인공지능 과제 2

수업 명: 인공지능

담당 교수님: 박철수 교수

학번: 2018202074

이름: 김상우

강의시간: 금 3,4

Introduction

해당 과제의 최종 목표는 주어진 grid word에서 Start에서 End로 가는 최소 루트를 찾는 코드를 만드는 것입니다. 이에 DP, 즉 dynamic Programming을 사용해야 하며 해당 결과를 찾음에 있어서 Policy iteration, Value iteration를 바탕으로 진행하여야 한다. Policy iteration의 경우 이를 위해 policy evaluation과 policy improvement를 구현해야 한다. 추가적으로 이 상황에서 random policy와 greedy policy를 진행하였을 때의 비교 또한 진행하여야 한다. 최종적으로 위 grid world에서 돌아가는 지 확인 및 위 인공지능 구조에 대한 이해를 하는 것을 목표로 한다.

Algorithm

Dynamic Programming

Dynamic Programming이란 시간에 따라 변하는 대상에 대해 사용하는 programming 기법 중 하나로, 큰 Problem에 대해 작은 problem(sub problem)들로 나누어 수행하고 이들의 solution을 기반으로 큰 problem을 해결해 나가는 프로그래밍 기법입니다. 분할정복과는 중복되는 sub problem을 이용해 programming한다는 차이가 있다.

Policy Evaluation

주어진 policy와 value를 기반으로 해당 policy에 대해 현재 상태의 가치를 구하는 방식이다. 각 station이 가능한 모든 action들에 대해 action이 일어나게 될 시 도달하는 state에서의 value값과 discount value(현재를 기준으로 했을 때 다음 state의 중요도를 고려하는 것이다.)의 곱과 행동으로 들게 되는 result를 더한 후, 이 결과에 이 action이 일어날 확률을 곱해 모두 더한 값이 해당 state의 value값이 되게 된다. 다음 state에서의 value값이 필요하다는 점에서 Dynamic programming을 사용할 수 있다.

Bellman equation을 이용하여 이를 나타낼 수 있으며 위에 설명을 공식으로 나타내면 아래와 같이 나온다.

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{s \in S} P_{ss'}^a V_k(s')\right)$$

출처: https://sumniya.tistory.com/10

Policy improvement

현재 상태 s에서 행동 a로의 진행시의 가치를 계산하여 정책을 결정하는 방법으로 기존에 사용하던 policy보다 변경시의 policy값이 더 크다면 이를 변경하는 방식으로 이를 진행한다. 현재 greedy policy를 해당 코드에서는 사용하고 있는데, 이 경우 지금 state에 action을 통해 가장 높은 value를 얻을 수 있는 방향으로 진행을 한다.

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned}
$$

출처:https://mclearninglab.tistory.com/103

Policy 자체의 유효성은 위와 같은 공식을 통해 구해낸 state s에서 a라는 action을 취할 시의 가치를 이용, 1/(해당 value가 가장 높은 것들의 수)의 값을 결과가 가장 높은 곳에 배분하여 다양한 방향으로의 처리 또한 진행할 수 있을 것이다.

Policy iteration

위의 Policy Evaluation을 반복하면 현재 policy에 대한 true value function을 얻어내는 것이 가능하다. 후 이를 바탕으로 Policy improvement를 진행하여 최적의 policy를 찾고 다시 해당 policy를 기반으로 policy evaluation를 반복하여 true value function을 찾아내는 행위를 반복하여 최적의 정책을 찾는 방법이다. 이 때문에 Policy Evaluation은 policy improvement의 판단하에 이전보다 나은 policy를 기반으로 진행할 수 있다는 장점이 있다.

Value Iteration

Policy iteration의 경우 policy evaluation와 policy improvement로 나뉘어 있다. 이를 보완한 형태로 evaluation과 improvement가 한번에 일어나는 최적의 정책을 찾는 방법이다. 벨만 공식을 기반으로 최적화 방식을 업데이트의 형태로 바꾼 것으로 policy iteration과 마찬가지로 특정 value로의 수렴을 위해 계속해 사용한다. 단, 합쳐진 만큼 전체 계산량이 늘어남을 확인할 수 있었다.

