DRL CCE Karthick

October 18, 2019

0.0.1 Assignment 1 - Deep Reinforcement Learning

Submitted by : Karthick Raja S

Roll Number: 22/230

0.0.2 Contents

- Question
- Glossary
- Initial condition
- Update condition
- Output
- Python code for state feature vector
- Solution 1.1, 1.2, 1.3
- Solution 2.1, 2.2, 2.3
- Something that i tried : compare random policy with **Epsilon greedy Policy** & **Thompson sampling policy**
- Conclusion

Question

- Consider a 5 x 5 with 25 state as shown below
- Possible actions are left, right, up and down
- All the actions can be employed in one step at a time
- States A and B are special states
  - Any action in state A (state:1) leads to A'(state:17) and gives a reward of 10
  - Any action in state B (state:3) leads to B'(state:14) and gives a reward of 5

$$\begin{bmatrix} * & A & * & B & * \\ * & * & * & * & * \\ * & * & * & B' & * \\ * & A' & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

- If we are in a boundry state and we pick up an action that takes us out of the grid, we stay there but the reward is -1

- For any other movement between the states fetches a reward of 0
- We are considering considering a equi-probable policy

$$\pi(\uparrow /s) = \pi(\downarrow /s) = \pi(\leftarrow /s) = \pi(\rightarrow /s) = \frac{1}{4} \forall s \in states$$

Considering 2 different kinds of state features: * $

$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \cdots & 0 & 0 \end{bmatrix}$

^{T} $ with 25 dimensions * scalar feature: $\frac{n}{25}$ where n ∈ {1...25}

Three different starting conditions: * Middle state : 13 * First state : 2 * Last state : 25

Note: I have taken the liberty to have state value matrix, instead of state value vector (as explained by professor) for the ease of writing the environment and program.

Glossary

- t time $\in T$ can be set by altering the variable EPOCH
- S states : 0 to 24
- R Rewards : 0 for all cell, -1 for moves going out of the grid and 10 , 5 special cases
- r_t reward at time t ∈ in R
- s_t state at time t ∈ in S
- j_t utility value at time t
- α_t learning parameter held constant for every 50 epochs
- $\vartheta(s_t)$ - value function / differential reward for state s and time t
- $ Epoch $ - Total number of moves

Initial condition

$$\hat{j_t} = 0$$

$$\vartheta_t(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Update conditions

$$\hat{j_{t+1}} = \hat{j_t} + \alpha_t(r_{t+1} - \hat{j_t})$$

$$\delta_t = r_{t+1} - \hat{j_{t+1}} + \gamma\vartheta(s_{t+1}) - \vartheta(s_t)$$

$$\vartheta(s_{t+1}) = \vartheta_t(s) + \alpha_t\delta_t \ \ \forall s \in S$$

Ouput

- Plot of utilization as a function of time
- Plot of average state-value as a function of time (for vector state space)
- Plot of state-value as a function of time (for scalar vector space)

Python code for Question 1

```
In [196]: import numpy as np
          from gridworld import gridworld
          import seaborn as sns
          import matplotlib.pyplot as plt

In [197]: EPOCHS = 10000

In [198]: # Defining random policy
          class Random:
              def __init__(self, n_arms):
                  self.n_arms = n_arms
                  self.chosen_arms = []

              def action(self):
                  arm = np.random.randint(self.n_arms)
                  self.chosen_arms.append(arm)
                  return arm

              def update(self, reward):
                  pass;

In [199]: sampling_random = Random(4)

In [200]: class play:

              def __init__(self, initial_state, sampling_technique ):
                  self.rewards = []
                  self.utility = [0]
                  self.value_matrix = np.zeros((5,5))
                  self.delta = []
                  self.states = []
                  self.initial_state = initial_state
                  self.states.append(initial_state)
                  self.value_function = []
                  self.env = gridworld(initial_state=initial_state)
                  self.EPOCHS = EPOCHS
                  self.print_epoch = EPOCHS/10
                  self.actions = []
                  self.sampling_technique = sampling_technique

              def update_value_matrix(self, observation, new_observation, reward, s_alpha):
                  '''Return the updated utility matrix
                  @param utility_matrix the matrix before the update
                  @return the updated utility matrix
                  '''
                  v_s= self.value_matrix[observation[0], observation[1]]
                  v_s1 = self.value_matrix[new_observation[0], new_observation[1]]
```

```
        self.utility.append(self.utility[-1]+s_alpha*(reward-self.utility[-1]))
        self.delta.append(reward - self.utility[-1] + v_s1 - v_s)
        new_value = v_s + s_alpha*self.delta[-1]
        self.value_matrix[observation[0], observation[1]] = new_value
        self.value_function.append(self.value_matrix.sum()/25)

    def alpha(self, t):
        return 1 / (divmod(t,50)[0]+1)

    def run_experiment(self):
        for step in range(self.EPOCHS+1):
            #Take the action from the action matrix
            action = self.sampling_technique.action()
            self.actions.append(action)
            #Move one step in the environment and get obs and reward
            reward, new_state = self.env.step(action)
            self.update_value_matrix(self.states[-1], new_state, reward,self.alpha(ste
            self.rewards.append(reward)
            self.states.append(new_state)
            self.sampling_technique.update(reward)
            if step%self.print_epoch==0:
                print(" {0:} | states: {1:}| rewards: {2:} | utility :{3:2f}| delta: {



    def plot_value_matrix(self):
        plt.figure(figsize=(15,7))
        sns.heatmap(self.value_matrix, annot=True, fmt='g')
        plt.title(r" $\vartheta$ state value")
        plt.xlabel("x-axis")
        plt.ylabel("y-axis")
        plt.tight_layout()
        plt.show()

    def plot_utility(self):
        fig, ax = plt.subplots()
        fig.set_figheight(7)
        fig.set_figwidth(15)
        plt.plot(self.utility)
        plt.title("Utility plot")
        textstr = '\n'.join((
            r'$\mu=%.2f$' % (np.mean(self.utility), ),
            r'$\mathrm{median}=%.2f$' % (np.median(self.utility), ),
            r'$last-value =%.2f$' % (self.utility[-1])))
        # these are matplotlib.patch.Patch properties
```

```
                props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
                ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                        verticalalignment='top',bbox =props)
                plt.xlabel(" Time ")
                plt.ylabel(" $\hat{J}   utility $")
                plt.show()

            def plot_value_function(self):
                fig, ax = plt.subplots()
                fig.set_figheight(7)
                fig.set_figwidth(15)
                plt.plot(self.value_function)
                plt.title("Average value plot")
                textstr = '\n'.join((
                    r'$\mu=%.2f$' % (np.mean(self.value_function), ),
                    r'$\mathrm{median}=%.2f$' % (np.median(self.value_function),),
                    r'$last-value =%.2f$' % (self.value_function[-1])))
                # these are matplotlib.patch.Patch properties
                props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
                ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                        verticalalignment='top',bbox =props)
                plt.xlabel(" Time ")
                plt.ylabel(r" $ \vartheta --- Average value $")
                plt.show()
```

