RL and Advanced DL: Домашнее задание 1

Anna Beketova, ML-32

About Blackjack

Blackjack is a card game where the goal is to beat the dealer by obtaining cards that sum to closer to 21 (without going over 21) than the dealers cards.

Description

Card Values:

```
- Face cards (Jack, Queen, King) have a point value of 10.
- Aces can either count as 11 (called a 'usable ace') or 1.
- Numerical cards (2-9) have a value equal to their number.
```

This game is played with an infinite deck (or with replacement). The game starts with the dealer having one face up and one face down card, while the player has two face up cards.

The player can request additional cards (hit, action=1) until they decide to stop (stick, action=0) or exceed 21 (bust, immediate loss). After the player sticks, the dealer reveals their facedown card, and draws until their sum is 17 or greater. If the dealer goes bust, the player wins. If neither the player nor the dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21.

Action Space

There are two actions: stick (0), and hit (1).

Observation Space

The observation consists of a 3-tuple containing: the player's current sum, the value of the dealer's one showing card (1-10 where 1 is ace), and whether the player holds a usable ace (0 or 1). This environment corresponds to the version of the blackjack problem described in Example 5.1 in Reinforcement Learning: An Introduction by Sutton and Barto (http://incompleteideas.net/book/the-book-2nd.html).

Rewards

- win game: +1
- lose game: -1
- draw game: 0
- win game with natural blackjack:

  +1.5 (if 'natural' is True)

  +1 (if 'natural' is False)

Source: https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py

In [1]:
```
import sys
import gym
from collections import defaultdict
```

```python
import tqdm
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

Some usefull functions

In [2]:

```python
def epsilon_greedy(Q, state, eps):
    if np.random.random() > eps:
        return np.argmax(Q[state])
    else:
        return env.action_space.sample()


def estimate_reward(Q, num_episodes=100000):
    reward = 0.

    for i in range(num_episodes):
        reward += generate_episode_from_Q(env, Q, 0, env.action_space.n)[-1][-1]

    return reward/num_episodes


def get_probs(Q_s, epsilon, n_actions):
    policy_s = np.ones(n_actions) * epsilon / n_actions
    best_a = np.argmax(Q_s)
    policy_s[best_a] = 1 - epsilon + (epsilon / n_actions)
    return policy_s


def policy_from_Q(Q):
    return dict((k,np.argmax(v)) for k, v in Q.items())


def generate_episode_from_Q(env, Q, epsilon, n_actions):
    episode = []
    state, _ = env.reset()
    terminated = False

    while True:
        action = np.random.choice(np.arange(n_actions), p=get_probs(Q[state], epsilon,
            if state in Q else env.action_space.sample()
        next_state, reward, terminated, _, _ = env.step(action)
        episode.append((state, action, reward))
        state = next_state

        if terminated:
            break

    return episode


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('bone'), vmin=0, vmax=2, extent=[10.5, 21
```

```python
        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=[0,1,2], cax=cax)
        cbar.ax.set_yticklabels(['0 (STICK)','1 (HIT)', '2 (DOUBLE)'])

    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()
```

Little demo on Blackjack environment

In [4]:

```python
env = gym.make('Blackjack-v1', natural=True)

state, _ = env.reset()
print(state, '\n')
print(f"Player's hand: {state[0]}")
print(f"Dealer's one card: {state[1]}")
print(f"If player has usable ace: {state[2]}\n")

action = 1 # hit - draw one more card
print("Action: hit\n")

next_state, reward, terminated, info, _ = env.step(action)
print(f"Next state: {next_state}")
print(f"Reward: {reward}")
print(f"If the game is terminated: {terminated}\n")

action = 0 # stick - stop
print("Action: stick\n")

next_state, reward, terminated, _, _ = env.step(action)
print(f"Next state: {next_state}")
print(f"Reward: {reward}")
print(f"If the game is terminated: {terminated}")
```

```
(8, 10, False)

Player's hand: 8
Dealer's one card: 10
If player has usable ace: False

Action: hit

Next state: (13, 10, False)
Reward: 0.0
If the game is terminated: False

Action: stick

Next state: (13, 10, False)
Reward: -1.0
If the game is terminated: True
```

In [3]:
```
N_verbose_batch_size = 1000
N_iterations = 100000
```

Часть первая, с блекджеком и стратегиями

Задания

1. Рассмотрим очень простую стратегию: говорить stand, если у нас на руках комбинация в 19, 20 или 21 очко, во всех остальных случаях говорить hit. Используйте методы Монте-Карло, чтобы оценить выигрыш от этой стратегии.
2. Реализуйте метод обучения с подкреплением без модели (можно Q-обучение, но рекомендую попробовать и другие, например Monte Carlo control) для обучения стратегии в блекджеке, используя окружение BlackjackEnv из OpenAI Gym.
3. Сколько выигрывает казино у вашей стратегии? Нарисуйте графики среднего дохода вашего метода (усреднённого по крайней мере по 100000 раздач, а лучше больше) по ходу обучения. Попробуйте подобрать оптимальные гиперпараметры.

Simple strategy: 'stick' action (value 0) if player has 19, 20, 21 in hand, 'hit' action (value 1) - otherwise.

In [5]:
```
def action_by_simple_strategy(player_hand_value):
    return int(player_hand_value < 19)

def generate_episode_simple_strategy(env):
    episode = []
    state, _ = env.reset()

    while True:
        players_hand = state[0]
        action = action_by_simple_strategy(players_hand)
        next_state, reward, terminated, _, _ = env.step(action)
        episode.append((next_state, action, reward))
        state = next_state

        if terminated:
            break

    return episode
```

In [6]:
```
env = gym.make('Blackjack-v1', natural=True)
```

In [7]:
```
reward = 0.
N_iterations = 1000000

for i in tqdm.tqdm(range(N_iterations)):
    reward += generate_episode_simple_strategy(env)[-1][-1]

print(f"Average reward for simple strategy: {reward / N_iterations}")
```

```
100%|██████████| 1000000/1000000 [02:43<00:00, 6106.97it/s]
Average reward for simple strategy: -0.198529
```

Q-learning

In [9]:
```
env = gym.make('Blackjack-v1', natural=True)

N_verbose_batch_size = 1000
N_iterations = 100000
```

In [10]:

```
def q_learning(env, num_episodes, alpha, gamma=1.0, eps_min=0.01,
               verbose=True, verbose_batch_size=50000, print_every=10):

    n_actions = env.action_space.n
    Q = defaultdict(lambda: np.zeros(n_actions))
    total_reward = 0.
    rewards = []

    print_counter = 0

    for i in range(num_episodes):
        state, _ = env.reset()
        eps = max(1.0 / (i+1), eps_min)

        while True:
            action = epsilon_greedy(Q, state, eps)
            next_state, cur_reward, terminated, _, _ = env.step(action)
            total_reward += cur_reward
            Q[state][action] = update_Q_qlearning(alpha, gamma, Q, state, action, cur_r
            state = next_state

            if terminated:
                break

        if i % verbose_batch_size == 0 and i > 0 and verbose:
            print(f"Episode {i} / {num_episodes}. Avg reward: {total_reward/verbose_bat
            total_reward = 0.
            rewards.append(estimate_reward(Q))
    return Q, rewards

def update_Q_qlearning(alpha, gamma, Q, state, action, reward, next_state=None):

    current = Q[state][action]
    Qsa_optimal_furure_value = np.max(Q[next_state]) if next_state is not None else 0
    new_value = current + alpha * (reward + (gamma * Qsa_optimal_furure_value) - curren

    return new_value
```

In [11]:

```
%%time

for eps_min in [0.001, 0.005, 0.01, 0.05, 0.1, 0.2]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = q_learning(env, num_episodes=1000, alpha=0.01, eps_min=eps_min, ve
        reward += estimate_reward(Q)

    print(f'epsilon={eps_min}, reward={(reward / tries):.5f}')
```

```
epsilon=0.001, reward=-0.13199
epsilon=0.005, reward=-0.14490
epsilon=0.01, reward=-0.13069
epsilon=0.05, reward=-0.10976
epsilon=0.1, reward=-0.13095
epsilon=0.2, reward=-0.12399
CPU times: user 7min 53s, sys: 750 ms, total: 7min 54s
Wall time: 7min 54s
```

In [12]:

```
%%time
for alpha in [0.0001, 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95]:
    tries = 5
    reward = 0

    for i in range(tries):
```

```python
        Q, rewards = q_learning(env, num_episodes=1000, alpha=alpha, eps_min=0.05, verb
        reward += estimate_reward(Q)

    print(f'alpha={alpha}, reward={(reward / tries):.5f}')
```

```
alpha=0.0001, reward=-0.12004
alpha=0.001, reward=-0.12150
alpha=0.01, reward=-0.12242
alpha=0.1, reward=-0.13476
alpha=0.25, reward=-0.15549
alpha=0.5, reward=-0.17024
alpha=0.75, reward=-0.19422
alpha=0.9, reward=-0.17715
alpha=0.95, reward=-0.17602
CPU times: user 12min 8s, sys: 1.05 s, total: 12min 9s
Wall time: 12min 9s
```

In [13]:

```python
%%time

N_verbose_batch_size = 1000
N_iterations = 100000

Q, rewards = q_learning(env, N_iterations, alpha=0.001, gamma=0.95,
                        eps_min=0.005, verbose_batch_size=N_verbose_batch_size)
```

