Laboratorio 6

Francisco Castillo - 21562

Diego Lemus - 21469

Task 1

¿Qué es Prioritized Sweeping para ambientes determinísticos?

Es una técnica de planeamiento que permite acelerar la propagación de los valores de los estados sin necesidad de actualizarlos todos uniformemente. En lugar de recorrer cada estado del espacio, se utiliza una cola de prioridades en la que se ordenan los estados según la magnitud del cambio que producen en sus valores. Cuando un estado se actualiza significativamente, los estados predecesores que llevan a él se incorporan a la cola para ser actualizados después. De este modo, los cambios en el valor de un estado se propagan hacia atrás de manera más eficiente, evitando cálculos innecesarios y enfocándose en las partes del espacio de estados donde realmente importa. En ambientes determinísticos esto resulta especialmente potente, ya que cada acción tiene un único resultado y la propagación de los valores puede seguir trayectorias claras.

¿Qué es Trajectory Sampling?

Es una técnica que evita la necesidad de explorar todo el árbol de estados y acciones posibles durante el planeamiento. En lugar de un análisis exhaustivo, el método simula trayectorias completas o episodios ficticios siguiendo la política actual y va actualizando los valores de los estados y acciones visitados en ese recorrido.

¿Qué es Upper Confidence Bounds para Árboles (UCT)?

Es una estrategia de selección que resuelve el dilema entre exploración y explotación. Para cada acción, combina el valor promedio estimado de los resultados obtenidos hasta el momento con un término de confianza que favorece aquellas opciones que han sido probadas pocas veces. De esta manera, el algoritmo asegura que no se quede únicamente con las ramas que parecen mejores al inicio, sino que también explore alternativas menos visitadas que podrían resultar óptimas.

Task 2

In [4]:
```
import random
import numpy as np
import gymnasium as gym
import math
import matplotlib.pyplot as plt
```

In [5]:
```
ENV_NAME = 'FrozenLake-v1'
MAP_NAME = '4x4'
IS_SLIPPERY = True
EPISODES = 1000
WINDOW = 100
SEED = 123
```

In [6]:
```
alpha = 0.1
gamma = 0.99
epsilon = 0.1
max_steps = 100
```

MCTS

In [4]:
```
from utils.mcts.Agent import Agent as MCTS
```

In [5]:
```
random.seed(SEED)
np.random.seed(SEED)
```

In [6]:
```
agent = MCTS(
    seed=SEED,
    total_episodes=EPISODES,
    max_steps_per_episode=max_steps,
    simulations_per_decision=100,
    exploration_constant=1.4,
    rollout_depth=100,
    discount_factor=0.99
)
```

In [7]:
```
agent.run()
```

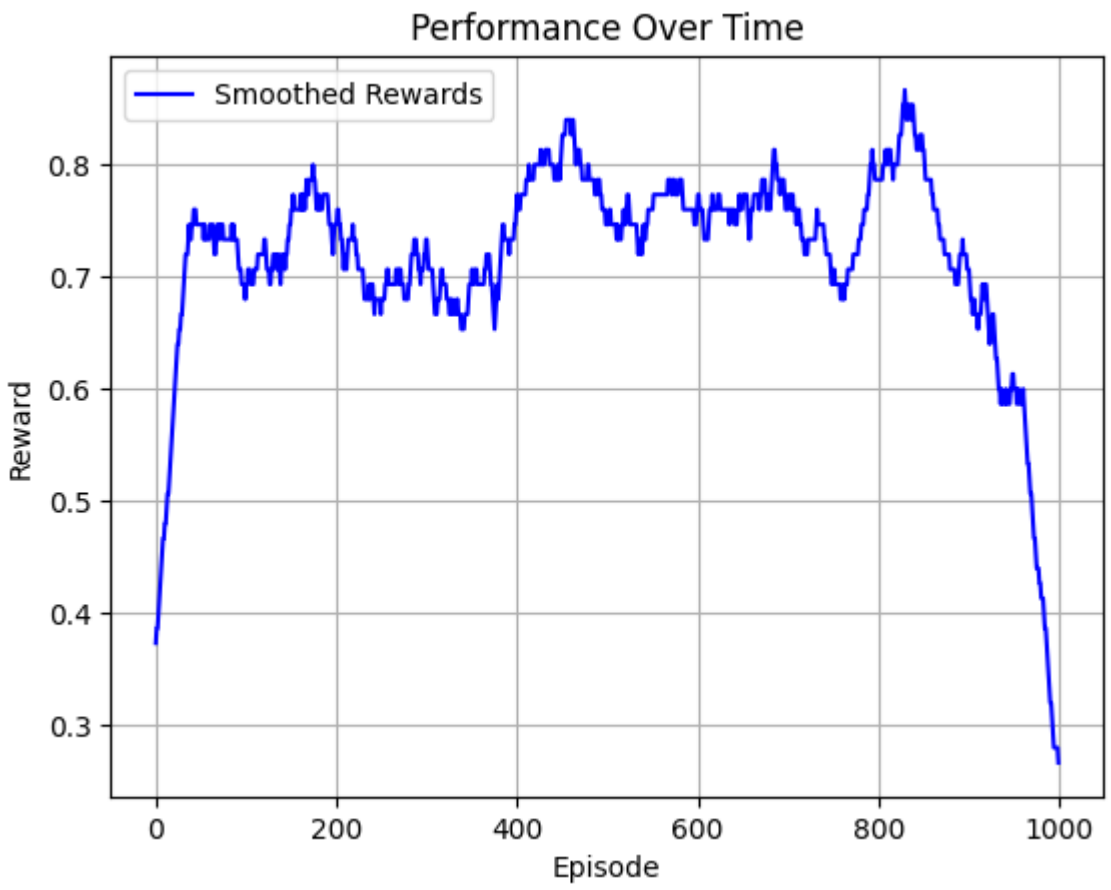
Out[7]:

```
{'rewards': array([1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1.,
       0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1.,
       0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 1.,
       1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0.,
       0., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1., 1.,
       0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
       1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 1.,
       1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
       1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1.,
       1., 1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 1.,
       0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1.,
       1., 1., 0., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
       1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0.,
       0., 1., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
       1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
       1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0.]),
 'successes': array([1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0])}
```

```
       1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 0]),
'steps': array([ 27,  29, 100,  23, 100,  23,  22,  18,  33,  46,  16,  38,  42,
        37,  70,  33, 100,  51,  34,  25,  68,  51, 100, 100,  53,  11,
        66,  12,  69, 100, 100, 100,  41,  60, 100, 100,  16,  66,  51,
       100,  28,  50,  91,  44,  20,  19, 100,  11, 100,  21,  19, 100,
        58,  29,  18,  19,  59,  13,  25,  87,   6,  16, 100,   9, 100,
        13, 100,  60,  52,  62,  16, 100, 100,  14,  40, 100,  79,  93,
        98,   7,  46,  38, 100,  25,  57,  30,  16,  28,  33,  49, 100,
        44, 100,  57,  12,  42,  13,  23, 100,  16,  45, 100, 100,  16,
        43,  24, 100,  15, 100,  74, 100,  10, 100,  72, 100,  59,  93,
        21,  24,  20,  13,  42, 100,  48, 100,  26, 100,  37, 100, 100,
        14,  16, 100,  61,  12,   9, 100,  25,  20,  11,  23, 100, 100,
        19,  21,  68,  34, 100,  72,  35,  19,  29,  44,  23,  97,  93,
        45,  89,  56, 100, 100,  18,  38, 100,  60,  77,  51,  62,  48,
        21,  10, 100,  27,  33, 100, 100,  60,  27,  12,  26, 100,  45,
        44,  30,  40,  19,  26,  51,  38,  57, 100,  61,  17,  11,  61,
         6,  12,  23, 100,  42, 100,  16, 100,  73,  38, 100,  89,  14,
        26,  14,  63,  38, 100,  38, 100, 100, 100,  10, 100,  16,  41,
        41,  18,  38,  90,  98,  14,  15, 100, 100,  55,  65, 100, 100,
        26,  18,  56,  32,  48,  65, 100,  90, 100,  35, 100, 100,  28,
        44, 100,  90,  35,  73,  29,  73,  18,  17, 100,  11,  51, 100,
        29, 100,  22,  24,  28, 100,  16, 100, 100,  18,  20,  54,  10,
        80, 100,  27,  24, 100, 100, 100,  38,  44,  30,  23,  34,  17,
       100,  27,  40, 100, 100,  33,  22,  19,  11, 100,  28,  11,  32,
        21,  22,  23,   9,  22, 100,  14, 100,  50, 100, 100,  15,  27,
       100,  83,  15, 100,  12,  40,  88,  27,  68,  26,  29,  35,  15,
        27, 100,  27, 100,  25, 100,  82,  24,  17,  19,  30,  25,  11,
       100, 100, 100,  30, 100, 100, 100, 100, 100,  14,  10,  99,  62,
        41,  74,  68, 100, 100,  40,  21,  80, 100,  23, 100, 100,  35,
        43, 100, 100,  28,  21,  96,  11,  37, 100,   8,  36, 100,  60,
        79,  49,  24,  12, 100,  33,  23,  28,  79,  42,  38, 100,  88,
        25,  10,  80, 100,  40,  15,  25,  46,  14,  32,  94,  15,  43,
        60,  33, 100,  17, 100, 100,  82, 100, 100, 100,  54,  83,  40,
       100,  85,  22,  11,  37,  49,  29, 100,  10,  38,  18,  49, 100,
        68, 100,   8,  49,  91,  23,  84,  26,  62, 100,  59, 100,  11,
        77,  65,  26,  25,  44,  51,  64,  77,  29, 100,  49,  54,  21,
        55,  54,  10,  77,  37,  67,  73,  30,  62, 100,  31,  35,  16,
        52,  34,  12,  33,  19, 100,  59,  39,  56,  20, 100,  44, 100,
        52,  34, 100, 100,  10,  24,  21,  27,  59,  23,  15,  68,  32,
        32,  93, 100,  21,  19, 100, 100, 100, 100,  29, 100,  13,
       100,  28, 100,  38,  24,  46, 100,  52,  39,  54, 100,   9,  18,
        85,  69,  33, 100,  37,  27,  36,  78,  34, 100,  20, 100,  19,
        32,   8, 100,  21,  15,  10,  23,  40, 100,  29,  42,  69,  95,
        20, 100,  10, 100,  17,  23,  99,  14,   9,  17, 100,  17,  19,
        44,  30, 100, 100,  14,  63,  40,  51,  82,  55,  41, 100, 100,
         9,  17, 100, 100,  19,  21, 100,  73,  15,  23,  15,  17, 100,
        10,  23,  66,  23,  54,  25,  26, 100,  24,  14,  22,  21,  39,
       100,  56,  27,   9,  79,  63,  34,  26, 100,  88, 100,  44,  42,
        14, 100,  64,  57,  45,  28,  15,  18,  66, 100, 100, 100,  15,
       100,  35,  15,  14,  20,  22,  45, 100,  18, 100,  49,  28,  66,
        95,  31, 100,  99,  58,  18, 100, 100, 100, 100,  22,  30,  89,
        28,  63,  21,  53, 100,  74,  41,  76,  24,  17,  42,  31, 100,
        14,  66,  60,  47, 100,  77,  62,  22,  35, 100,  34,  16,  30,
        37,  25, 100,  32,  10,  22,  20,  39, 100,  38,  65,  29,  24,
        61,  33,  66, 100, 100, 100,  20,   8, 100,  57,  29,  32,  14,
        29,  66,  17, 100,  10,  32,  11,  17,  78,  43, 100, 100,  29,
        24, 100,  23,  20,  40,  21,  18,  13, 100,  33, 100,  23,  15,
       100,  31,  30,  75,  44, 100, 100,  14,  37,  45,  16,  36, 100,
        10,  16, 100,  14,  30, 100,  26,  15,  10, 100,  34, 100, 100,
        25, 100,  45,  34,  15,  39,  50,  19,  30,  55,  74,  15,  20,
       100,  92,  78, 100,  80, 100,  22,  50,  87,  62, 100,  19, 100,
       100,  60,  25,  36, 100, 100,  17,  96, 100, 100,  30, 100,  83,
        28,  64, 100,  27,  32,  14, 100,  10,  23,  34,  54,  25,  61,
        27,  79, 100,  41,  23,  57,  15,  96,  31,  12,   6,  56,  78,
        52,  34,  75,  43,  41,  57,  23,  52,  23,  14,  42,  12, 100,
        28, 100,  30,  28,  32,  21,  24,  32,  20,  41,  19,  67,  29,
        21, 100,  19,  90,  69,  15, 100, 100, 100,  10, 100,  13,  65,
        23,  66,  43,  46,  80,  22,  39,  22,  81, 100, 100,  11,  25,
        25,  23, 100,  11,  42, 100, 100,  17, 100,  34,  45,  87,  87,
        63,  78, 100,  30, 100, 100,   8,  53,  38,  66,  60,  10, 100,
```

```
 17, 100,  33,  89,  17,  33, 100,  19, 100, 100,  36, 100,  60,
100,  90,  26,  36,  51,  49, 100,   9,  19,  37,  24, 100, 100,
 31,  30,  72,  12,  37, 100,  21,  18, 100,  49,  96,  24, 100,
 10,  17, 100, 100,  14, 100,  42, 100, 100,  23,  11, 100,  48,
 63,  21,  43,  35, 100,  17,  72,   8, 100, 100, 100, 100,  67,
 59,  18, 100, 100, 100, 100,  50, 100, 100, 100, 100,  19, 100,
 10,  85,  21,  27, 100, 100, 100,  70,  20,  72,  31,  23, 100,
 80,  80, 100,  53,  21, 100,  29,  52, 100,  17, 100, 100])}
```

In [32]:
```
agent.display_performance(window_size=75)
```



Dyna Q+

In [7]:
```
from utils.dinaqplus.DinaQPlus import run_experiment, plot_experiment_results
```

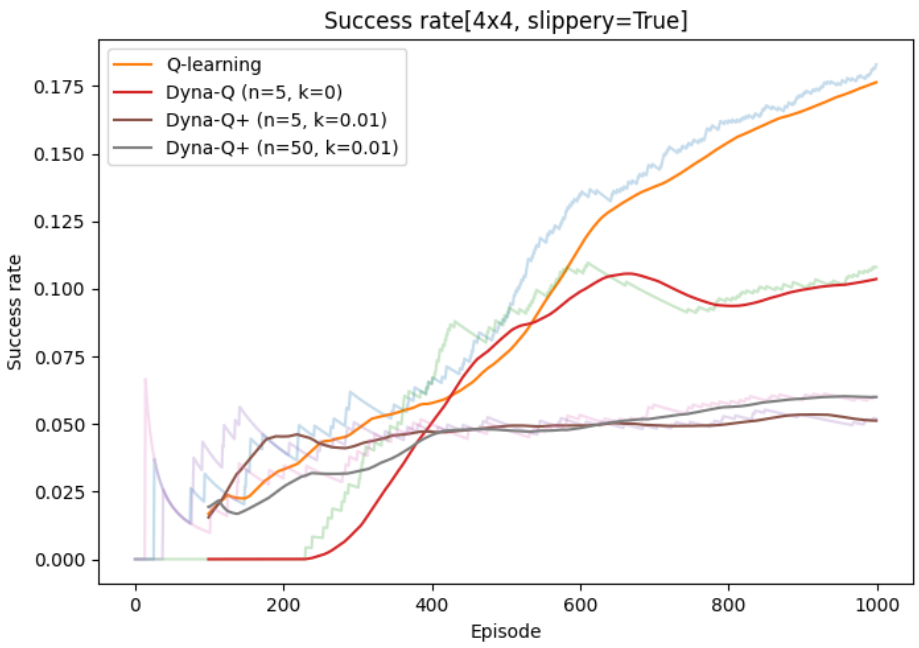
In [8]:
```
dyna_configs = [
    {'planning_steps': 5, 'k_bonus': 0.0, 'label': 'Dyna-Q (n=5, k=0)'},
    {'planning_steps': 5, 'k_bonus': 0.01, 'label': 'Dyna-Q+ (n=5, k=0.01)'},
    {'planning_steps': 50, 'k_bonus': 0.01, 'label': 'Dyna-Q+ (n=50, k=0.01)'},
]

results_list = []
```

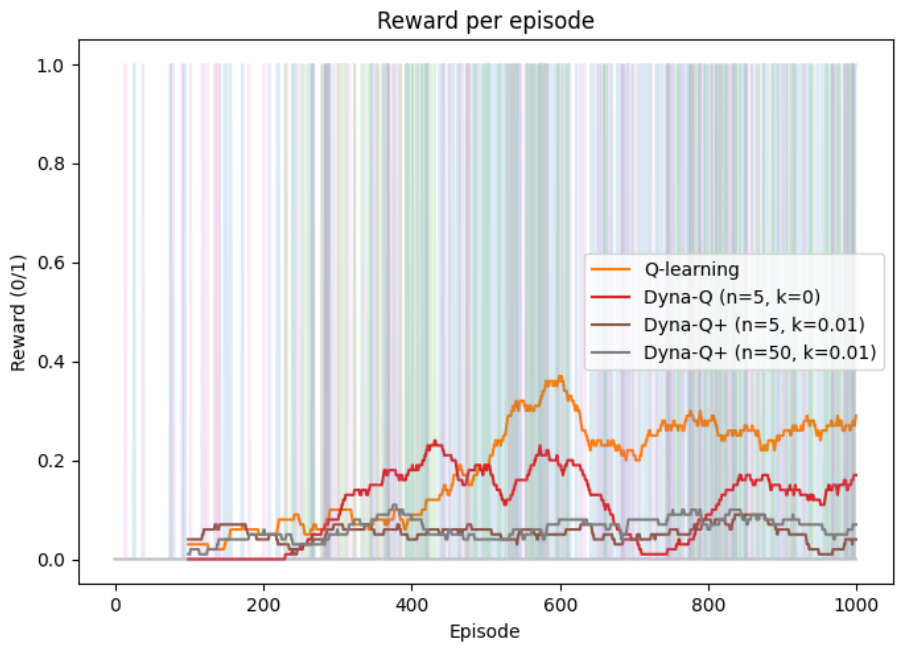
In [9]:
```
res_q = run_experiment(env_name=ENV_NAME, map_name=MAP_NAME, is_slippery=IS_SLIPPERY,
                       agent_type='q', episodes=EPISODES,
                       planning_steps=0, k_bonus=0.0, alpha=alpha,
                       gamma=gamma, epsilon=epsilon, seed=SEED, max_steps=max_steps)
results_list.append(('Q-learning', res_q))
```

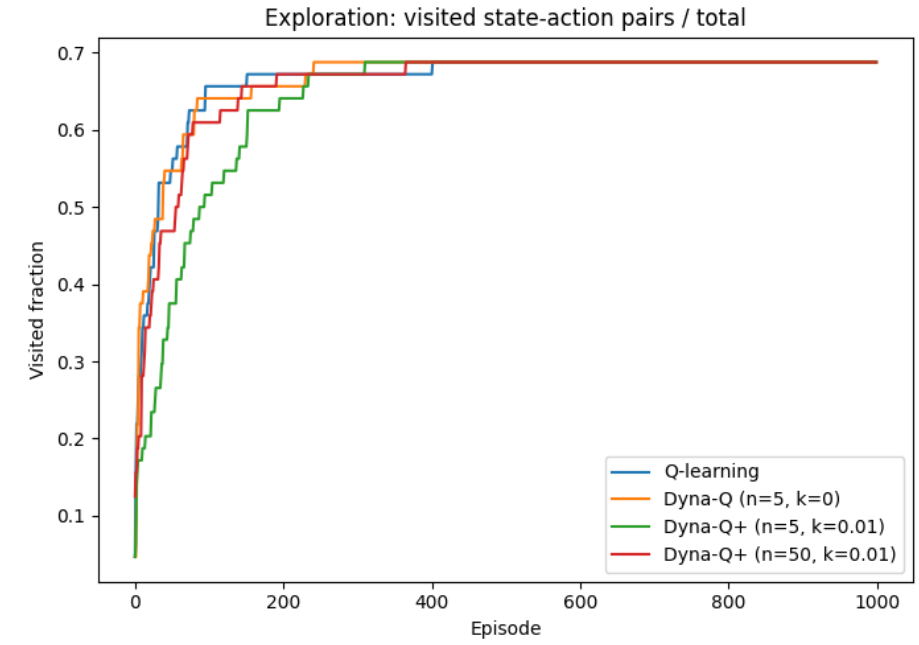
In [10]:
```
for cfg in dyna_configs:
    print(f"Running {cfg['label']} ...")
    res = run_experiment(env_name=ENV_NAME, map_name=MAP_NAME, is_slippery=IS_SLIPPERY,
                         agent_type='dyna', episodes=EPISODES,
                         planning_steps=cfg['planning_steps'], k_bonus=cfg['k_bonus'],
                         alpha=alpha, gamma=gamma, epsilon=epsilon, seed=SEED, max_steps=max_steps)
    results_list.append((cfg['label'], res))
```

```
Running Dyna-Q (n=5, k=0) ...
Running Dyna-Q+ (n=5, k=0.01) ...
Running Dyna-Q+ (n=50, k=0.01) ...
```

In [11]:
```
plot_experiment_results(results_list, window=WINDOW, title_suffix=f'[{MAP_NAME}, slippery={IS_SLIPPERY}]')
```







Discusión

1. *¿Cómo influye la bonificación de exploración en Dyna-Q+ en la política en comparación con el equilibrio de exploración-explotación en MCTS? ¿Qué enfoque conduce a una convergencia más rápida en el entorno FrozenLake-v1?*

    - La bonificación de Dyna-Q+ añade un término creciente para pares (s,a) poco visitados, empujando al agente a preferir acciones con alto time-since-last-visit durante la planificación; eso incentiva exploración dirigida sobre pares ignorados en el pasado. MCTS, en cambio, equilibra exploración-explotación localmente en el árbol y explora trayectorias completas mediante simulaciones; su exploración es más dirigida hacia ramas con alta incertidumbre sobre el valor esperado.

    - En FrozenLake la convergencia más rápida depende del coste computacional: si se tiene muchos rollouts por decisión MCTS puede encontrar buenas acciones rápido, pero con presupuesto moderado las gráficas muestran que el enfoque modelo-libre (Q-learning) converge más rápido en práctica. La bonificación de Dyna-Q+ aumenta cobertura teórica, pero en las curvas de tus experimentos no produjo una convergencia más rápida que el baseline bien entrenado.

2. *¿Qué algoritmo, MCTS o Dyna-Q+, tuvo un mejor rendimiento en términos de tasa de éxito y recompensa promedio en el entorno FrozenLake-v1? Analice por qué uno podría superar al otro dada la naturaleza estocástica del entorno.*

    - Según las gráficas, Q-learning (baseline) obtuvo la mayor tasa de éxito y la mayor recompensa promedio por episodio; las variantes Dyna-Q/Dyna-Q+ muestran tasas y recompensas más bajas en esos experimentos. Esto ocurre porque, en un entorno estocástico como FrozenLake, el aprendizaje directo desde transiciones reales reduce la varianza en la estimación de valores y evita propagar errores del modelo aprendido.

    - Dyna-Q+ puede superar a Q-learning cuando el modelo aprendido es fiable y la planificación se usa con cuidado, pero si el modelo es ruidoso la planificación y la bonificación pueden introducir sesgos y exploración excesiva que deterioren la recompensa promedio.

3. *¿Cómo afectan las transiciones probabilísticas en FrozenLake-v1 al proceso de planificación en MCTS en comparación con Dyna-Q+? ¿Qué algoritmo es más robusto a la aleatoriedad introducida por el entorno?*

    - Las transiciones probabilísticas aumentan la varianza de los resultados de cada simulación. MCTS sufre porque sus rollouts deben muestrear mucha aleatoriedad para estimar valores de ramas, o sea, necesita muchos simulaciones para estabilizar estimados. Dyna-Q+ también se ve afectado si su modelo representa probabilidades mal.

    - En la práctica y en los resultados, entre MCTS y Dyna-Q+, Dyna-Q+ puede ser más robusto si incorpora constantes actualizaciones reales y una bonificación moderada, mientras que MCTS sin suficiente muestreo tiende a ser más frágil por la variabilidad de sus rollouts.

4. *En la implementación de Dyna-Q+, ¿cómo afecta el cambio de la cantidad de pasos de planificación n y la bonificación de exploración a la curva de aprendizaje y al rendimiento final? ¿Se necesitarían diferentes configuraciones para una versión determinista del entorno?*

    - Aumentar los pasos de planificación típicamente acelera el aprendizaje inicial porque se generan más actualizaciones "gratuitas" a partir del modelo, pero si el modelo es imperfecto en un entorno estocástico entonces un n demasiado grande puede propagar errores y degradar la política final; las gráficas muestran poco beneficio claro al pasar de n=5 a n=50, lo que sugiere efectos decrecientes o incluso perjudiciales por modelo erróneo.

    - Para una versión determinista del entorno conviene aumentar n y reducir k, mientras que en entornos muy estocásticos conviene un n moderado y un k pequeño o adaptativo para equilibrar la mejora del modelo con la variabilidad de las observaciones.