# Universidad de Concepción

(503649)

Aprendizaje por refuerzo

Profesor Julio Godoy

---

## Tarea1

---

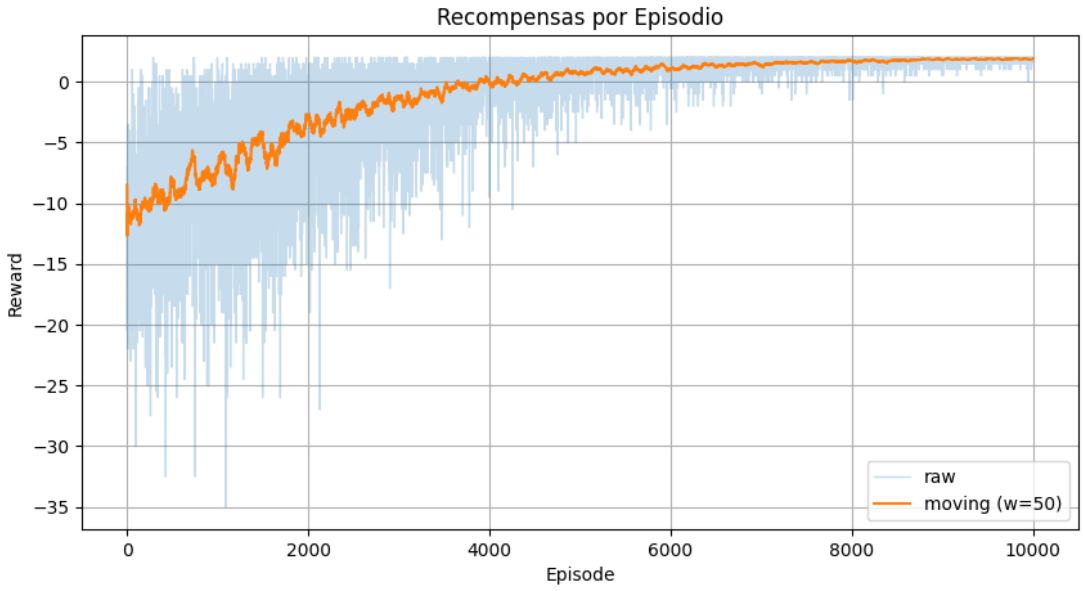
*Autores:*
Brendan Rubilar Vivanco - 2021750657

Concepción, 2025

# 1. Problema 1

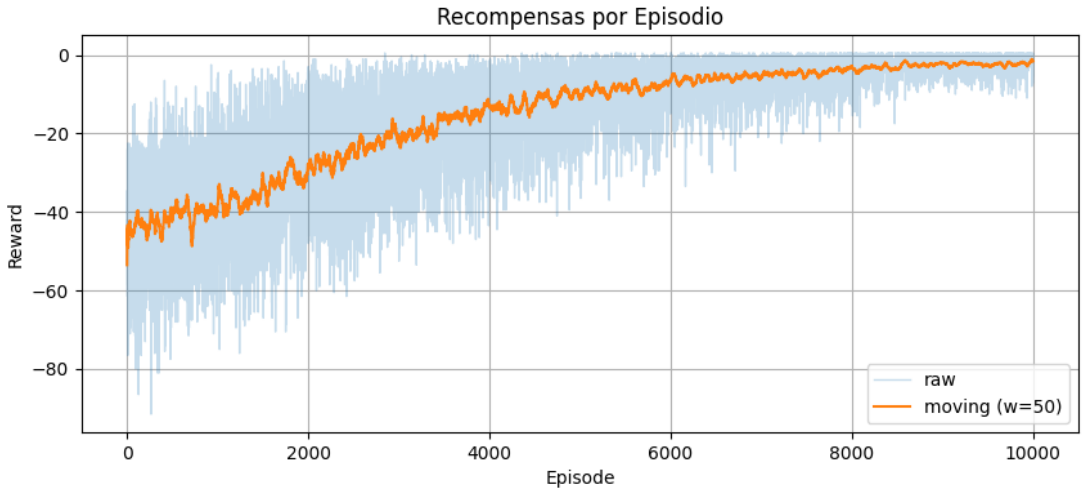
## 1.1. Muestre la curva de aprendizaje en cada caso

