Aprendizaje por diferencia temporal (TD)

En este ejercicio vamos a implementar el método de diferencia temporal para para resolver MDPs con transiciones y recompensas desconocidas.

El método de TD se basa en el cálculo de los valores para los estados de acuerdo con la fórmula:

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

donde α corresponde a la taza de aprendizaje.

Task 1

Para implementar TD definimos `td_learning.py` como una extensión del ambiente de Gridworld. Dentro de esta extensión debemos asegurarnos que:

- Seguimos una política, dada como un parámetro del ambiente.
- Cada paso de la muestra ejecuta la política para el estado actual, obteniendo un estado de llegada y una recompensa. Tenga en cuenta que las acciones no son determinísticas y la ejecución de cada acción depende de un factor de ruido (en el caso de Gridworld, tomaremos un factor de ruido de 0.2 para las acciones abajo e izquierda y 0.3 para las acciones arriba y derecha, desconocida para el agente). Por ejemplo, el agente tiene una probabilidad de 0.8 de moverse a la izquierda y abajo y terminar en el estado correspondiente y probabilidad de 0.2 de terminar en cualquiera de las otras tres direcciones.
- A partir de los valores obtenidos de diferentes muestras, obtenga una nueva política.
- Utilice una taza de aprendizaje de `0.7`

Responda las preguntas

1. ¿Cuántas iteraciones son necesarias para que la política de las muestras se estabilice?
2. ¿Cómo se compara la política obtenida con la calculada utilizando iteración de valores o iteración de políticas? ¿Existe alguna diferencia? ¿Porqué?

In [1]:
```
from assignment_td_sarsa.environment_world import EnvironmentWorld, Action
from assignment_td_sarsa.td_learning import TDLearning

cliff_world = EnvironmentWorld([
    ['-1'] * 12,
    ['-1'] * 12,
    ['-1'] * 12,
    ['S'] + ['-100'] * 10 + ['1000']
```

```
],
    terminal_states=[(x, 3) for x in range(1, 12)], action_noise={
        Action.UP: 0.3,
        Action.DOWN: 0.2,
        Action.LEFT: 0.2,
        Action.RIGHT: 0.3
    })
```

In [2]:
```
cliff_world
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	SC	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1000

In [3]:
```
from random import choice

actions = cliff_world.ACTIONS

td_learning = TDLearning(cliff_world , lambda s: choice(actions), 0.7, 0.96)
```

In [4]:
```
td_learning.iterate_learning(num_steps=10000000)
```

```
100%|██████████| 10000000/10000000 [01:54<00:00, 87323.64it/s]
```

In [5]:
```
td_learning.print_values()
```

```
          0          1          2          3          4          5   \
0 -51.313448 -59.839199 -56.088128 -63.624704 -66.599113 -60.012854
1 -77.250012 -82.625253 -63.821178 -68.850574 -79.668051 -76.597852
2 -66.946054 -88.049476 -90.139071 -74.394359 -95.987255 -91.962725
3 -72.640919   0.000000   0.000000   0.000000   0.000000   0.000000

          6          7          8          9          10          11
0 -69.946293 -51.569629 -34.906770  -7.087166   28.889009   55.659964
1 -60.778289 -57.673284 -51.226461  36.371327  101.248186  704.372941
2 -72.578859 -57.268722 -88.678313 -39.761121  212.872839  253.459005
3   0.000000   0.000000   0.000000   0.000000    0.000000    0.000000
```

1. ¿Cuántas iteraciones son necesarias para que la política de las muestras se estabilice?

2. ¿Cómo se compara la política obtenida con la calculada utilizando iteración de valores o iteración de políticas? ¿Existe alguna diferencia? ¿Porqué?