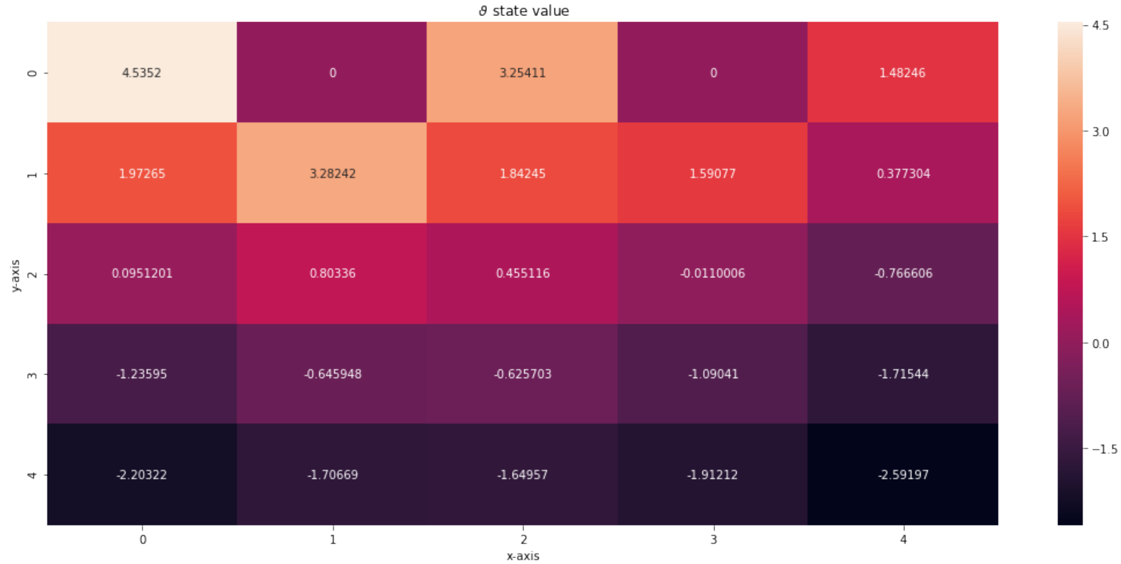
Solution 1

starting at position 13: 2,2

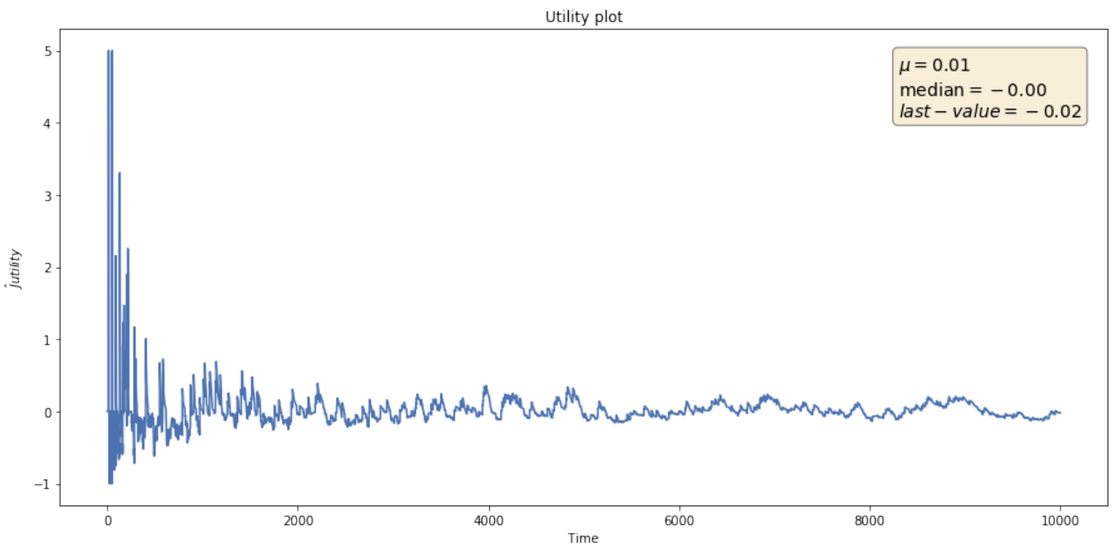
```
In [201]: q1_s1 = play([2,2],sampling_technique=sampling_random)
          q1_s1.run_experiment()
```

```
step:0 | states: [1, 2]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [4, 2]| rewards: 0 | utility :0.011729| delta: -0.38 | value_function: 0.13
step:2000 | states: [3, 2]| rewards: 0 | utility :-0.039797| delta: 0.50 | value_function: 0.13
step:3000 | states: [2, 3]| rewards: 0 | utility :-0.120334| delta: -1.05 | value_function: 0.1
step:4000 | states: [2, 2]| rewards: 0 | utility :0.226619| delta: -0.59 | value_function: 0.15
step:5000 | states: [4, 2]| rewards: -1 | utility :-0.080669| delta: -0.92 | value_function: 0.
step:6000 | states: [2, 2]| rewards: 0 | utility :0.003243| delta: 0.40 | value_function: 0.14
step:7000 | states: [1, 4]| rewards: 0 | utility :0.165174| delta: -1.49 | value_function: 0.15
step:8000 | states: [4, 2]| rewards: 0 | utility :-0.086608| delta: 0.29 | value_function: 0.14
step:9000 | states: [3, 4]| rewards: -1 | utility :0.170079| delta: -1.17 | value_function: 0.1
step:10000 | states: [2, 4]| rewards: 0 | utility :-0.019366| delta: 0.97 | value_function: 0.1
```

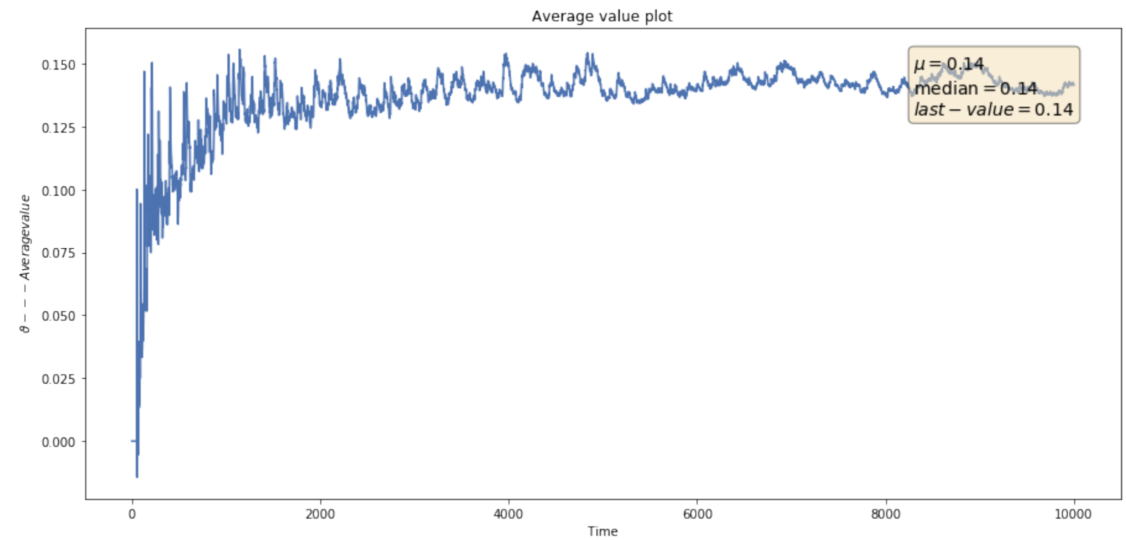
```
In [202]: q1_s1.plot_value_matrix()
```




```
In [203]: q1_s1.plot_utility()
```




```
In [204]: q1_s1.plot_value_function()
```



