Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio IA

02 de Mayo de 2024

Continuous Mountain Car **3**
Hipótesis sobre la discretización e hiperparámetros en Continuous Mountain Car 3
Discretización de estados y acciones 3
Valores de hiperparámetros 4
gamma (Factor de descuento): 4
epsilon(Exploración-explotación): 4
alpha(Tasa de aprendizaje): 4
Resultados obtenidos 5
Discretización de estados y acciones 5
Conclusión sobre la discretización de estados y acciones en Q-learning 9
Reflexión general: 10
Valores de hiperparametros: 11
Conclusión sobre los valores de los hiperparametros 15
Three Musketeers **17**
Descripción General del Juego **17**
Reglas Principales 17
Resumen de la Tarea 17
Bitácora 17
Resultados **18**
Gráficas 18
Implementación de Funciones 18
Minimax 18
Expectimax 18
Funciones de Evaluación 19
Hipótesis 19
Interpretación de los Resultados 19
Conclusiones 20
Notas de Advertencia 20
Recomendación Final 20
Anexo **21**
Minimax-default vs Pete 21
Minimax-default vs Random 21
Minimax-mobility vs Pete 22
Minimax-mobility vs Random 22
Expectimax-default vs Pete 23
Expectimax-default vs Random 23
Expectimax-mobility vs Pete 24
Expectimax-mobility vs Random 24
Juegos 25

Continuous Mountain Car

Hipótesis sobre la discretización e hiperparámetros en Continuous Mountain Car

Discretización de estados y acciones

1. **Granularidad y Generalización**:
    - Una discretización con muchos estados y acciones permite capturar con mayor precisión los detalles del entorno, pero puede requerir más episodios para que el aprendizaje converja debido al mayor tamaño de la tabla de estados y sus acciones.
    - Una discretización con menos estados y acciones simplifica la representación del espacio de estados, lo que podría acelerar la convergencia y facilitar la generalización, pero con el riesgo de perder precisión en la política aprendida.
2. **Relevancia de disminuir los Estados y Acciones**:
    - Al achicar los estados y acciones, el sistema reduce su complejidad y, por ende, la cantidad de decisiones que el algoritmo debe tomar. Esto puede ser beneficioso en entornos donde las transiciones son suaves y no requieren una resolución tan alta.
    - En el caso del Continuous Mountain Car, los cambios en velocidad y posición entre pasos son graduales, lo que sugiere que una discretización fina podría ser innecesaria.
3. **Relación con el Rendimiento**:
    - Una discretización excesivamente fina puede generar una tabla de estados enorme, llevando a problemas de memoria y a que el algoritmo necesite muchos episodios para explorar adecuadamente.
    - Una discretización con menos estados y acciones permite un aprendizaje más rápido porque cada estado y acción tiene más probabilidades de ser visitados repetidamente, reforzando los valores aprendidos.
4. **Propuesta de Hipótesis**:
    - **Hipótesis**: Para el problema Continuous Mountain Car, una discretización más gruesa de los estados (e.g., 10-20 estados por variable) y de las acciones (e.g., 3-5 acciones) resulta más eficiente, ya que equilibra la capacidad de generalización con la velocidad de convergencia al simplificar la complejidad del entorno percibido por el algoritmo.

Valores de hiperparámetros

gamma (Factor de descuento):

- **Rango propuesto**: 0.8 a 1.
- **Justificación**:

- Un valor cercano a 1 prioriza las recompensas a largo plazo, crucial en problemas como el Continuous Mountain Car donde el objetivo requiere múltiples pasos para alcanzarse.
- Valores menores (gamma<0.8) tienden a priorizar recompensas inmediatas, lo que podría llevar a políticas subóptimas en este problema.

epsilon(Exploración-explotación):

- **Estrategia**: Inicial alto (epsilon≈1.0) para explorar el espacio de estados, disminuyendo gradualmente hasta 0.1 aproximadamente.
- **Justificación**:
  - Durante las primeras fases del entrenamiento, una alta exploración ayuda a descubrir regiones del espacio de estados que contienen políticas efectivas.
  - Reducir epsilon en fases posteriores asegura que el algoritmo aproveche el conocimiento adquirido para explotar políticas prometedoras.

alpha(Tasa de aprendizaje):

- **Estrategia**: Inicial alto (αinicial≈0.5−0.7) y decreciente para estabilizarse en un valor bajo (αfinal≈0.01).
- **Justificación**:
  - Al principio, un alpha alto permite grandes actualizaciones en la tabla estado/acción, acelerando el aprendizaje inicial.
  - Reducir alpha en etapas finales minimiza las oscilaciones, permitiendo una convergencia más precisa hacia la política óptima.

Resultados obtenidos

Discretización de estados y acciones

Para las siguientes gráficas corrimos el algoritmo con los siguientes valores:

- Alpha: 0.9
- Gamma: 0.9
- Epsilon: 0.9
- Episodios: 1000
- Duración aproximada de cada ejecución: 2 horas

Aclaraciones: Cuando se habla de estados o acciones de tamaño “x” se refiere a que tanto para las velocidades, valores en el eje de las x y acciones se discretizan los valores con un tamaño igual a x. En código python seria lo siguiente:

1. x_space = np.linspace(-1.2, 0.6, x)
2. vel_space = np.linspace(-0.07, 0.07, x)
3. actions = list(np.linspace(-1, 1, x))

