인공지능

Project2

담당 교수님: 박철수

강의 시간: 금3,4

학 과: 컴퓨터정보공학부

학 번: 2020202055

성 명: 최소윤

1. Introduction

Dynamic programming을 이용하여 environment의 model을 푸는 과제로 큰 순차적인 행동 결정을 작은 process로 나누어 제시된 grid world에서 Policy iteration과 Value iteration을 policy를 최적화하는 것이 목표이다. 7 x 7의 시작점과 끝점을 가지는 grid-world로 action은 상하좌우이다. 이번 프로젝트에서 Policy evaluation과 Policy Improvement, Value Iteration을 구현해보도록 한다.

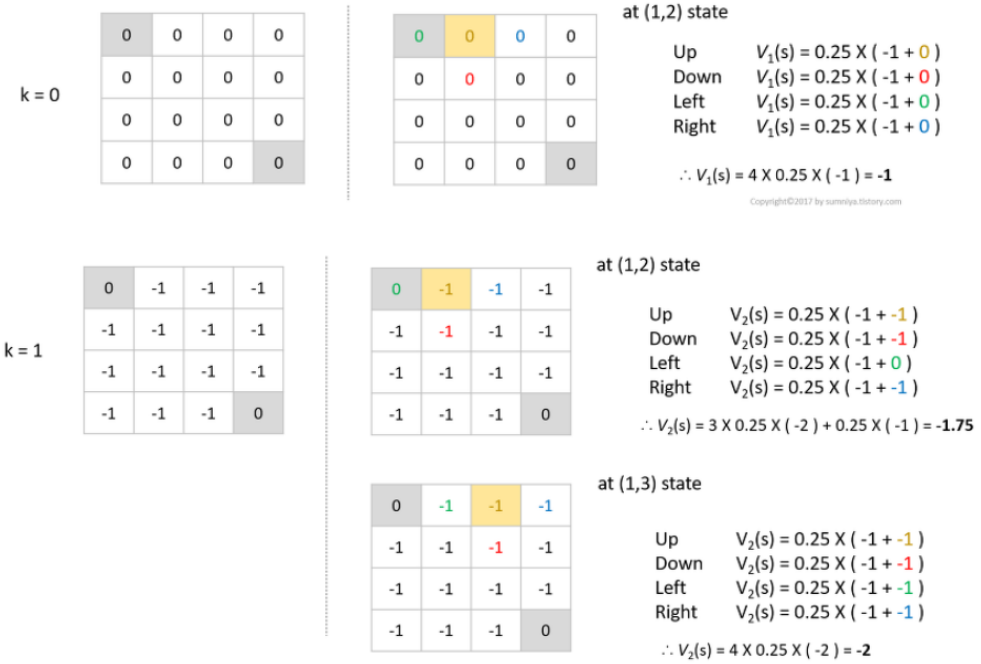
2. Algorithm

Policy evaluation – random Policy

Matrix가 주어지면 각 state에 대한 ture value function을 찾는 것이 목표이며 이는 현재 따르는 policy가 맞는지 틀린지 판단하기 위해 value function을 사용하며 value function을 계속해서 update하여 true value function을 찾는다.

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

State의 k+1번째 value function은 state에서 이동 가능한 다음 state에 대해서 k번째 구해놓은 다음 state의 value function을 이용한다.



위 그림과 같이 초기 value function을 0으로 설정하고 state에서 현재 V0에서 one-step씩 이동 가능한 모든 다음 state의 value function으로 다음 iteration의 value function(V1)을 구하여 update할 수 있다. 위, 아래, 왼쪽, 오른쪽 각 행동에

