width):
            #'G'이면 goal
            if frozen_lake[i][j] == 'G':
                #state의 번호는 (가로길이 -1) + (세로길이 -1)*가로길이
                return i*width + j
```

5. Qlearning(Qtable, frozen_lake, height, width) : frozen_lake를 Q-learning하면서 Qtable을 작성하는 함수

```
def Qlearning(Qtable, frozen_lake, height, width):
   #총 30000번의 learning을 통해서 학습을 진행하며 Qtable을 작성함
    num_trial = 30000

   #중간에 일종의 loop (회전하는 상황) 에 빠지는 경우를 거르기 위해서
   #각 action별로 최대 step은 99회로 제한함
    max_step = 99

   #alpha value는 0.5로 지정함
    discount = 0.5

   #3000번의 learning을 취하게 하는 for 문
    for trial in range(num_trial):
       #step 수 초기화
        step = 0

       #Goal이나 구덩이를 만나면 True로 변경하여
       #각 action에 대한 for 문을 break함
        done = False

       #각 시행에 대한 전체 reward를 저장하는 변수 : total_reward
        total_reward = 0

       #첫번째 state는 항상 start지점
       #위에서 선언한 findStart 함수를 이용하여 시작 지점을 찾음
        state = findStart(frozen_lake, height, width)

       #한 번의 learning이 최대 step 수 99 동안 진행(action)되게 하는 for 문
        for step in range(max_step):
            #reward를 0으로 초기화
            reward = 0

            #현재의 state에 대해 좌/우/상/하의 Q value 리스트 : lrud
            lrud = Qtable[state]

           #현재 state에서 갈 수 있는 경로는 좌/우/상/하
           #이 중 벽으로 가는 경우를 제거하고
           #나머지 중 해당 action을 취했을 때의 Q value들 중 최대값이 있는 방향으로
           #action을 취할 수 있도록 이동 방향을 임시로 저장해두는 변수 : tmp_direction_index

            #0 : left, 1 : right, 2 : up, 3 : down
            tmp_direction_index = [0,1,2,3]
            #벽으로 가는 경우 제거
```

```
            #왼쪽 벽
            if state % width == 0:
                tmp_direction_index.remove(0)
            #오른쪽 벽
            if state % width == width-1:
                tmp_direction_index.remove(1)
            #가장 위의 벽
            if state // width == 0:
                tmp_direction_index.remove(2)
            #가장 아래의 벽
            if state // width == height-1:
                tmp_direction_index.remove(3)

            #lrud 리스트 중 최대값을 저장하는 변수 : tmp_max
            tmp_max = max(lrud)

            #이동이 가능한 방향들만 남긴 리스트 변수 : possible_directoin_index
            possible_direction_index = tmp_direction_index

            #위쪽에서 구한 tmp_max와 동일한 값을 갖고 있는 방향만
            #possible_direction_index에 남기기
            for possible_direction in tmp_direction_index:
                if lrud[possible_direction] != tmp_max:
                    possible_direction_index.remove(possible_direction)

            #이유 : max 값을 갖고 있는 방향이 여러 개일 때
            #단순하게 max(index)를 해버리면 항상 첫번째 최댓값으로 결정됨
            #초기에는 Qtable이 모두 0으로 초기화 되어있기 때문에
            #max 값을 갖고 있는 방향들 중 random하게 이동방향을 결정해야 학습이 진행됨
            action = random.choice(possible_direction_index)

            #0 : left
            if action == 0:
                #새로운 state는 현재 state에서 좌측으로 한 칸 이동
                new_state = state - 1

            #1 : right
            elif action == 1:
                #새로운 state는 현재 state에서 우측으로 한 칸 이동
                new_state = state + 1

            #2 : up
            elif action == 2:
                #새로운 state는 현재 state에서 윗줄로 이동
                new_state = state - width

```

```
            #3 : down
            elif action == 3:
                #새로운 state는 현재 state에서 아랫줄로 이동
                new_state = state + width

            #new state의 frozen lake에서의 좌표 구하기
            new_state_width = new_state % width
            new_state_height = new_state // width

            #new state 살펴보기!
            #구덩이에 빠진 경우, reward -1, 해당 step 종료, done을 True로 변경
            if frozen_lake[new_state_height][new_state_width] == 'H':
                reward = -1
                done = True

            #목적지에 도착한 경우, reward +1, 해당 step 종료, done을 Truen로 변경
            if frozen_lake[new_state_height][new_state_width] == 'G':
                reward = 1
                done = True

            #그냥 정상적인 길이거나 시작점일 경우 reward는 0, 해당 learning을 지속
            if frozen_lake[new_state_height][new_state_width] == 'F' or frozen_lake[new_state_height][new_state_width] == 'S':
                reward = 0
                done = False

            #Q(state, action) = reward + 0.5 * maxarg(Q(next state, next action)) 을 수행
            Qtable[state][action] = reward + discount * max(Qtable[new_state])

            total_reward = total_reward + reward

            #다음번 for loop의 state는 새로운 state가 되어야함
            state = new_state

            #구덩이를 만나거나 목적지에 도착했으면 해당 learning은 종료, 다음 learning으로 이동
            if done == True:
                break;

    #30000번의 학습이 완료된 Qtable을 반환함
    return Qtable
```

6. findPath(result_Qtable, frozen_lake, height, width) : 5번의 Qlearning 함수를 통해 반환된 Qtable을 result Qtable이라고 보고, 출발지에서 목적지까지 구덩이에 빠지지 않고 가는 경로를 찾는 함수.

   Qtable을 이용하여 경로를 결정하게 되는데, 이 때 Q value가 큰 방향으로 이동한다.

```
def findPath(result_Qtable, frozen_lake, height, width):
    #경로를 저장할 리스트 : path
    path = []

    #출발 지점
    position = findStart(frozen_lake, height, width)
    #목적 지점
    goal = findGoal(frozen_lake, height, width)

    #경로 리스트에 출발지를 추가함
    path.append(position)

    #목적지에 도착하기 전까지 경로 리스트에 경로를 추가함
    while True:
       #다음으로 이동할 위치는 result_Qtable의 현재 postion state의 상/하/좌/우 action에 대한 Q value들 중 최대값을 지닌 방향
        new_position_direction = result_Qtable[position].index(max(result_Qtable[position]))

        #이동 방향이 left
        if new_position_direction == 0:
            position = position - 1
        #이동 방향이 right
        elif new_position_direction == 1:
            position = position + 1
        #이동 방향이 up
        elif new_position_direction == 2:
            position = position - width
        #이동 방향이 down
        elif new_position_direction == 3:
            position = position + width

        #목적지에 도착한 경우
        if position == goal:
            #경로 리스트에 목적지를 추가하고
            path.append(position)
            #while 문을 종료함
            break

        #목적지에 도착하지 않은 경우
        #경로에 이동한 position을 추가함
        path.append(position)

```

```
    #경로를 반환함
    return path
```

7. makeResultFrozenLake (frozen_lake, path, height, width) : 위의 함수가 반환한 경로를 frozen_lake에 적용 하여 경로를 'R'로 표현하는 함수. `frozen_lake[i][j] = 'R'`을 사용했더니 계속 오류가 발생해서 결국 frozen_lake에서 path에 해당하지 않는 요소들과 path의 요소들을 순서를 지키며 하나의 list (lake_list)에 담아두고 다시 frozen_lake의 가로, 세로 길이를 지키며 result_frozen_lake를 만들었다.

```
def makeResultFrozenLake(frozen_lake, path, height, width):
    lake_list = []
    path.pop(0)
    path.pop(len(path)-1)

    for i in range(height):
        for j in range(width):
            if i*width + j not in path:
                lake_list.append(frozen_lake[i][j])
            elif i*width+j in path:
                lake_list.append('R')

    result_frozen_lake = []

    while len(lake_list) != 0:
        tmp = lake_list[:width]
        result_frozen_lake.append(tmp)
        lake_list = lake_list[width:]

    return result_frozen_lake
```

8. makeResultFile(result_frozen_lake, lakeInfo, height, width) : 위의 함수의 결과인 result_frozen_lake를 txt 파일로 저장하는 함수

```
def makeResultFile(result_frozen_lake, lakeInfo, height, width):
    Info = list(lakeInfo.split())
    version = Info[0]

    res_file_name = "FrozenLake_" + version + "_output.txt"
    result_file = open(res_file_name, 'w', encoding="UTF-8")

    result_file.write(lakeInfo)
    result_file.write('\n')

    for i in range(height):
        for j in range(width):
            result_file.write(result_frozen_lake[i][j])
```

```
14          result_file.write('\n')
```

9. main 함수

```
if __name__ == '__main__':
    frozen_lake, lakeInfo = openFile('FrozenLake_3.txt')
    version, state_num, Qtable, height, width = initQtable(lakeInfo)
    result_Qtable = Qlearning(Qtable, frozen_lake, height, width)

    path = findPath(result_Qtable, frozen_lake, height, width)
    result_frozen_lake = makeResultFrozenLake(frozen_lake, path,
height, width)

    makeResultFile(result_frozen_lake, lakeInfo, height, width)
```

## 3. Result

1. version 1


2. version 2


3. version 3

```
3 11 4
HFRG
FFRH
FHRF
HHRF
FHRR
HSHR
FRHR
HRRR
FHFF
HHFH
FHFH
```