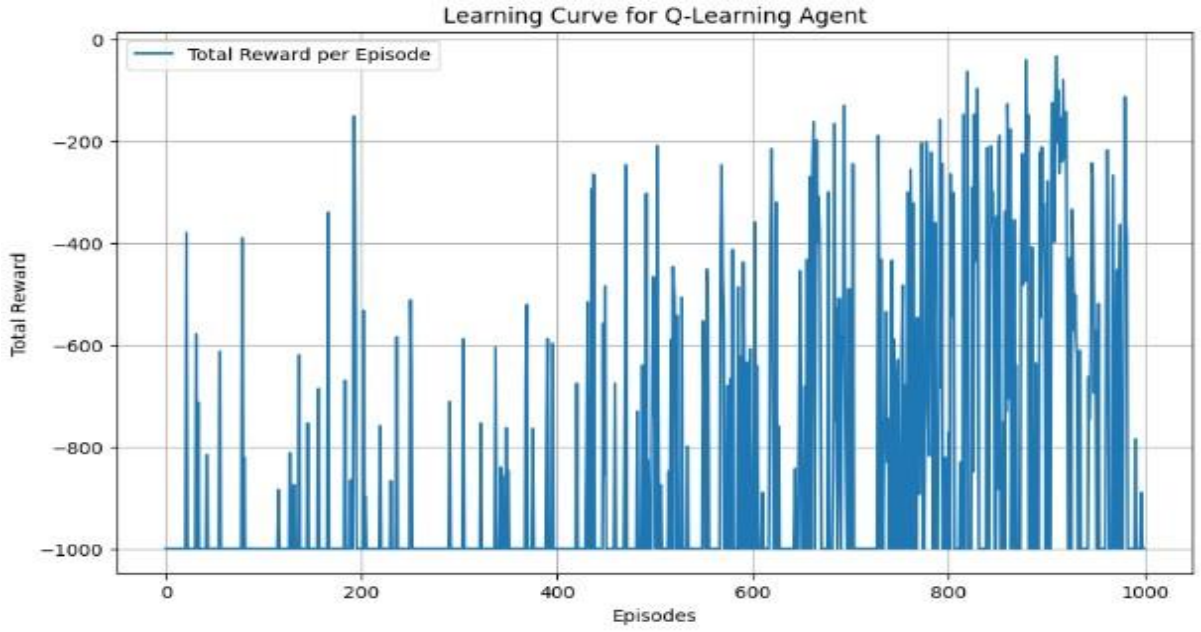
Como limite de acciones y estados pusimos como minimos y maximos:

1. Acciones: 3 - 20
2. Estados: 10 - 100

1 - Pocos estados y pocas acciones:

Para esta ejecución consideramos los siguientes valores:

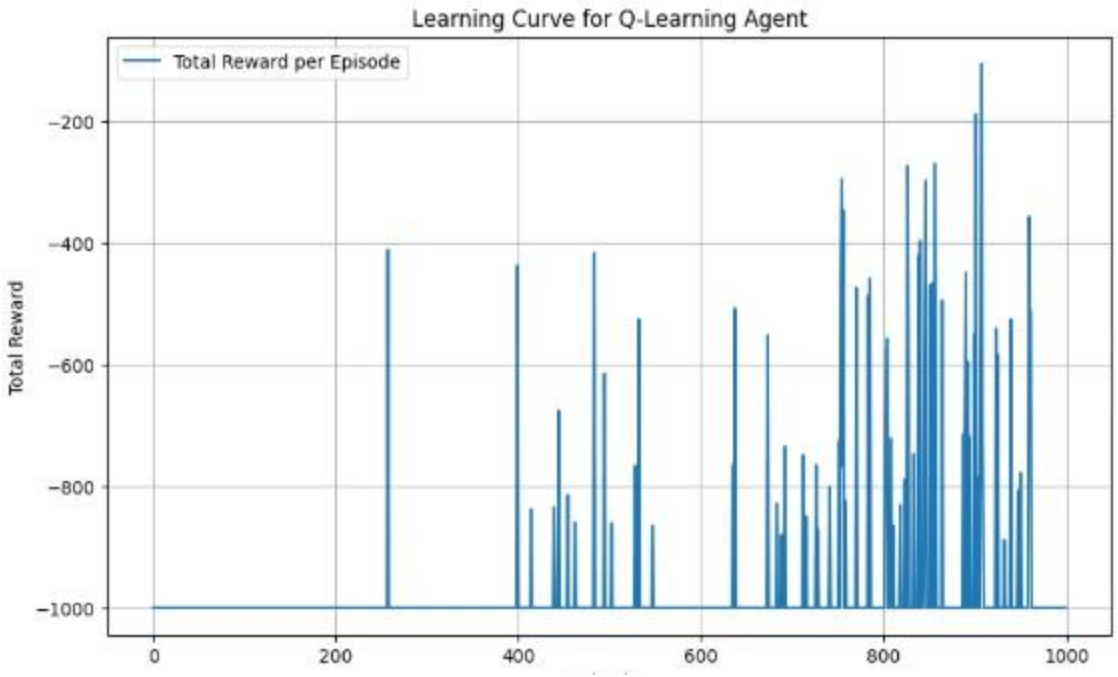
- Posibles espacios de las x: 10
- Posibles espacio de las velocidades: 10
- Posibles acciones: 3



2 - Pocos estados y muchas acciones:

Para esta ejecución consideramos los siguientes valores:

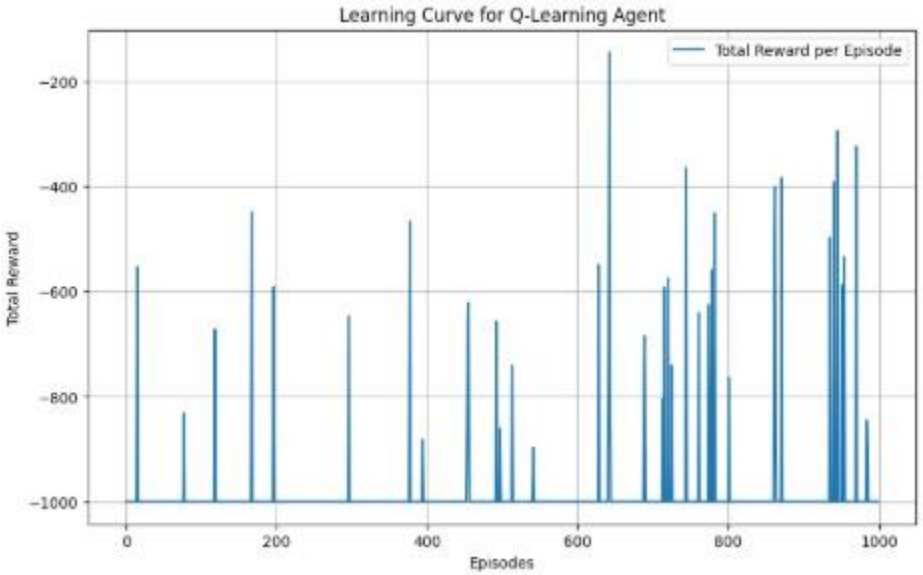
- Posibles espacios de las x: 10
- Posibles espacio de las velocidades: 10
- Posibles acciones: 20



3 - Varios estados y pocas acciones:

Para esta ejecución consideramos los siguientes valores:

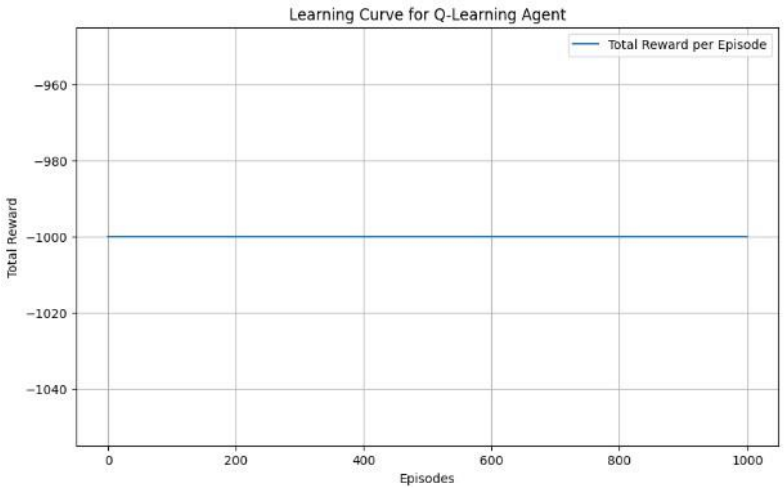
- Posibles espacios de las x: 25
- Posibles espacio de las velocidades: 25
- Posibles acciones: 5



4 - Muchos estados y varias acciones:

Para esta ejecución consideramos los siguientes valores:

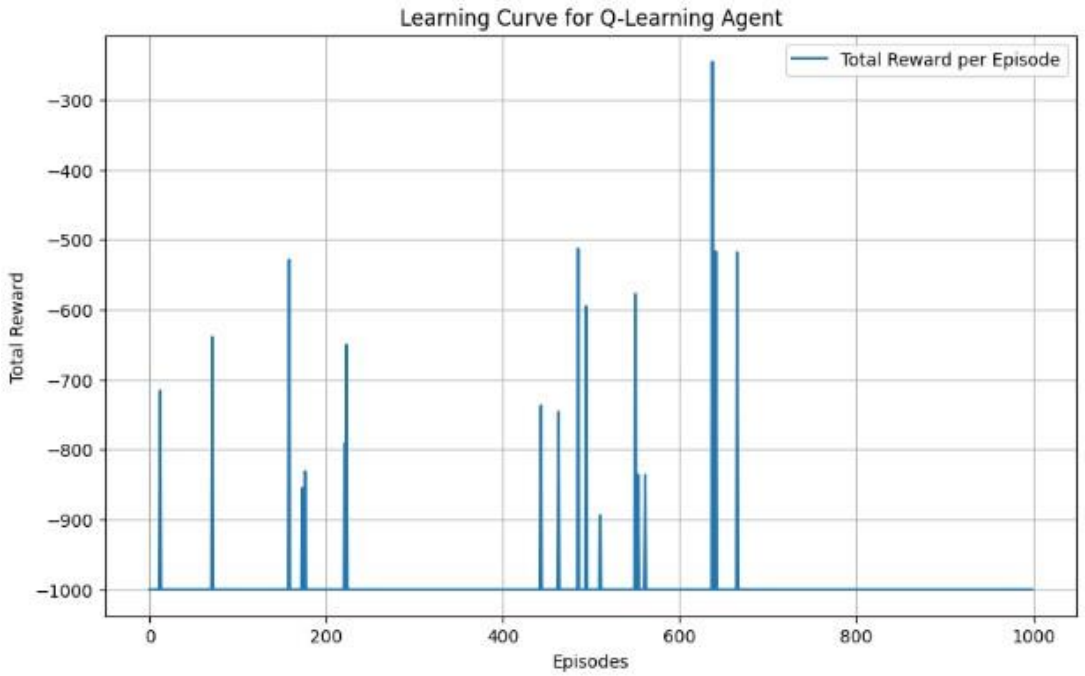
- Posibles espacios de las x: 50
- Posibles espacio de las velocidades: 50
- Posibles acciones: 10



5 - Muchos estados y pocas acciones:

Para esta ejecución consideramos los siguientes valores:

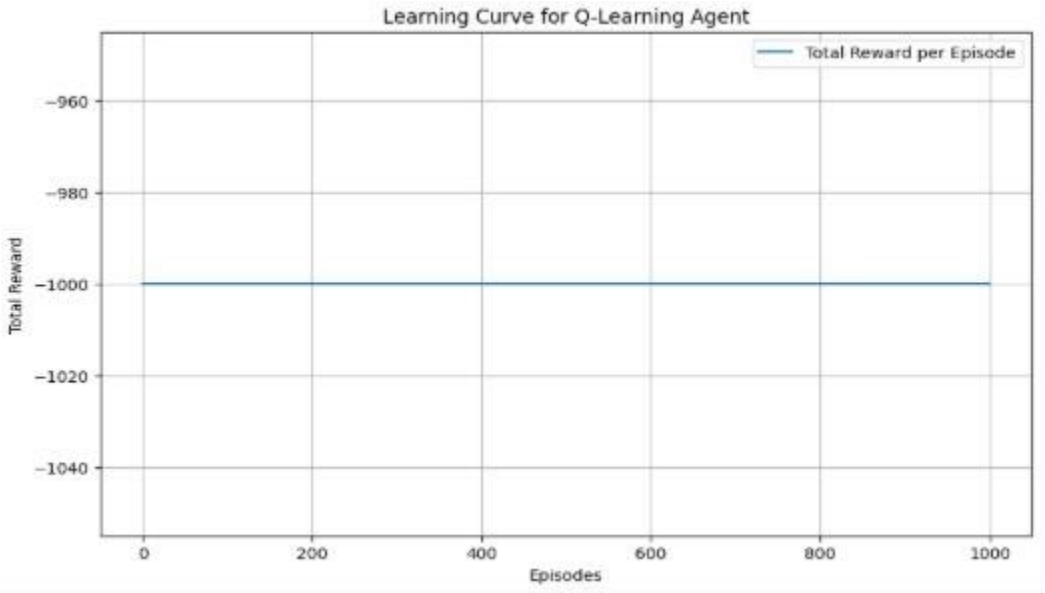
- Posibles espacios de las x: 50
- Posibles espacio de las velocidades: 50
- Posibles acciones: 3



6 - Límite de estados y de acciones:

Para esta ejecución consideramos los siguientes valores:

- Posibles espacios de las x: 100
- Posibles espacio de las velocidades: 100
- Posibles acciones: 20



Conclusión sobre la discretización de estados y acciones en Q-learning

Con base en las gráficas obtenidas para las diferentes configuraciones de discretización, podemos extraer las siguientes conclusiones:

1. **Pocos estados y pocas acciones (10 x 10 estados y 3 acciones):**

   La gráfica muestra un progreso lento, con un aprendizaje inicial limitado y oscilaciones significativas. Esto se debe más que nada a que probablemente son pocos estados y acciones y tenemos un epsilon muy alto, o sea explora de más.

2. **Pocos estados y muchas acciones (10 x 10 estados y 20 acciones):**

   Acá se observa un comportamiento errático, con picos de recompensa muy dispersos y poco consistentes. El agente parece tener dificultad para encontrar patrones significativos debido al alto número de acciones disponibles con una representación de estados limitada.

3. **Varios estados y pocas acciones (25 x 25 estados y 5 acciones):**

   El aprendizaje mejora considerablemente, con una tendencia más consistente hacia recompensas altas en episodios posteriores. Sin embargo, sigue habiendo cierta variabilidad en los resultados.

4. **Muchos estados y varias acciones (50 x 50 estados y 10 acciones):**

   Es uno de los peores, podemos llegar a entender que o la subida de estados o de acciones es lo que llevó a resultados tan malos.

5. **Muchos estados y pocas acciones (50 x 50 estados y 3 acciones):**

   Aunque hay cierto progreso, no es del todo bueno, sin embargo el hecho de que las acciones sean pocas y que los estados sigan siendo la misma cantidad que en el punto 4, nos hace llegar a concluir que hay que reducir la cantidad de acciones posibles por estado, y que la eficacia en si va a depender de la cantidad de episodios que dispongamos según los estados que tenga el algoritmo.

6. **Muchos estados y pocas acciones (100 x 100 estados y 20 acciones):**

   Solo con ver la gráfica se puede observar que la gran cantidad de estados y acciones son muy ineficaces con pocos episodios, además con estos grandes números es muy difícil llegar a algo hablando de costos computacionales.

Reflexión general:

- **Impacto de la discretización baja:** Configuraciones con pocos estados o acciones resultan en un aprendizaje lento e ineficiente debido a la falta de información suficiente para modelar el entorno adecuadamente. Sin embargo en problemas simples como estos, al entrenar con pocos episodios(1000 en nuestro caso), resultaron totalmente eficientes, llegando a obtener buenos resultados en test. En problemas donde los estados tienen variaciones más finas o múltiples dimensiones críticas, una discretización con pocos estados puede simplificar demasiado el espacio de búsqueda. Esto hace que el agente no sea capaz de distinguir entre situaciones diferentes que requieren estrategias específicas, el cual claramente no es el caso del Continuous Mountain Car donde solo se requiere empujar el coche en la dirección correcta.
- **Impacto de la discretización alta:** Configuraciones con muchos estados y acciones tienden a ofrecer mejores resultados, pero a costa de un mayor tiempo de cómputo y memoria requerida, mas de 50 estados y 10 acciones nos parece que es una cantidad muy grande a considerar para cualquier

cantidad de episodios y no consideramos que sea apropiado para este ambiente.

- **Balance ideal:** La configuración de discretización debe balancear los estados y acciones además de la complejidad computacional. Configuraciones intermedias como 25 x 25 estados y 5 acciones ofrecen un compromiso razonable entre rendimiento y costo computacional, sin embargo en nuestro caso preferimos quedarnos con 10 x 10 estados y 3 acciones ya que brindó muy buenos resultados, teniendo en cuenta que el epsilon era fijo de 0.9, o

  sea exploraba mucho, lo cual es totalmente negativo para pocos estados y acciones, pero aun así obtuvo los mejores resultados.