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

출처: https://mclearninglab.tistory.com/107

위는 value iteration에서 사용될 식으로 결과를 보면 해당 state에서 진행할 수 있는 action들에대해 reward와 value값 * discount의 합을 구하고 이중 가장 큰 값을 받아와 value로서 사용하는 것을 확인할 수 있다.

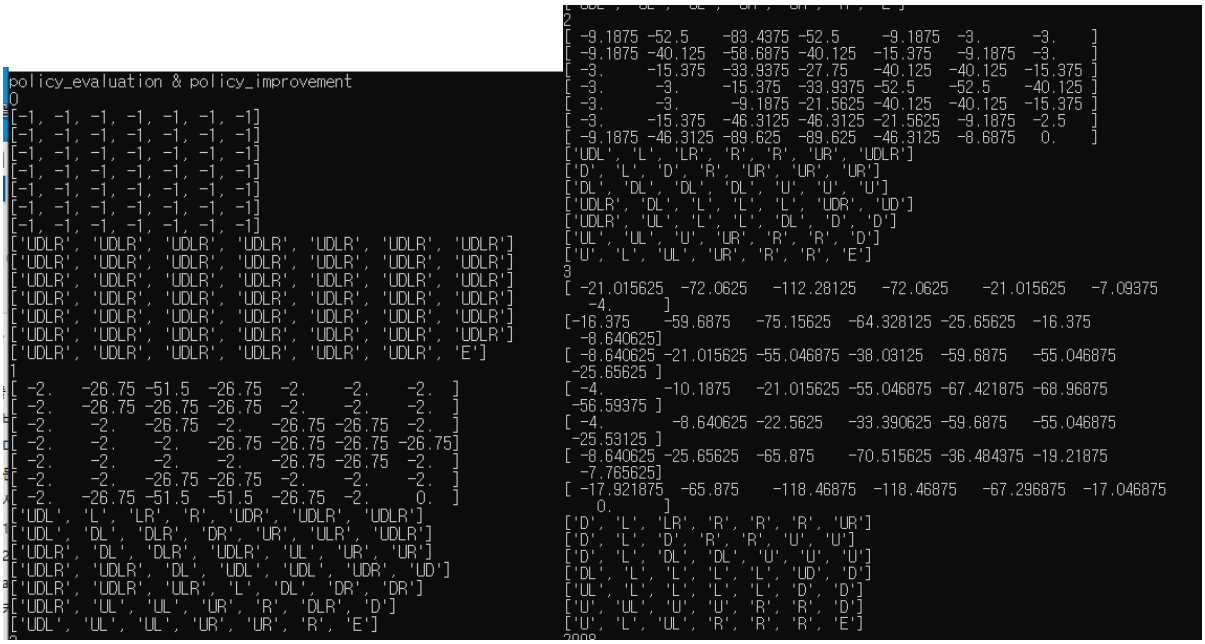
Result

해당 과제를 수행하기 위해 result를 pdf에 나와있는 값으로 초기화하고 2차원 배열의 value를 end point를 제외하고 모두 -1으로 해주었다. end point가 우측 하단으로 잡혀있기에 우측 하단의 value는 모든 경우에서 0으로 유지하였다. policy관련 출력을 진행할 시, 우측 하단은 End를 의미하는 E로 표시되게 하였다. 또한 방향에 대해 위, 아래, 왼쪽, 오른쪽 순으로 U, D, L, R로 표현하였으며 확률이 같을 경우에는 묶어 출력을 진행하였다.

