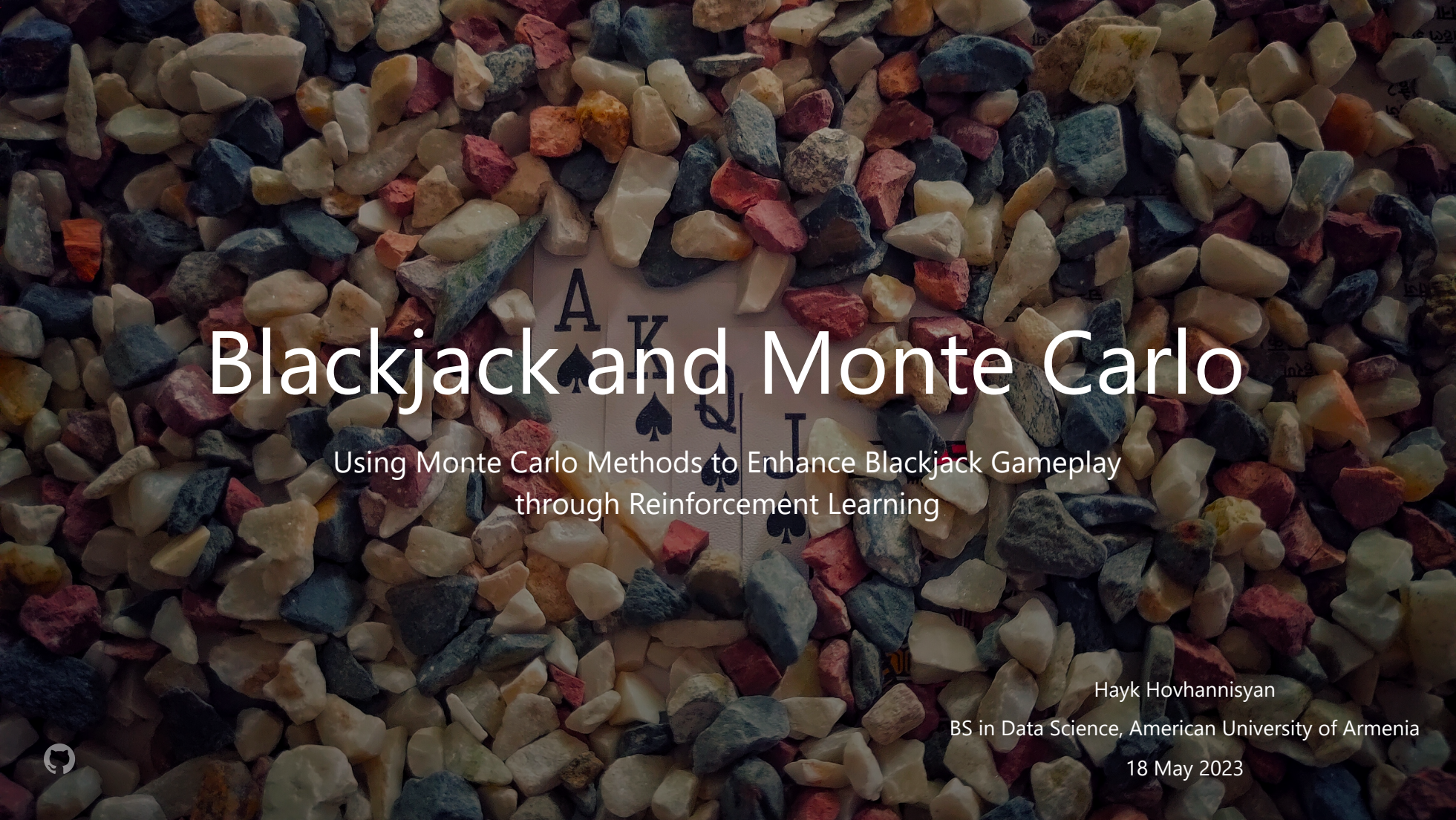


Blackjack and Monte Carlo

Using Monte Carlo Methods to Enhance Blackjack Gameplay through Reinforcement Learning

Hayk Hovhannisyan

BS in Data Science, American University of Armenia

18 May 2023

Table of contents

Blackjack

Rules

- ♣ Objective
  - ♦ Get close to 21
- ♥ Values
  - ♦ J Q K → 10
  - ♥ A → 1 or 11
- ♠ Player 2 card
- ♣ Dealer 1 face up 1 face down
- ♠ No suits