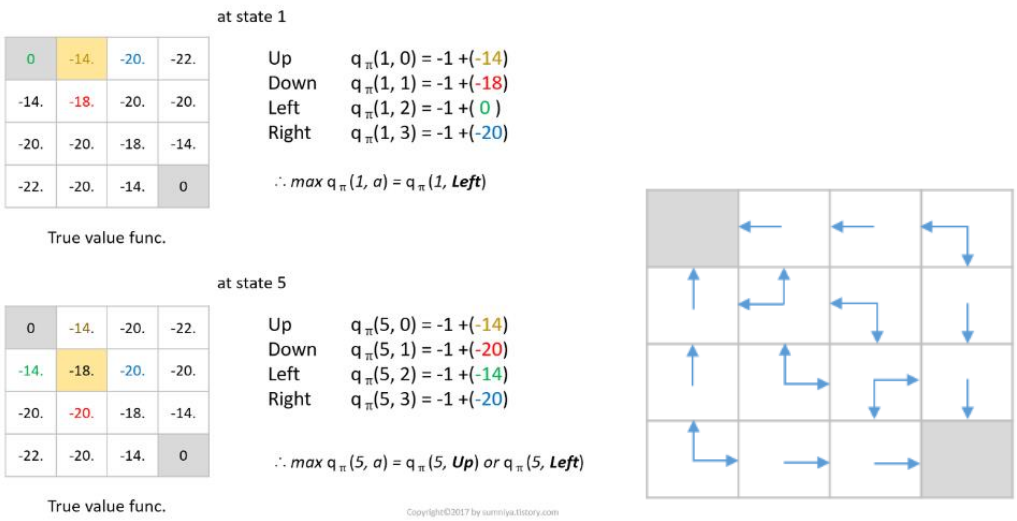
대한 action이 가능하며 각 action에 대한 value function을 구한 뒤 더한 값을 저장한다. 위 행동을 반복하면 수렴하는 값이 나오게 되는데 이는 ture value function이 된다.

Policy Improvement

Iteration을 반복하여 true value function을 찾은 후에 해당 policy를 따르는 것이 좋은 것인지 판단하고 policy를 update해야한다. 이 과정을 통해 optimal policy를 찾을 수 있다. Greedy policy improvement는 가장 널리 알려진 방법으로, 가치가 가장 높은 state를 선택하겠다는 의미로 max값을 가져온다.

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s \in S} P_{ss'}^a V_\pi(s')$$

policy evaluation에서 state-value function을 이용하여 모든 state에 대한 value값을 구했으면 policy에 따른 action을 취해야한다. 현재 state에서 갈 수 있는 가장 높은 state로 이동하는 action을 취해야 하며 action을 정량화하여 선택하는 기준을 만드는 q-function을 이용해야 한다.



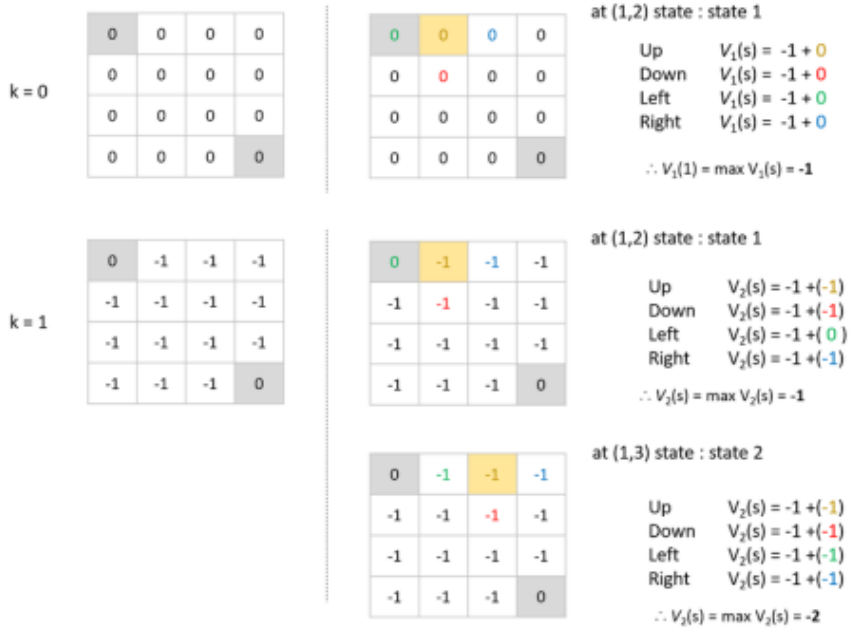
state별로 취할 수 있는 action에 대해 q-function을 구한다. 이는 greedy policy improvement를 통해 선택된 action이며 이를 모든 state에 대해 적용하면 오른쪽 그림과 같이 화살표로 action을 표현할 수 있으며 optimal policy를 찾게 된다.