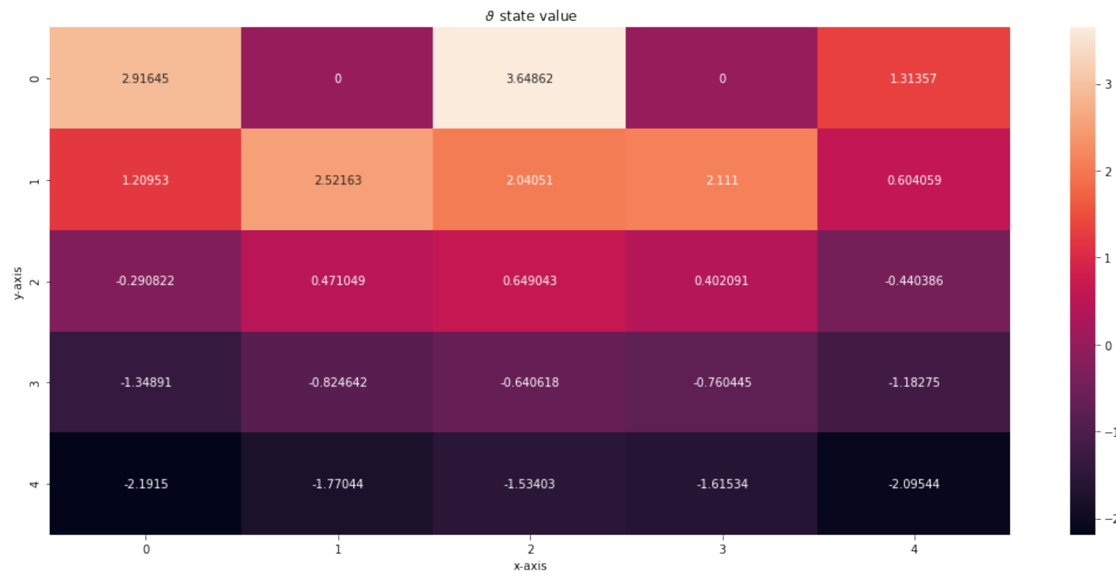
Solution 1

starting at position 2: 0,1

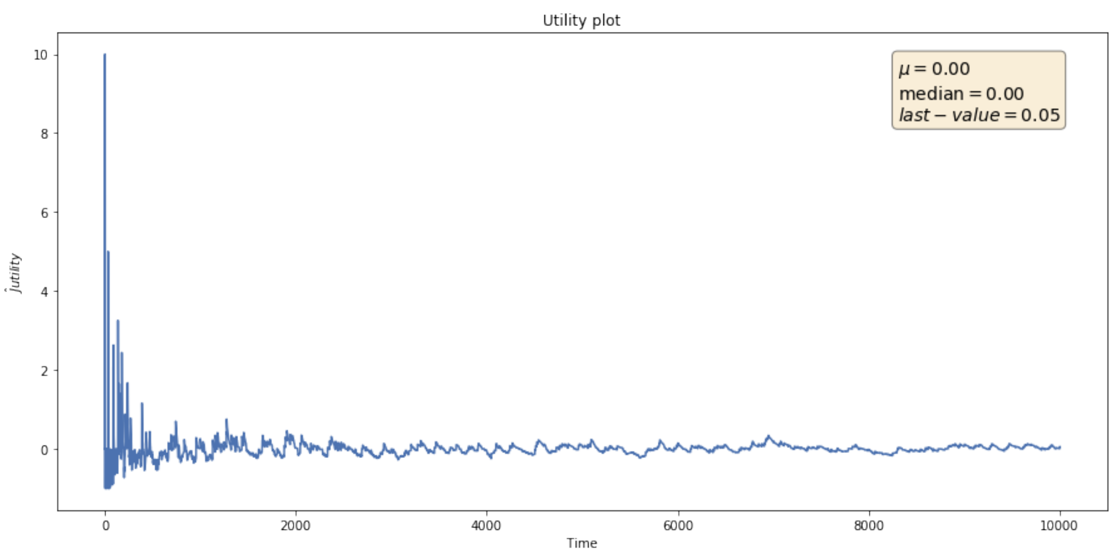
```
In [205]: q1_s2 = play([0,1], sampling_technique=sampling_random)
          q1_s2.run_experiment()
```

```
step:0 | states: [0, 0]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [3, 2]| rewards: 0 | utility :0.232730| delta: -0.18 | value_function: 0.12
step:2000 | states: [2, 3]| rewards: 0 | utility :0.019455| delta: 0.61 | value_function: 0.12
step:3000 | states: [3, 4]| rewards: 0 | utility :-0.008528| delta: 0.44 | value_function: 0.12
step:4000 | states: [4, 1]| rewards: 0 | utility :-0.062007| delta: -0.26 | value_function: 0.1
step:5000 | states: [3, 0]| rewards: -1 | utility :0.053836| delta: -1.05 | value_function: 0.1
step:6000 | states: [3, 1]| rewards: 10 | utility :0.146652| delta: 6.51 | value_function: 0.13
step:7000 | states: [2, 3]| rewards: 5 | utility :0.198219| delta: 3.98 | value_function: 0.13
step:8000 | states: [4, 3]| rewards: 0 | utility :-0.016491| delta: -0.05 | value_function: 0.1
step:9000 | states: [2, 3]| rewards: 5 | utility :0.089313| delta: 3.98 | value_function: 0.13
step:10000 | states: [4, 1]| rewards: 0 | utility :0.054571| delta: -1.01 | value_function: 0.1
```

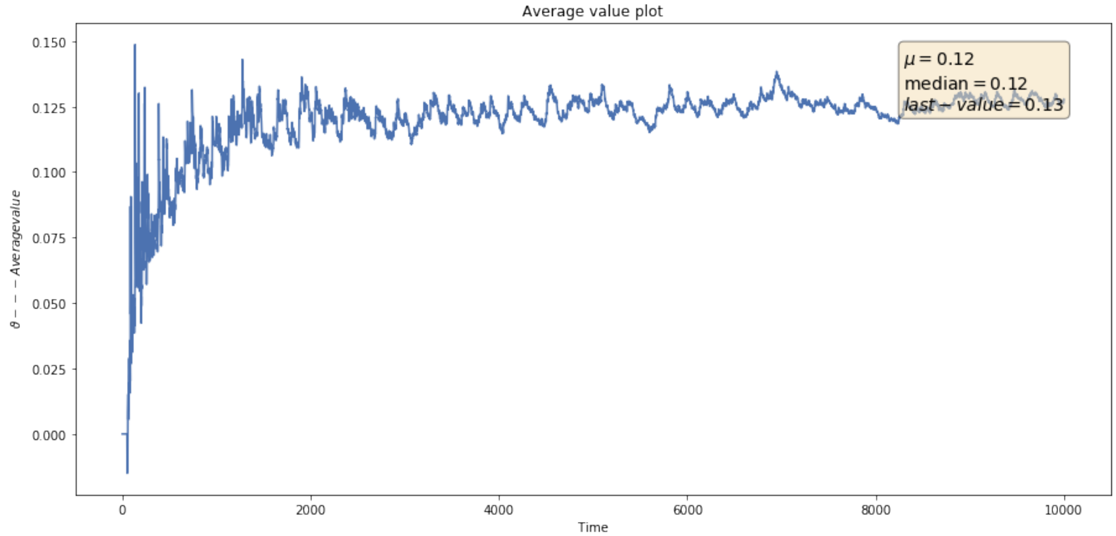
```
In [206]: q1_s2.plot_value_matrix()
```



In [207]: `q1_s2.plot_utility()`



In [208]: `q1_s2.plot_value_function()`



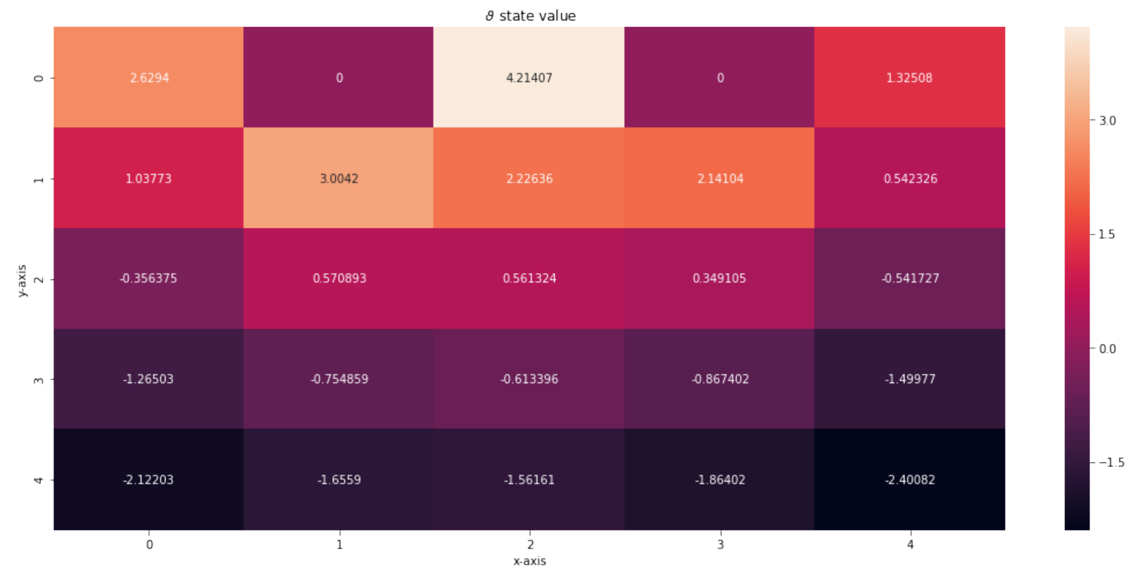
Solution 1

starting at position 25: 4,4

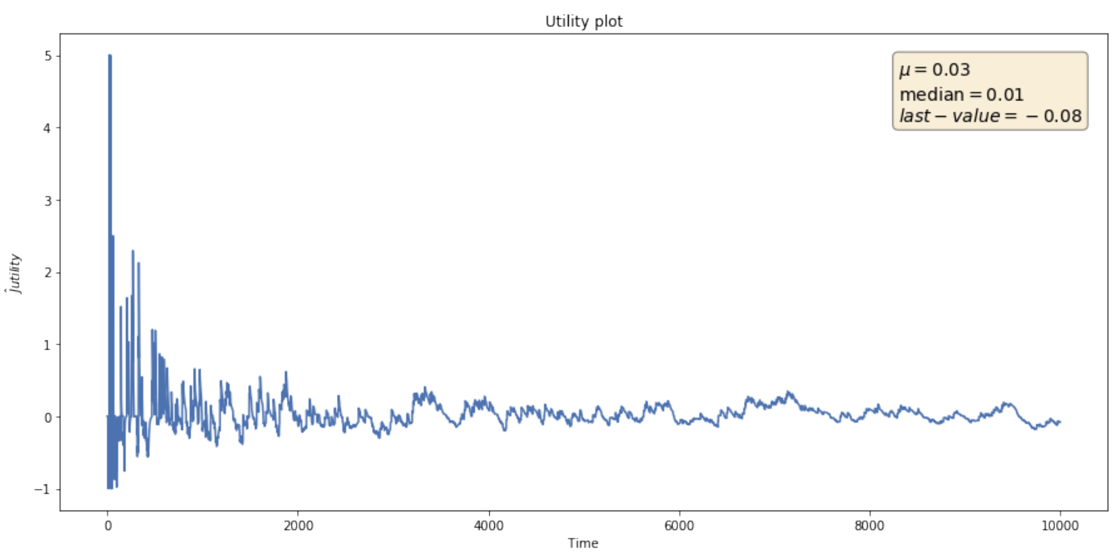
In [209]:
```
q1_s3 = play([4,4],sampling_technique=sampling_random)
q1_s3.run_experiment()
```

```
step:0 | states: [3, 4]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [2, 0]| rewards: 0 | utility :-0.197388| delta: 0.84 | value_function: 0.10
step:2000 | states: [4, 4]| rewards: -1 | utility :-0.118360| delta: -0.88 | value_function: 0.
step:3000 | states: [4, 2]| rewards: 0 | utility :0.054302| delta: -0.76 | value_function: 0.13
step:4000 | states: [4, 3]| rewards: -1 | utility :0.086272| delta: -1.09 | value_function: 0.1
step:5000 | states: [2, 4]| rewards: 0 | utility :-0.057735| delta: 0.66 | value_function: 0.12
step:6000 | states: [3, 2]| rewards: 0 | utility :-0.102569| delta: 0.19 | value_function: 0.12
step:7000 | states: [2, 2]| rewards: 0 | utility :0.190463| delta: 0.11 | value_function: 0.13
step:8000 | states: [1, 0]| rewards: 0 | utility :0.095634| delta: -2.12 | value_function: 0.13
step:9000 | states: [1, 1]| rewards: 0 | utility :-0.014648| delta: 2.41 | value_function: 0.13
step:10000 | states: [4, 2]| rewards: 0 | utility :-0.082392| delta: 0.18 | value_function: 0.1
```