Entrenamiento en Mapa 1 Usando Q-Learning:




Entrenamiento en Mapa 1 Usando Sarsa:

Entrenamiento en Mapa 2 Usando Q-Learning:




Entrenamiento en Mapa 2 Usando Sarsa:


## 1.2. Modificación de hiperparámetros para dar solución al mapa 2

Se probaron diferentes combinaciones de parámetros, esto usando QLearning. Inicialmente se bajó el Gamma para ver cómo reaccionaba el agente, al tener un peor resultado, se bajó el Learning rate, esto mostró un mejor resultado, así finalmente se combinó un aumento de gamma (1)y disminución de learning rate (0.1), así el agente logra resolver el problema con ambos algoritmos pero con una recompensa de 0.5 al pasar por un obstáculo.
Para obtener la política óptima se puede utilizar un learning rate menor (0.05) con el mismo gamma anterior (1), así el agente esquiva la trampa y obtiene una recompensa de 2.
El agente logra resolver el problema al modificar estos parámetros, ya que al bajar la velocidad de aprendizaje y aumentar el gamma, el agente prioriza recompensas a largo plazo y aprende de manera más lenta. Esto evita que aprenda a tomar malos caminos.

# 2. Problema 2

## 2.1. Cambie el código de GridWorld.py para que la transición de acciones sea estocástica

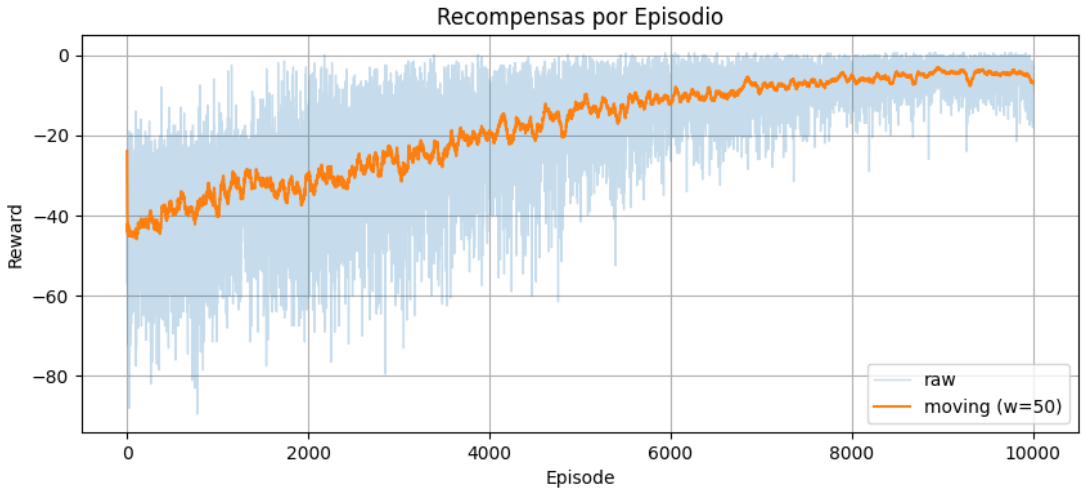
*Dado que no se pide quitar esta modificación en los problemas siguientes, se mantendrá este cambio para los problemas 3 y 4.*
Implementado en código de la siguiente forma:

```
r = np.random.rand()
if r < 0.7:
    actual_action = action
elif r < 0.85:
    actual_action = (action - 1) % 4  # izquierda relativa
else:
    actual_action = (action + 1) % 4  # derecha relativa
```