Value Iteration

Policy Iteration과는 다르게 Bellman Optimality Equation을 사용한다. 이는 evaluation을 한 번만 진행하며 이동 가능한 다음 state에 대한 모든 value function 중에서 max값만 가져와 greedy하게 value function을 구한 후 improve 하였다.

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

위 식은 max값을 취하는 optimal value function이다.



위 그림을 보면 각 취할 수 있는 action에 대해 다음 state에 대한 value값을 모두 더하는 것이 아니라 최대값을 선택하여 value 값을 취한다. 이를 통해 빠른 수렴이 가능하다.

3. Result

Policy evaluation – random Policy

```
Iteration: 0
[[  0  -1 -100  -1  -1  -1  -1]
 [ -1  -1 -100  -1  -1  -1  -1]
 [ -1  -1  -1   -1  -1  -1  -1]
 [ -1  -1  -1   -1 -100 -100 -1]
 [ -1  -1  -1   -1  -1  -1  -1]
 [ -1  -1  -1   -1  -1  -1  -1]
 [ -1  -1 -100 -100 -1  -1   0]]

Iteration: 1
[[  0.   -26.5  -51.5  -26.75  -2.    -2.    -2.  ]
 [ -1.75 -26.75 -26.75 -26.75  -2.    -2.    -2.  ]
 [ -2.    -2.   -26.75  -2.    -26.75 -26.75  -2.  ]
 [ -2.    -2.    -2.    -26.75 -26.75 -26.75 -26.75]
 [ -2.    -2.    -2.    -2.    -26.75 -26.75  -2.  ]
 [ -2.    -2.   -26.75 -26.75  -2.    -2.    -1.75]
 [ -2.   -26.75 -51.5  -51.5  -26.75  -1.75   0.  ]]

Iteration: 2
[[  0.    -27.188 -33.875 -27.75   -9.188  -3.     -3.   ]
 [ -8.625 -15.25  -33.938 -15.375 -15.375  -9.188  -3.   ]
 [ -2.938 -15.375  -9.188 -27.75  -15.375 -15.375 -15.375]
 [ -3.     -3.    -15.375  -9.188 -27.75  -27.75  -15.375]
 [ -3.     -3.     -9.188 -21.562 -15.375 -15.375 -15.312]
 [ -3.    -15.375 -21.562 -21.562 -21.562  -9.062  -2.438]
 [ -9.188 -21.562 -40.125 -40.125 -21.5    -8.625   0.   ]]

Iteration: 3
[[  0.    -20.078 -31.688 -22.547 -14.628  -7.094  -4.   ]
 [ -7.703 -22.282 -19.422 -27.203 -13.282 -10.186  -6.641]
 [ -6.484  -6.594 -24.11  -13.282 -22.562 -17.922 -13.281]
 [ -3.984 -10.188  -8.641 -24.109 -17.922 -19.469 -19.453]
 [ -4.     -8.641 -16.375 -14.828 -22.562 -17.875 -13.125]
 [ -8.641 -13.281 -22.562 -27.203 -17.875 -13.     -7.703]
 [-11.734 -22.562 -31.844 -31.828 -23.953 -10.797   0.   ]]

Iteration: 4
[[  0.    -19.512 -24.434 -25.066 -15.438 -10.028  -6.934]
 [-10.617 -14.949 -27.321 -18.133 -19.695 -12.735 -10.028]
 [ -8.191 -17.266 -13.485 -25.496 -16.602 -17.375 -15.824]
 [ -7.664  -8.465 -19.696 -14.668 -23.176 -19.293 -17.332]
 [ -7.316 -11.961 -14.668 -23.562 -18.125 -18.039 -15.539]
 [-10.414 -16.602 -23.176 -22.773 -22.68  -14.562  -9.457]
 [-14.668 -20.855 -28.199 -29.707 -22.113 -12.938   0.   ]]

Iteration: 5
[[  0.    -15.724 -25.083 -21.768 -18.557 -12.284  -9.481]
 [ -9.439 -19.679 -18.75  -25.394 -16.727 -15.282 -12.38 ]
 [-11.934 -12.272 -23.445 -16.722 -22.435 -17.114 -16.14 ]
 [ -8.909 -15.147 -13.621 -23.982 -18.172 -19.98  -17.997]
 [-10.339 -12.763 -20.599 -18.559 -22.864 -17.88  -16.092]
 [-13.25  -17.602 -21.56  -25.781 -20.393 -16.778 -10.89 ]
 [-16.151 -21.081 -26.484 -26.698 -22.86  -13.403   0.   ]]
```