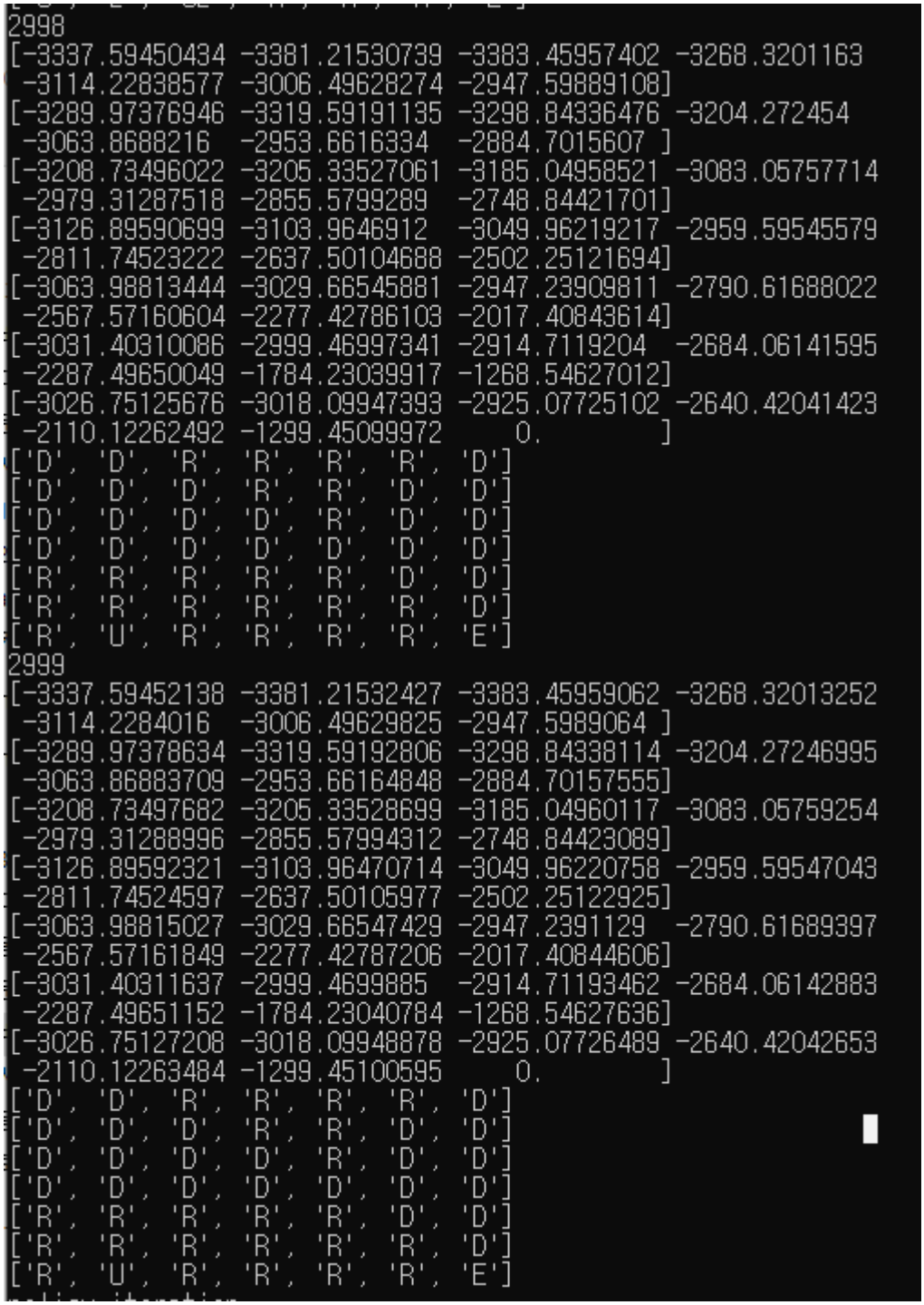
1.Policy evalution & policy_improvement(random policy)

위 Policy evaluation과 policy improvement의 식을 바탕으로 코드를 작성하였다.