```
Episode 1000 / 100000. Avg reward: -0.159
Episode 2000 / 100000. Avg reward: -0.139
Episode 3000 / 100000. Avg reward: -0.096
Episode 4000 / 100000. Avg reward: -0.134
Episode 5000 / 100000. Avg reward: -0.123
Episode 6000 / 100000. Avg reward: -0.091
Episode 7000 / 100000. Avg reward: -0.05
Episode 8000 / 100000. Avg reward: -0.058
Episode 9000 / 100000. Avg reward: -0.068
Episode 10000 / 100000. Avg reward: -0.101
Episode 11000 / 100000. Avg reward: -0.091
Episode 12000 / 100000. Avg reward: -0.122
Episode 13000 / 100000. Avg reward: -0.054
Episode 14000 / 100000. Avg reward: -0.082
Episode 15000 / 100000. Avg reward: -0.054
Episode 16000 / 100000. Avg reward: -0.137
Episode 17000 / 100000. Avg reward: -0.131
Episode 18000 / 100000. Avg reward: -0.076
Episode 19000 / 100000. Avg reward: -0.098
Episode 20000 / 100000. Avg reward: -0.026
Episode 21000 / 100000. Avg reward: -0.069
Episode 22000 / 100000. Avg reward: -0.06
Episode 23000 / 100000. Avg reward: -0.054
Episode 24000 / 100000. Avg reward: -0.094
Episode 25000 / 100000. Avg reward: -0.089
Episode 26000 / 100000. Avg reward: -0.042
Episode 27000 / 100000. Avg reward: -0.11
Episode 28000 / 100000. Avg reward: -0.099
Episode 29000 / 100000. Avg reward: -0.101
Episode 30000 / 100000. Avg reward: -0.108
Episode 31000 / 100000. Avg reward: -0.123
Episode 32000 / 100000. Avg reward: -0.144
Episode 33000 / 100000. Avg reward: -0.093
Episode 34000 / 100000. Avg reward: -0.059
Episode 35000 / 100000. Avg reward: -0.077
Episode 36000 / 100000. Avg reward: -0.048
Episode 37000 / 100000. Avg reward: -0.103
Episode 38000 / 100000. Avg reward: -0.092
Episode 39000 / 100000. Avg reward: -0.095
Episode 40000 / 100000. Avg reward: -0.086
Episode 41000 / 100000. Avg reward: -0.092
Episode 42000 / 100000. Avg reward: -0.062
Episode 43000 / 100000. Avg reward: -0.104
Episode 44000 / 100000. Avg reward: -0.07
Episode 45000 / 100000. Avg reward: -0.095
Episode 46000 / 100000. Avg reward: -0.144
Episode 47000 / 100000. Avg reward: -0.081
```

```
Episode 48000 / 100000. Avg reward: -0.163
Episode 49000 / 100000. Avg reward: -0.085
Episode 50000 / 100000. Avg reward: -0.07
Episode 51000 / 100000. Avg reward: -0.126
Episode 52000 / 100000. Avg reward: -0.093
Episode 53000 / 100000. Avg reward: -0.105
Episode 54000 / 100000. Avg reward: -0.093
Episode 55000 / 100000. Avg reward: -0.045
Episode 56000 / 100000. Avg reward: -0.117
Episode 57000 / 100000. Avg reward: -0.065
Episode 58000 / 100000. Avg reward: -0.112
Episode 59000 / 100000. Avg reward: -0.102
Episode 60000 / 100000. Avg reward: -0.129
Episode 61000 / 100000. Avg reward: -0.095
Episode 62000 / 100000. Avg reward: -0.075
Episode 63000 / 100000. Avg reward: -0.123
Episode 64000 / 100000. Avg reward: -0.133
Episode 65000 / 100000. Avg reward: -0.09
Episode 66000 / 100000. Avg reward: -0.124
Episode 67000 / 100000. Avg reward: -0.098
Episode 68000 / 100000. Avg reward: -0.088
Episode 69000 / 100000. Avg reward: -0.12
Episode 70000 / 100000. Avg reward: -0.078
Episode 71000 / 100000. Avg reward: -0.081
Episode 72000 / 100000. Avg reward: -0.058
Episode 73000 / 100000. Avg reward: -0.073
Episode 74000 / 100000. Avg reward: -0.102
Episode 75000 / 100000. Avg reward: -0.08
Episode 76000 / 100000. Avg reward: -0.115
Episode 77000 / 100000. Avg reward: -0.095
Episode 78000 / 100000. Avg reward: -0.128
Episode 79000 / 100000. Avg reward: -0.072
Episode 80000 / 100000. Avg reward: -0.09
Episode 81000 / 100000. Avg reward: -0.111
Episode 82000 / 100000. Avg reward: -0.105
Episode 83000 / 100000. Avg reward: -0.107
Episode 84000 / 100000. Avg reward: -0.114
Episode 85000 / 100000. Avg reward: -0.123
Episode 86000 / 100000. Avg reward: -0.07
Episode 87000 / 100000. Avg reward: -0.066
Episode 88000 / 100000. Avg reward: -0.098
Episode 89000 / 100000. Avg reward: -0.087
Episode 90000 / 100000. Avg reward: -0.116
Episode 91000 / 100000. Avg reward: -0.07
Episode 92000 / 100000. Avg reward: -0.04
Episode 93000 / 100000. Avg reward: -0.146
Episode 94000 / 100000. Avg reward: -0.114
Episode 95000 / 100000. Avg reward: -0.092
Episode 96000 / 100000. Avg reward: -0.058
Episode 97000 / 100000. Avg reward: -0.069
Episode 98000 / 100000. Avg reward: 0.006
Episode 99000 / 100000. Avg reward: -0.072
CPU times: user 27min 5s, sys: 2.35 s, total: 27min 7s
Wall time: 27min 8s
```

In [14]:

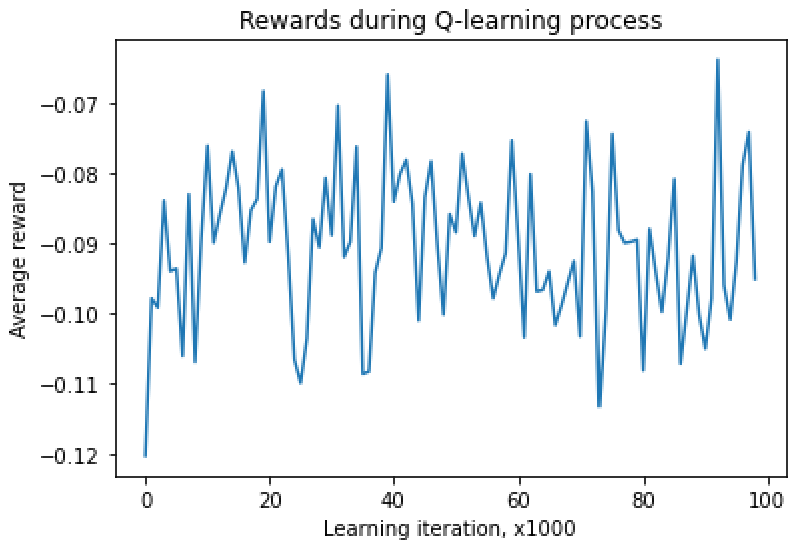
```
# estimated reward
estimate_reward(Q)
```

Out[14]:

```
-0.109
```

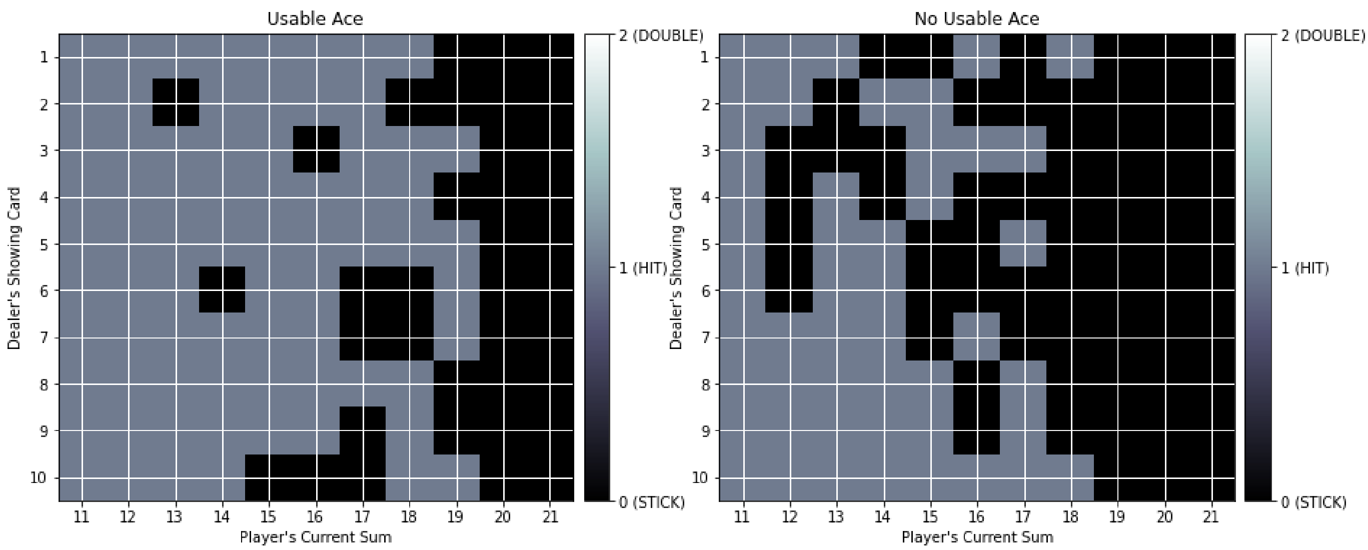
In [15]:

```
plt.title("Rewards during Q-learning process")
plt.plot(list(range(len(rewards))), rewards)
plt.ylabel("Average reward")
plt.xlabel("Learning iteration, x1000")
plt.show()
```



In [16]:
```
policy = policy_from_Q(Q)
```

In [17]:
```
plot_policy(policy)
```



Monte Carlo control

In [18]:
```
def update_Q(env, episode, Q, alpha, gamma):
    states, actions, rewards = zip(*episode)
    discounts = np.array([gamma**i for i in range(len(rewards) + 1)])

    for i, state in enumerate(states):
        old_Q = Q[state][actions[i]]
        Q[state][actions[i]] = old_Q + alpha * (sum(rewards[i:] * discounts[:-(1+i)]) -

    return Q

def montecarlo_control(env, num_episodes, alpha, gamma=1.0, eps_min=0.01, verbose=True,
    n_actions = env.action_space.n
    Q = defaultdict(lambda: np.zeros(n_actions))
    total_reward = 0.
    rewards = []

    for i in range(num_episodes):
        eps = max(1.0 / (i+1), eps_min)
        episode = generate_episode_from_Q(env, Q, eps, n_actions)
        Q = update_Q(env, episode, Q, alpha, gamma)
        cur_reward = episode[-1][-1]
        total_reward += cur_reward
```