Valores de hiperparametros:

Para las siguientes gráficas corrimos el algoritmo con los siguientes valores:

- Episodios: 1000
- Tamaño de estados: 10 (sacado del análisis previo de discretización)
- Tamaño de acciones: 3 (sacado del análisis previo de discretización)
- Duración aproximada de cada ejecución: 2 horas

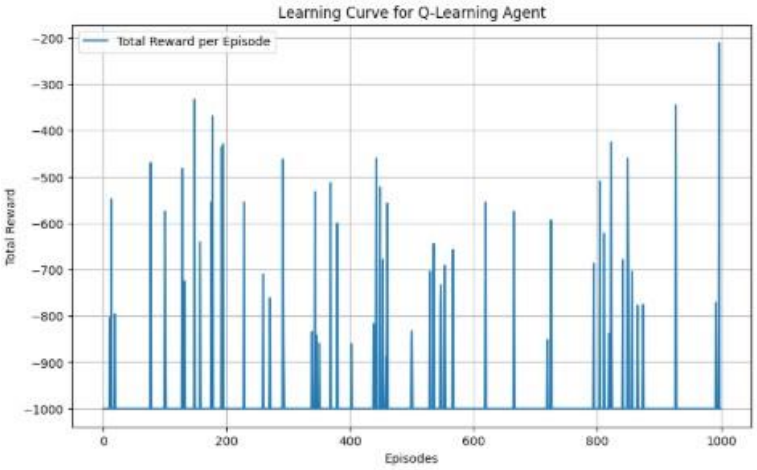
Aclaraciones: Cuando se habla de estados o acciones de tamaño “x” se refiere a que tanto para las velocidades, valores en el eje de las x y acciones se discretizan los valores con un tamaño igual a x. En código python seria lo siguiente:

4. x_space = np.linspace(-1.2, 0.6, x)
5. vel_space = np.linspace(-0.07, 0.07, x)
6. actions = list(np.linspace(-1, 1, x))

1 - Primera prueba

Para esta ejecución consideramos los siguientes valores:

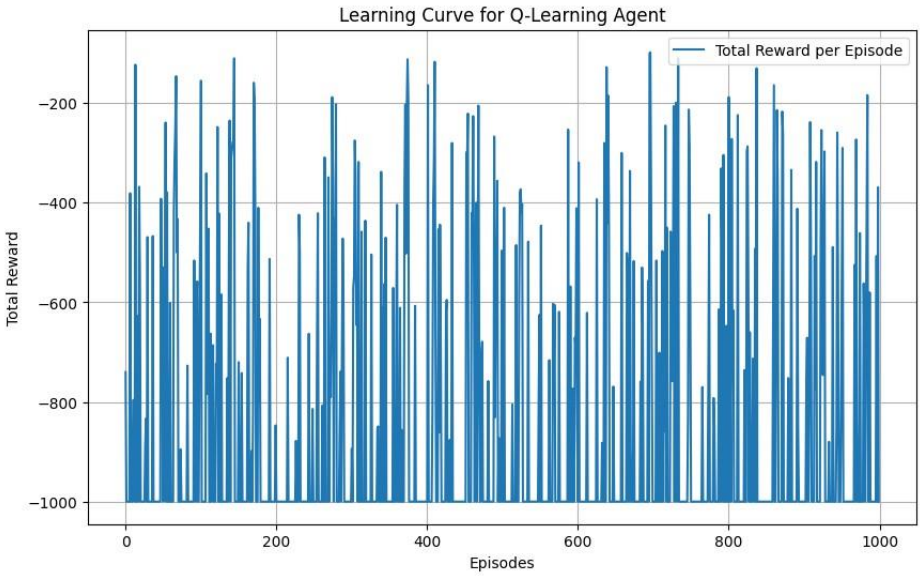
- Epsilon: 0.9 fijo
- Gamma: 0.9 fijo ● Alpha: 0.9 fijo



2 - Segunda prueba

Para esta ejecución consideramos los siguientes valores:

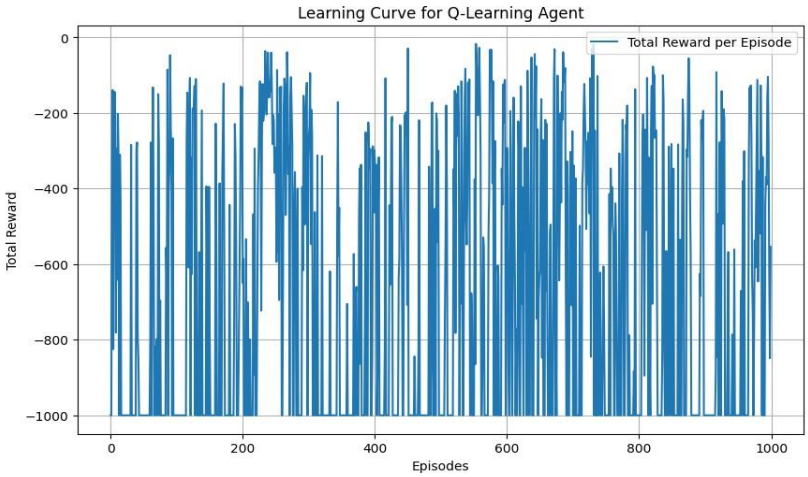
- Epsilon: 0.5 fijo
- Gamma: 0.9 fijo ● Alpha: 0.9 fijo



3 - Tercera prueba

Para esta ejecución consideramos los siguientes valores:

- Epsilon: 0.2 fijo
- Gamma: 0.9 fijo
- Alpha: 0.9 fijo


Learning Curve for Q-Learning Agent
Total Reward per Episode
Total Reward
Episodes
0
−200
−400
−600
−800
−1000
0
200
400
600
800
1000

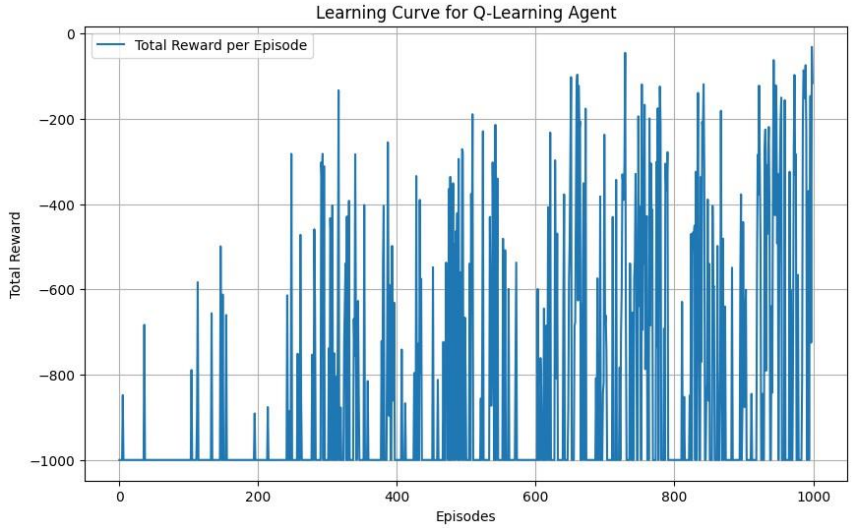
-

ejecución consideramos los siguientes valores:

4 Cuarta prueba

Para esta

- Epsilon: Comienza en 0.9 y tiene decaimiento lineal hasta 0.2
- Gamma: 0.9 fijo ● Alpha: 0.9 fijo



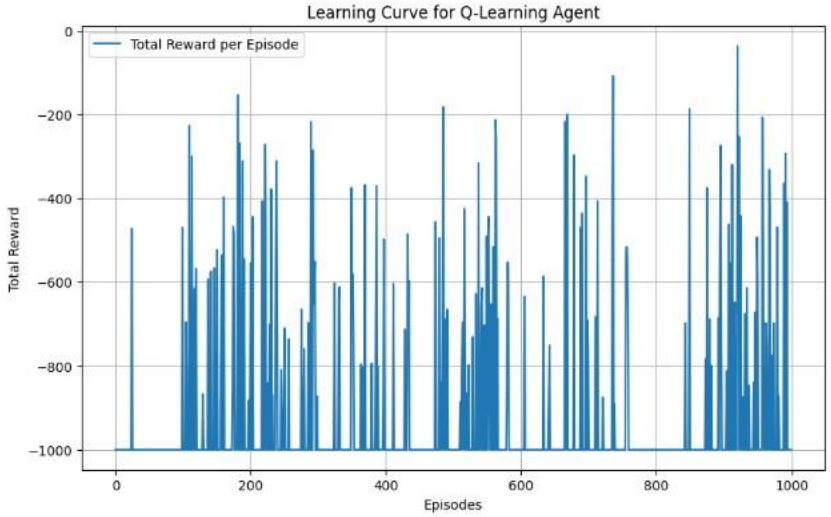
5 - Quinta prueba

Para esta ejecución consideramos los siguientes valores:

- Epsilon: Comienza en 0.9 y tiene decaimiento lineal hasta 0.2
- Gamma: 0.9 fijo
- Alpha: 0.2 fijo

-

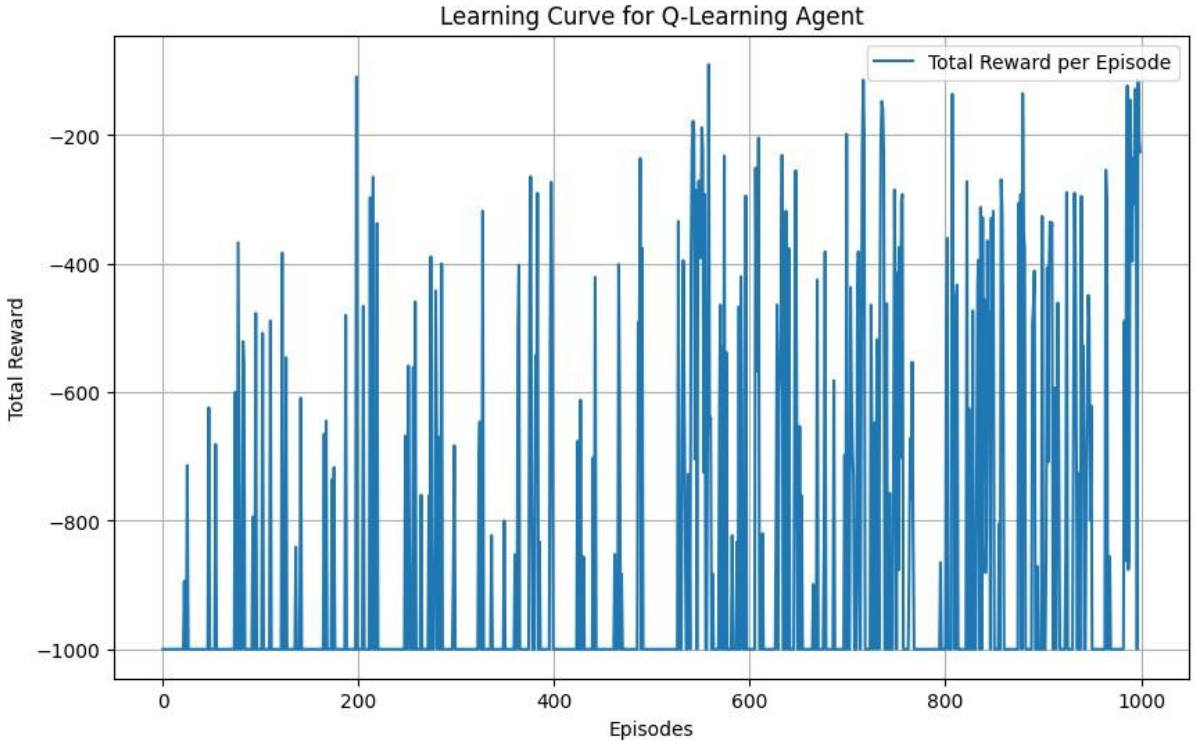
ejecución consideramos los siguientes valores:



6 Sexta prueba

Para esta

- Epsilon: Comienza en 0.9 y tiene decaimiento lineal hasta 0.2
- Gamma: 0.9 fijo
- Alpha: Comienza en 0.9 y tiene decaimiento lineal hasta 0.7