Environment

♣ Infinite Deck

Action Space

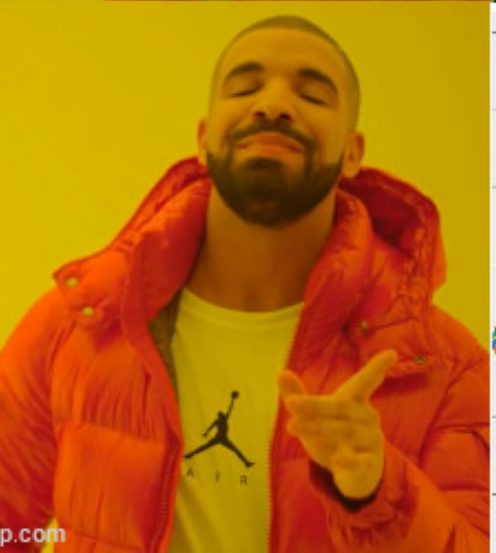
Hit (1) or Stick (0)

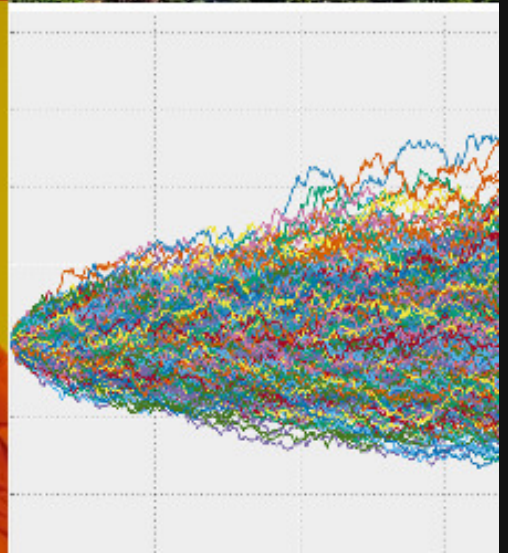
Observation space

(**Players Sum** , **Dealers Up card**, **Usable Ace**)
(`int` , `int`, `bool`)

Terminal statets

Win (1) Lose (-1) Draw (0)





Monte Carlo Simulation

♥ ~~Prior knowledge~~ Experience

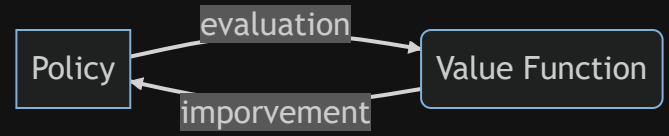
♦ ~~Transitions Probabilty Distribution~~

Sample Transitions

♣ Averaging Sampling Results

♠ Discrete finite Episodes → Experience

♣ Policy Control



My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns(S_t, A_t))

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to Returns$(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

My implementation

```python
import gym
import numpy as np
from collections import defaultdict

###

def mc_control(num_ep, gamma, eps, eps_min, decay):
    Q = defaultdict(lambda: np.ones(2)*-np.inf)
    r = defaultdict(lambda: list())

    for _ in range(num_episodes):
        episode = generate_episode(Q, eps)
        G = 0
        for t in range(len(episode)-1,-1,-1):
            state, action, reward = episode[t]

            G = gamma*G+reward
            r[state, action].append(G)
            Q[state][action] = np.mean(r[state,action])

        eps = max(eps_min, eps*eps_decay)
    return Q
```

MC Control

Initialize:

$\pi(s) \in A(s)(\text{arbitrarily}), \forall s \in S$

$Q(s,a) \in \mathbb{R}(\text{arbitrarily}), \forall s \in S, a \in A(s)$

$\text{Returns}(s,a) \longleftarrow \text{empty list}, \forall s \in S, a \in A(s)$

Loop forever ($\forall$ epsisode:)

Choose $S_0 \in S, A_0 \in A(S_0)$ randomly, s.d. $P(S_0, A_0) > 0$

Generate episode from S_0, A_0 following π :

$S_0, A_0, R_1, ..., S_{T-1}, A_{T_1}, R_T$

$G \leftarrow 0$

Loop ove each episode,$t = T-1, T-2, ..., 0$:

$G \leftarrow \gamma G + R_{S_t, A_t}$:

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Append$G$ to $\text{Returns}(S_t, A_t))$