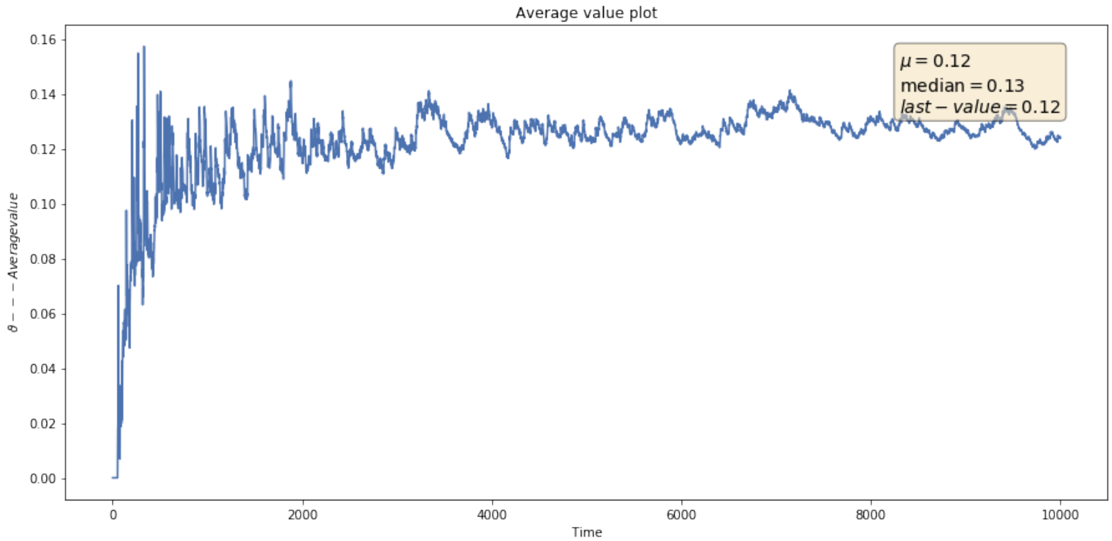
In [210]:
```
q1_s3.plot_value_matrix()
```




```
In [211]: q1_s3.plot_utility()
```




```
In [212]: q1_s3.plot_value_function()
```



Python code for Question 2

```
In [213]: class play_scalar:

              def __init__(self, initial_state, sampling_techique):
                  self.rewards = []
                  self.utility = [0]
                  self.value_function = []
                  self.delta = []
                  self.states = []
                  self.initial_state = initial_state
                  self.state_matrix = (np.array(range(25)).reshape(5,5)+1)/25
                  self.state_no = self.state_matrix[self.initial_state[0],self.initial_state[1]]
                  self.states.append(self.state_no)
                  self.env = gridworld(initial_state=self.initial_state)
                  self.EPOCHS = EPOCHS
                  self.print_epoch = EPOCHS/10
                  self.actions = []
                  self.sampling_techique = sampling_techique

              def update_value_matrix(self, old_state, new_state, reward, s_alpha):
                  '''Return the updated utility matrix
                  @param utility_matrix the matrix before the update
                  @return the updated utility matrix
                  '''
                  v_s= old_state
                  v_s1 = new_state
                  self.utility.append(self.utility[-1]+s_alpha*(reward-self.utility[-1]))
                  self.delta.append(reward - self.utility[-1] + v_s1 - v_s)
```

```python
        new_value = v_s + s_alpha*self.delta[-1]
        self.value_function.append(new_value)

    def alpha(self, t):
        return 1 / (divmod(t,50)[0]+1)

    def run_experiment(self):
        for step in range(self.EPOCHS+1):
            #Take the action from the action matrix
            action = self.sampling_techique.action()
            self.actions.append(action)
            #Move one step in the environment and get obs and reward
            reward, new_state = self.env.step(action)
            new_state = self.state_matrix[new_state[0],new_state[1]]
            self.update_value_matrix(self.states[-1], new_state, reward,self.alpha(ste
            self.sampling_techique.update(reward)
            self.rewards.append(reward)
            self.states.append(new_state)
            if step%self.print_epoch==0:
                print(" {0:} | states: {1:}| rewards: {2:} | utility :{3:2f}| delta: {





    def plot_utility(self):
        fig, ax = plt.subplots()
        fig.set_figheight(7)
        fig.set_figwidth(15)
        plt.plot(self.utility)
        plt.title("Utility plot")
        textstr = '\n'.join((
            r'$\mu=%.2f$' % (np.mean(self.utility), ),
            r'$\mathrm{median}=%.2f$' % (np.median(self.utility), ),
            r'$last-value =%.2f$' % (self.utility[-1])))
        # these are matplotlib.patch.Patch properties
        props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
        ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                verticalalignment='top',bbox =props)
        plt.xlabel(" Time ")
        plt.ylabel(" $\hat{J}   utility $")
        plt.show()

    def plot_value_function(self):
        fig, ax = plt.subplots()
        fig.set_figheight(7)
        fig.set_figwidth(15)
```

```
                plt.plot(self.value_function)
                plt.title("Average value plot")
                textstr = '\n'.join((
                    r'$\mu=%.2f$' % (np.mean(self.value_function), ),
                    r'$\mathrm{median}=%.2f$' % (np.median(self.value_function),),
                    r'$last-value =%.2f$' % (self.value_function[-1])))
                # these are matplotlib.patch.Patch properties
                props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
                ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                        verticalalignment='top',bbox =props)
                plt.xlabel(" Time ")
                plt.ylabel(r" $ \vartheta --- Average value $")
                plt.show()
```