상하좌우로 이동할 수 있으며 각 확률은 0.25로 같다. 7 x 7 grid world에서 random policy에 대한 true value function을 구할 수 있었으며 이를 계산하여 iteration을 1000으로 설정하고 결과 값을 출력했을 때 다음 그림과 같이 나왔다. Iteration 0에서와 같이 초기값을 start와 end에서는 0으로 설정하였고 나머지는 -1로 초기화하고 중간에 -100으로 함정을 만들었다. one step씩 각 state의 value function을 update하면 초반에는 값이 점점 작아지면서 값이 계속 변하는 것을 알 수 있다.

```
Iteration: 80
[[  0.    -31.444 -48.239 -57.758 -63.133 -65.962 -67.153]
 [-31.448 -42.792 -52.618 -59.25  -63.178 -65.18  -65.962]
 [-48.25  -52.625 -57.396 -60.837 -62.636 -63.178 -63.134]
 [-57.778 -59.265 -60.845 -61.479 -60.837 -59.25  -57.76 ]
 [-63.161 -63.201 -62.653 -60.845 -57.397 -52.619 -46.24 ]
 [-65.997 -65.211 -63.201 -59.266 -52.626 -42.793 -31.445]
 [-67.192 -65.997 -63.161 -57.778 -48.25  -31.449   0.   ]]

Iteration: 90
[[  0.    -33.106 -50.859 -60.963 -66.692 -69.719 -70.997]
 [-33.109 -45.085 -55.492 -62.542 -66.731 -68.676 -69.719]
 [-50.866 -55.497 -60.56  -64.22  -66.146 -66.732 -66.692]
 [-60.975 -62.551 -64.226 -64.901 -64.221 -62.542 -60.964]
 [-66.708 -66.746 -66.156 -64.226 -60.561 -55.493 -50.86 ]
 [-69.739 -68.896 -66.746 -62.552 -55.497 -45.086 -33.107]
 [-71.02  -69.739 -66.706 -60.975 -50.866 -33.109   0.   ]]

Iteration: 100
[[  0.    -34.592 -53.199 -63.824 -69.869 -73.072 -74.428]
 [-34.593 -47.134 -58.059 -65.481 -69.904 -72.178 -73.072]
 [-53.203 -58.061 -63.386 -67.242 -69.278 -69.904 -69.869]
 [-63.832 -65.487 -67.245 -67.956 -67.242 -65.482 -63.625]
 [-69.88  -69.912 -69.265 -67.245 -63.387 -58.06  -53.199]
 [-73.084 -72.19  -69.912 -65.487 -58.062 -47.134 -34.592]
 [-74.442 -73.084 -69.879 -63.832 -53.203 -34.593   0.   ]]

Iteration: 500
[[  0.    -46.866 -72.533 -87.467 -96.112 -100.764 -102.758]
 [-46.865 -64.069 -79.276 -89.768 -96.118 -99.436 -100.764]
 [-72.533 -79.276 -86.746 -92.221 -95.166 -96.117 -96.112]
 [-87.466 -89.768 -92.221 -93.218 -92.221 -89.768 -87.467]
 [-96.112 -96.117 -95.166 -92.221 -86.746 -79.276 -72.533]
 [-100.764 -99.436 -96.118 -89.768 -79.276 -64.069 -46.865]
 [-102.758 -100.764 -96.112 -87.467 -72.533 -46.866   0.   ]]

Iteration: 520
[[  0.    -46.892 -72.576 -87.519 -96.17  -100.826 -102.821]
 [-46.892 -64.106 -79.323 -89.821 -96.176 -99.498 -100.826]
 [-72.576 -79.323 -86.798 -92.277 -95.224 -96.176 -96.17 ]
 [-87.52  -89.821 -92.277 -93.274 -92.277 -89.821 -87.519]
 [-96.17  -96.176 -95.224 -92.277 -86.798 -79.323 -72.576]
 [-100.826 -99.498 -96.176 -89.821 -79.323 -64.106 -46.892]
 [-102.821 -100.826 -96.17  -87.52  -72.576 -46.892   0.   ]]

Iteration: 540
[[  0.    -46.914 -72.61  -87.561 -96.216 -100.874 -102.87 ]
 [-46.914 -64.136 -79.361 -89.864 -96.222 -99.546 -100.874]
 [-72.61  -79.361 -86.839 -92.322 -95.27  -96.222 -96.216]
 [-87.562 -89.864 -92.322 -93.319 -92.321 -89.864 -87.561]
 [-96.217 -96.222 -95.27  -92.322 -86.839 -79.361 -72.61 ]
 [-100.875 -99.547 -96.222 -89.864 -79.361 -64.136 -46.914]
 [-102.87 -100.875 -96.217 -87.562 -72.61  -46.914   0.   ]]
```

```
Iteration: 960
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]

Iteration: 980
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]

Iteration: 1000
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]
```