```
        if i % verbose_batch_size == 0 and i > 0 and verbose:
            print(f"Episode {i} / {num_episodes}. Avg reward: {total_reward/verbose_bat
            total_reward = 0.
            rewards.append(estimate_reward(Q))

    return Q, rewards
```

In [19]:

```
%%time

for eps_min in [0.001, 0.005, 0.01, 0.05, 0.1, 0.2]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000, alpha=0.01, eps_min=eps
        reward += estimate_reward(Q)

    print(f'epsilon={eps_min}, reward={(reward / tries):.5f}')
```

```
epsilon=0.001, reward=-0.15756
epsilon=0.005, reward=-0.14115
epsilon=0.01, reward=-0.14720
epsilon=0.05, reward=-0.15388
epsilon=0.1, reward=-0.14432
epsilon=0.2, reward=-0.15152
CPU times: user 7min 37s, sys: 567 ms, total: 7min 37s
Wall time: 7min 37s
```

In [20]:

```
%%time
for alpha in [0.0001, 0.001, 0.01, 0.1, 0.2]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000, alpha=alpha, eps_min=0.
        reward += estimate_reward(Q)

    print(f'alpha={alpha}, reward={(reward / tries):.5f}')
```

```
alpha=0.0001, reward=-0.13885
alpha=0.001, reward=-0.13602
alpha=0.01, reward=-0.14210
alpha=0.1, reward=-0.15054
alpha=0.2, reward=-0.14151
CPU times: user 6min 21s, sys: 458 ms, total: 6min 21s
Wall time: 6min 21s
```

In [21]:

```
%%time
for gamma in [0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000,
                                        alpha=0.1, eps_min=0.01, gamma=gamma, verbose=False)
        reward += estimate_reward(Q)

    print(f'gamma={gamma}, reward={(reward / tries):.5f}')
```

```
gamma=0.5, reward=-0.13250
gamma=0.6, reward=-0.13789
gamma=0.7, reward=-0.15016
gamma=0.8, reward=-0.13507
gamma=0.9, reward=-0.14390
gamma=0.95, reward=-0.14882
gamma=1.0, reward=-0.15643
```

```
CPU times: user 9min 4s, sys: 688 ms, total: 9min 4s
Wall time: 9min 4s
```

In [22]:
```
%%time

Q, rewards = montecarlo_control(env, N_iterations,
                                alpha=0.1, eps_min=0.01, gamma=0.8,
                                verbose_batch_size=N_verbose_batch_size)
```

```
Episode 1000 / 100000. Avg reward: -0.18
Episode 2000 / 100000. Avg reward: -0.123
Episode 3000 / 100000. Avg reward: -0.131
Episode 4000 / 100000. Avg reward: -0.105
Episode 5000 / 100000. Avg reward: -0.139
Episode 6000 / 100000. Avg reward: -0.078
Episode 7000 / 100000. Avg reward: -0.084
Episode 8000 / 100000. Avg reward: -0.1
Episode 9000 / 100000. Avg reward: -0.068
Episode 10000 / 100000. Avg reward: -0.112
Episode 11000 / 100000. Avg reward: -0.172
Episode 12000 / 100000. Avg reward: -0.144
Episode 13000 / 100000. Avg reward: -0.093
Episode 14000 / 100000. Avg reward: -0.101
Episode 15000 / 100000. Avg reward: -0.05
Episode 16000 / 100000. Avg reward: 0.001
Episode 17000 / 100000. Avg reward: -0.067
Episode 18000 / 100000. Avg reward: -0.072
Episode 19000 / 100000. Avg reward: -0.098
Episode 20000 / 100000. Avg reward: -0.022
Episode 21000 / 100000. Avg reward: -0.088
Episode 22000 / 100000. Avg reward: -0.059
Episode 23000 / 100000. Avg reward: -0.093
Episode 24000 / 100000. Avg reward: -0.028
Episode 25000 / 100000. Avg reward: -0.085
Episode 26000 / 100000. Avg reward: -0.048
Episode 27000 / 100000. Avg reward: -0.115
Episode 28000 / 100000. Avg reward: -0.019
Episode 29000 / 100000. Avg reward: -0.055
Episode 30000 / 100000. Avg reward: -0.018
Episode 31000 / 100000. Avg reward: -0.065
Episode 32000 / 100000. Avg reward: -0.096
Episode 33000 / 100000. Avg reward: -0.095
Episode 34000 / 100000. Avg reward: -0.05
Episode 35000 / 100000. Avg reward: -0.079
Episode 36000 / 100000. Avg reward: -0.024
Episode 37000 / 100000. Avg reward: -0.113
Episode 38000 / 100000. Avg reward: -0.083
Episode 39000 / 100000. Avg reward: -0.058
Episode 40000 / 100000. Avg reward: -0.13
Episode 41000 / 100000. Avg reward: -0.039
Episode 42000 / 100000. Avg reward: -0.089
Episode 43000 / 100000. Avg reward: -0.047
Episode 44000 / 100000. Avg reward: -0.047
Episode 45000 / 100000. Avg reward: -0.145
Episode 46000 / 100000. Avg reward: -0.032
Episode 47000 / 100000. Avg reward: -0.07
Episode 48000 / 100000. Avg reward: -0.092
Episode 49000 / 100000. Avg reward: -0.078
Episode 50000 / 100000. Avg reward: -0.051
Episode 51000 / 100000. Avg reward: -0.055
Episode 52000 / 100000. Avg reward: -0.11
Episode 53000 / 100000. Avg reward: -0.047
Episode 54000 / 100000. Avg reward: -0.063
Episode 55000 / 100000. Avg reward: -0.123
Episode 56000 / 100000. Avg reward: -0.053
Episode 57000 / 100000. Avg reward: -0.049
Episode 58000 / 100000. Avg reward: -0.078
Episode 59000 / 100000. Avg reward: -0.049
Episode 60000 / 100000. Avg reward: -0.022
Episode 61000 / 100000. Avg reward: -0.085
Episode 62000 / 100000. Avg reward: -0.119
Episode 63000 / 100000. Avg reward: -0.069
Episode 64000 / 100000. Avg reward: -0.081
```

```
Episode 65000 / 100000. Avg reward: -0.053
Episode 66000 / 100000. Avg reward: -0.102
Episode 67000 / 100000. Avg reward: -0.002
Episode 68000 / 100000. Avg reward: -0.07
Episode 69000 / 100000. Avg reward: -0.032
Episode 70000 / 100000. Avg reward: -0.104
Episode 71000 / 100000. Avg reward: -0.067
Episode 72000 / 100000. Avg reward: -0.058
Episode 73000 / 100000. Avg reward: -0.059
Episode 74000 / 100000. Avg reward: -0.136
Episode 75000 / 100000. Avg reward: -0.069
Episode 76000 / 100000. Avg reward: -0.06
Episode 77000 / 100000. Avg reward: -0.07
Episode 78000 / 100000. Avg reward: -0.065
Episode 79000 / 100000. Avg reward: -0.061
Episode 80000 / 100000. Avg reward: -0.059
Episode 81000 / 100000. Avg reward: -0.049
Episode 82000 / 100000. Avg reward: -0.085
Episode 83000 / 100000. Avg reward: -0.035
Episode 84000 / 100000. Avg reward: -0.059
Episode 85000 / 100000. Avg reward: -0.049
Episode 86000 / 100000. Avg reward: -0.068
Episode 87000 / 100000. Avg reward: -0.088
Episode 88000 / 100000. Avg reward: -0.044
Episode 89000 / 100000. Avg reward: -0.12
Episode 90000 / 100000. Avg reward: -0.152
Episode 91000 / 100000. Avg reward: -0.085
Episode 92000 / 100000. Avg reward: -0.023
Episode 93000 / 100000. Avg reward: -0.053
Episode 94000 / 100000. Avg reward: -0.024
Episode 95000 / 100000. Avg reward: -0.05
Episode 96000 / 100000. Avg reward: 0.012
Episode 97000 / 100000. Avg reward: -0.054
Episode 98000 / 100000. Avg reward: -0.121
Episode 99000 / 100000. Avg reward: -0.03
CPU times: user 27min 19s, sys: 2.17 s, total: 27min 22s
Wall time: 27min 22s
```

In [23]:

```
# estimated reward
estimate_reward(Q)
```

Out[23]: -0.05551

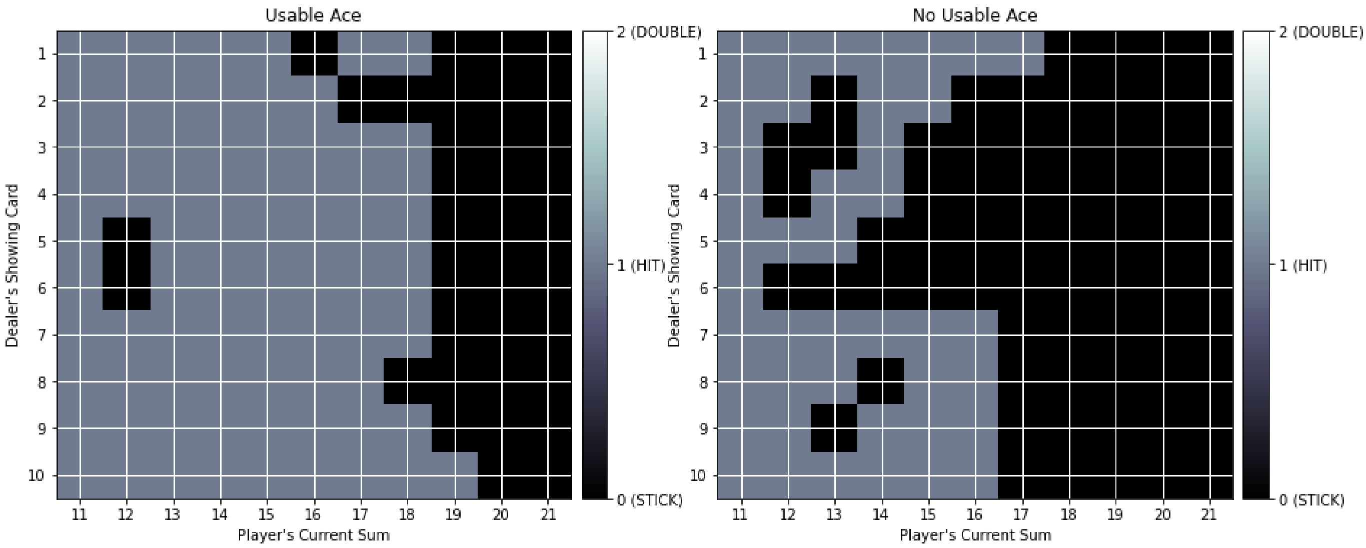
In [24]:

```
plt.plot(list(range(len(rewards))), rewards)
plt.title("rewards during learning process")
plt.ylabel("average reward")
plt.xlabel("learning iteration, x1000")
plt.show()
```

In [25]:

```
policy = policy_from_Q(Q)
```

In [26]:
```python
plot_policy(policy)
```



Часть вторая, удвоенная

В базовый блекджек, описанный в предыдущем разделе, обыграть казино вряд ли получится. Но, к счастью, на этом история не заканчивается. Описанные выше правила были упрощёнными, а на самом деле у игрока есть ещё и другие возможности. Реализовывать split может оказаться непросто, поэтому давайте ограничимся удвоением ставки. Итак, у игрока появляется дополнительное действие:

- double — удвоить ставку; при этом больше действий делать нельзя, игроку выдаётся ровно одна дополнительная карта, а выигрыш или проигрыш удваивается.

1. Реализуйте новый вариант блекджека на основе окружения BlackjackEnv из OpenAI Gym, в котором разрешено удвоение ставки.
2. Реализуйте метод обучения с подкреплением без модели для этого варианта, постройте графики, аналогичные п.2.

In [28]:
```python
from blackjack_double import BlackjackEnvDouble

env = BlackjackEnvDouble(natural=True)
```

In [29]:
```python
env.reset()

# check that new action 'double' is available
env.step(2)
```

Out[29]:
```
((12, 10, False), -2.0, True, False, {})
```

In [30]:
```python
%%time

for eps_min in [0.001, 0.005, 0.01, 0.05, 0.1, 0.2]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000, alpha=0.01, eps_min=eps
        reward += estimate_reward(Q)

    print(f'epsilon={eps_min}, reward={(reward / tries):.5f}')
```

```
epsilon=0.001, reward=-0.21927
epsilon=0.005, reward=-0.19345
epsilon=0.01, reward=-0.23956
epsilon=0.05, reward=-0.22511
epsilon=0.1, reward=-0.21255
epsilon=0.2, reward=-0.22328
CPU times: user 6min 4s, sys: 423 ms, total: 6min 4s
Wall time: 6min 4s
```

In [31]:

```
%%time
for alpha in [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000, alpha=alpha, eps_min=0.
        reward += estimate_reward(Q)

    print(f'alpha={alpha}, reward={(reward / tries):.5f}')
```

```
alpha=0.0001, reward=-0.21334
alpha=0.001, reward=-0.24168
alpha=0.01, reward=-0.20544
alpha=0.1, reward=-0.22031
alpha=0.2, reward=-0.23946
alpha=0.3, reward=-0.20341
alpha=0.4, reward=-0.19888
alpha=0.5, reward=-0.22823
alpha=0.6, reward=-0.20052
alpha=0.7, reward=-0.22011
alpha=0.8, reward=-0.21293
alpha=0.9, reward=-0.20870
alpha=0.95, reward=-0.20882
CPU times: user 13min 7s, sys: 905 ms, total: 13min 8s
Wall time: 13min 8s
```

In [32]:

```
%%time
for gamma in [0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0]:
    tries = 5
    reward = 0

    for i in range(tries):
        Q, rewards = montecarlo_control(env, num_episodes=1000,
                                        alpha=0.2, eps_min=0.1, gamma=gamma, verbose=False)
        reward += estimate_reward(Q)

    print(f'gamma={gamma}, reward={(reward / tries):.5f}')
```

```
gamma=0.5, reward=-0.18257
gamma=0.6, reward=-0.20146
gamma=0.7, reward=-0.20154
gamma=0.8, reward=-0.20597
gamma=0.9, reward=-0.22093
gamma=0.95, reward=-0.20628
gamma=1.0, reward=-0.20435
CPU times: user 7min 12s, sys: 510 ms, total: 7min 12s
Wall time: 7min 12s
```

In [33]:

```
%%time

Q, rewards = montecarlo_control(env, N_iterations,
                                alpha=0.1, eps_min=0.01, gamma=0.8,
                                verbose_batch_size=N_verbose_batch_size)
```

```
Episode 1000 / 100000. Avg reward: -0.241
Episode 2000 / 100000. Avg reward: -0.195
Episode 3000 / 100000. Avg reward: -0.116
Episode 4000 / 100000. Avg reward: -0.162
Episode 5000 / 100000. Avg reward: -0.162
Episode 6000 / 100000. Avg reward: -0.074
```

```
Episode 7000 / 100000. Avg reward: -0.069
Episode 8000 / 100000. Avg reward: -0.13
Episode 9000 / 100000. Avg reward: -0.176
Episode 10000 / 100000. Avg reward: -0.145
Episode 11000 / 100000. Avg reward: -0.073
Episode 12000 / 100000. Avg reward: -0.154
Episode 13000 / 100000. Avg reward: -0.087
Episode 14000 / 100000. Avg reward: -0.119
Episode 15000 / 100000. Avg reward: -0.106
Episode 16000 / 100000. Avg reward: -0.11
Episode 17000 / 100000. Avg reward: -0.048
Episode 18000 / 100000. Avg reward: -0.159
Episode 19000 / 100000. Avg reward: -0.112
Episode 20000 / 100000. Avg reward: -0.092
Episode 21000 / 100000. Avg reward: -0.084
Episode 22000 / 100000. Avg reward: -0.056
Episode 23000 / 100000. Avg reward: -0.051
Episode 24000 / 100000. Avg reward: -0.096
Episode 25000 / 100000. Avg reward: -0.074
Episode 26000 / 100000. Avg reward: -0.108
Episode 27000 / 100000. Avg reward: -0.086
Episode 28000 / 100000. Avg reward: -0.076
Episode 29000 / 100000. Avg reward: -0.16
Episode 30000 / 100000. Avg reward: -0.031
Episode 31000 / 100000. Avg reward: -0.081
Episode 32000 / 100000. Avg reward: -0.039
Episode 33000 / 100000. Avg reward: -0.048
Episode 34000 / 100000. Avg reward: -0.042
Episode 35000 / 100000. Avg reward: -0.092
Episode 36000 / 100000. Avg reward: -0.09
Episode 37000 / 100000. Avg reward: -0.093
Episode 38000 / 100000. Avg reward: -0.148
Episode 39000 / 100000. Avg reward: -0.067
Episode 40000 / 100000. Avg reward: -0.134
Episode 41000 / 100000. Avg reward: -0.023
Episode 42000 / 100000. Avg reward: -0.1
Episode 43000 / 100000. Avg reward: 0.0
Episode 44000 / 100000. Avg reward: -0.043
Episode 45000 / 100000. Avg reward: -0.09
Episode 46000 / 100000. Avg reward: -0.052
Episode 47000 / 100000. Avg reward: -0.121
Episode 48000 / 100000. Avg reward: -0.048
Episode 49000 / 100000. Avg reward: -0.058
Episode 50000 / 100000. Avg reward: -0.002
Episode 51000 / 100000. Avg reward: -0.109
Episode 52000 / 100000. Avg reward: -0.136
Episode 53000 / 100000. Avg reward: -0.14
Episode 54000 / 100000. Avg reward: -0.079
Episode 55000 / 100000. Avg reward: -0.083
Episode 56000 / 100000. Avg reward: -0.083
Episode 57000 / 100000. Avg reward: -0.081
Episode 58000 / 100000. Avg reward: -0.121
Episode 59000 / 100000. Avg reward: -0.04
Episode 60000 / 100000. Avg reward: -0.048
Episode 61000 / 100000. Avg reward: -0.111
Episode 62000 / 100000. Avg reward: -0.075
Episode 63000 / 100000. Avg reward: -0.118
Episode 64000 / 100000. Avg reward: -0.073
Episode 65000 / 100000. Avg reward: -0.082
Episode 66000 / 100000. Avg reward: -0.06
Episode 67000 / 100000. Avg reward: -0.068
Episode 68000 / 100000. Avg reward: -0.049
Episode 69000 / 100000. Avg reward: 0.023
Episode 70000 / 100000. Avg reward: -0.034
Episode 71000 / 100000. Avg reward: -0.117
Episode 72000 / 100000. Avg reward: -0.02
Episode 73000 / 100000. Avg reward: -0.09
Episode 74000 / 100000. Avg reward: -0.056
Episode 75000 / 100000. Avg reward: -0.017
Episode 76000 / 100000. Avg reward: -0.044
Episode 77000 / 100000. Avg reward: -0.042
Episode 78000 / 100000. Avg reward: -0.112
Episode 79000 / 100000. Avg reward: -0.093
Episode 80000 / 100000. Avg reward: -0.039
```