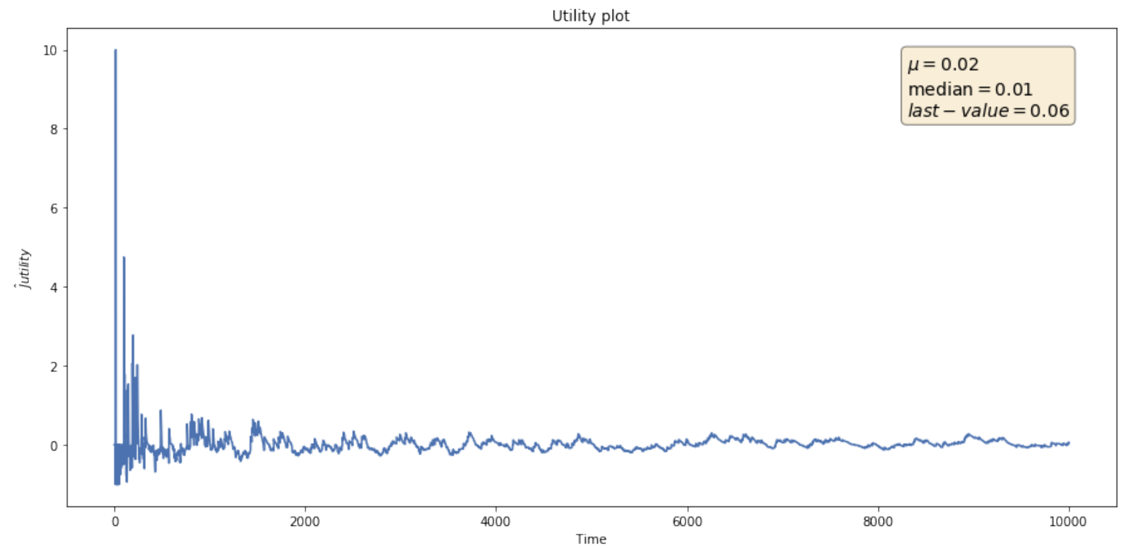
Solution 2

Starting position : 13

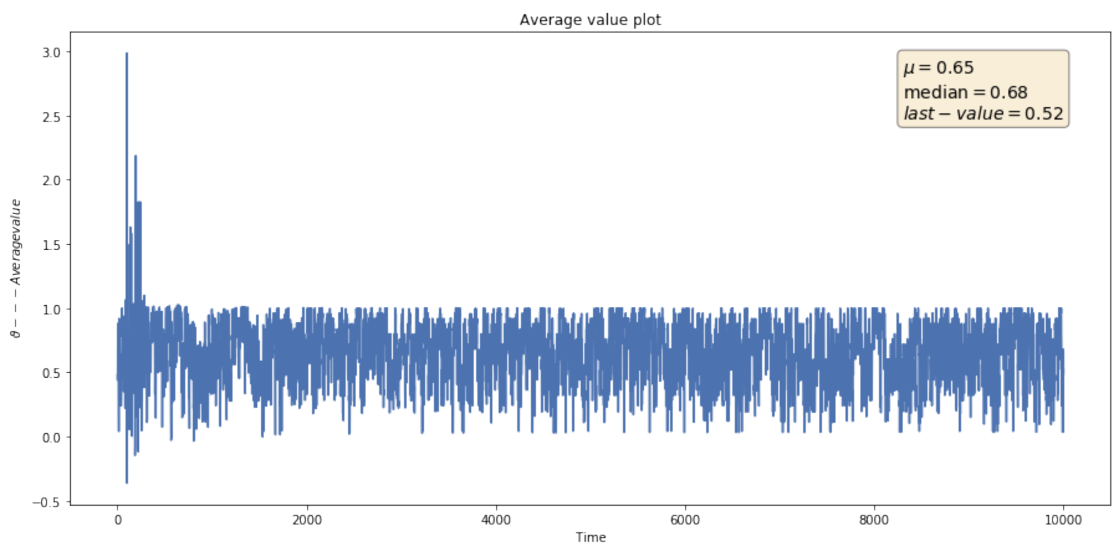
```
In [214]: q2_s1 = play_scalar([2,2], sampling_techique=sampling_random)
          q2_s1.run_experiment()
```

```
step:0 | states: 0.48| rewards: 0 | utility :0.000000| delta: -0.04 | value_function: 0.48 | up
step:1000 | states: 0.92| rewards: -1 | utility :0.059927| delta: -1.06 | value_function: 0.87
step:2000 | states: 0.32| rewards: 0 | utility :-0.099748| delta: -0.10 | value_function: 0.52
step:3000 | states: 0.64| rewards: 0 | utility :0.226502| delta: -0.43 | value_function: 0.83 |
step:4000 | states: 0.68| rewards: 0 | utility :0.059714| delta: 0.14 | value_function: 0.48 |
step:5000 | states: 0.68| rewards: 0 | utility :0.043617| delta: -0.00 | value_function: 0.64 |
step:6000 | states: 1.0| rewards: -1 | utility :0.127050| delta: -1.13 | value_function: 0.99 |
step:7000 | states: 0.92| rewards: 0 | utility :0.057832| delta: -0.02 | value_function: 0.88 |
step:8000 | states: 0.68| rewards: 0 | utility :-0.023922| delta: -0.02 | value_function: 0.72
step:9000 | states: 0.8| rewards: -1 | utility :0.185359| delta: -1.19 | value_function: 0.79 |
step:10000 | states: 0.72| rewards: 0 | utility :0.059977| delta: 0.14 | value_function: 0.52 |
```

```
In [215]: q2_s1.plot_utility()
```




```
In [216]: q2_s1.plot_value_function()
```



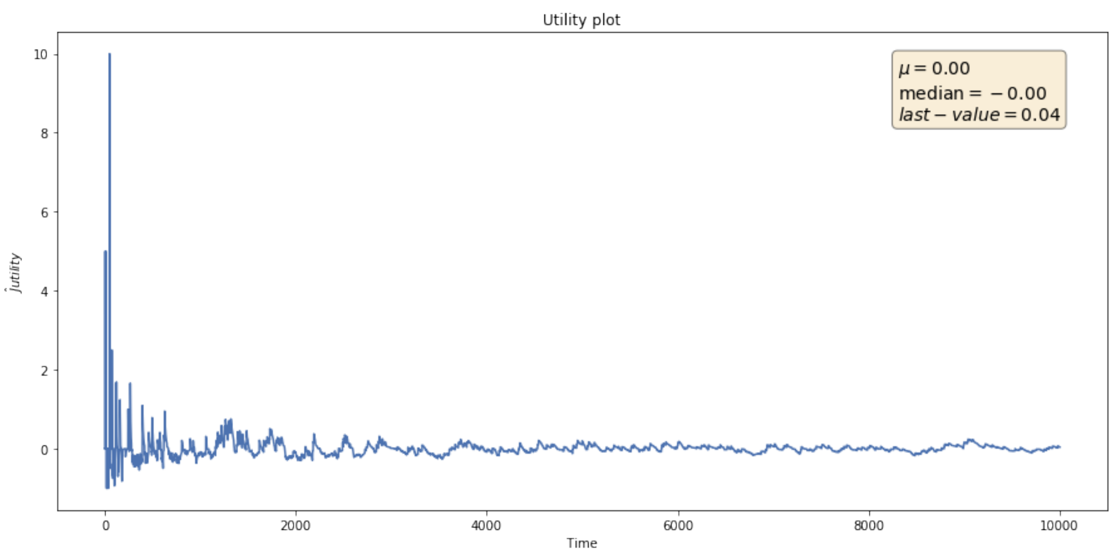
Solution 2

Starting position : 2

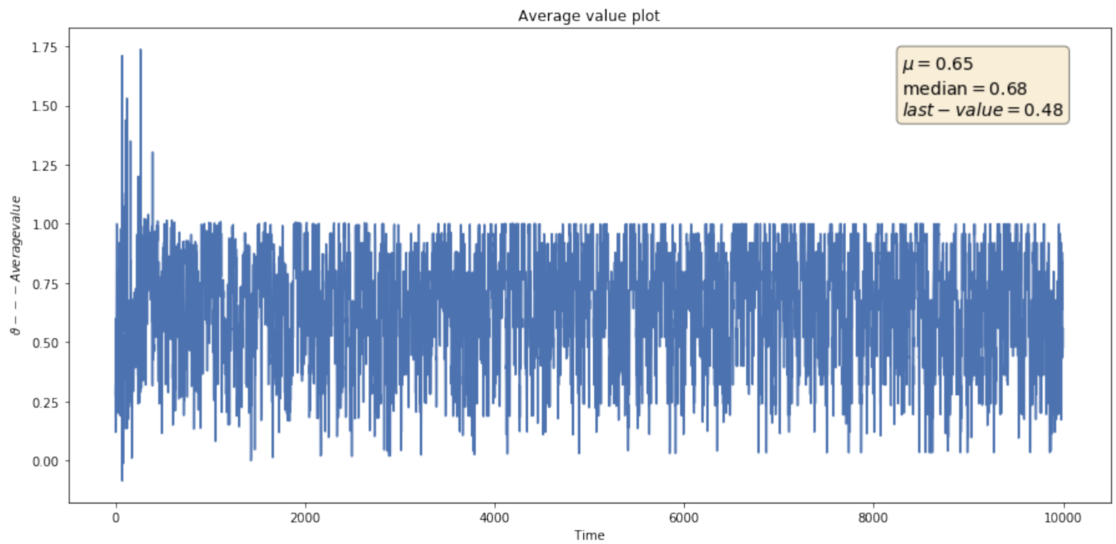
```
In [217]: q2_s2 = play_scalar([0,1], sampling_techique=sampling_random)
          q2_s2.run_experiment()
```

```
step:0 | states: 0.28| rewards: 0 | utility :0.000000| delta: 0.20 | value_function: 0.28 | upd
step:1000 | states: 0.8| rewards: 0 | utility :-0.083295| delta: 0.12 | value_function: 0.77 |
step:2000 | states: 0.88| rewards: -1 | utility :-0.118272| delta: -0.88 | value_function: 0.86
step:3000 | states: 0.52| rewards: 0 | utility :0.012383| delta: 0.19 | value_function: 0.32 |
step:4000 | states: 0.6| rewards: 0 | utility :-0.078025| delta: -0.12 | value_function: 0.80 |
step:5000 | states: 0.64| rewards: -1 | utility :0.186699| delta: -1.19 | value_function: 0.63
step:6000 | states: 0.12| rewards: 0 | utility :0.072009| delta: -0.27 | value_function: 0.32 |
step:7000 | states: 0.68| rewards: 0 | utility :0.055853| delta: -0.02 | value_function: 0.64 |
step:8000 | states: 0.64| rewards: 0 | utility :0.025215| delta: 0.17 | value_function: 0.44 |
step:9000 | states: 0.64| rewards: 0 | utility :0.167435| delta: -0.21 | value_function: 0.68 |
step:10000 | states: 0.28| rewards: 0 | utility :0.042055| delta: -0.24 | value_function: 0.48
```

In [218]: q2_s2.plot_utility()



In [219]: q2_s2.plot_value_function()



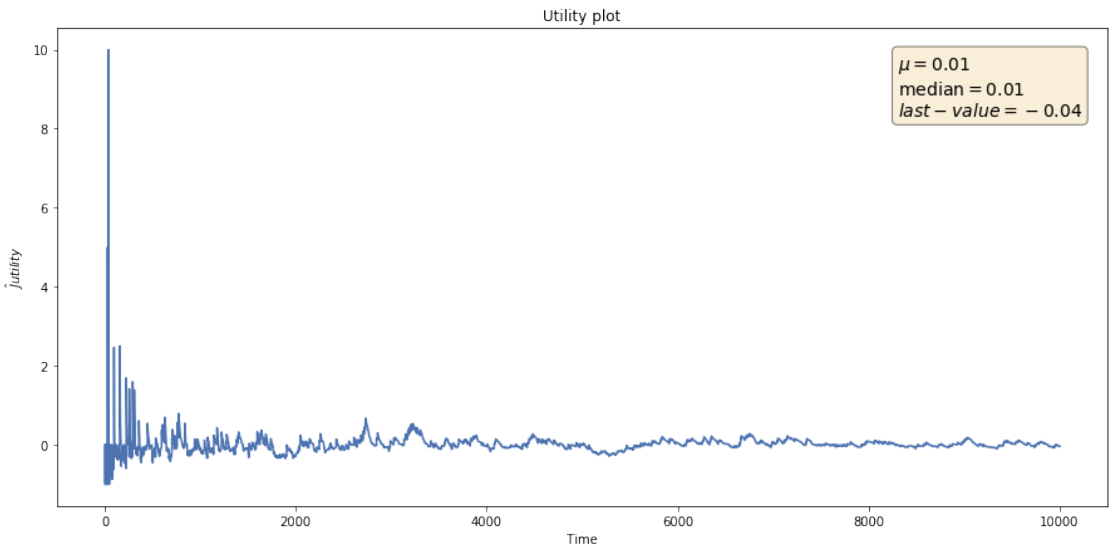
Solution 2

Starting position : 25

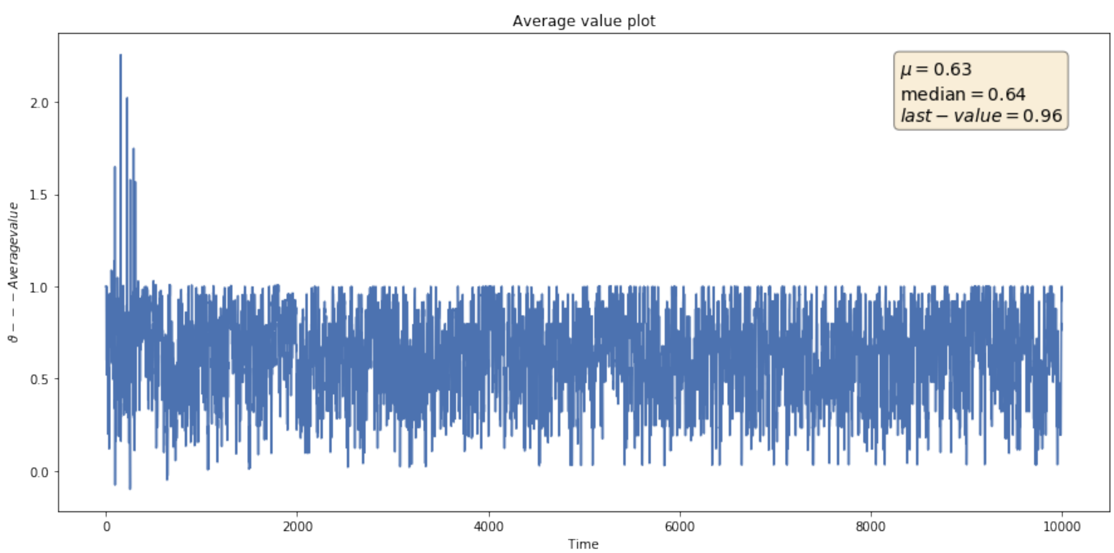
```
In [220]: q2_s3 = play_scalar([4,4], sampling_techique=sampling_random)
          q2_s3.run_experiment()
```

```
step:0 | states: 1.0| rewards: -1 | utility :-1.000000| delta: 0.00 | value_function: 1.00 | up
step:1000 | states: 0.96| rewards: 0 | utility :-0.117880| delta: 0.16 | value_function: 0.93 |
step:2000 | states: 0.48| rewards: 0 | utility :-0.262693| delta: 0.22 | value_function: 0.53 |
step:3000 | states: 0.92| rewards: 0 | utility :0.079772| delta: -0.04 | value_function: 0.88 |
step:4000 | states: 0.2| rewards: 0 | utility :-0.024573| delta: -0.18 | value_function: 0.40 |
step:5000 | states: 0.32| rewards: 0 | utility :-0.010936| delta: -0.03 | value_function: 0.36
step:6000 | states: 0.36| rewards: 0 | utility :0.044383| delta: -0.00 | value_function: 0.32 |
step:7000 | states: 0.88| rewards: 0 | utility :0.068561| delta: -0.11 | value_function: 0.92 |
step:8000 | states: 0.92| rewards: 0 | utility :0.077211| delta: -0.04 | value_function: 0.88 |
step:9000 | states: 0.32| rewards: 0 | utility :0.091712| delta: -0.29 | value_function: 0.52 |
step:10000 | states: 1.0| rewards: 0 | utility :-0.039593| delta: 0.08 | value_function: 0.96 |
```

```
In [221]: q2_s3.plot_utility()
```



In [222]: `q2_s3.plot_value_function()`



Experimenting with policy

Notes on epsilon greedy algorithm

- Choose an arbitrary low probability value $e = 0.1$
- Randomly chose actions for n trails
- On each trial estimate the reward for each variant
- After n trails:

  - select $1 - eExploit)onerandomly(Explore)$

Notes on Thompson sampling

The Bayesian way

- Set a uniform prior distribution between 0 and 1 for each action's reward
- Draw a parametere θ from the prior distribution
- Select the action that is associated with the highest parameter θ
- Observe the reward and update the distribution parameter

Note Both the Action policy method are for commonly used for bernouli bandits, But here i have assumed receiving positive reward as 1 and any other reward is 0

In [223]:
```
from sampling_algorithm import Thompson, eGreedy, UCB
```

In [224]:
```
sampling_greedy = eGreedy(4, 0.1)
sampling_thompson = Thompson(4)
sampling_random = Random(4)
```

In [225]:
```
class play:

        def __init__(self, initial_state, sampling_technique):
            self.rewards = []
            self.utility = [0]
            self.value_matrix = np.zeros((5,5))
            self.delta = []
            self.states = []
            self.initial_state = initial_state
            self.states.append(initial_state)
            self.value_function = []
            self.env = gridworld(initial_state=initial_state)
            self.EPOCHS = EPOCHS
            self.print_epoch = EPOCHS/10
            self.actions = []
            self.sampling_tech = sampling_technique

        def update_value_matrix(self, observation, new_observation, reward, s_alpha):
            '''Return the updated utility matrix
            @param utility_matrix the matrix before the update
            @return the updated utility matrix
            '''
            v_s= self.value_matrix[observation[0], observation[1]]
            v_s1 = self.value_matrix[new_observation[0], new_observation[1]]
            self.utility.append(self.utility[-1]+s_alpha*(reward-self.utility[-1]))
            self.delta.append(reward - self.utility[-1] + v_s1 - v_s)
            new_value = v_s + s_alpha*self.delta[-1]
            self.value_matrix[observation[0], observation[1]] = new_value
            self.value_function.append(self.value_matrix.sum()/25)
```

```python
def alpha(self, t):
    return 1 / (divmod(t,50)[0]+1)

def run_experiment(self):
    for step in range(self.EPOCHS+1):
        #Take the action from the action matrix
        #action = np.random.randint(4)
        action = self.sampling_tech.action()
        self.actions.append(action)
        #Move one step in the environment and get obs and reward
        reward, new_state = self.env.step(action)
        self.sampling_tech.update(reward)
        self.update_value_matrix(self.states[-1], new_state, reward,self.alpha(ste
        self.rewards.append(reward)
        self.states.append(new_state)
        if step%self.print_epoch==0:
            print(" {0:} | states: {1:}| rewards: {2:} | utility :{3:2f}| delta: {




def plot_value_matrix(self):
    plt.figure(figsize=(15,7))
    sns.heatmap(self.value_matrix, annot=True,fmt='g')
    plt.title(r" $\vartheta$ state value")
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    plt.tight_layout()
    plt.show()

def plot_utility(self):
    fig, ax = plt.subplots()
    fig.set_figheight(7)
    fig.set_figwidth(15)
    plt.plot(self.utility)
    plt.title("Utility plot")
    textstr = '\n'.join((
        r'$\mu=%.2f$' % (np.mean(self.utility), ),
        r'$\mathrm{median}=%.2f$' % (np.median(self.utility), ),
        r'$last-value =%.2f$' % (self.utility[-1])))
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
    ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
            verticalalignment='top',bbox =props)
    plt.xlabel(" Time ")
    plt.ylabel(" $\hat{J}   utility $")
```

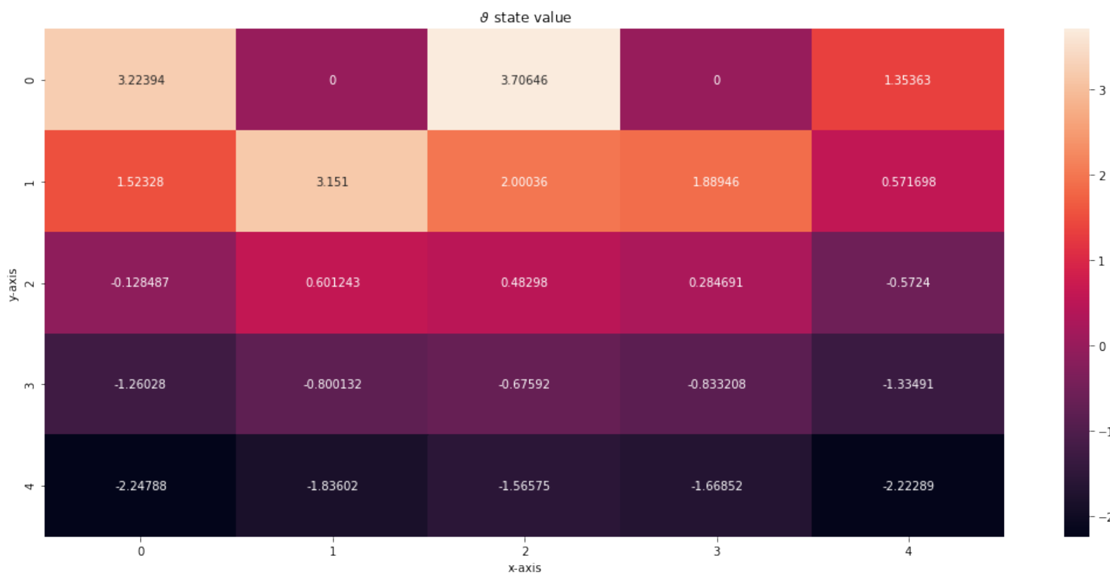
```
            plt.show()

        def plot_value_function(self):
            fig, ax = plt.subplots()
            fig.set_figheight(7)
            fig.set_figwidth(15)
            plt.plot(self.value_function)
            plt.title("Average value plot")
            textstr = '\n'.join((
                r'$\mu=%.2f$' % (np.mean(self.value_function), ),
                r'$\mathrm{median}=%.2f$' % (np.median(self.value_function),),
                r'$last-value =%.2f$' % (self.value_function[-1])))
            # these are matplotlib.patch.Patch properties
            props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
            ax.text(0.8, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                    verticalalignment='top',bbox =props)
            plt.xlabel(" Time ")
            plt.ylabel(r" $ \vartheta --- Average value $")
            plt.show()
```

In [226]:
```
q1_thompson = play([2,2], sampling_thompson)
q1_egreedy = play([2,2], sampling_greedy)
q1_random = play([2,2], sampling_random)
q1_thompson.run_experiment()
q1_egreedy.run_experiment()
q1_random.run_experiment()
```

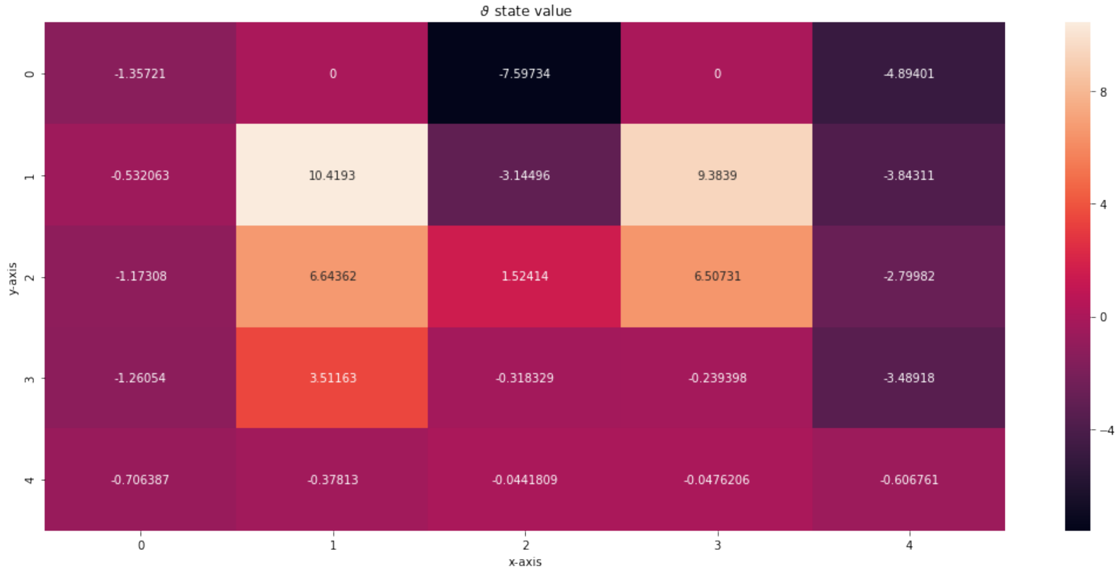
```
step:0 | states: [2, 1]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [0, 4]| rewards: 0 | utility :0.547327| delta: 0.37 | value_function: 0.22
step:2000 | states: [3, 1]| rewards: 10 | utility :0.963507| delta: 3.08 | value_function: 0.25
step:3000 | states: [3, 2]| rewards: 0 | utility :0.885020| delta: -0.42 | value_function: 0.25
step:4000 | states: [1, 0]| rewards: -1 | utility :1.449031| delta: -2.45 | value_function: 0.2
step:5000 | states: [2, 3]| rewards: 5 | utility :1.275350| delta: 1.61 | value_function: 0.27
step:6000 | states: [0, 0]| rewards: 0 | utility :0.627895| delta: -0.22 | value_function: 0.25
step:7000 | states: [1, 3]| rewards: 0 | utility :0.774370| delta: 1.25 | value_function: 0.26
step:8000 | states: [3, 1]| rewards: 10 | utility :0.783008| delta: 3.74 | value_function: 0.26
step:9000 | states: [0, 2]| rewards: -1 | utility :0.562080| delta: -1.56 | value_function: 0.2
step:10000 | states: [1, 0]| rewards: 0 | utility :0.743429| delta: -3.12 | value_function: 0.2
step:0 | states: [3, 2]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [2, 0]| rewards: 0 | utility :-0.218975| delta: -2.97 | value_function: 0.1
step:2000 | states: [1, 3]| rewards: 0 | utility :1.776034| delta: 0.80 | value_function: 0.24
step:3000 | states: [3, 4]| rewards: -1 | utility :-0.028903| delta: -0.97 | value_function: 0.
step:4000 | states: [3, 1]| rewards: 10 | utility :2.589210| delta: 0.52 | value_function: 0.28
step:5000 | states: [0, 4]| rewards: -1 | utility :-0.602078| delta: -0.40 | value_function: 0.
step:6000 | states: [0, 2]| rewards: -1 | utility :0.874062| delta: -1.87 | value_function: 0.2
step:7000 | states: [3, 3]| rewards: 0 | utility :0.089544| delta: -0.07 | value_function: 0.19
step:8000 | states: [2, 2]| rewards: 0 | utility :0.327589| delta: -5.33 | value_function: 0.20
step:9000 | states: [1, 1]| rewards: 0 | utility :0.511686| delta: 3.64 | value_function: 0.21
```

```
step:10000 | states: [2, 3]| rewards: 5 | utility :0.880549| delta: 1.25 | value_function: 0.22
step:0 | states: [1, 2]| rewards: 0 | utility :0.000000| delta: 0.00 | value_function: 0.00 | u
step:1000 | states: [1, 1]| rewards: 0 | utility :-0.185525| delta: 1.78 | value_function: 0.13
step:2000 | states: [3, 3]| rewards: 0 | utility :-0.039641| delta: -0.54 | value_function: 0.1
step:3000 | states: [4, 1]| rewards: 0 | utility :-0.129948| delta: -0.16 | value_function: 0.1
step:4000 | states: [4, 0]| rewards: 0 | utility :-0.150245| delta: -0.58 | value_function: 0.1
step:5000 | states: [4, 2]| rewards: -1 | utility :-0.065794| delta: -0.93 | value_function: 0.
step:6000 | states: [4, 1]| rewards: 0 | utility :0.023740| delta: 0.46 | value_function: 0.15
step:7000 | states: [3, 4]| rewards: 0 | utility :-0.145821| delta: -0.39 | value_function: 0.1
step:8000 | states: [4, 2]| rewards: 0 | utility :0.078479| delta: 0.11 | value_function: 0.15
step:9000 | states: [4, 0]| rewards: 0 | utility :0.125277| delta: -0.54 | value_function: 0.15
step:10000 | states: [3, 3]| rewards: 0 | utility :-0.008535| delta: 0.85 | value_function: 0.1
```