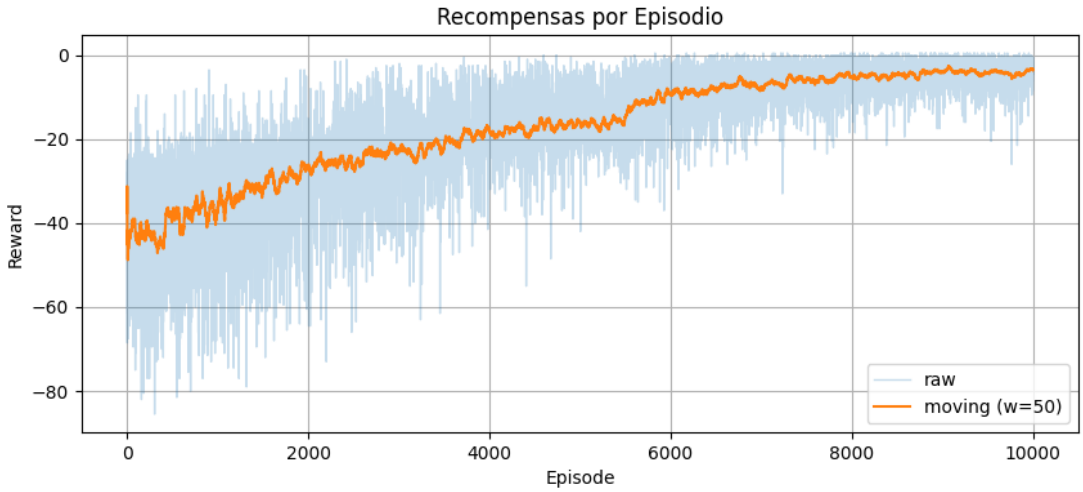
La variable r tiene un valor entre 0 y 1 (con distribución uniforme), así se puede usar como porcentaje para tomar una decisión con el movimiento. Un valor menor a 0.7 permitirá al agente tomar la acción que él desea; en los otros casos, se mueve a la izquierda de la dirección deseada o a la derecha.

## 2.2. Entrene nuevamente Q-learning y SARSA. ¿Se mantiene la relación entre el desempeño de ambos algoritmos en el ambiente estocástico? ¿por qué?

*Para este punto se volvieron a usar los hiperparametros con los que se encuentra la política optima*
El algoritmo Q-Learning estocastico tiene el siguiente resultado en su entrenamiento:



El algoritmo Sarsa estocastico tiene el siguiente resultado en su entrenamiento:




Ambos algoritmos permiten al agente encontrar la solución, pero Sarsa presenta menos ruido en el grafico de entrenamiento, esto se debe a que adopta decisiones más conservadoras. De todas formas, en ambos casos la estocasticidad del entorno incrementa la probabilidad de errores en comparación con el escenario determinista.

# 3. Problema 3

## 3.1. Convertir Q-learning en Double Q-learning

Para transformar el algoritmo Q-Learning en Double Q-Learning, se implementaron tres modificaciones:
**1. Inicialización de dos Tablas Q** En lugar de una única tabla de valor, se inicializan dos tablas Q, denominadas $Q_1$ y $Q_2$.

```
Q1 = np.zeros((STATES, ACTIONS))
Q2 = np.zeros((STATES, ACTIONS))
```

**2. Selección de Acciones** Para la política de explotación (cuando no se explora aleatoriamente), la acción a ejecutar en el estado actual $s_t$ se selecciona considerando la información de ambas tablas. Se elige la acción que maximiza la suma de los valores de $Q_1$ y $Q_2$:

$$a_t = \arg\max_a (Q_1(s_t, a) + Q_2(s_t, a))$$

```
action = np.argmax(Q1[state, :] + Q2[state, :])
```