```
Episode 81000 / 100000. Avg reward: -0.067
Episode 82000 / 100000. Avg reward: -0.086
Episode 83000 / 100000. Avg reward: -0.122
Episode 84000 / 100000. Avg reward: -0.074
Episode 85000 / 100000. Avg reward: -0.059
Episode 86000 / 100000. Avg reward: -0.128
Episode 87000 / 100000. Avg reward: -0.09
Episode 88000 / 100000. Avg reward: -0.09
Episode 89000 / 100000. Avg reward: -0.081
Episode 90000 / 100000. Avg reward: -0.05
Episode 91000 / 100000. Avg reward: -0.104
Episode 92000 / 100000. Avg reward: -0.12
Episode 93000 / 100000. Avg reward: -0.099
Episode 94000 / 100000. Avg reward: -0.06
Episode 95000 / 100000. Avg reward: -0.092
Episode 96000 / 100000. Avg reward: -0.061
Episode 97000 / 100000. Avg reward: -0.027
Episode 98000 / 100000. Avg reward: -0.096
Episode 99000 / 100000. Avg reward: -0.106
CPU times: user 22min 43s, sys: 1.65 s, total: 22min 45s
Wall time: 22min 45s
```

In [34]:

```
# estimated reward
estimate_reward(Q)
```

Out[34]:

```
-0.06394
```

In [35]:

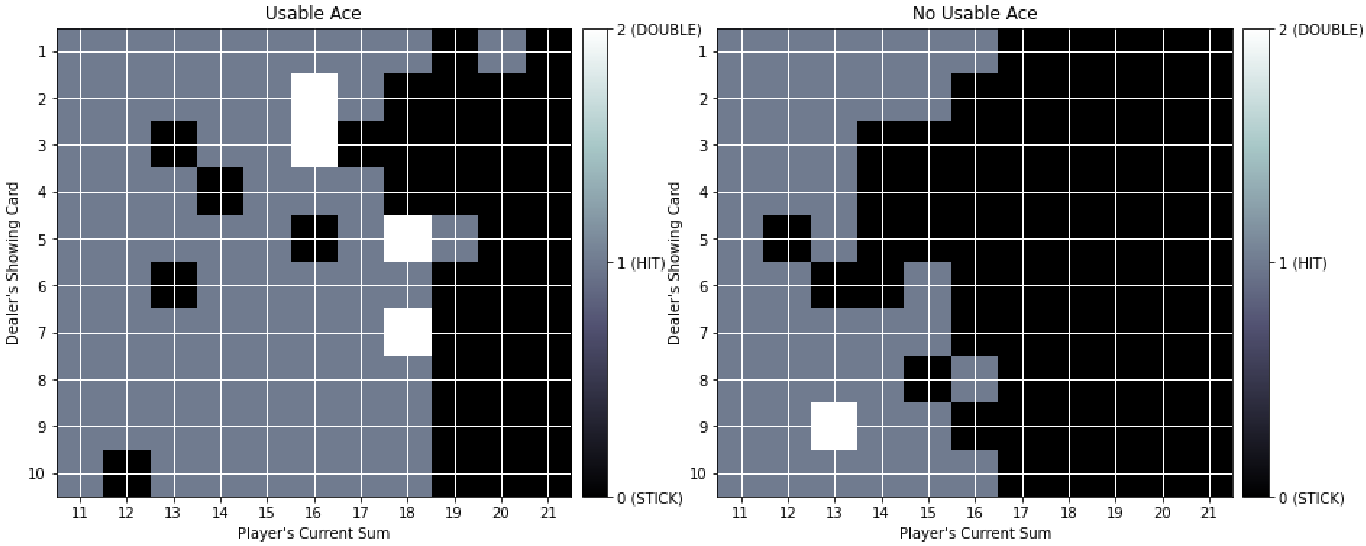
```
plt.plot(list(range(len(rewards))), rewards)
plt.title("rewards during learning process")
plt.ylabel("average reward")
plt.xlabel("learning iteration, x1000")
plt.show()
```

In [36]:

```
policy = policy_from_Q(Q)
```

In [37]:

```
plot_policy(policy)
```



Часть третья, в главной роли — Дастин Хоффман

А теперь давайте вспомним, как играют в блекджек настоящие профессионалы. Дело в том, что в оффлайн-казино обычно не перемешивают колоду после каждой раздачи — это слишком замедляло бы игру. После раздачи карты просто раздаются дальше с верха колоды до тех пор, пока карт не останется слишком мало, и только тогда колода перемешивается; давайте для определённости считать, что наше казино будет перемешивать колоду, в которой осталось меньше 15 карт.

Действительно, если вы будете запоминать, какие карты уже вышли, у вас будет информация о том, какие карты ещё остались, а это позволяет лучше понять, когда нужно удваивать ставку или делать split, а когда лучше не стоит. В настоящем казино могут раздавать карты сразу из нескольких колод, и заслуга Rain Man'а была в том, что он смог считать карты в шести колодах одновременно. Но мы с вами вооружены компьютерами, так что подсчёт можно считать автоматическим.

1. Реализуйте вариант окружения BlackjackEnv из предыдущей части (с удвоением), в котором игрок имеет возможность “считать карты” в колоде. Это можно сделать разными способами; возможно, вам поможет статья википедии о блекджеке (а возможно, и нет).
2. Реализуйте метод обучения с подкреплением без модели для этого варианта, постройте графики, аналогичные п.2.

In [4]:

```
from blackjack_count import BlackjackEnvCount

env = BlackjackEnvCount(natural=True)
```

In [5]:

```
# validate that card countring works
assert len(env.deck) == 52
env.reset()
assert len(env.deck) == 48
env.step(1)
assert len(env.deck) == 47
env.reset()
assert len(env.deck) == 43

for _ in range(14, 43):
    env.step(1)

assert len(env.deck) == 14
env.reset()
```

```python
    # deck shuffled and reseted
    assert len(env.deck) > 40
    len(env.deck)
```

Out[5]: 48

In [6]:

```python
# counting cards with the simplest schema

# 2,3,4,5,6 - +1
# 7,8,9 - +0
# 10, 11 - -1

cards_counter = 0

def count_card(card):
    global cards_counter
    if card < 7:
        cards_counter += 1
    if card >= 10:
        cards_counter -= 1

def count_cards(cards):
    for card in cards:
        count_card(card)
```

In [10]:

```python
def generate_episode_from_Q(env, Q, epsilon, nA):
    global cards_counter
    episode = []
    state, is_deck_shuffled = env.reset()

    if is_deck_shuffled:
        cards_counter = 0

    # we can see only one dealer's card, but both our cards
    count_cards((env.dealer[0], env.player[0], env.player[1]))

    while True:
        state = (*state, cards_counter)
        action = np.random.choice(np.arange(nA), p=get_probs(Q[state], epsilon, nA)) \
            if state in Q else env.action_space.sample()
        next_state, reward, done, _, _ = env.step(action)
        count_card(env.player[-1])
        episode.append((state, action, reward))
        state = next_state

        if done:
            break

    # now we can see all dealer's cards
    count_cards(env.dealer[1:])
    return episode


def montecarlo_control(env, num_episodes, alpha, gamma=1.0, eps_min=0.01, verbose=True,
    n_actions = env.action_space.n
    Q = defaultdict(lambda: np.zeros(n_actions))
    total_reward = 0.
    rewards = []

    for i in range(num_episodes):
        eps = max(1.0 / (i+1), eps_min)
        episode = generate_episode_from_Q(env, Q, eps, n_actions)
        Q = update_Q(env, episode, Q, alpha, gamma)
        cur_reward = episode[-1][-1]
        total_reward += cur_reward
```

```python
        if i % verbose_batch_size == 0 and i > 0 and verbose:
            print(f"Episode {i} / {num_episodes}. Avg reward: {total_reward/verbose_bat
            total_reward = 0.
            rewards.append(estimate_reward(Q))

    return Q, rewards
```

In [11]:

```python
%%time

N_verbose_batch_size = 50000
N_iterations = N_verbose_batch_size * 200

Q, rewards = montecarlo_control(env, N_iterations,
                                alpha=0.1, gamma=1, eps_min=0.001,
                                verbose_batch_size=N_verbose_batch_size)
```

```
Episode 50000 / 10000000. Avg reward: -0.18242
Episode 100000 / 10000000. Avg reward: -0.12163
Episode 150000 / 10000000. Avg reward: -0.09967
Episode 200000 / 10000000. Avg reward: -0.0826
Episode 250000 / 10000000. Avg reward: -0.07732
Episode 300000 / 10000000. Avg reward: -0.0746
Episode 350000 / 10000000. Avg reward: -0.069
Episode 400000 / 10000000. Avg reward: -0.05633
Episode 450000 / 10000000. Avg reward: -0.06511
Episode 500000 / 10000000. Avg reward: -0.05844
Episode 550000 / 10000000. Avg reward: -0.05991
Episode 600000 / 10000000. Avg reward: -0.05344
Episode 650000 / 10000000. Avg reward: -0.04527
Episode 700000 / 10000000. Avg reward: -0.05208
Episode 750000 / 10000000. Avg reward: -0.04449
Episode 800000 / 10000000. Avg reward: -0.06154
Episode 850000 / 10000000. Avg reward: -0.04685
Episode 900000 / 10000000. Avg reward: -0.05638
Episode 950000 / 10000000. Avg reward: -0.0551
Episode 1000000 / 10000000. Avg reward: -0.04822
Episode 1050000 / 10000000. Avg reward: -0.03369
Episode 1100000 / 10000000. Avg reward: -0.05164
Episode 1150000 / 10000000. Avg reward: -0.04377
Episode 1200000 / 10000000. Avg reward: -0.03752
Episode 1250000 / 10000000. Avg reward: -0.04406
Episode 1300000 / 10000000. Avg reward: -0.04223
Episode 1350000 / 10000000. Avg reward: -0.05554
Episode 1400000 / 10000000. Avg reward: -0.04337
Episode 1450000 / 10000000. Avg reward: -0.04774
Episode 1500000 / 10000000. Avg reward: -0.04158
Episode 1550000 / 10000000. Avg reward: -0.04749
Episode 1600000 / 10000000. Avg reward: -0.03749
Episode 1650000 / 10000000. Avg reward: -0.04767
Episode 1700000 / 10000000. Avg reward: -0.03207
Episode 1750000 / 10000000. Avg reward: -0.04195
Episode 1800000 / 10000000. Avg reward: -0.04353
Episode 1850000 / 10000000. Avg reward: -0.04613
Episode 1900000 / 10000000. Avg reward: -0.04244
Episode 1950000 / 10000000. Avg reward: -0.0427
Episode 2000000 / 10000000. Avg reward: -0.04389
Episode 2050000 / 10000000. Avg reward: -0.04611
Episode 2100000 / 10000000. Avg reward: -0.04206
Episode 2150000 / 10000000. Avg reward: -0.04146
Episode 2200000 / 10000000. Avg reward: -0.04134
Episode 2250000 / 10000000. Avg reward: -0.04702
Episode 2300000 / 10000000. Avg reward: -0.04729
Episode 2350000 / 10000000. Avg reward: -0.04096
Episode 2400000 / 10000000. Avg reward: -0.03517
Episode 2450000 / 10000000. Avg reward: -0.04387
Episode 2500000 / 10000000. Avg reward: -0.04511
Episode 2550000 / 10000000. Avg reward: -0.04087
Episode 2600000 / 10000000. Avg reward: -0.03758
Episode 2650000 / 10000000. Avg reward: -0.04296
Episode 2700000 / 10000000. Avg reward: -0.03531
Episode 2750000 / 10000000. Avg reward: -0.04528
```