위와 같이 iteration 1000번째에는 수렴한 값이 나오는 것을 확인할 수 있으며 iteration 711번째에서 값이 수렴하였다.

Policy Improvement

```
Iteration: 960
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]

Iteration: 980
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]

Iteration: 1000
[[   0.     -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]
```

Policy evaluation을 통해 iteration을 반복하여 true value function을 찾았고, Policy Improvement를 통해 어떤 policy를 따르는 것이 좋을지 안 좋을지 판단하여 알맞은 Policy를 update하여 이를 반영해야 한다. 현재 policy보다 더 나은 policy를 찾아가면 optimal policy에 가까워진다.

```
Updated policy is :
[[[0.   0.   0.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.5  0.5  0.  ]]

 [[1.   0.   0.   0.  ]
  [0.5  0.   0.5  0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.5  0.5  0.  ]
  [0.   1.   0.   0.  ]]

 [[1.   0.   0.   0.  ]
  [1.   0.   0.   0.  ]
  [0.5  0.   0.5  0.  ]
  [0.   0.   1.   0.  ]
  [0.   0.5  0.5  0.  ]
  [0.   1.   0.   0.  ]
  [0.   1.   0.   0.  ]]

 [[1.   0.   0.   0.  ]
  [1.   0.   0.   0.  ]
  [1.   0.   0.   0.  ]
  [0.25 0.25 0.25 0.25]
  [0.   1.   0.   0.  ]
  [0.   1.   0.   0.  ]
  [0.   1.   0.   0.  ]]]
```

```
[[[1.   0.   0.   0.  ]
  [1.   0.   0.   0.  ]
  [0.5  0.   0.   0.5 ]
  [0.   0.   0.   1.  ]
  [0.   0.5  0.   0.5 ]
  [0.   1.   0.   0.  ]
  [0.   1.   0.   0.  ]]

 [[1.   0.   0.   0.  ]
  [0.5  0.   0.   0.5 ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   1.   0.   0.  ]]

 [[0.5  0.   0.   0.5 ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   1.  ]
  [0.   0.   0.   0.  ]]]
```

Greedy policy improvement로 구현하여 max값만 선택한다. 그리고 해당 policy에 따른 action을 취해야한다. 즉 현재 state에서 가장 높은 곳으로 이동하는 action을 취한다. 위의 그림은 update된 policy를 의미한다.

```
at each state, chosen action is :
[['T' 'Left' 'Left' 'Left' 'Left' 'Left' 'Down']
 ['Up' 'Up' 'Left' 'Left' 'Left' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Left' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Up' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Right' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Right' 'Right' 'Right' 'Right' 'Down']
 ['Up' 'Right' 'Right' 'Right' 'Right' 'Right' 'T']]
```

위 그림은 각각의 state에서의 action을 matrix로 나타낸 모습이다.

Value Iteration

```
Iteration: 0
[[   0   -1 -100   -1   -1   -1   -1]
 [  -1   -1 -100   -1   -1   -1   -1]
 [  -1   -1   -1   -1   -1   -1   -1]
 [  -1   -1   -1   -1 -100 -100   -1]
 [  -1   -1   -1   -1   -1   -1   -1]
 [  -1   -1   -1   -1   -1   -1   -1]
 [  -1   -1 -100 -100   -1   -1    0]]

Iteration: 1
[[ 0. -1. -2. -2. -2. -2. -2.]
 [-1. -2. -2. -2. -2. -2. -2.]
 [-2. -2. -2. -2. -2. -2. -2.]
 [-2. -2. -2. -2. -2. -2. -2.]
 [-2. -2. -2. -2. -2. -2. -2.]
 [-2. -2. -2. -2. -2. -2. -1.]
 [-2. -2. -2. -2. -2. -1.  0.]]

Iteration: 2
[[ 0. -1. -2. -3. -3. -3. -3.]
 [-1. -2. -3. -3. -3. -3. -3.]
 [-2. -3. -3. -3. -3. -3. -3.]
 [-3. -3. -3. -3. -3. -3. -3.]
 [-3. -3. -3. -3. -3. -3. -2.]
 [-3. -3. -3. -3. -3. -2. -1.]
 [-3. -3. -3. -3. -2. -1.  0.]]

Iteration: 3
[[ 0. -1. -2. -3. -4. -4. -4.]
 [-1. -2. -3. -4. -4. -4. -4.]
 [-2. -3. -4. -4. -4. -4. -4.]
 [-3. -4. -4. -4. -4. -4. -3.]
 [-4. -4. -4. -4. -4. -3. -2.]
 [-4. -4. -4. -4. -3. -2. -1.]
 [-4. -4. -4. -3. -2. -1.  0.]]

Iteration: 4
[[ 0. -1. -2. -3. -4. -5. -5.]
 [-1. -2. -3. -4. -5. -5. -5.]
 [-2. -3. -4. -5. -5. -5. -4.]
 [-3. -4. -5. -5. -5. -4. -3.]
 [-4. -5. -5. -5. -4. -3. -2.]
 [-5. -5. -5. -4. -3. -2. -1.]
 [-5. -5. -4. -3. -2. -1.  0.]]

Iteration: 5
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1. -2. -3. -4. -5. -6. -5.]
 [-2. -3. -4. -5. -6. -5. -4.]
 [-3. -4. -5. -6. -5. -4. -3.]
 [-4. -5. -6. -5. -4. -3. -2.]
 [-5. -6. -5. -4. -3. -2. -1.]
 [-6. -5. -4. -3. -2. -1.  0.]]

Iteration: 960
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1. -2. -3. -4. -5. -6. -5.]
 [-2. -3. -4. -5. -6. -5. -4.]
 [-3. -4. -5. -6. -5. -4. -3.]
 [-4. -5. -6. -5. -4. -3. -2.]
 [-5. -6. -5. -4. -3. -2. -1.]
 [-6. -5. -4. -3. -2. -1.  0.]]

Iteration: 980
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1. -2. -3. -4. -5. -6. -5.]
 [-2. -3. -4. -5. -6. -5. -4.]
 [-3. -4. -5. -6. -5. -4. -3.]
 [-4. -5. -6. -5. -4. -3. -2.]
 [-5. -6. -5. -4. -3. -2. -1.]
 [-6. -5. -4. -3. -2. -1.  0.]]

Iteration: 1000
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1. -2. -3. -4. -5. -6. -5.]
 [-2. -3. -4. -5. -6. -5. -4.]
 [-3. -4. -5. -6. -5. -4. -3.]
 [-4. -5. -6. -5. -4. -3. -2.]
 [-5. -6. -5. -4. -3. -2. -1.]
 [-6. -5. -4. -3. -2. -1.  0.]]
```

