In [60]:
```
import numpy as np
import random
```

In [61]:
```
class Field:
    def __init__(self) -> None:
        """
        Initialize field and set a random start state
        """
        self.states = [
            [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [-1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
            ]
        self.state = (random.randrange(0, len(self.states)), random.randrange(0, len(self.states[0])))

    def done(self):
        """
        Check if it isn't in a neutral state
        """
        if self.states[self.state[0]][self.state[1]] != 0:
            return True
        else:
            return False

    def get_possible_actions(self):
        """
        Return possible actions in state

        Action:
               0 => left
               1 => right
               2 => up
               3 => down
        """
        actions = [0, 1, 2, 3]
        if self.state[0] == 0:
            actions.remove(2)
        if self.state[0] == len(self.states) -1:
            actions.remove(3)
        if self.state[1] == 0:
            actions.remove(0)
        if self.state[1] == len(self.states[0]) -1:
            actions.remove(1)
        return actions

    def update_next_state(self, action):
        """
        Update state according to action -> Return state and reward
        """
        x, y = self.state

        if action == 0:
            if y == 0:
                return self.state, -10
            self.state = x, y - 1
        if action == 1:
            if y == len(self.states) -1:
                return self.state, -10
            self.state = x, y + 1
        if action == 2:
            if x == 0:
                return self.state, -10
            self.state = x - 1, y
        if action == 3:
            if self.state == len(self.states) -1:
                return self.state, -10
            self.state = x + 1, y
        reward = self.states[self.state[0]][self.state[1]]
        return self.state, reward
```

In [62]:
```
field = Field()
field.state, field.done(), field.get_possible_actions()
```

Out[62]:
```
((0, 7), False, [0, 1, 3])
```

In [63]:
```
field.update_next_state(2)
field.state, field.done(), field.get_possible_actions()
```

Out[63]:
```
((0, 7), False, [0, 1, 3])
```

In [64]:

```
#Train the model
field = Field()
q_table = np.zeros((len(field.states), len(field.states[0]), 4))

alpha = .5
epsilon = .5
gamma = .5

for _ in range(10000):
    field = Field()
    while not field.done():
        actions = field.get_possible_actions()
        if random.uniform(0, 1) < epsilon:
            action = random.choice(actions)
        else:
            action = np.argmax(q_table[field.state])

        cur_x, cur_y = field.state
        (next_x, next_y), reward = field.update_next_state(action)
        q_table[cur_x, cur_y, action] = (1 - alpha)*q_table[cur_x, cur_y, action] + alpha*(reward + gamma*np.max(
```

In [65]:

```
q_table
```

Out[65]:

```
array([[[  0.      ,   0.      ,   0.      ,   0.      ],
        [ -1.      ,   0.      ,   0.      ,   0.      ],
        [  0.      , -10.      ,   0.      ,   0.      ],
        [  0.      ,   0.03125 ,   0.      ,   0.03125 ],
        [  0.015625,   0.0625  ,   0.      ,   0.0625  ],
        [  0.03125 ,   0.125   ,   0.      ,   0.125   ],
        [  0.0625  ,   0.25    ,   0.      ,   0.25    ],
        [  0.125   ,   0.5     ,   0.      ,   0.5     ],
        [  0.25    ,   0.25    ,   0.      ,   1.      ],
        [  0.5     ,   0.125   ,   0.      ,   0.5     ],
        [  0.25    ,   0.0625  ,   0.      ,   0.25    ],
        [  0.125   ,   0.      ,   0.      ,   0.125   ]],

       [[  0.      ,   0.      ,   0.      ,   0.      ],
        [ -1.      ,   0.      ,   0.      ,   0.      ],
        [  0.      , -10.      ,   0.      ,   0.      ],
        [  0.      ,   0.0625  ,   0.015625,   0.015625],
        [  0.03125 ,   0.125   ,   0.03125 ,   0.03125 ],
        [  0.0625  ,   0.25    ,   0.0625  ,   0.0625  ],
        [  0.125   ,   0.5     ,   0.125   ,   0.125   ],
        [  0.25    ,   1.      ,   0.25    ,   0.25    ],
        [  0.      ,   0.      ,   0.      ,   0.      ],
        [  1.      ,   0.25    ,   0.25    ,   0.25    ],
        [  0.5     ,   0.125   ,   0.125   ,   0.125   ],
        [  0.25    ,   0.      ,   0.0625  ,   0.0625  ]],

       [[ -5.      ,   0.      ,  -1.      ,   0.      ],
        [  0.      ,   0.      ,   0.      ,   0.      ],
        [  0.      , -10.      ,   0.      ,   0.      ],
        [  0.      ,   0.03125 ,   0.03125 ,   0.      ],
        [  0.015625,   0.0625  ,   0.0625  ,   0.      ],
        [  0.03125 ,   0.125   ,   0.125   ,   0.      ],
        [  0.0625  ,   0.25    ,   0.25    ,   0.      ],
        [  0.125   ,   0.5     ,   0.5     ,   0.      ],
        [  0.25    ,   0.25    ,   1.      ,   0.      ],
        [  0.5     ,   0.125   ,   0.5     ,   0.      ],
        [  0.25    ,   0.0625  ,   0.25    ,   0.      ],
        [  0.125   ,   0.      ,   0.125   ,   0.      ]]])
```

In [ ]: