Sieci samouczące się

Laboratorium 1

Przemysław Rośleń 180150

Problem żeglarza rozwiązałem stosując 2 różne warianty algorytmu monte Carlo dla uczenia ze wzmocnieniem Q-learning:

- Iteracja strategii
- Iteracji wartości

Poniżej przedstawiłem dla każdej z tych dwóch metod:

- kod źródłowy funkcji rozwiązującej problem żeglarza stosując dany algorytm
- wyniki w postaci średnich sum nagród uzyskanych w środowiskach:
    - map_small
    - map_middle
    - map_spiral
- wyniki w postaci wizualizacji uzyskanej strategii (zwroty strzałek na mapie)

Na końcu raportu umieściłem wnioski porównując oba podejścia pod kątem złożoności obliczeniowej, stabilności oraz uzyskanych wyników w eksperymentach.

Iteracja strategii:

Kod powstał w głównej mierze na bazie pseudokodu przedstawionego na wykładzie.

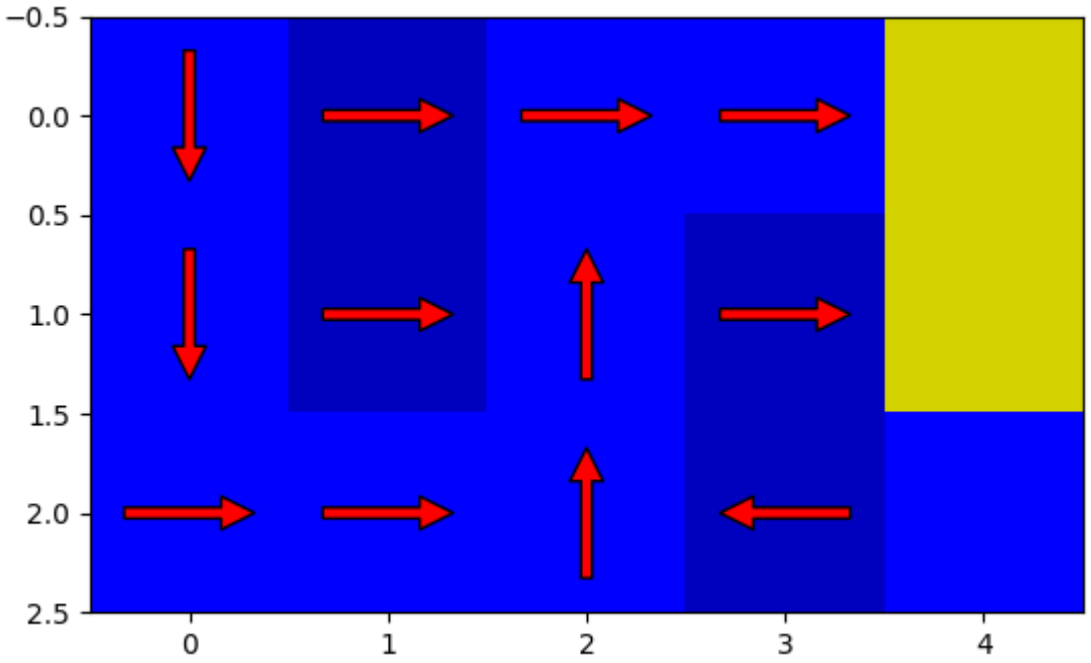
Kod:

```
def sailor_train_strategy_iteration(reward_map, Q,
num_of_episodes, gamma, init_state, init_action, first_training):
    num_of_rows, num_of_columns = reward_map.shape
    num_of_steps_max = int(5 * (num_of_rows + num_of_columns))  #
maximum number of steps in an episode
    sum_of_rewards = np.zeros([num_of_episodes], dtype=float)

    for episode in range(num_of_episodes):
        state = init_state #np.zeros([2], dtype=int)  # initial
state here [1 1] but rather random due to exploration
        the_end = False
        nr_pos = 0
        while the_end == False:
            nr_pos = nr_pos + 1  # move number

            # Action choosing (1 - right, 2 - up, 3 - left, 4 -
bottom):
            if nr_pos == 1:
                action = 1 + init_action
            else:
                if first_training:
                    action = 1 + np.random.randint(0, 4)
                else:
                    action = 1 + np.argmax(Q[state[0], state[1],
:])
            state_next, reward = environment(state, action,
reward_map)
            state = state_next  # going to the next state

            # end of episode if maximum number of steps is reached
or last column
            # is reached
            if (nr_pos == num_of_steps_max) | (state[1] >=
num_of_columns - 1):
                the_end = True

            sum_of_rewards[episode] += (reward * np.power(gamma,
nr_pos))

    for episode in range(num_of_episodes):
        Q[init_state[0]][init_state[1]][init_action] +=
np.mean(sum_of_rewards)
    return Q
```

```
# ITERACJA STRATEGII
epochs = 50
for i in range(epochs):
   print(f'epoch {i + 1}')
   for row in reversed(range(num_of_columns)):
       for col in range(num_of_rows):
           for action in range(num_of_actions):
               if i == 0:
                   Q =
sf.sailor_train_strategy_iteration(reward_map, Q,
number_of_episodes, gamma, [row, col], action, True)
               else:
                   Q =
sf.sailor_train_strategy_iteration(reward_map, Q,
number_of_episodes, gamma, [row, col], action, False)
   sf.sailor_test(reward_map, Q, number_of_episodes)
   sf.draw(reward_map, Q)
```

