**Artificial Intelligence Assignment2 Report**

2016025078 강덕영

assignment2는 Q-learning을 사용하여 시작지점에서 목표지점까지 최적의 경로로 이동하는 알고리즘을 구현하는 것입니다.

**-Code Explanation**

알고리즘은 2016025078\_assignment\_2.py 에 구현되어있습니다. 해당 소스코드는 맵의 각 칸을 나타내는 state 클래스와, 맵을 Q-learning으로 학습한 후 시작부터 끝까지의 최적의 경로를 찾는 main 함수로 이루어져 있습니다.

**State Class**

우선, 맵의 각 칸을 하나의 state 라고 생각하였고, 이를 구현하기 위해 State 클래스를 선언하였습니다. 만약 5x5 맵이라면 1번부터 25번까지 총 25개의 state 객체가 생성됩니다. 객체가 생성될 때 초기화되는 변수들은 다음과 같습니다.

*1. valid* 배열

valid 배열은 특정 칸에서 이동 가능한 방향을 저장한 배열입니다. 맵은 NxN 이차원 배열이며, 특정 칸에서 상, 하, 좌, 우로 이동할 수 있습니다. 하지만 모든 칸에서 상, 하, 좌, 우로 이동할 수 없습니다. 예를 들어, 가장 왼쪽 위의 칸에선 오른쪽과 아랫쪽으로, 가장 오른쪽 위의 칸에선 왼쪽과 아랫쪽으로만 이동할 수 있습니다. 벽을 뚫고 이동할 순 없기 때문입니다. 따라서 state 에 따라 현재의 위치에서 움직일 수 있는 방향이 다르기 때문에, 이를 나타내기 위해 valid 배열을 사용합니다.

상, 하, 좌, 우를 각각 0, 1, 2, 3으로 나타내었습니다. 만약 어떤 state의 valid 배열이 [1, 3] 이라면, 해당 state에서는 아래쪽과 오른쪽으로만 이동할 수 있다는 뜻입니다.

*2. actions* 배열

actions 배열은 state에서 특정 방향으로 이동했을 때의 Q value 값을 가집니다. 만약 어떤 state의 actions 배열이 [-1, 0, 1, 2] 이라면, 위로 움직였을 때의 Q value 값은 -1, 아래로 움직였을 때의 Q value 값은 0 이라는 것입니다. 즉, actions 배열의 네개의 값은 각각 상, 하, 좌, 우로 움직였을 때의 Q value 값을 의미합니다. (물론 이 예시의 state는 상, 하, 좌, 우로 다 이동 가능하다고 가정한 것이며, 이때 valid 배열은 [0, 1, 2, 3] 일 것입니다.) actions 배열은 Q table과 동일합니다.

state에 따라 valid, actions 배열이 다르게 초기화되며, 생성한 state 객체는 states 배열에 추가하여 이후에 접근하게 됩니다.

**Main**

파일 입력, Q learning, path finding, 파일 출력의 모든 과정을 main 함수에 구현하였습니다.

우선, input.txt 파일을 읽어와서 rewards 배열을 초기화합니다. rewards 배열은 state의 보상 값을 가지고 있으며, state는 1번부터 N\*N 까지 존재하므로 rewards배열의 index도 1부터 N\*N까지 존재합니다. rewards[1]은 1번 state의 보상값입니다.

그리고 N\*N개의 state 객체를 생성하고 states 배열에 추가합니다. states의 배열의 index도 1부터 N\*N까지 존재합니다. states[2]는 2번 state 객체입니다.

rewards, states 배열의 초기화가 끝나면, Q-learning을 시작합니다. 총 학습은 백만 번 이루어지며, 각 학습마다 최대 천번의 탐색이 허용됩니다. explorations 변수가 총 학습의 수를, steps 변수는 학습마다 최대 반복 수를 나타냅니다. (컴퓨터 사양에 따라 학습시간이 오래 걸릴 수 있습니다)