$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$

$\pi(S_t) \leftarrow (argmax)_a Q(S_t, a)$

Source: Sutton et al

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont'd

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

EPISODE GENERATION

```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

Cont’d

GREEDY POLICY

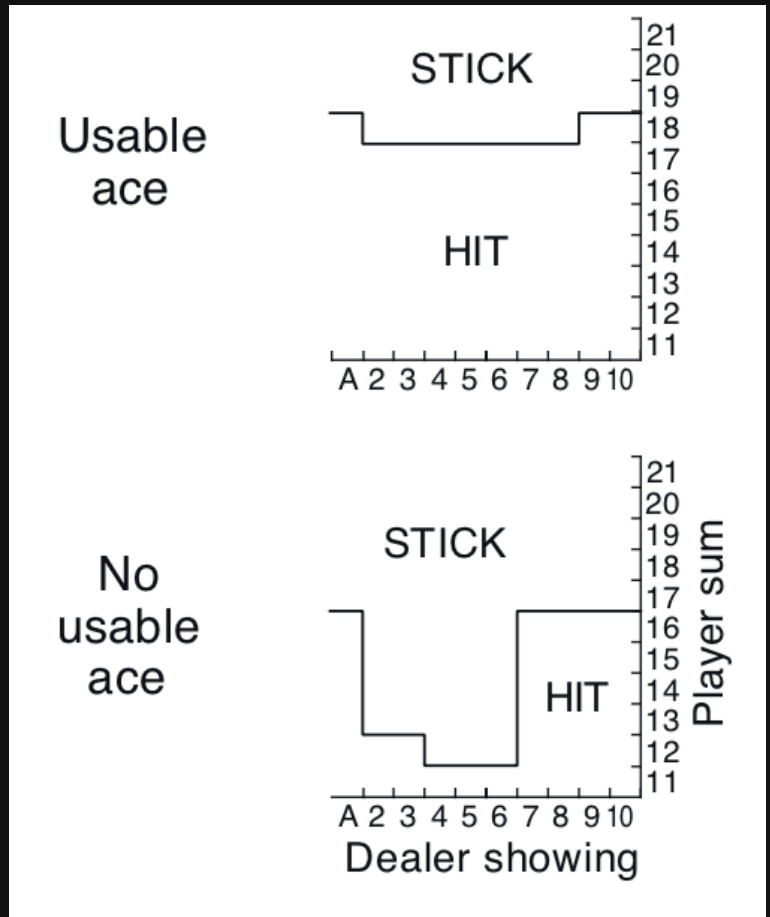
```
def eps_greedy(Q, state, epsilon):
    if state[0]<12:
        return 1
    elif np.random.random() < epsilon:
        # explore
        return np.random.choice([0,1])
    else:
        # exploit
        return np.argmax(Q[state])

#
```

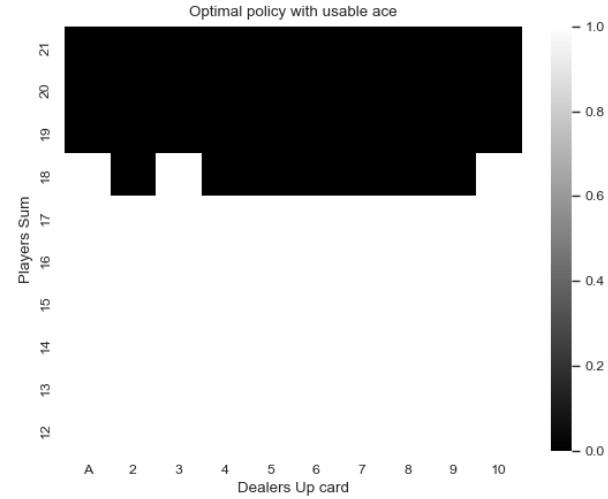
EPISODE GENERATION

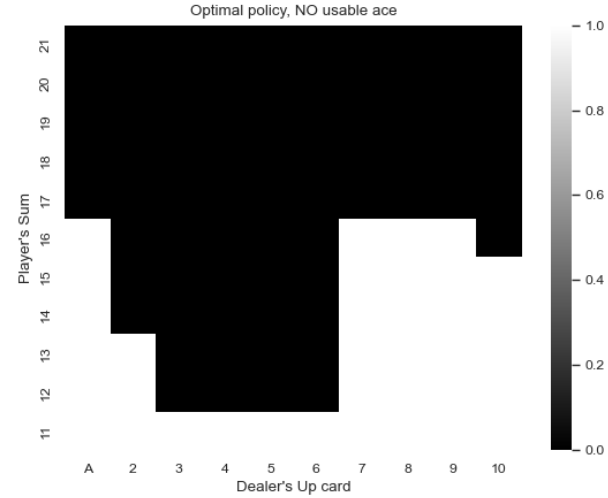
```
def generate_episode(Q, eps):
    state = env.reset()[0]
    episode = []