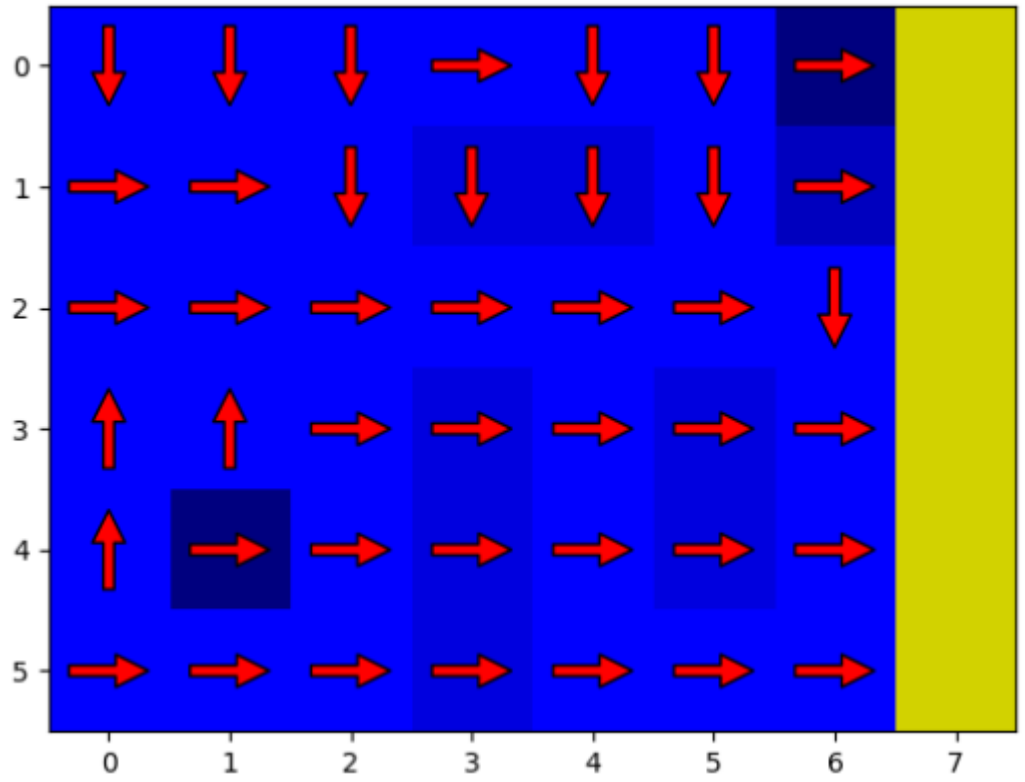
1) map_small (50 epok -> 1000 iteracji każda):

```
epoch 45
test-1000 mean sum of rewards = 1.9776000000000002
epoch 46
test-1000 mean sum of rewards = 1.4905
epoch 47
test-1000 mean sum of rewards = 1.6452
epoch 48
test-1000 mean sum of rewards = 1.8085000000000002
epoch 49
test-1000 mean sum of rewards = 1.7387
epoch 50
test-1000 mean sum of rewards = 1.6016

Process finished with exit code 0
```