Esta aproximación combina el conocimiento de ambas tablas para obtener una estimación de valor más robusta y estable.
**3. Separar la Selección de la evaluación** Esta es la modificación central del algoritmo. En cada paso de tiempo, se actualiza solo una de las dos tablas Q, elegida de forma aleatoria con un 50 % de probabilidad. El concepto clave es elegir qué acción parece ser la mejor en el futuro usando una tabla y luego con la otra se calcula cuánto vale realmente esa acción.

```
if np.random.uniform(0, 1) < 0.5:
    best_next_action = np.argmax(Q1[next_state, :])
    # Evalua el valor de esa accion con Q2
    target_value = Q2[next_state, best_next_action]
    Q1[state, action] = Q1[state, action] + LEARNING_RATE * \
        (reward + GAMMA * target_value - Q1[state, action])
```

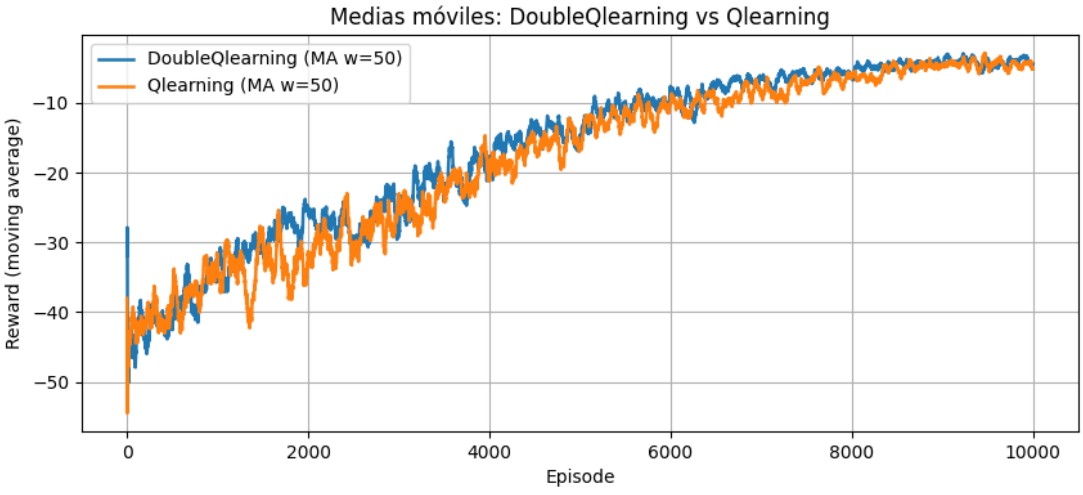
```
else:
    best_next_action = np.argmax(Q2[next_state, :])
    target_value = Q1[next_state, best_next_action]
    Q2[state, action] = Q2[state, action] + LEARNING_RATE * \
        (reward + GAMMA * target_value - Q2[state, action])
```

Este mecanismo de doble evaluación evita la sobreestimación de los valores Q que puede ocurrir en el algoritmo Q-Learning estándar.

### 3.2. Grafique la curva de aprendizaje en map2 y compárela con la de Q-learning

Resultado del entrenamiento de Double Q-Learning:


Se hizo una comparativa de las medias móviles de ambas implementaciones para ver las similitudes o diferencias con mayor facilidad:




Se aprecia una mejora en la implementación DoubleQlearning, aunque no tan alejada de la implementación simple, ambas mantienen un crecimiento similar en el gráfico.

### 3.3. Explique el por qué de la diferencia (o similitud) entre ambos resultados

Double Q-Learning reduce el sesgo de sobreestimación cuando el escenario es más estocástico, dado que estamos trabajando en un escenario estocástico, donde los obstáculos y muros siempre están fijos, no existe tanto sesgo para corregir. Además el entorno es pequeño, por lo que rápidamente ambos métodos convergen a la solución.

### 3.4. En caso de ser similares, ¿qué cambio del ambiente cree ud. que llevaría a resultados distintos entre Q-learning y Double Q-learning? Justifique

Si las trampas tuvieran un patrón de aparición, por ejemplo si fuera probable que aparezcan en algunas casillas de un sector en especifico en vez de que siempre estén fijas, añadiriamos un poco más de aleatoriedad al escenario, y en ese caso podría mostrar mayor ventaja la implementación de doble Q-learning.