    done = False
    while not done:
        action = eps_greedy(Q, state, eps)
        next_state, reward, done, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state
    return episode
```

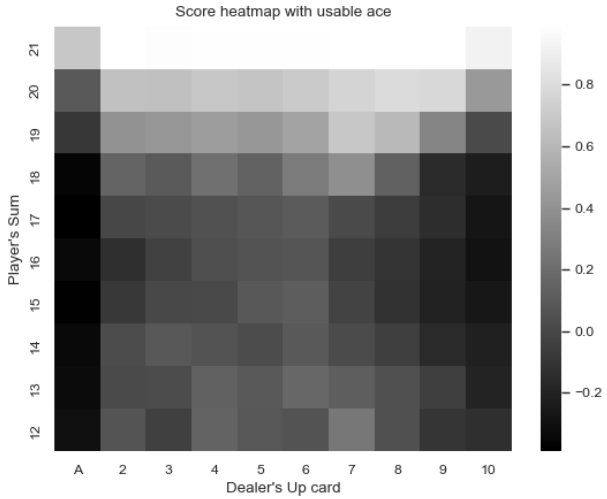
Results

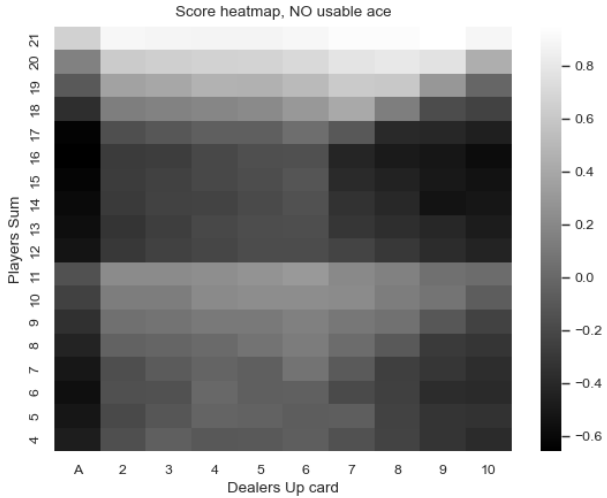
Optimal Policy



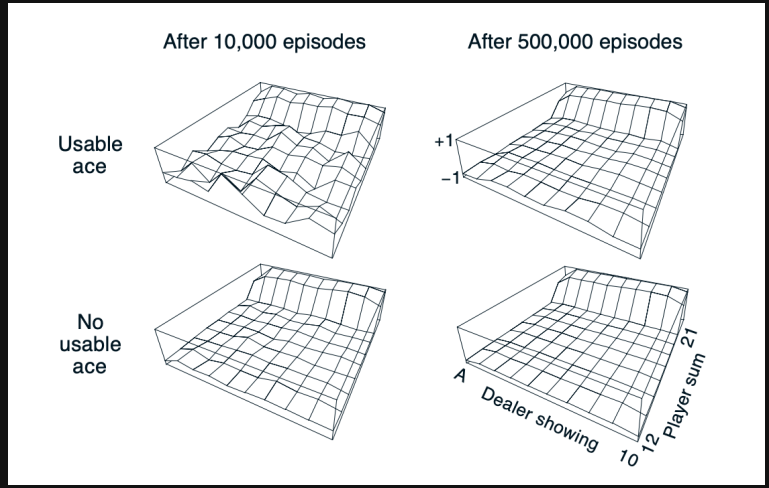





Score heatmap with usable ace
Player's Sum
21
20
19
18
17
16
15
14
13
12
A 2 3 4 5 6 7 8 9 10
Dealer's Up card
0.8
0.6
0.4
0.2
0.0
−0.2


Score heatmap, NO usable ace
Players Sum
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
A 2 3 4 5 6 7 8 9 10
Dealers Up card
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

State Score


After 10,000 episodes
After 500,000 episodes
Usable ace
No usable ace
+1
−1
A
Dealer showing
10
12
Player sum
21

Discussion

Explore-exploit trade-off

Convergence

About optimality

Off-policy learning

♣ ♥ ♠ ♦

Thank You For Attention!

Questions?

♣ ♥ ♠ ♦

END