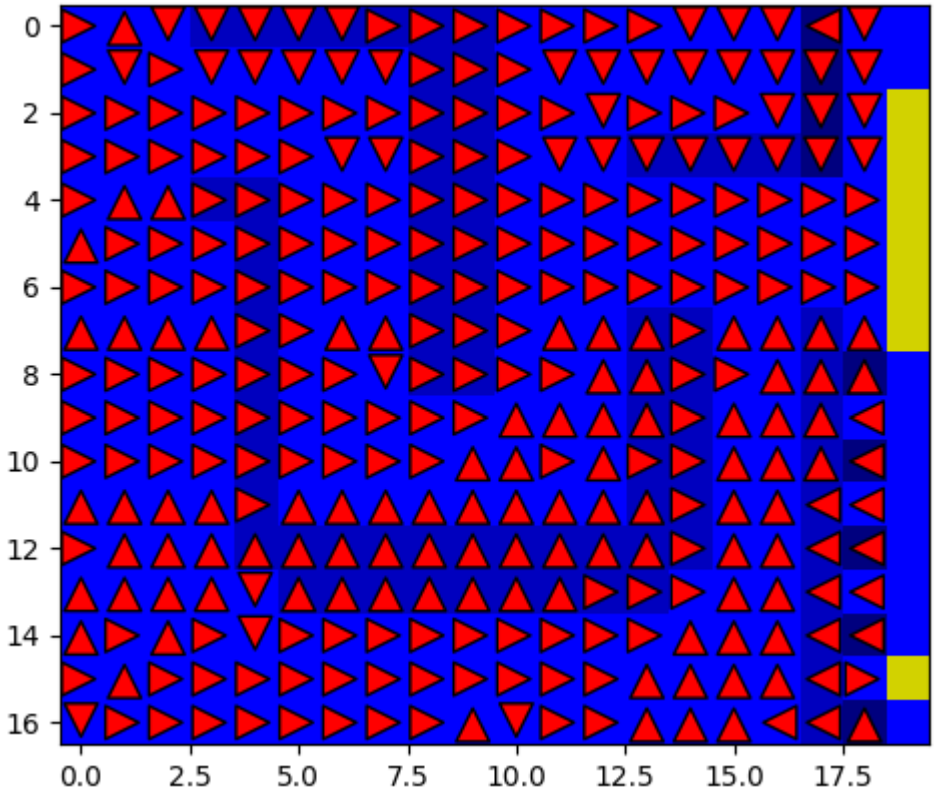
2) map_middle (50 epok -> 1000 iteracji każda):

```
epoch 45
test-1000 mean sum of rewards = 6.6711
epoch 46
test-1000 mean sum of rewards = 6.6048
epoch 47
test-1000 mean sum of rewards = 6.6406
epoch 48
test-1000 mean sum of rewards = 6.6883
epoch 49
test-1000 mean sum of rewards = 6.640200000000001
epoch 50
test-1000 mean sum of rewards = 6.7989

Process finished with exit code 0
```



3) map_spiral (30 epok -> 100 iteracji każda):

```
epoch 25
test-100 mean sum of rewards = 170.868
epoch 26
test-100 mean sum of rewards = 164.245
epoch 27
test-100 mean sum of rewards = 163.90400000000002
epoch 28
test-100 mean sum of rewards = 166.81599999999997
epoch 29
test-100 mean sum of rewards = 167.77700000000002
epoch 30
test-100 mean sum of rewards = 172.065

Process finished with exit code 0
```



Iteracja wartości:

Kod powstał w głównej mierze na bazie pseudokodu przedstawionego na wykładzie.

Kod:

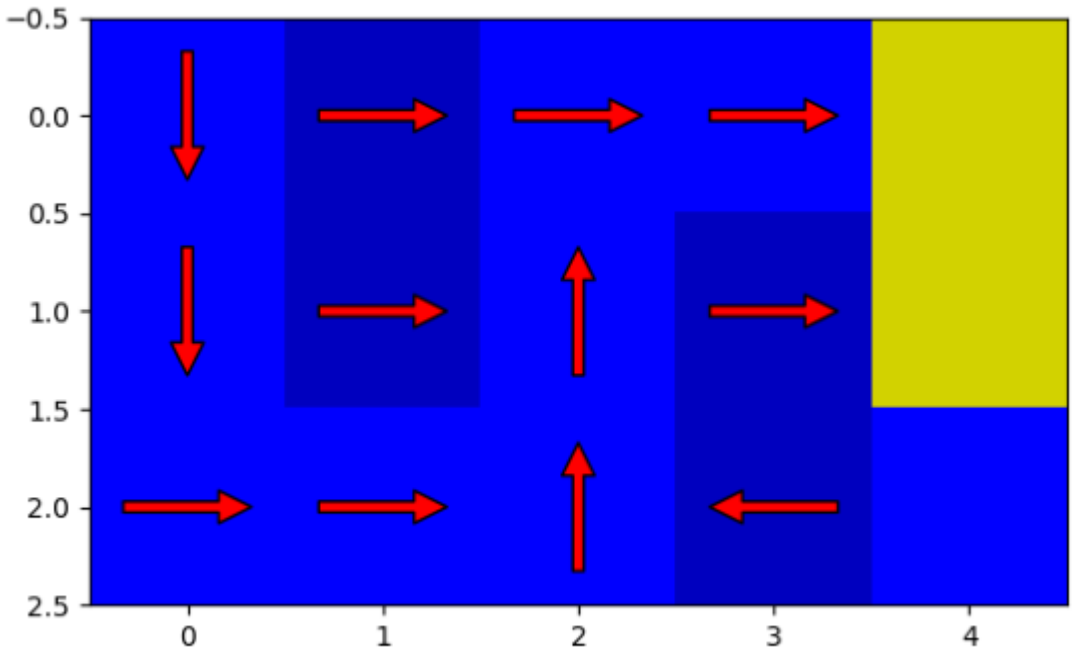
```
def sailor_train_value_iteration(reward_map, Q, gamma, init_state,
init_action, first_training, alpha):
    num_of_rows, num_of_columns = reward_map.shape
    num_of_steps_max = int(5 * (num_of_rows + num_of_columns))  #
maximum number of steps in an episode
    sum_of_rewards = 0

    state = init_state #np.zeros([2], dtype=int)  # initial state
here [1 1] but rather random due to exploration
    the_end = False
    nr_pos = 0
    while the_end == False:
        nr_pos = nr_pos + 1  # move number

        # Action choosing (1 - right, 2 - up, 3 - left, 4 -
bottom):
```

```
        action = (init_action if (state[0] == init_state[0] and
state[1] == init_state[1]) else np.argmax(Q[state[0], state[1]]))
+ 1
        state_next, reward = environment(state, action, reward_map)

        sum_of_rewards += np.power(gamma, nr_pos - 1) * reward
        state = state_next  # going to the next state

        # end of episode if maximum number of steps is reached or
last column
        # is reached
        if (nr_pos == num_of_steps_max) | (state[1] >=
num_of_columns - 1):
            the_end = True

    Q[init_state[0]][init_state[1]][init_action] = (1 - alpha) *
Q[init_state[0]][init_state[1]][init_action] + (alpha *
sum_of_rewards)
    return Q
```

```
# ITERACJA WARTOŚCI
epochs = 1000
for i in range(epochs):
    iteration = 0
    for row in range(num_of_rows):
        for col in range(num_of_columns):
            for action in range(num_of_actions):
                iteration += 1
                if i == 0:
                    Q = sf.sailor_train_value_iteration(reward_map,
Q, gamma, [row, col], action, True, 1 / iteration)
                else:
                    Q = sf.sailor_train_value_iteration(reward_map,
Q, gamma, [row, col], action, False, 1 / iteration)
    if i % 50 == 0:
        print(f'epoch {i + 1}')
        sf.sailor_test(reward_map, Q, number_of_episodes)
        sf.draw(reward_map, Q)
sf.sailor_test(reward_map, Q, number_of_episodes)
sf.draw(reward_map, Q)
```

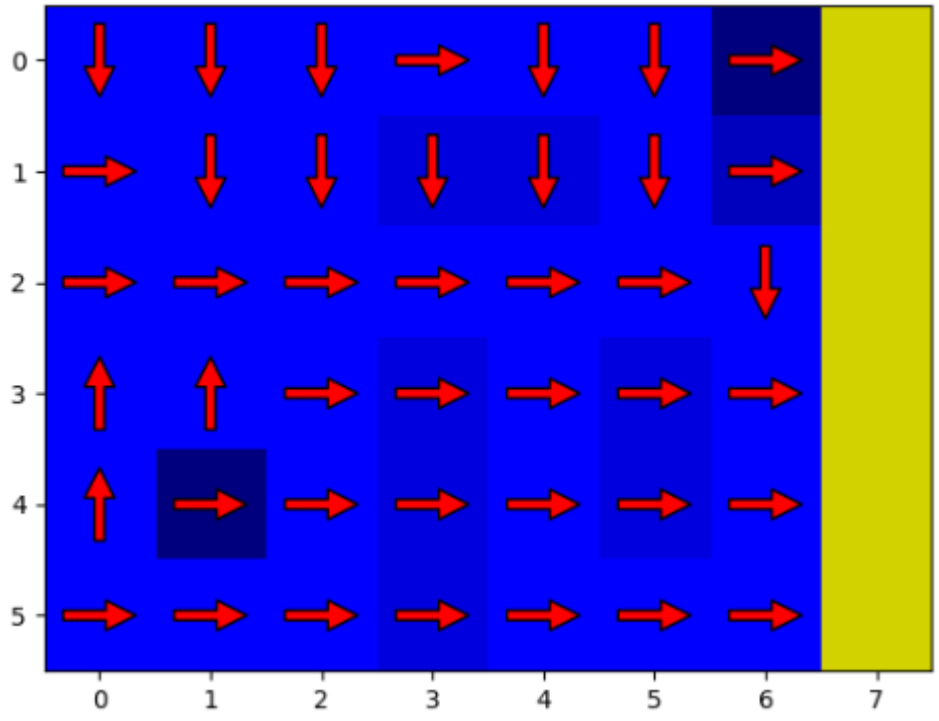
1) map_small (1000 epok):

```
test-1000 mean sum of rewards = 2.0644
epoch 801
test-1000 mean sum of rewards = 1.7828
epoch 851
test-1000 mean sum of rewards = 1.5888000000000002
epoch 901
test-1000 mean sum of rewards = 1.5684
epoch 951
test-1000 mean sum of rewards = 1.8504
test-1000 mean sum of rewards = 2.1159

Process finished with exit code 0
```