위 그림은 Value Iteration을 적용한 것이며 이는 Bellman Optimally equation을 이용하여 계산한다. Evaluation 과정에서 max값을 취해서 greedy하게 value function을 구한다. Policy evaluation과 비교하면 Value Iteration은 Iteration 5번째부터 수렴했으며 더 빨리 수렴한 것을 볼 수 있다.

Random policy & Greedy policy

```
[[   0.      -46.987  -72.724  -87.701  -96.372 -101.039 -103.038]
 [ -46.987  -64.237  -79.486  -90.008  -96.378  -99.708 -101.039]
 [ -72.724  -79.486  -86.977  -92.469  -95.423  -96.378  -96.372]
 [ -87.701  -90.008  -92.469  -93.469  -92.469  -90.008  -87.701]
 [ -96.372  -96.378  -95.423  -92.469  -86.977  -79.486  -72.724]
 [-101.039  -99.708  -96.378  -90.008  -79.486  -64.236  -46.986]
 [-103.038 -101.039  -96.372  -87.701  -72.724  -46.987    0.   ]]

at each state, chosen action is :
[['T' 'Left' 'Left' 'Left' 'Left' 'Left' 'Down']
 ['Up' 'Up' 'Left' 'Left' 'Left' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Left' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Up' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Up' 'Right' 'Down' 'Down' 'Down']
 ['Up' 'Up' 'Right' 'Right' 'Right' 'Right' 'Down']
 ['Up' 'Right' 'Right' 'Right' 'Right' 'Right' 'T']]
```

random policy에서는 첫번째 그림과 같이 수렴하는 true value가 나왔으며, Policy Improvement를 구현하여 greedy policy에서는 두번째 그림과 같이 optimal policy를 도출할 수 있었다. 결과적으로는 policy Improvement를 greedy policy로 구현

하여 optimal한 action을 찾을 수 있었으며 첫번째 그림의 true value의 최대값에 따라 action을 결정하면 두번째 그림과 같이 action이 나온다. 그러므로 결국 optimal policy는 두번째 그림과 같이 action을 취하면 reward를 최대로 받아야하는 agent는 최단거리로 종료점까지 갈 수 있다.

4. Consideration

Policy evaluation과 Policy Improvement와 Value Iteration 용어와 개념이 헷갈려 이해하는 데 어려움이 있었다. 이번 과제를 통해서 강의 자료를 복습하면서 이해를 하고 프로젝트를 진행하면서 코드로 구현하고, 한 줄씩 이해해보는 과정을 통해 완전히 이해할 수 있었다. 또한 공식도 이해하기 어려웠는데 한 칸씩 계산해보면서 해당 value가 어떻게 나왔는지 직접 계산해보았고 이를 통해 코드를 작성할 때 좀 더 수월하게 작성할 수 있었다.