# FrozenLake 5

## UWAGA

Wczytaj do Colab plik **frozen_lake.py** (intrukcja w pliku **COLAB_instrukcja.pdf**).

```
from frozen_lake import FrozenLakeEnv
#from frozen_lake_slippery import FrozenLakeEnv
import numpy as np
import random

env = FrozenLakeEnv()
```

## Implementacja algorytmu

Algorytm wygląda następująco (objaśnienia do algorytmu w wykładzie 5):

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(termina$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

Funkcję **Q** inicjujemy **zerami** za pomocą tablicy o wymiarach **16x4**:

```
Q = np.zeros([env.nS,env.nA])
print(Q)
```

```
[[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. 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. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Definicja funkcji, która dla danego stanu **S** zwraca akcję zgodnię z **polityką epsilon-zachłanną**:

```
def epsilon_greedy_action(env,Q,state,epsilon=0.3):
    n = random.uniform(0,1)
    if n<= epsilon:
        return np.random.randint(env.action_space.n)
    else:
        return np.argmax(Q[state])
```

**Objaśnienie**: losujemy liczbę **n** z przedziału **(0,1)**. Jeżeli **n<epsilon** wówczas funkcja zwraca **losow** zwraca **akcję o największym zwrocie**.

## Polecenie 1 (do uzupełnienia)

Uzupełnij poniższą funkcję implementującą algorytm **SARSA**:

```
def SARSA_Q(env, episodes=1000, gamma=0.91, alpha=0.1):

    Q = np.zeros([env.nS,env.nA])

    for i in range(episodes):
        finished = False

        env.reset()

        S = env.s
        A = epsilon_greedy_action(env,Q,S)

        while not finished:

            next_S, R, finished, _ = env.step(A)
            next_A = epsilon_greedy_action(env,Q,next_S)

            Q[S][A] = Q[S][A] + alpha * (R + gamma * Q[next_S, next_A] - Q[S, A])
```

```
            S = next_S
            A = next_A

    return Q
```

Test:

```
Q = SARSA_Q(env,2000)
print(np.round(Q,2))   #zaokrąglamy do dwóch miejsc po przecinku
```

```
[[0.32 0.4  0.23 0.35]
 [0.35 0.   0.03 0.15]
 [0.12 0.   0.   0.01]
 [0.01 0.   0.   0.  ]
 [0.4  0.44 0.   0.32]
 [0.   0.   0.   0.  ]
 [0.   0.68 0.   0.  ]
 [0.   0.   0.   0.  ]
 [0.51 0.   0.57 0.39]
 [0.42 0.71 0.63 0.  ]
 [0.51 0.86 0.   0.41]
 [0.   0.   0.   0.  ]
 [0.   0.   0.   0.  ]
 [0.   0.66 0.86 0.61]
 [0.64 0.87 1.   0.63]
 [0.   0.   0.   0.  ]]
```

# Polecenie 2 (do uzupełnienia)

Przestestuj działanie algorytmu **SARSA** dla dwóch wartości parametru gamma (**0.1** i **0.99**) i różny
algorytmu? Czy wybór w każdym stanie akcji związanej z największym zwrotem gwarantuje dotar

```
Q = SARSA_Q(env,5000,0.1)
print(np.round(Q,2))
```

```
      [[0.   0.   0.   0.  ]
       [0.   0.   0.   0.  ]
       [0.   0.   0.   0.  ]

Q = SARSA_Q(env,3000,0.99)
print(np.round(Q,2))
```

```
 [[0.42 0.46 0.29 0.41]
  [0.41 0.   0.   0.06]
  [0.04 0.   0.   0.  ]
  [0.   0.   0.   0.  ]
  [0.41 0.52 0.   0.41]
  [0.   0.   0.   0.  ]
  [0.   0.63 0.   0.  ]
  [0.   0.   0.   0.  ]
  [0.49 0.   0.63 0.42]
  [0.5  0.52 0.68 0.  ]
  [0.53 0.9  0.   0.33]
  [0.   0.   0.   0.  ]
  [0.   0.   0.   0.  ]
  [0.   0.09 0.95 0.13]
  [0.55 0.65 1.   0.67]
  [0.   0.   0.   0.  ]]
```

(DO UZUEPEŁNIENIA)

Według mnie dla gammy równej 0.1 nie działa poprawnie, ponieważ w większości stanów nieterm
wynoszą 0. Natomiast dla gammy równej 0.99 wybór w każdym stanie nieterminalnym akcji z naj
celu.