```
Episode 2800000 / 10000000. Avg reward: -0.03816
Episode 2850000 / 10000000. Avg reward: -0.04076
Episode 2900000 / 10000000. Avg reward: -0.04314
Episode 2950000 / 10000000. Avg reward: -0.04058
Episode 3000000 / 10000000. Avg reward: -0.05052
Episode 3050000 / 10000000. Avg reward: -0.0355
Episode 3100000 / 10000000. Avg reward: -0.0414
Episode 3150000 / 10000000. Avg reward: -0.04287
Episode 3200000 / 10000000. Avg reward: -0.03635
Episode 3250000 / 10000000. Avg reward: -0.03383
Episode 3300000 / 10000000. Avg reward: -0.0397
Episode 3350000 / 10000000. Avg reward: -0.03589
Episode 3400000 / 10000000. Avg reward: -0.03609
Episode 3450000 / 10000000. Avg reward: -0.03764
Episode 3500000 / 10000000. Avg reward: -0.03621
Episode 3550000 / 10000000. Avg reward: -0.03707
Episode 3600000 / 10000000. Avg reward: -0.03456
Episode 3650000 / 10000000. Avg reward: -0.03817
Episode 3700000 / 10000000. Avg reward: -0.04034
Episode 3750000 / 10000000. Avg reward: -0.03783
Episode 3800000 / 10000000. Avg reward: -0.0245
Episode 3850000 / 10000000. Avg reward: -0.03253
Episode 3900000 / 10000000. Avg reward: -0.0409
Episode 3950000 / 10000000. Avg reward: -0.03398
Episode 4000000 / 10000000. Avg reward: -0.03199
Episode 4050000 / 10000000. Avg reward: -0.03501
Episode 4100000 / 10000000. Avg reward: -0.03687
Episode 4150000 / 10000000. Avg reward: -0.03456
Episode 4200000 / 10000000. Avg reward: -0.02152
Episode 4250000 / 10000000. Avg reward: -0.03057
Episode 4300000 / 10000000. Avg reward: -0.02758
Episode 4350000 / 10000000. Avg reward: -0.03403
Episode 4400000 / 10000000. Avg reward: -0.03984
Episode 4450000 / 10000000. Avg reward: -0.03552
Episode 4500000 / 10000000. Avg reward: -0.03429
Episode 4550000 / 10000000. Avg reward: -0.03568
Episode 4600000 / 10000000. Avg reward: -0.03802
Episode 4650000 / 10000000. Avg reward: -0.02842
Episode 4700000 / 10000000. Avg reward: -0.0379
Episode 4750000 / 10000000. Avg reward: -0.03599
Episode 4800000 / 10000000. Avg reward: -0.03208
Episode 4850000 / 10000000. Avg reward: -0.03027
Episode 4900000 / 10000000. Avg reward: -0.03789
Episode 4950000 / 10000000. Avg reward: -0.04107
Episode 5000000 / 10000000. Avg reward: -0.03272
Episode 5050000 / 10000000. Avg reward: -0.035
Episode 5100000 / 10000000. Avg reward: -0.03245
Episode 5150000 / 10000000. Avg reward: -0.03418
Episode 5200000 / 10000000. Avg reward: -0.0285
Episode 5250000 / 10000000. Avg reward: -0.04059
Episode 5300000 / 10000000. Avg reward: -0.02872
Episode 5350000 / 10000000. Avg reward: -0.04562
Episode 5400000 / 10000000. Avg reward: -0.0356
Episode 5450000 / 10000000. Avg reward: -0.03313
Episode 5500000 / 10000000. Avg reward: -0.03315
Episode 5550000 / 10000000. Avg reward: -0.04142
Episode 5600000 / 10000000. Avg reward: -0.04261
Episode 5650000 / 10000000. Avg reward: -0.0374
Episode 5700000 / 10000000. Avg reward: -0.02733
Episode 5750000 / 10000000. Avg reward: -0.03847
Episode 5800000 / 10000000. Avg reward: -0.03586
Episode 5850000 / 10000000. Avg reward: -0.02881
Episode 5900000 / 10000000. Avg reward: -0.04035
Episode 5950000 / 10000000. Avg reward: -0.03551
Episode 6000000 / 10000000. Avg reward: -0.04215
Episode 6050000 / 10000000. Avg reward: -0.0344
Episode 6100000 / 10000000. Avg reward: -0.03566
Episode 6150000 / 10000000. Avg reward: -0.0395
Episode 6200000 / 10000000. Avg reward: -0.03562
Episode 6250000 / 10000000. Avg reward: -0.03466
Episode 6300000 / 10000000. Avg reward: -0.02975
Episode 6350000 / 10000000. Avg reward: -0.0375
Episode 6400000 / 10000000. Avg reward: -0.03692
Episode 6450000 / 10000000. Avg reward: -0.03309
```

```
Episode 6500000 / 10000000. Avg reward: -0.03008
Episode 6550000 / 10000000. Avg reward: -0.03392
Episode 6600000 / 10000000. Avg reward: -0.03307
Episode 6650000 / 10000000. Avg reward: -0.03432
Episode 6700000 / 10000000. Avg reward: -0.03387
Episode 6750000 / 10000000. Avg reward: -0.03861
Episode 6800000 / 10000000. Avg reward: -0.02651
Episode 6850000 / 10000000. Avg reward: -0.02735
Episode 6900000 / 10000000. Avg reward: -0.03696
Episode 6950000 / 10000000. Avg reward: -0.03378
Episode 7000000 / 10000000. Avg reward: -0.04045
Episode 7050000 / 10000000. Avg reward: -0.03583
Episode 7100000 / 10000000. Avg reward: -0.03984
Episode 7150000 / 10000000. Avg reward: -0.03217
Episode 7200000 / 10000000. Avg reward: -0.03269
Episode 7250000 / 10000000. Avg reward: -0.03602
Episode 7300000 / 10000000. Avg reward: -0.03759
Episode 7350000 / 10000000. Avg reward: -0.03502
Episode 7400000 / 10000000. Avg reward: -0.03555
Episode 7450000 / 10000000. Avg reward: -0.03677
Episode 7500000 / 10000000. Avg reward: -0.03788
Episode 7550000 / 10000000. Avg reward: -0.03692
Episode 7600000 / 10000000. Avg reward: -0.04084
Episode 7650000 / 10000000. Avg reward: -0.03516
Episode 7700000 / 10000000. Avg reward: -0.03678
Episode 7750000 / 10000000. Avg reward: -0.03034
Episode 7800000 / 10000000. Avg reward: -0.04219
Episode 7850000 / 10000000. Avg reward: -0.03428
Episode 7900000 / 10000000. Avg reward: -0.03621
Episode 7950000 / 10000000. Avg reward: -0.03989
Episode 8000000 / 10000000. Avg reward: -0.03001
Episode 8050000 / 10000000. Avg reward: -0.02625
Episode 8100000 / 10000000. Avg reward: -0.04478
Episode 8150000 / 10000000. Avg reward: -0.03642
Episode 8200000 / 10000000. Avg reward: -0.02953
Episode 8250000 / 10000000. Avg reward: -0.03313
Episode 8300000 / 10000000. Avg reward: -0.02862
Episode 8350000 / 10000000. Avg reward: -0.03551
Episode 8400000 / 10000000. Avg reward: -0.03041
Episode 8450000 / 10000000. Avg reward: -0.02837
Episode 8500000 / 10000000. Avg reward: -0.03429
Episode 8550000 / 10000000. Avg reward: -0.03552
Episode 8600000 / 10000000. Avg reward: -0.0335
Episode 8650000 / 10000000. Avg reward: -0.03096
Episode 8700000 / 10000000. Avg reward: -0.03909
Episode 8750000 / 10000000. Avg reward: -0.03311
Episode 8800000 / 10000000. Avg reward: -0.02655
Episode 8850000 / 10000000. Avg reward: -0.03696
Episode 8900000 / 10000000. Avg reward: -0.03676
Episode 8950000 / 10000000. Avg reward: -0.03668
Episode 9000000 / 10000000. Avg reward: -0.03841
Episode 9050000 / 10000000. Avg reward: -0.0367
Episode 9100000 / 10000000. Avg reward: -0.03527
Episode 9150000 / 10000000. Avg reward: -0.03946
Episode 9200000 / 10000000. Avg reward: -0.03891
Episode 9250000 / 10000000. Avg reward: -0.03495
Episode 9300000 / 10000000. Avg reward: -0.0413
Episode 9350000 / 10000000. Avg reward: -0.03747
Episode 9400000 / 10000000. Avg reward: -0.03502
Episode 9450000 / 10000000. Avg reward: -0.02883
Episode 9500000 / 10000000. Avg reward: -0.04468
Episode 9550000 / 10000000. Avg reward: -0.03503
Episode 9600000 / 10000000. Avg reward: -0.03065
Episode 9650000 / 10000000. Avg reward: -0.03483
Episode 9700000 / 10000000. Avg reward: -0.03455
Episode 9750000 / 10000000. Avg reward: -0.03422
Episode 9800000 / 10000000. Avg reward: -0.03382
Episode 9850000 / 10000000. Avg reward: -0.031
Episode 9900000 / 10000000. Avg reward: -0.02958
Episode 9950000 / 10000000. Avg reward: -0.03226
CPU times: user 1h 22min 55s, sys: 12.3 s, total: 1h 23min 7s
Wall time: 1h 23min 24s
```

In [12]:

```
# estimated reward
```

```
estimate_reward(Q, 1000000)
```

Out[12]: -0.0310635

In [14]:

```
plt.plot(list(range(len(rewards))), rewards)
plt.title("Rewards during learning process")
plt.ylabel("Average reward")
plt.xlabel("Learning iteration, x200")
plt.show()
```

In [15]:

```
policy = policy_from_Q(Q)
```

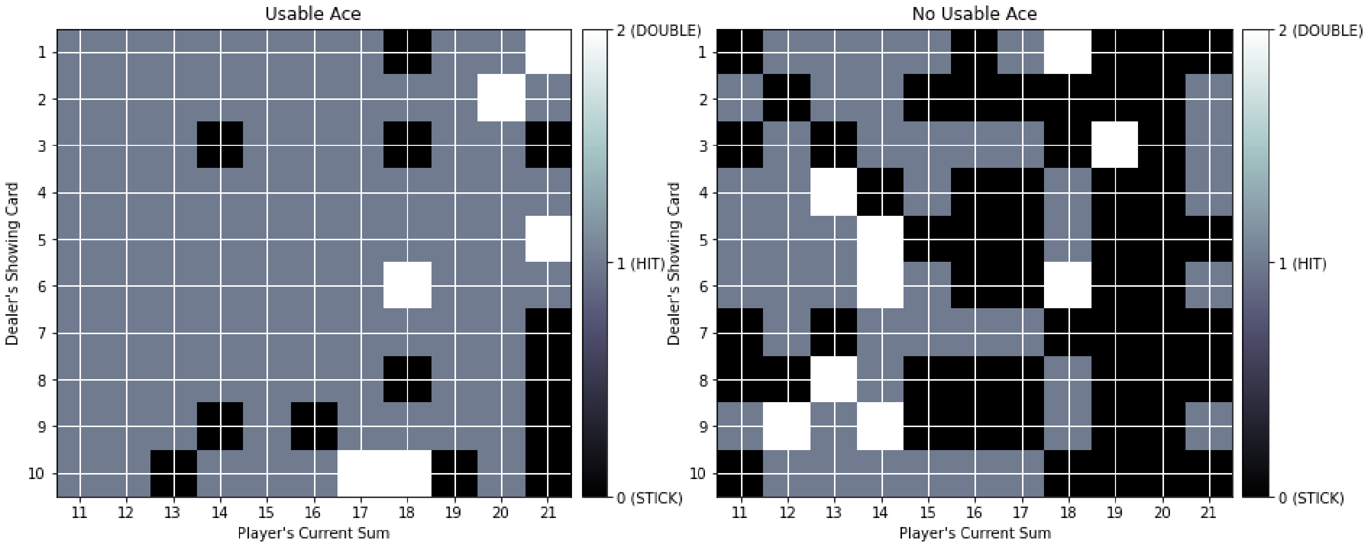
In [16]:

```
def plot_policy(policy, card_count):

    def get_Z(x, y, usable_ace, card_count):
        if (x,y,usable_ace, card_count) in policy:

            return policy[(x,y,usable_ace, card_count)]
        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, card_count) for x in x_range] for y in y_r
        surf = ax.imshow(Z, cmap=plt.get_cmap('bone'), vmin=0, vmax=2, extent=[10.5, 21
        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=[0,1,2], cax=cax)
        cbar.ax.set_yticklabels(['0 (STICK)','1 (HIT)', '2 (DOUBLE)'])

    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()
```

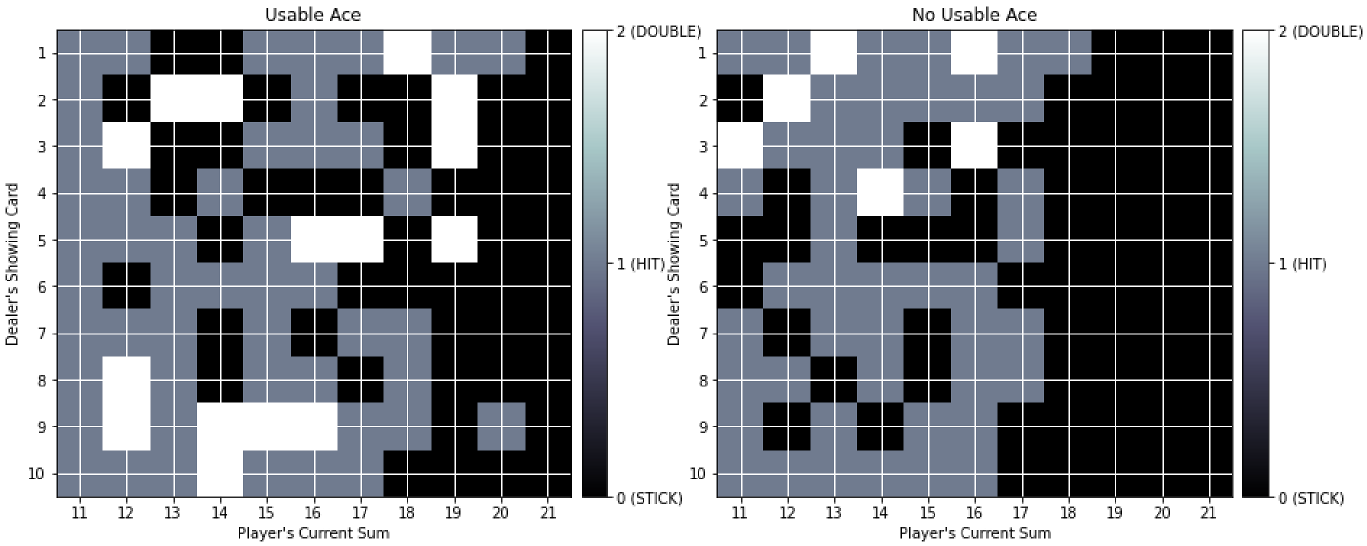
In [17]:

```python
for card_count in range(-10, 10, 3):
    print(f"card count: {card_count}")
    plot_policy(policy, card_count)
```

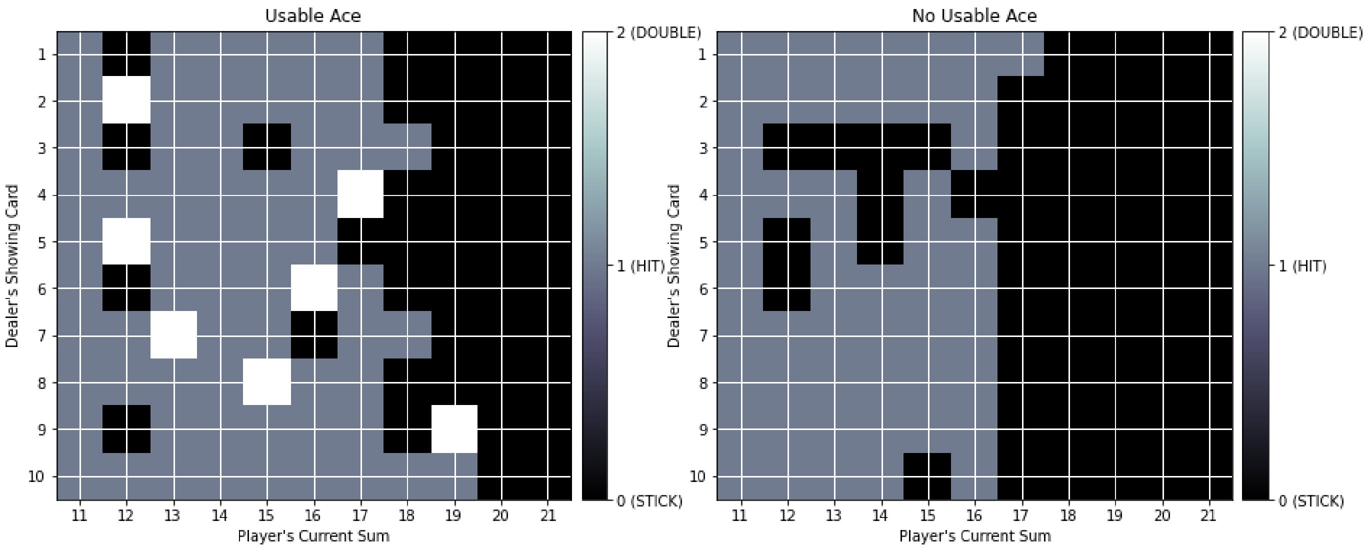
card count: -10



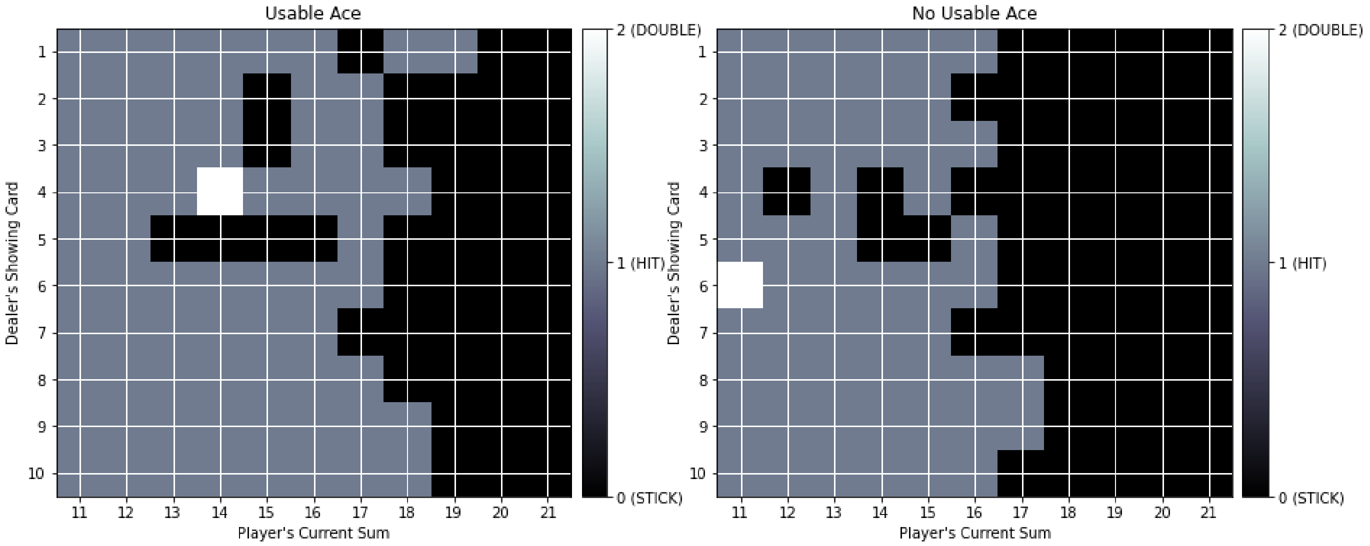
card count: -7



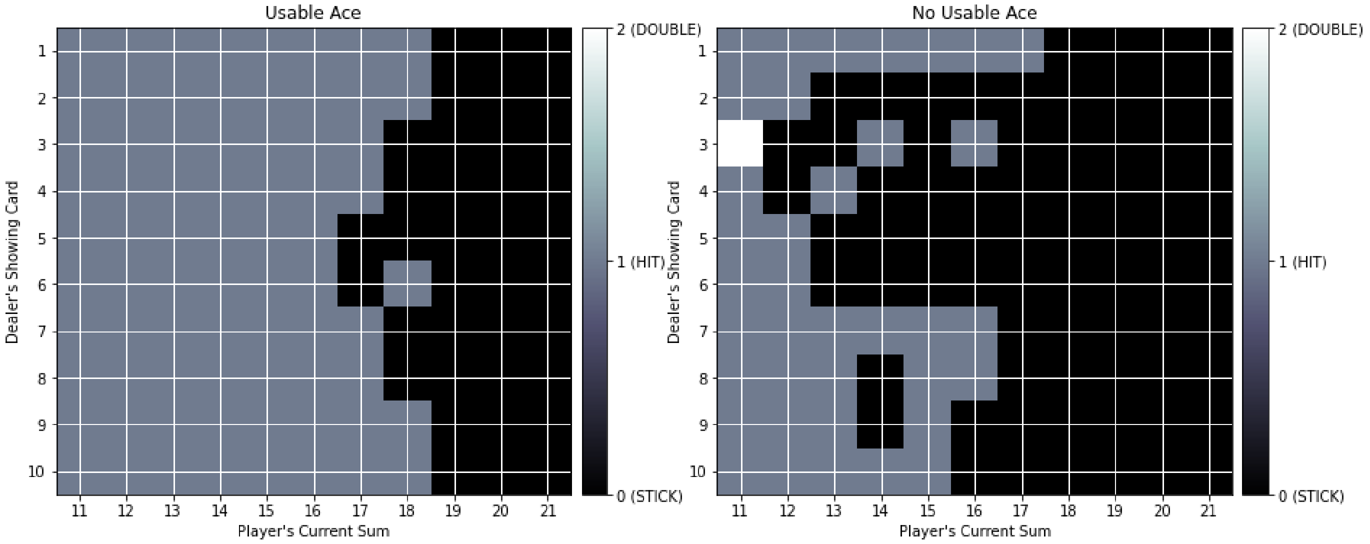
card count: -4



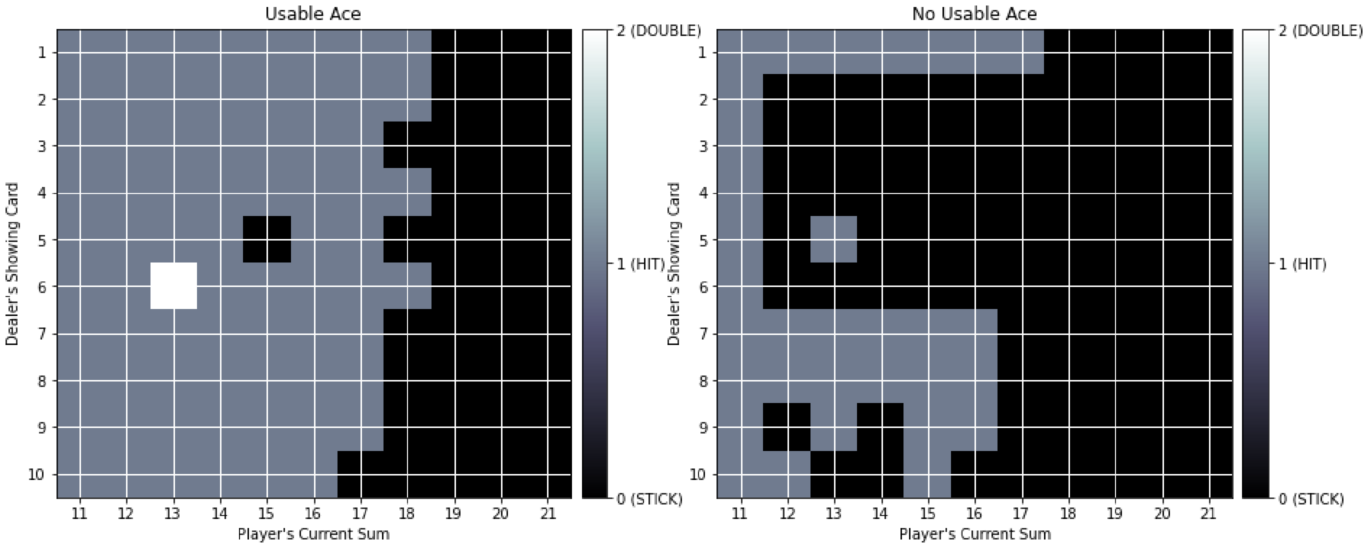
card count: -1



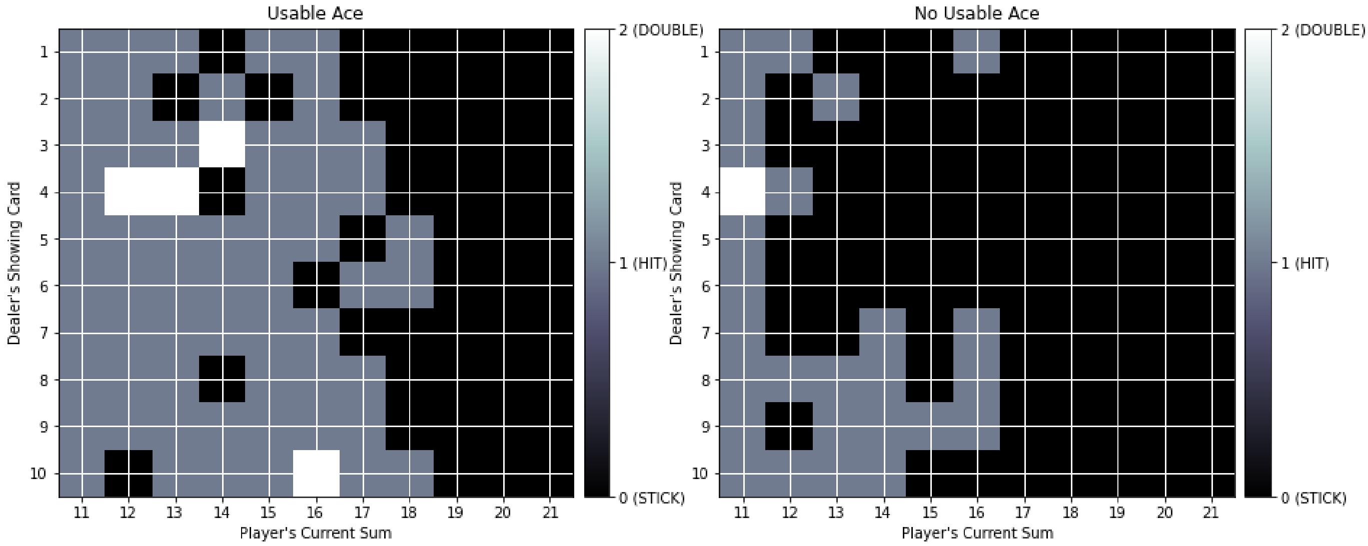
card count: 2



card count: 5



card count: 8



With simple strategy "19, 20, 21" the reward is -0.198529

On basic Blackjack environment Q-learning reached -0.109 and Monte Carlo control gave -0.05551. Because of these scores I decided that MC control is more promising and evaluated later environments with this algorithm.

After adding "double" to the environment my calculations with MC control showed -0.06394, I think it just didn't converge compeletely or some other hyperparameters are optimal.

Blackjack with counts and double reached average reward of -0.0310635. It still loses money!

```
In [ ]:
```