4방향으로의 향할 확률을 1/4로 동일하게 가정하고 이를 바탕으로 진행하였다. Policy_evalution의 결과를 policy_improvement를 통해 측정하긴 하지만 policy의 값을 업데이트해주지 않는다.



위는 이를 실행했을 때의 결과이다. Reward = -100인 함정과 value값이 언제나 0인 end point를 기점으로 영향을 받는 모습이 확인된다. 실제로 함정 근처의 값들이 낮아지는 모습을, end point근처의 값이 상대적으로 적어지는 모습을 확인할 수 있었다.



위는 충분히 값을 늘려 policy가 수렴된 구간의 일부를 가져온 것이다.(2998-2999번째 시도) 여전히 end point에서 멀수록, start point에 가까울수록 비교적 낮은 값을 갖는 모습이며, 또한 함정에 영향을 계속 받아와 함정 근처에서의 값 또한 낮은 값을 갖게 된 것을 확인할 수 있다. 위의 policy를 확인하면 어느 시점에서도 이를 따라갈 시 함정을 피해 End point에 도달할 수 있음을 확인할 수 있다. 다만, 우리가 판단하기에 어느 쪽으로 가도 괜찮은 경우에 대해서는 판별하지 못하고 한쪽 방향만을 나타내고 있다.

2.Policy Iteration(Policy evalution & policy_improvement)(greedy policy)

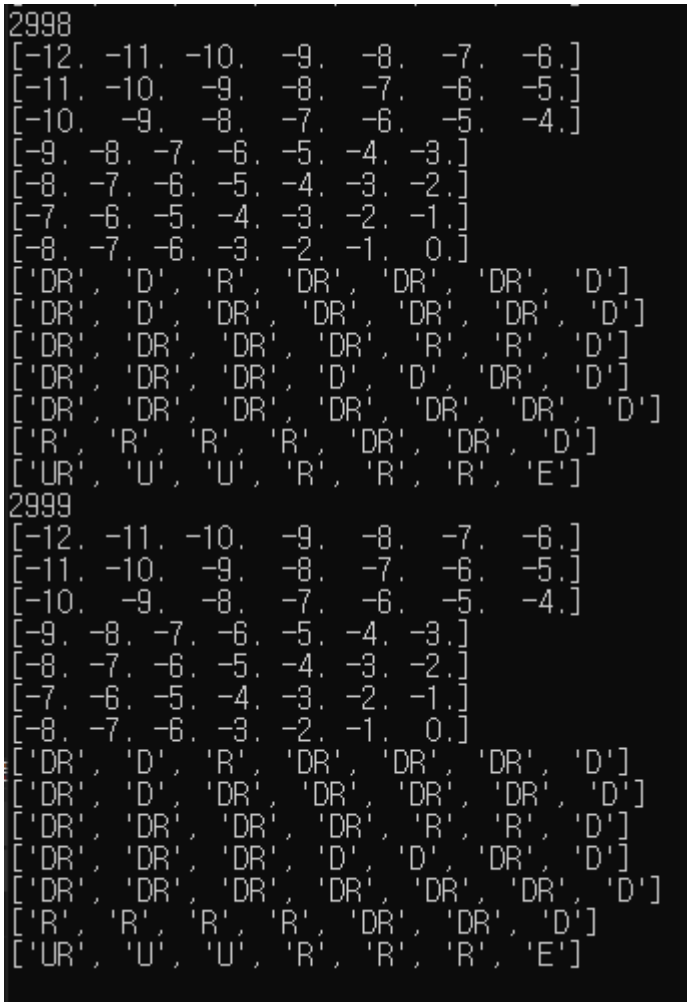
위 1과 동일한 policy evalution, policy improvement 함수를 사용하여 이를 진행하였다. 단, policy evalution을 반복하여 value가 수렴할 때 까지 얻은 값을 기반으로 policy improvement를 진행하고, 이렇게 policy_improvement가 생성한 현재 state에서 당장 가장 높은 value들을 가져오는 action들과 이를 기반으로 return하는 policy를 다음 policy evalution에 적용하는 코드를 작성하였다. (greedy policy)