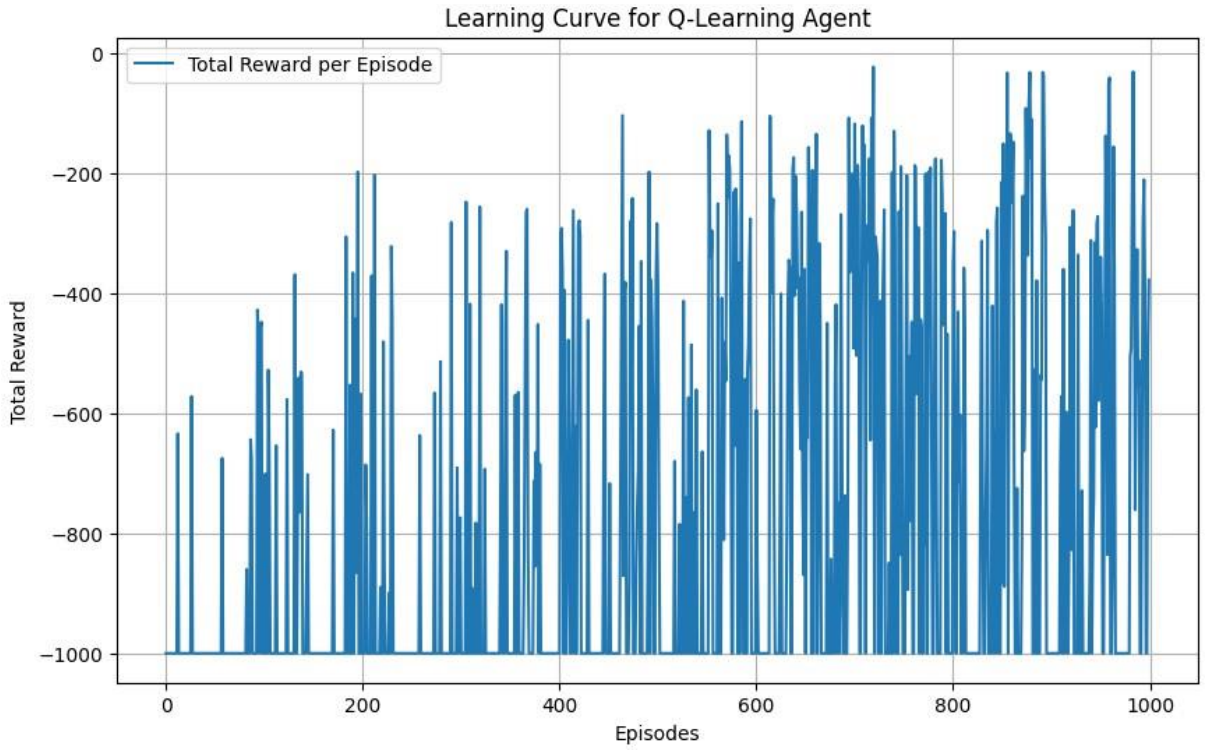
-

ejecución consideramos los siguientes valores:

6 Septima prueba

Para esta

- Epsilon: Comienza en 0.9 y tiene decaimiento exponencial hasta 0.2
- Gamma: 0.9 fijo
- Alpha: Comienza en 0.9 y tiene decaimiento lineal hasta 0.7



Conclusión sobre los valores de los hiperparametros

Tras los experimentos realizados en el problema Continuous Mountain Car con Q-learning, se pueden extraer importantes conclusiones que ayudan a entender mejor el impacto de los hiperparámetros en el rendimiento del modelo:

1. **Alpha:**

-

ejecución consideramos los siguientes valores:

- Un alpha alto (fijo en 0.9) proporcionó los mejores resultados, asegurando actualizaciones rápidas y efectivas de los valores de la tabla estado/acción. Esto resultó en un aprendizaje más eficiente y un mejor desempeño en el testeo.

- Aunque variar alpha entre 0.9 y 0.6 también fue efectivo, la velocidad de testeo fue significativamente más lenta, lo que sugiere que un alpha fijo alto simplifica el ajuste y acelera la convergencia hacia una política efectiva.
- Un alpha bajo dio lugar a un mal desempeño en el testeo, lo que indica que las oscilaciones iniciales son necesarias para explorar eficientemente el espacio de estados y acciones, se necesitaran muchos episodios para lograr que converja con un alpha bajo.

2. **Exploración-explotación (epsilon):**
   - Confirmando la hipótesis inicial, mantener un epsilon alto al inicio y reducirlo gradualmente mediante un decaimiento, ya sea lineal o exponencial, fue clave para lograr buenos resultados.
   - El decaimiento lineal mostró una ventaja significativa en el testeo, al completar el entrenamiento en menos tiempo y con mayor éxito, en comparación con el decaimiento exponencial. Esto sugiere que la transición gradual y constante hacia la explotación es más efectiva para este entorno.
3. **Gamma:**
   - Fijar gamma en 0.9 fue suficiente para capturar las recompensas a largo plazo sin comprometer el desempeño. Los cambios entre valores altos no mostraron un impacto significativo, lo que valida que priorizar objetivos a largo plazo es crucial en este problema.
4. **Balance entre Aprendizaje y Testeo:**
   - Aunque variar alpha y usar decaimiento exponencial para epsilon permitió buenos resultados, estas estrategias ralentizaron el proceso, especialmente en la fase de testeo. Esto nos hace ver que, en escenarios prácticos, priorizar configuraciones simples y eficientes como un alpha fijo alto y un decaimiento lineal puede ser más beneficioso.

Three Musketeers

Descripción General del Juego

El juego de Los Tres Mosqueteros involucra dos bandos: los Mosqueteros (M) y los Guardias (G). El objetivo principal es evaluar la interacción entre ambos bandos, donde los Mosqueteros intentan evitar ser alineados mientras los Guardias buscan dicha alineación.

Reglas Principales

- Si es el turno de los Mosqueteros y no pueden moverse, ganan los Mosqueteros.
- Si es el turno de los Guardias y no pueden moverse, ganan los Mosqueteros.
- Si los tres Mosqueteros quedan alineados (en una misma fila o columna) después de cualquier movimiento, ganan los Guardias inmediatamente.

