**Artificial Intelligence Assignment02 Report**

컴퓨터소프트웨어학부 박제우

1. 함수 및 코드 설명

제출한 코드에서는 하나의 main함수에서 파일입력, Q table 초기화, Q learning, 결과출력 모두를 수행하도록 하였다. 각각의 코드에 대한 설명은 다음과 같다.

(1) 파일 입력 및 변수 초기화

1 with open("input.txt", "r") as inputFile:

2 inputLines = inputFile.readlines()

3 inputArr = []

4 mapArr = [] # 2차원 배열 inputArr을 1차원 배열로 나타낸 것

5 MAP\_SIZE = 5 # 5\*5 크기 기준

6 qTable = [{} for \_ in range(MAP\_SIZE \* MAP\_SIZE)]

7

8 BOMB\_REWARD = -100

9 GOAL\_REWARD = 100

10 BONUS\_REWARD = 1

11 gamma = 0.9

12

13 startPos = 0

14 goalPos = 24

15

16 for inputLine in inputLines:

17 inputArr.append(list(inputLine.rstrip()))

18

19 for i in range(MAP\_SIZE):

20 for j in range(MAP\_SIZE):

21 if inputArr[i][j] == "S":

22 startPos = i \* MAP\_SIZE + j

23 elif inputArr[i][j] == "G":

24 goalPos = i \* MAP\_SIZE + j

25 mapArr.append(inputArr[i][j])

line 2-13에서는 필요한 변수/배열들을 선언 및 초기화하고 있다. 입력되는 맵을 저장하는 배열inputArr과 mapArr 두 개가 있는데, 처음에 inputArr에 이차원 배열로 입력받았다가 이후 연산을 편리하게 하도록 1차원 배열로 바꿔서 mapArr에 저장한다. inputArr[i][j]는 i행 j열에 대응하며 mapArr[MAP\_SIZE \* i+j]에 해당한다.

MAP\_SIZE는 맵의 가로 세로 크기를 의미하며 여기서는 5\*5 맵을 입력으로 받으므로 5로 지정한다. 또한 qTable에서 qTable[i]는 mapArr[i]에 해당하는 Q table을 딕셔너리 형태로 가지고 있다. 수업에서는 Q-Learning 예시에서 S1~S6으로 나누고 table을 S1,a12 식으로 저장했으나, 여기서는 0번~24번 구역으로 나누고 table을 qTable[a][b] 식으로 저장하였다. 또한 각 원소를 딕셔너리로 저장한 것은 해당 지역에서 이동가능한 칸들에 대해서만 원소를 갖게 해서 메모리 공간도 줄이고 random.choice로 랜덤 이동을 쉽게 구현할 수 있기 때문이다.

이후 line 8-11에서는 리워드 값을 지정하고, line 16-25에서는 입력을 받으며 mapArr에 저장하고 startPos와 goalPos에 각각 출발지점, 목표지점의 위치를 기록한다.

(2) Q 테이블 초기화

1 for i in range(MAP\_SIZE):

2 for j in range(MAP\_SIZE):

3 mapNumber = i \* MAP\_SIZE + j # 5\*5기준 0번~24번

4

5 if mapNumber == goalPos: # goal 번호에 대한 QTable은 만들지 않음

6 continue

7

8 if i - 1 >= 0:

9 qTable[mapNumber][(i - 1) \* MAP\_SIZE + j] = 0

10 if j - 1 >= 0:

11 qTable[mapNumber][i \* MAP\_SIZE + (j - 1)] = 0

12 if i + 1 < MAP\_SIZE:

13 qTable[mapNumber][(i + 1) \* MAP\_SIZE + j] = 0

14 if j + 1 < MAP\_SIZE:

15 qTable[mapNumber][i \* MAP\_SIZE + (j + 1)] = 0

Q learning을 실시하기 전 우선 Q 테이블을 0으로 초기화해준다. 상/하/좌/우 방향 모두 이동할 수 있는 경우에 한해서 qTable의 원소에 해당하는 딕셔너리에 원소를 추가해준다.