처음 policy의 경우는 4방향으로의 향할 확률을 1/4로 동일하게 가정하고 이를 바탕으로 진행하였다. 위에서 언급하였듯 Policy_evalution의 결과를 policy_improvement를 통해 측정하며 policy의 값을 업데이트한다.

```
policy_iteration
0
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1]
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR']
['UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'UDLR', 'E']
1
[-3337.59823577 -3381.21900473 -3383.4632084  -3268.32366911
 -3114.23185315 -3006.49967835 -2947.60224539]
[-3289.9774668  -3319.59557003 -3298.84695136 -3204.27594579
 -3063.87221199 -2953.66493651 -2884.70481243]
[-3208.73859461 -3205.33885721 -3185.05308124 -3083.06095068
 -2979.31611252 -2855.58304326 -2748.84725539]
[-3126.8994598  -3103.96818298 -3049.96556571 -2959.59866317
 -2811.74824414 -2637.50386863 -2502.25391048]
[-3063.99160182 -3029.66884919 -2947.24233545 -2790.61989214
 -2567.57433224 -2277.43027662 -2017.41060744]
[-3031.40649647 -2999.47327652 -2914.71503476 -2684.06423769
 -2287.49891608 -1784.23229817 -1268.5476352 ]
[-3026.75461107 -3018.10272566 -2925.0802894  -2640.42310777
 -2110.12479621 -1299.4523648     0.        ]
['D', 'D', 'R', 'R', 'R', 'R', 'D']
['D', 'D', 'D', 'R', 'R', 'D', 'D']
['D', 'D', 'D', 'D', 'R', 'D', 'D']
['D', 'D', 'D', 'D', 'D', 'D', 'D']
['R', 'R', 'R', 'R', 'R', 'D', 'D']
['R', 'R', 'R', 'R', 'R', 'R', 'D']
['R', 'U', 'R', 'R', 'R', 'R', 'E']
2
```

```
2
[-12. -11. -10.  -9.  -8.  -7.  -6.]
[ -11.  -10.   -9. -107. -106. -105.   -5.]
[ -10.   -9.   -8.   -7. -105. -104.   -4.]
[-9. -8. -7. -6. -5. -4. -3.]
[-8. -7. -6. -5. -4. -3. -2.]
[-7. -6. -5. -4. -3. -2. -1.]
[  -8.   -7. -103.   -3.   -2.   -1.    0.]
['DR', 'D', 'R', 'R', 'R', 'R', 'D']
['DR', 'D', 'D', 'D', 'U', 'R', 'D']
['DR', 'DR', 'DR', 'D', 'L', 'R', 'D']
['DR', 'DR', 'DR', 'D', 'D', 'DR', 'D']
['DR', 'DR', 'DR', 'DR', 'DR', 'DR', 'D']
['R', 'R', 'R', 'R', 'DR', 'DR', 'D']
['UR', 'U', 'U', 'R', 'R', 'R', 'E']
3
[-12. -11. -10.  -9.  -8.  -7.  -6.]
[-11. -10.  -9.  -8.  -9.  -6.  -5.]
[-10.  -9.  -8.  -7.  -8.  -5.  -4.]
[-9. -8. -7. -6. -5. -4. -3.]
[-8. -7. -6. -5. -4. -3. -2.]
[-7. -6. -5. -4. -3. -2. -1.]
[-8. -7. -6. -3. -2. -1.  0.]
['DR', 'D', 'R', 'DR', 'R', 'DR', 'D']
['DR', 'D', 'DR', 'D', 'R', 'DR', 'D']
['DR', 'DR', 'DR', 'D', 'R', 'R', 'D']
['DR', 'DR', 'DR', 'D', 'D', 'DR', 'D']
['DR', 'DR', 'DR', 'DR', 'DR', 'DR', 'D']
['R', 'R', 'R', 'R', 'DR', 'DR', 'D']
['UR', 'U', 'U', 'R', 'R', 'R', 'E']
2998
```

