Monte Carlo Methods - Off Policy

In [26]:
```
import sys
import gym
import numpy as np
from collections import defaultdict
```

Configuring the Blackjack environment

In [27]:
```
env = gym.make('Blackjack-v0')
```

In [28]:
```
print(vars(env))
```

```
{'action_space': Discrete(2), 'observation_space': Tuple(Discrete(32), Discrete
(11), Discrete(2)), 'np_random': RandomState(MT19937) at 0x7F3084802380, 'natur
al': False, 'dealer': [5, 8], 'player': [5, 8], 'spec': EnvSpec(Blackjack-v0)}
```

In [29]:
```
#Generating Random Policy
for i in range(5):
    print('Episode : ', i+1)
    state = env.reset()
    step = 0
    while True:
        step +=1
        # takes random action from environment's action space
        action = env.action_space.sample()
        print('Step = {}\t State = {}\t Action Taken = {}'.format(step, state, ac
        state, reward, done, info = env.step(action)
        if done:
            print('Game Ended...')
            if reward > 0: print('Agent Won!\n')
            else: print('Agent Lost!\n')
            break
```

```
Episode :  1
Step = 1         State = (20, 8, False)  Action Taken = 0
Game Ended...
Agent Won!

Episode :  2
Step = 1         State = (11, 6, False)  Action Taken = 0
Game Ended...
Agent Lost!

Episode :  3
Step = 1         State = (13, 2, False)  Action Taken = 0
Game Ended...
Agent Lost!

Episode :  4
Step = 1         State = (12, 9, False)  Action Taken = 1
Game Ended...
Agent Lost!

Episode :  5
Step = 1         State = (11, 2, False)  Action Taken = 1
Step = 2         State = (19, 2, False)  Action Taken = 1
Game Ended...
Agent Lost!
```

In [30]:
```
#Random Policy
def random_policy(nA):
    A = np.ones(nA, dtype=float) / nA
    def policy_fn(observation):
        return A
    return policy_fn
```

In [31]:
```
#Greedy Policy
def greedy_policy(Q):
    def policy_fn(state):
        A = np.zeros_like(Q[state], dtype=float)
        best_action = np.argmax(Q[state])
        A[best_action] = 1.0
        return A
    return policy_fn
```

In [32]:
```
#Defining Monte Carlo Off Policy
def mc_off_policy(env, num_episodes, behavior_policy, max_time=100, discount_fact
    Q = defaultdict(lambda:np.zeros(env.action_space.n))
    C = defaultdict(lambda:np.zeros(env.action_space.n))

    target_policy = greedy_policy(Q)

    for i_episode in range(1, num_episodes+1):
        if i_episode % 1000 == 0:
            print("\rEpisode {}/{}.".format(i_episode, num_episodes), end="")
            sys.stdout.flush()

        episode = []
        state = env.reset()
        for t in range(max_time):
            probs = behavior_policy(state)
            action = np.random.choice(np.arange(len(probs)), p=probs)
            next_state, reward, done, _ = env.step(action)
            episode.append((state, action, reward))
            if done:
                break
            state = next_state

        G = 0.0
        W = 1.0
        for t in range(len(episode))[::-1]:
            state, action, reward = episode[t]
            G = discount_factor * G + reward
            C[state][action] += W
            Q[state][action] += (W / C[state][action]) * (G - Q[state][action])
            if action !=  np.argmax(target_policy(state)):
                break
            W = W * 1./behavior_policy(state)[action]
    return Q, target_policy
```

In [33]:
```
random_policy = random_policy(env.action_space.n)
Q, policy = mc_off_policy(env, num_episodes=1000000, behavior_policy=random_polic
```

```
Episode 1000000/1000000.
```

In [34]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable


def plot_policy(policy):

    def get_Z(x, y, usable_ace):
        if (x,y,usable_ace) in policy:
            return policy[x,y,usable_ace]
        else:
            return 1

    def get_figure(usable_ace, ax):
        x_range = np.arange(11, 22)
        y_range = np.arange(10, 0, -1)
        X, Y = np.meshgrid(x_range, y_range)
        Z = np.array([[get_Z(x,y,usable_ace) for x in x_range] for y in y_range])
        surf = ax.imshow(Z, cmap=plt.get_cmap('Pastel2', 2), vmin=0, vmax=1, exte
        plt.xticks(x_range)
        plt.yticks(y_range)
        plt.gca().invert_yaxis()
        ax.set_xlabel('Player\'s Current Sum')
        ax.set_ylabel('Dealer\'s Showing Card')
        ax.grid(color='w', linestyle='-', linewidth=1)
        divider = make_axes_locatable(ax)
        cax = divider.append_axes("right", size="5%", pad=0.1)
        cbar = plt.colorbar(surf, ticks=[0,1], cax=cax)
        cbar.ax.set_yticklabels(['0 (STICK)','1 (HIT)'])

    fig = plt.figure(figsize=(15, 15))
    ax = fig.add_subplot(121)
    ax.set_title('Usable Ace')
    get_figure(True, ax)
    ax = fig.add_subplot(122)
    ax.set_title('No Usable Ace')
    get_figure(False, ax)
    plt.show()
```

In [35]:
```
policy = dict((k,np.argmax(v)) for k, v in Q.items())
plot_policy(policy)
```