(3) Q learning

1 NUMBER\_OF\_REPETITIONS = 200 # 총 Learning할 횟수

2

3 for i in range(NUMBER\_OF\_REPETITIONS):

4 currentPos = startPos

5

6 while currentPos != goalPos:

7 # qTable[currentPos].keys()는 현재 위치에서 갈 수 있는 곳임

8 nextPos = random.choice(list(qTable[currentPos].keys()))

9 immediateReward = 0

10 if mapArr[nextPos] == "G":

11 immediateReward = GOAL\_REWARD

12 elif mapArr[nextPos] == "T":

13 immediateReward = BONUS\_REWARD

14 elif mapArr[nextPos] == "B":

15 immediateReward = BOMB\_REWARD

16

17 if nextPos == goalPos:

18 qTable[currentPos][nextPos] = immediateReward

19 else:

20 qTable[currentPos][nextPos] = immediateReward + (gamma \* max(qTable[nextPos].values()))

21 currentPos = nextPos

line 1에서는 시작지점부터 목표지점까지 도착하면서 qTable을 갱신하는 것을 1번의 learning으로 봤을 때 총 몇 번 learning할 것인지 지정해준다. 5\*5 크기에서는 10번 정도만 learning을 해도 충분히 수렴하긴 하나 맵의 크기가 작으므로 learning 횟수를 넉넉히 잡았다.

각각의 learning 과정은 수업에서 다뤘던 방법과 동일한 방식으로 진행된다. 출발지점에서 시작하여(line 4) 현재 갈 수 있는 곳 중 랜덤하게 이동하여(line 8) 즉각적인 reward를 얻는다(line 9-15). 만약 목표지점에 도달했을 때는 이 값 그대로 qTable에 저장되나(line 17-18), 그 외의 경우 다음 장소에서 얻을 수 있는 Q함수 최댓값과 gamma값을 곱한 값이 추가로 더해져서 qTable에 저장된다(line19-20). 이후 currentPos값을 갱신하고(line 21) 목표지점에 도달할 때까지 반복한다(line 6).

(4) 파일 출력

1 with open("output.txt", "w") as outputFile:

2 currentPos = startPos

3 while currentPos != goalPos:

4 outputFile.write(str(currentPos) + " ")

5 currentPos = random.choice([k for k, v in qTable[currentPos].items() if max(qTable[currentPos].values()) == v]) # qTable이 최대가 되는 key값중 무작위

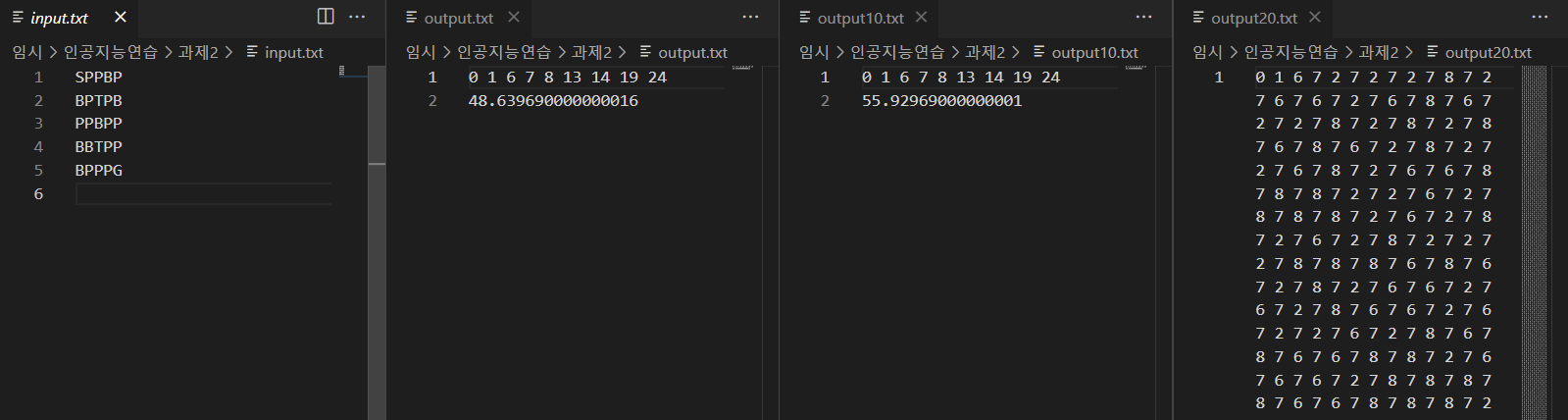
6 outputFile.write(str(currentPos) + "\n")