## 4. Problema 4

### 4.1. Implemente Q($\lambda$) y SARSA($\lambda$) en tarea1.py)

Los algoritmos utilizando lambda implementan trazas de elegibilidad, asi se asignan créditos a acciones pasadas y así en lugar de actualizar solo el valor del último par estado-acción, una parte del error de predicción se propaga hacia atrás a los pares visitados recientemente.
**Sarsa($\lambda$):** En esta implementación la traza de elegibilidad simplemente se acumula para los estados visitados y decae en cada paso, distribuyendo el crédito de manera uniforme según las acciones que la política actual está tomando.

```
#Se elige la proxima accion segun la politica actual
if np.random.uniform(0, 1) < epsilon:
    next_action = env.action_space.sample()
else:
    next_action = np.argmax(Q[next_state, :])

#Calcular el error de predicci n usando la accion que realmente se tomara
delta = reward + GAMMA * Q[next_state, next_action] - Q[state, action]

#Se incrementa la traza para el par (s, a) actual
e[state, action] += 1.0

#Se actualizan todos los valores Q segun su traza de elegibilidad
Q += LEARNING_RATE * delta * e

#Luego deben decaer todas las trazas de elegibilidad
e *= GAMMA * lam

#Se Actualiza el estado y accion para el siguiente paso
state, action = next_state, next_action
```

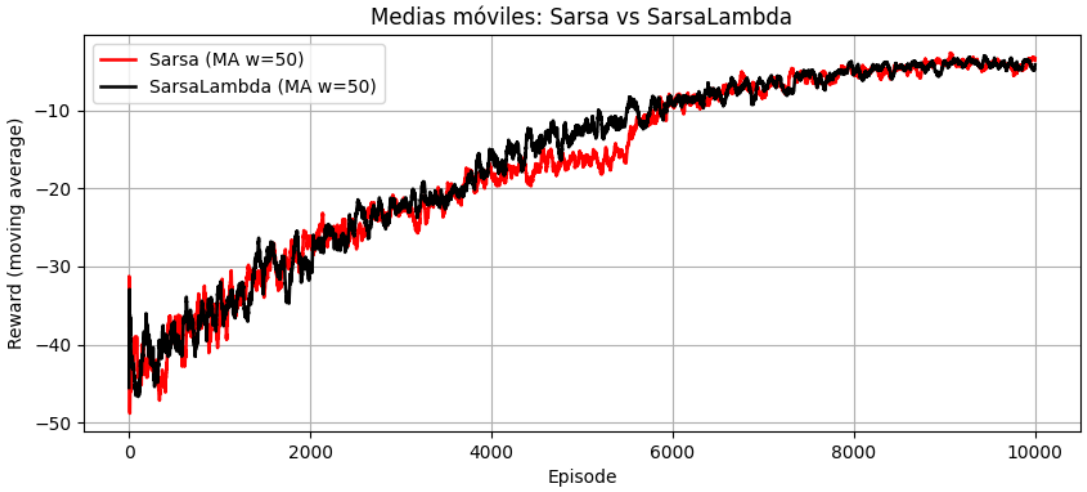
**Q($\lambda$) implementación basada en pseudocodigo de Watkins:**
La principal diferencia con Sarsa($\lambda$) es el como maneja las trazas de elegibilidad, la actualización del valor Q siempre utiliza la mejor acción posible en el siguiente estado, independientemente de la acción que se vaya a tomar.
Y la característica distintiva de la versión de Watkins es que si la acción que se elige para el siguiente paso no es la acción greedy, las trazas de elegibilidad se resetean a cero. Así se corta la propagación del crédito del pasado, ya que el agente ha dejado de seguir la política óptima que está intentando aprender.