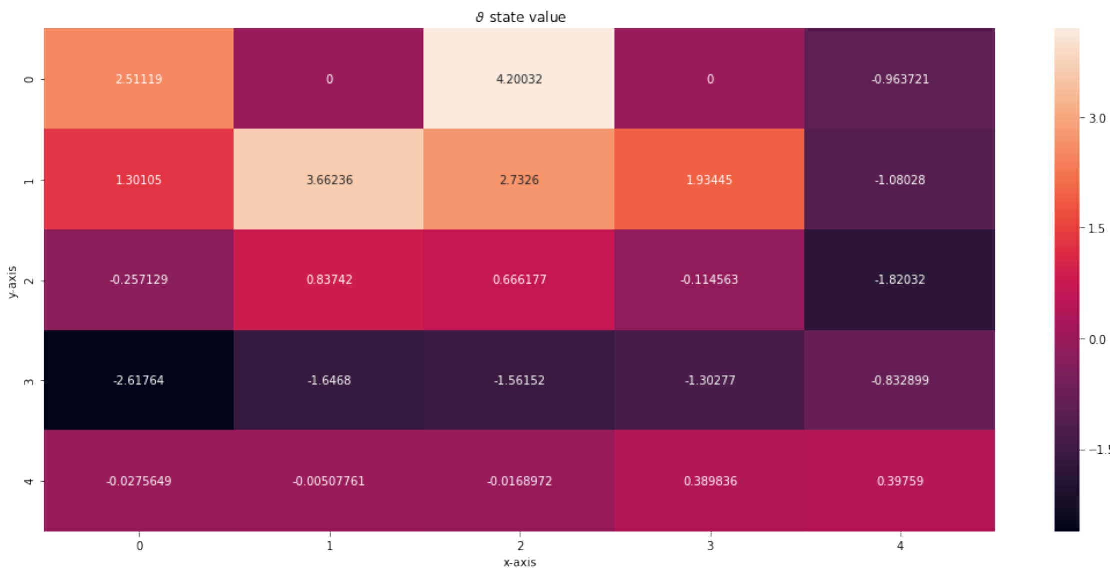
In [227]: q1_random.plot_value_matrix()



In [228]: q1_egreedy.plot_value_matrix()

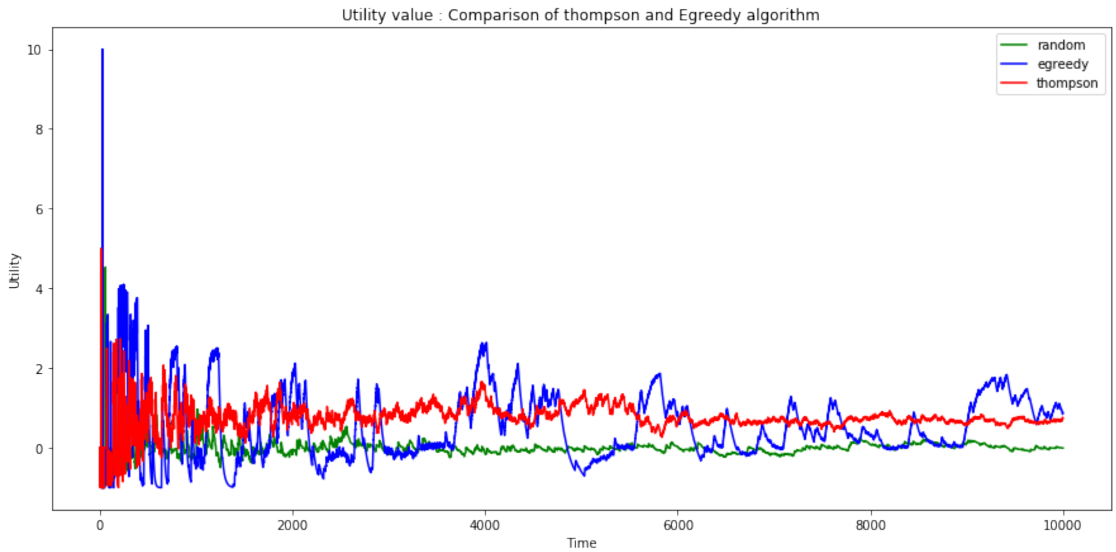



```
In [229]: q1_thompson.plot_value_matrix()
```

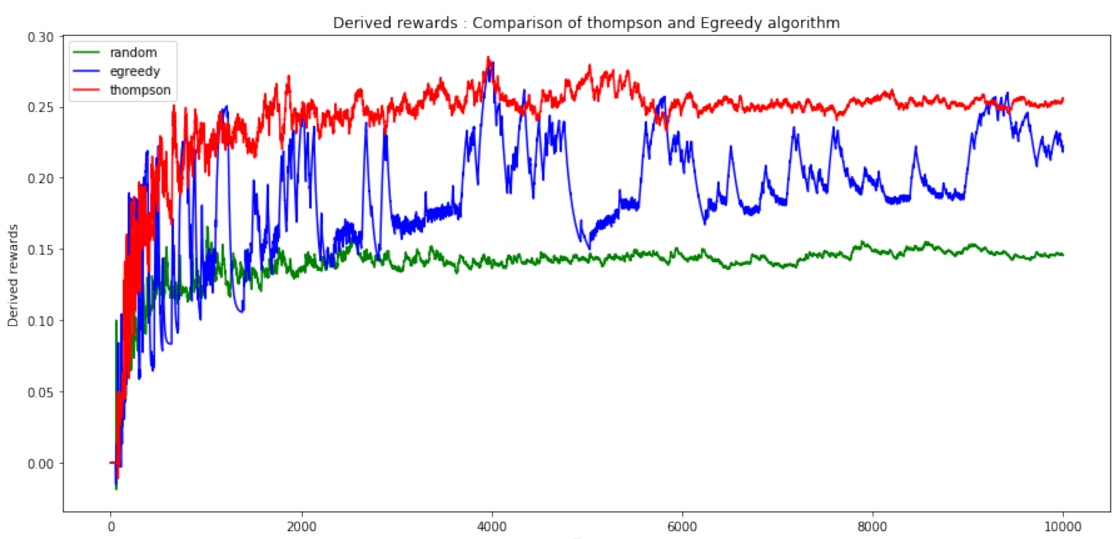



```
In [230]: plt.figure(figsize=(15,7))
          plt.plot(q1_random.utility, color='green', label="random")
          plt.plot(q1_egreedy.utility, color='blue', label="egreedy")
          plt.plot(q1_thompson.utility, color='red', label="thompson")
          plt.legend()
          plt.title("Utility value : Comparison of thompson and Egreedy algorithm")
```

```
plt.xlabel("Time")
plt.ylabel("Utility")
plt.show()
```




```
In [231]: plt.figure(figsize=(15,7))
          plt.plot(q1_random.value_function, color="green", label="random")
          plt.plot(q1_egreedy.value_function, color='blue', label="egreedy")
          plt.plot(q1_thompson.value_function, color='red', label="thompson")
          plt.legend()
          plt.title("Derived rewards : Comparison of thompson and Egreedy algorithm")
          plt.ylabel("Derived rewards")
          plt.xlabel("Time")
          plt.show()
```



```
In [233]: # Creates four polar axes, and accesses them through the returned array
          plt.figure(figsize=(15,7))
          plt.style.use('seaborn-deep')
          plt.hist([sampling_random.chosen_arms, sampling_greedy.chosen_arms, sampling_thompson.
                   label=['Random','EGreedy', 'Thompson'])
          plt.xticks(range(4),["up","right","down","left"])
          plt.legend(loc='upper right')
          plt.title("Actions taken by different action policies")
          plt.show()
```

Conclusion

- The starting state has no influence on the Average value function and utility value over time
- On comparsing random action policy with epsilon greedy (classic appraoch) and Thompson sampling (Bayesian approach) with random action policy, Thompson sampling seems to be getting long term reward
- Random policy gives equal probability for all the actions. While epsilon greedy algorithm and thompson sampling algorithms chooses actions based on the reward received by them
- Thompson algorithm chooses **up** most of the time and **left** and **right** the second most and **down** becomes a less reasonable option. Which is expected, since the moving up is the only way to gain more reward once the robot gets out the special state S'.
- Epsilon greedy algorithm chooses the most profitable action (as expected) which is **up** and if it receives negative on taking the same action multiple times goes for other options **right** and **left** but explores lesser compared to the thompson sampling and also depends on a external parameter α which is chosen as 0.1 in this case.

Reference:

- Reinforcement Learning - Richard S Sutton
- Artifical Intelligence - Modern approach - Stuart Russell & Peter Norvig
- Comparison thompson sampling and epsilon greedy