7

8 outputFile.write(str(max(qTable[startPos].values()))) # 시작지점 Q함수 최댓값

Q learning이 종료되었으면 시작지점에서 시작하여 qTable이 최대가 되는 key값으로 이동하여 목표지점까지 도달한다. qTable이 최대가 되는 key값이 여러 개 있을 경우 해당 key값들 중 무작위로 선정한다. 목표지점으로 가는 과정과 시작지점의 Q 함수 최댓값을 output.txt에 기록한다.

2. 실행 결과 및 분석



첨부된 파일 input.txt에 대하여 reward가 1일 때, 10일 때, 20일 때 결과는 각각 output.txt, output10.txt, output20.txt와 같다. 이 때 입력 특성상 실행할 때마다 서로 다른 결과를 출력할 수 있는데, 이는 주어진 input.txt의 13번과 18번 지점에서 qTable값이 최대가 되는 지점이 두 군데 존재하기 때문이다 (qTable값이 같을 경우 최대가 되는 지점들 중 무작위로 가는 것으로 앞에서 작성한 것과 같이 설계하였었다). 예를 들어 0 1 6 7 8 13 14 19 24도 최적 경로가 될 수 있고 0 1 6 7 8 13 18 23 24도 최적의 경로가 될 수 있다. 그리고 reward가 1일 때와 10일 때 모두 18번 지점에서 바로 왼쪽에 있는 보너스로 가지않고 19번이나 23번으로 가는 것을 확인할 수 있는데 이는 목표지점에 비해 보너스 reward가 작아 몇 칸 더 가더라도 목표지점에 도착하는 것이 더 큰 reward를 얻을 수 있기 때문이다.

reward가 20일 때는 목표지점을 찾지 못하는 무한 루프에 빠져 실행이 제대로 완료되지 않았는데, learning 완료 직후 qTable을 확인한 결과 출발지점의 최대 Q값은 85.2631578947368이 나왔다.

reward가 1일 때와 10일 때는 정상적으로 목표지점까지의 최적 경로를 찾아서 올바르게 간 것을 확인할 수 있었다. 하지만 reward가 20일 때는 목표지점을 찾지 못하고 왔다갔다 하면서 무한루프에 빠져 정상적으로 종료되지 못하였는데, 이는 보너스 리워드가 너무 커서 멀리 있는 목표지점에 도착하는 것보다 가까이 있는 보너스 지점에 가는 것이 리워드가 더 크기 때문이다. 보너스 reward가 1일 때와 20일 때 7번 지점에서의 qTable을 보면 다음과 같다.

- 보너스 reward = 1인 경우

{2: 54.044100000000014, 6: 54.044100000000014, 12: -33.489999999999995, 8: 65.61000000000001}

- 보너스 reward = 20인 경우

{2: 94.73684210526312, 6: 94.73684210526312, 12: -5.263157894736878, 8: 94.73684210526312}

보너스 reward가 1일 때는 목표지점으로 향하는 방향인 8번 방향일 때 q값이 최대가 되나, 보너스 reward가 20일 때는 폭탄을 제외한 어느 방향을 가더라도 q값이 최대가 된다. 그 이유는 2번, 6번, 8번 지점의 qTable을 보면 알 수 있는데,

- 2번 지점

{1: 85.2631578947368, 7: 105.2631578947368, 3: -14.736842105263193}

- 6번 지점

{1: 85.2631578947368, 5: -14.736842105263193, 11: 85.2631578947368, 7: 105.2631578947368}

- 8번 지점

{3: -14.736842105263193, 7: 105.2631578947368, 13: 85.2631578947368, 9: -14.736842105263193}

2, 6, 8번 중 어느 곳으로 가더라도 7번지점으로 갈 때 q값이 최대가 된다. 이는 목표지점의 reward인 100을 얻기 위해서는 여러 곳을 지나야 하나 보너스 지점의 reward인 20을 얻기 위해서는 한 칸만 가면 되기 때문이다. 그래서 결국 위 실행 결과와 같이 목표지점을 찾지 못하고 무한루프에 빠지게 된다.