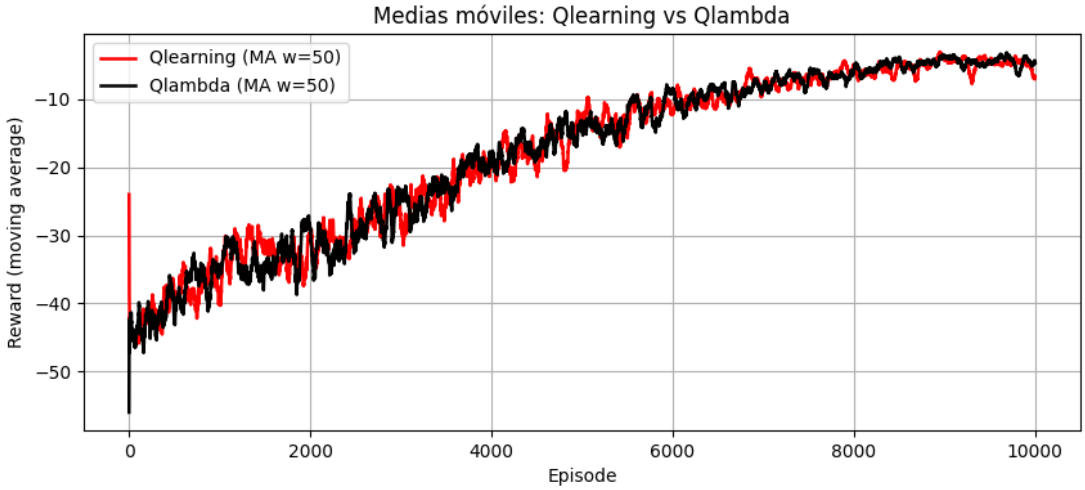
```
#Se toma la mejor accion posible en el siguiente estado
best_next = np.argmax(Q[next_state, :])

#Calcular el error de prediccion usando la mejor accion
```

```
delta = reward + GAMMA * Q[next_state, best_next] - Q[state, action]

#Se incrementa la traza para el par (s, a) actual
e[state, action] += 1.0

#Se actualizan todos los valores Q
Q += LEARNING_RATE * delta * e

#Se elige la accion que realmente se ejecutara
if np.random.uniform(0, 1) < epsilon:
    next_action = env.action_space.sample()
else:
    next_action = np.argmax(Q[next_state, :])

#Y aqui entra la idea de Watkins,
# Si la accion a tomar no es la greedy, se rompe la politica optima
# y se resetean las trazas para asi no propagar credito incorrecto.
if next_action != best_next:
    e[:] = 0.0
else:
    #Por otro lado, si se sigue la politica greedy, las trazas decaen normalmente
    e *= GAMMA * lam
```

### 4.2. Utilice los métodos con lambda de 0.5, y compare los resultados con los métodos evaluados en la pregunta 2, grafique la curva de aprendizaje en cada caso

Se pueden observar las comparartivas de medias moviles de las 4 implementaciones, las versiones lambda presentan una media movil más estable, pero muy similares.






## 4.3. Instrucciones de ejecución

**1. Configuración del Entorno:** Primero, es necesario crear y activar un entorno Conda con las dependencias específicas.

**Creación del entorno:** Se debe crear un entorno llamado `tarea1rl` con Python 3.6.

```
conda create --name tarea1rl python=3.6
```

**Activación del entorno:** Una vez creado, se debe activar el entorno.

```
conda activate tarea1rl
```

**Instalación de librerías:** Finalmente, instale las librerías requeridas con las versiones exactas.

```
pip install gym==0.21.0
pip install pyglet==1.5.11
```

**2. Ejecución de la Tarea** Con el entorno activado y las librerías instaladas, ejecute el script principal.

```
python tarea1.py
```

**3. Notas sobre la Configuración** Se pueden modificar ciertos parámetros en los archivos del proyecto:

- **Cambio de Mapa:** En el archivo GridWorld.py, se puede alternar entre los mapas disponibles (map1.txt y map2.txt).
- **Selección de Algoritmo:** En tarea1.py, se puede elegir el algoritmo a utilizar y ajustar sus hiperparámetros.