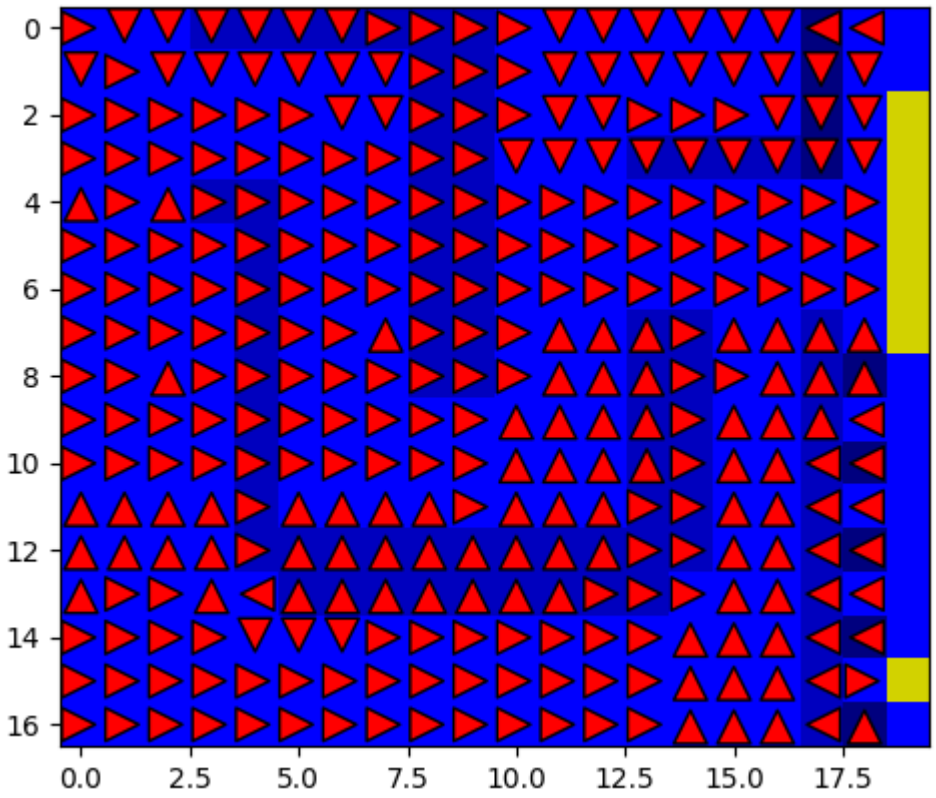
2) map_middle (1000 epok):

```
epoch 701
test-1000 mean sum of rewards = 6.835
epoch 751
test-1000 mean sum of rewards = 6.7878
epoch 801
test-1000 mean sum of rewards = 6.897600000000001
epoch 851
test-1000 mean sum of rewards = 6.779400000000001
epoch 901
test-1000 mean sum of rewards = 6.649900000000001
epoch 951
test-1000 mean sum of rewards = 6.722600000000001
test-1000 mean sum of rewards = 6.737

Process finished with exit code 0
```



3) map_spiral (1000 epok):

```
epoch 651
test-1000 mean sum of rewards = 169.59890000000001
epoch 701
test-1000 mean sum of rewards = 168.84570000000002
epoch 751
test-1000 mean sum of rewards = 168.53090000000003
epoch 801
test-1000 mean sum of rewards = 168.86090000000002
epoch 851
test-1000 mean sum of rewards = 168.396
epoch 901
test-1000 mean sum of rewards = 168.74660000000003
epoch 951
test-1000 mean sum of rewards = 168.9236
test-1000 mean sum of rewards = 168.5979

Process finished with exit code 0
```



Wnioski:

1. Jeżeli chodzi o złożoność obliczeniowom dwóch powyższych podejść to iteracja wartości charakteryzuje się mniejszą złożonością obliczeniową od iteracji strategii. Iteracja strategii zawiera o jedną więcej zagnieżdżoną pętle for po episodach co sprawia że jej złożoność obliczeniowa jest n razy większa od iteracji wartości (gdzie n oznacza liczbę episodów dla każdej pary (stan, akcja).

2. Większą stabilnością charakteryzuje się algorytm iteracji strategii. Jest on bardziej przewidywalny i mniej podatny na niedeterminizm środowiska. Ponadto, daje nieco dokładniejsze wyniki w bardziej złożonych środowiskach (mapa spiral).