위는 policy iteration을 k만큼 실행했을 때의 결과들이다. Random policy를 사용했을 때와 달리, 초반에는 evaluation이 함정의 영향을 받지만, 후반으로 갈수록 함정이 가지는 reward = -100의 영향을 value가 받지 않는 모습이다. 그러면서도 end point에 있는 value 0에는 영향을 받아 end point를 기준으로 endpoint에 가까울수록 값이 상대적으로 점점 낮아지도록 바뀌는 모습을 확인할 수 있다. Policy를 보면 2-3회차에서 value가 수렴하는 모습을 보인다. 그럼에도 방향은 policy에 기반하여 end로 향하는 모습이다.



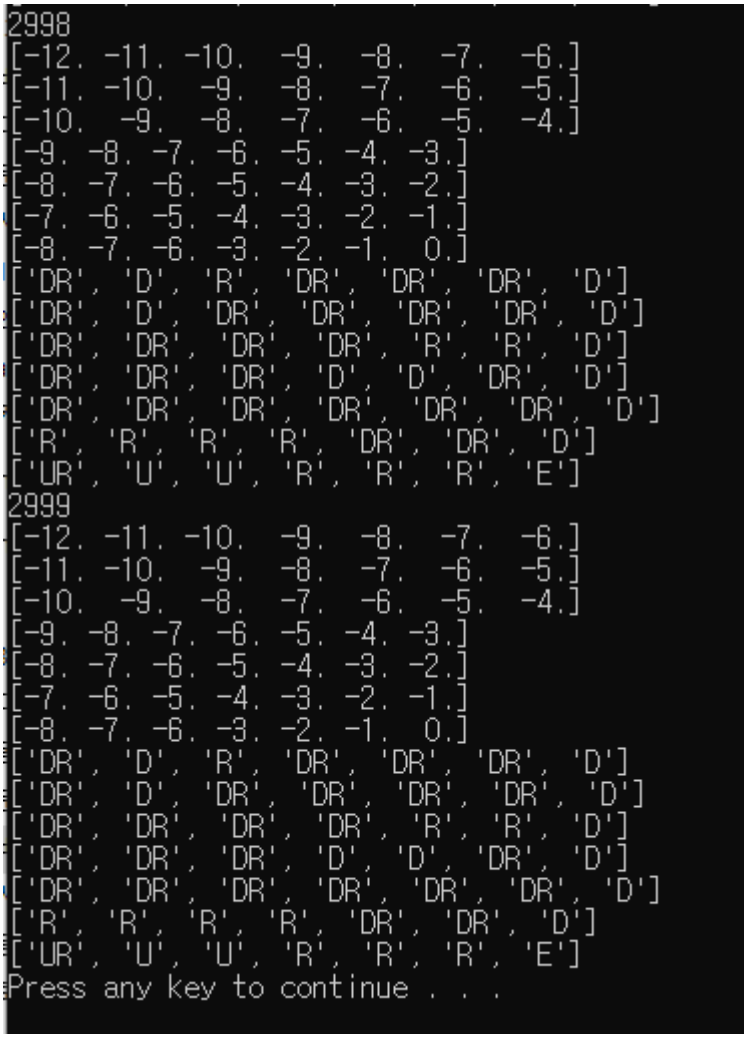
위는 충분히 값을 늘려 policy가 수렴된 구간의 일부를 가져온 것이다.(2998-2999번째 시도) 이전에 보았던 policy를 유지하는 모습이다. 여전히 end point에서 멀수록, start point에 가까울수록 비교적 낮은 값을 갖는 모습이며, policy를 업데이트 하지 않은 random policy에서는 함정에 영향을 계속 받아와 함정 근처에서의 값 또한 낮은 값을 갖게 된 것에 반하여 greedy policy를 사용한 policy iteration의 경우, 함정을 피하되 값이 영향을 받지 않아 값이 비교적 고르게 분포됨을 알 수 있었다. 이러한 여파로 다양한 선택지가 있는 경우 여러 방향으로 갈수 있음을 나타내고 있었다. 그럼에도 random policy의 경우와 마찬가지로 policy를 확인하면 어느 시점에서도 이를 따라갈 시 함정을 피하며 Result가 최소가 되게 End point에 도달할 수 있음을 확인할 수 있다.