학습의 시작은 1번 state에서 시작하여 랜덤한 방향으로 이동하며 state의 Q value를 갱신하는 방식으로 이루어집니다. (모든 state에서 상, 하, 좌, 우로 이동 가능한 것은 아닙니다. 따라서 state마다 움직일 수 있는 방향을 나타내기 위해 valid 배열을 초기화 했었습니다.) 우선 state의 valid 배열에서 무작위로 값을 꺼내서 이동할 다음 state(next\_state라고 하겠습니다)를 결정합니다. 그리고 next\_state의 Q value 중 최대 값을 구합니다. 이렇게 next\_state와 next\_state의 최대 Q value 값을 구하게 되면, 현재 state의 Q value 값을 갱신할 수 있습니다. Q-learning에 따라, next\_state의 보상값 + decaying factor(0.9) \* next\_state의 최대 Q value 값으로 갱신합니다. 그리고 next\_state로 이동 후에 이 과정을 반복하게 됩니다. 만약 이 과정을 반복하다 goal state나 bomb에 도달하게 되면 하나의 학습이 종료되고, 학습을 처음부터 다시 시작하게 됩니다.

학습이 진행되면서 모든 state의 Q value 값이 갱신됩니다. 다시 말해, state의 actions 배열의 원소들이 갱신됩니다. 학습이 끝나면, state 마다 상, 하, 좌, 우로 이동했을 때 각각의 Q value 값이 수렴됩니다. 따라서, Q value 값 중 가장 큰 값에 해당하는 방향으로 이동하는 것이 path finding의 규칙이 됩니다. 그러면 특정 state의 다음 state들 중 가장 reward가 큰 쪽으로 이동하게 되므로 optimal path를 찾게 됩니다.

마지막으로, path와 1번 state의 Q value 최대 값을 output.txt 파일에 쓰고 main 함수는 종료됩니다.

**-Code Analysis**

*맵의**보너스**지점의**보상**값에**따른**결과**값*

input.txt

SPPBP

BPTPB

PPBPP

BBTPP

BPPPG

1. reward = 1

path finding => 0 1 6 7 8 13 18 23 24

first state’s max Q value => 48.639690000000016

2. reward = 10

path finding => 0 1 6 7 8 13 18 23 24

first state’s max Q value => 55.92969000000001

3. reward = 20

path finding => ….. 3 8 3 8 3 8 3 8….. fail

first state’s max Q value => 85.2631578947368

보너스 지점의 보상 값이 커지면 커질 수록, 시작 state의 Q value 최대값은 증가합니다. 이는 Q-learning 알고리즘에 의해 당연한 것입니다. 보상 값이 1과 10일 때, [3][2]에 위치한 보너스 지점을 지나지 않는 이유는 goal state 방향으로 이동했을 때보다 보상 값이 작다고 판단하기 때문입니다. 실제로 [3][3]에 위치한 state의 Q value를 확인해보면, 보너스 지점으로 향하는 왼쪽 방향보다 오른쪽 방향의 Q value 값이 더 큼을 확인할 수 있습니다. 그런데 보상값이 20일 때, 특정 state에서 왔다 갔다하며 goal state까지 도달하지 못하는 상황이 발생합니다. 만약 특정 칸의 보상값이 많이 크면 그 주변 칸의 Q value 값도 많이 커지게 됩니다. 그러면 주변 칸을 거쳐 다시 해당 칸으로 이동해서 큰 보상값을 받으려고 하기 때문에 goal state 방향으로 이동하지 않고 특정 부분에 갇히는 현상이 발생합니다.

*학습 수에 따른 Q-learning 결과의 차이*

goal state까지의 거리가 멀고 bomb이 많을 수록 학습 도중 bomb state에 도달하여 학습이 중단되는 경우가 많습니다. 그러면 특정 state의 Q value가 갱신되지 않은 채 학습이 종료되는 경우가 생겨서 이후에 path finding이 되지 않는 현상이 존재합니다. 따라서 학습 반복 수를 늘림으로써 Q value를 수렴시킵니다.