3.Value Iteration

위에 제시된 value iteration의 식을 기반으로 이를 제작하였다.

policy의 경우는 4방향으로의 향할 확률을 1/4로 동일하게 가정하고 이를 바탕으로 진행하였다. 해당 policy는 업데이트 되지 않은, 즉 위의 1/4 상태를 유지한다. 위에서 언급한 것처럼 action을 취할 시 가장 높은 가치를 창출해내는 경우를 찾아 현재 value로 이를 사용하는 방식으로 진행이 될 것이다.

위는 value iteration을 k(=0,1,2,3)횟수 만큼 실행한 모습이다. 값을 정할 때 함정이 가지는 reward = -100의 영향을 받지 않는 모습이다. 그러면서도 end point에 있는 value 0에는 영향을 받아 end point를 기준으로 endpoint에 가까울수록 값이 상대적으로 점점 낮아지도록 바뀌는 모습을 확인할 수 있다. 위를 policy improvement를 이용해 나타내는 방향을 측정하였는데, policy improvement를 통한 policy update를 하지않았음에도 value iteration자체에서 업데이트를 진행하여 결과가 최적화되는 모습을 확인할 수 있었다. 이는 value iteration의 evaluation과 improvement를 동시에 처리하는 특성 때문임을 확인할 수 있었다.



위는 충분히 값을 늘려 policy가 수렴된 구간의 일부를 가져온 것이다.(2998-2999번째 시도) 여전히 end point에서 멀수록, start point에 가까울수록 비교적 낮은 값을 갖는 모습이며, 함정을 피하되 값이 영향을 받지 않아 값이 비교적 고르게 분포됨을 알 수 있었다. 이러한 여파로 다양한 선택지가 있는 경우 여러 방향으로 갈수 있음을 나타내고 있었다. 다른 case와 마찬가지로 policy를 확인하면 어느 시점에서도 이를 따라갈 시 함정을 피해 End point에 도달할 수 있음을 확인할 수 있다.

결과적으로 greedy policy를 사용한 결과와 동일한 것을 확인할 수 있었다.

Consideration

해당 과제를 진행하며 인공지능 학습에서 주어진 policy, value, reward의 관계와 어떤 식으로 학습이 진행되는지, 각 변수들이 어떻게 영향을 줄 수 있는지 알 수 있었습니다. 특히, 위에서 여러 방식의 policy를 사용해 보며 어떤 방식을 선택하느냐 에 따라 얻을 수 있는 결과가 다르며 이때문에 문제에 따라 선택이 중요하게 작용할 수 있다는 것을 알 수 있었습니다. 실제로 policy update를 하지 않은 쪽이 좋지 않아 보이기도 하지만, 각 상황에서 함정에서 멀어지는 등의 행위가 필요할 경우 policy를 업데이트 하지 않은 것이 유리할 수도 있는 것이다. 결과적으로 policy와 방법론의 중요성을 알 수 있었습니다. 기회가 된다면 greedy policy뿐만 아닌 장기적인 미래를 보고 확률적으로 지금 당장 좋지 않은 방향도 가보는 policy또한 사용해 보고 싶습니다.

Reference

Reinforcement Learning_part III_IV.pdf

https://sumniya.tistory.com/10

https://mclearninglab.tistory.com/103

https://mclearninglab.tistory.com/107