Resumen de la Tarea

Objetivo: Evaluar el desempeño del `StrategicAgent` utilizando las estrategias (`minimax`, `expectimax`) con heurísticas seleccionadas (`default`, `mobility`). Los experimentos compararon el desempeño del agente contra dos oponentes: `CaptainPete` y `RandomAgent`, ejecutando 1000 partidas por configuración. Esto permitió analizar cómo las combinaciones de estrategias y heurísticas impactan el rendimiento en el simulador `ThreeMusketeersEnv`.

Bitácora

1. **Configuración Inicial:**
   - **Heurísticas:** `default`, `mobility`.
   - **Estrategias:** `minimax`, `expectimax`.
   - **Oponentes:**
     - `CaptainPete`: Un oponente estratégico preconfigurado.
     - `RandomAgent`: Un oponente que realiza movimientos aleatorios.
   - **Simulador:** `ThreeMusketeersEnv`.
2. **Parámetros:**
   - Profundidad fija en los algoritmos `minimax` y `expectimax`: 3.
   - Número de partidas por configuración: 1000.
   - Jugador 1 (`Musketeers`): `StrategicAgent`.
   - Jugador 2 (`Guards`): `CaptainPete` o `RandomAgent`.
3. **Tiempo de Ejecución:**
   - El tiempo promedio por configuración varió entre **30 segundos y 6 minutos**, dependiendo de la estrategia utilizada.
4. **Resultados Obtenidos:**
   - Los resultados se registraron en términos de número de victorias para cada jugador, incluyendo porcentajes para facilitar la interpretación.

Resultados

Estrategia-Heurística	Oponente	Vict. Musketeers	% Vict. Musketeers	Vict. Guards	% Vict. Guards
Minimax-Default	Pete	866	86.6%	134	13.4%

Minimax-Default	Random	736	73.6%	264	26.4%
Minimax-Mobility	Pete	159	15.9%	841	84.1%
Minimax-Mobility	Random	158	15.8%	842	84.2%
Expectimax-Default	Pete	891	89.1%	109	10.9%
Expectimax-Default	Random	821	82.1%	179	17.9%
Expectimax-Mobility	Pete	349	34.9%	651	65.1%
Expectimax-Mobility	Random	334	33.4%	666	66.6%

Gráficas

Las siguientes gráficas muestran el número de victorias para los jugadores en cada configuración:

1. **Minimax-default vs Pete**
2. **Minimax-default vs Random**
3. **Minimax-mobility vs Pete**
4. **Minimax-mobility vs Random**
5. **Expectimax-default vs Pete**
6. **Expectimax-default vs Random**
7. **Expectimax-mobility vs Pete**
8. **Expectimax-mobility vs Random**

Implementación de Funciones

Minimax

El algoritmo `minimax` fue implementado con poda alfa-beta, reduciendo el espacio de búsqueda al descartar ramas que no son óptimas. Evalúa movimientos desde la perspectiva del jugador actual, asumiendo que el oponente siempre toma las mejores decisiones posibles.

Expectimax

El algoritmo `expectimax` amplía `minimax` al calcular movimientos del oponente como un promedio ponderado en lugar de asumir las mejores decisiones. Es más adecuado para situaciones donde los movimientos del oponente no son completamente deterministas.

Funciones de Evaluación

1. **Default:** Penaliza el número de guards restantes y verifica si los Musketeers están alineados.

2. **Mobility:** Calcula la diferencia en la cantidad de movimientos disponibles entre el jugador y el oponente.

Hipótesis

1. **Default Heuristic:** Las Musketeers obtendrán menos victorias contra `CaptainPete`, dado su enfoque limitado en alineación y reducción de guards.
2. **Mobility Heuristic:** La heurística `mobility` debería superar a `default`, ya que prioriza movimientos estratégicos que restringen al oponente.

Interpretación de los Resultados

1. **Minimax-Default:**
   - Esta configuración tuvo un **alto rendimiento** contra ambos oponentes.
   - Contra `Pete` logró **86.6%** de victorias, demostrando que la heurística `default` combinada con la búsqueda exhaustiva de Minimax es robusta.
   - Contra `RandomAgent`, obtuvo **73.6%** de victorias, ligeramente menos debido a la naturaleza impredecible del oponente aleatorio.
2. **Minimax-Mobility:**
   - La heurística `mobility` tuvo **un bajo rendimiento**. Contra `Pete`, alcanzó apenas **15.9%** de victorias, y similar desempeño contra `RandomAgent` (**15.8%**).
   - Esto sugiere que priorizar movimientos disponibles no necesariamente genera ventaja frente a un oponente que sigue estrategias simples pero efectivas.
3. **Expectimax-Default:**
   - Esta combinación fue **la más exitosa**. Contra `Pete`, obtuvo un **89.1%** de victorias.
   - Contra `RandomAgent`, también tuvo un desempeño sólido con **82.1%**.
   - El uso del promedio ponderado en Expectimax parece optimizar la toma de decisiones bajo incertidumbre, y la heurística `default` sigue siendo efectiva.
4. **Expectimax-Mobility:**
   - Aunque mejor que `Minimax-Mobility`, esta combinación aún fue insuficiente, con **34.9%** de victorias contra `Pete` y **33.4%** contra `RandomAgent`.
   - La heurística `mobility` prioriza movimientos disponibles, pero no contempla el control efectivo del tablero.

Conclusiones

1. **Estrategia Ganadora:** La combinación `Expectimax-Default` fue **la más efectiva**, superando significativamente a `Pete` y `RandomAgent`.
2. **Heurística Mobility:** Mostró un desempeño **muy bajo**, destacando que la movilidad no es suficiente para dominar el juego.
3. **Robustez de Expectimax:** La capacidad de manejar incertidumbre en los movimientos del oponente hace que Expectimax sea más eficiente que Minimax en este contexto.

Notas de Advertencia

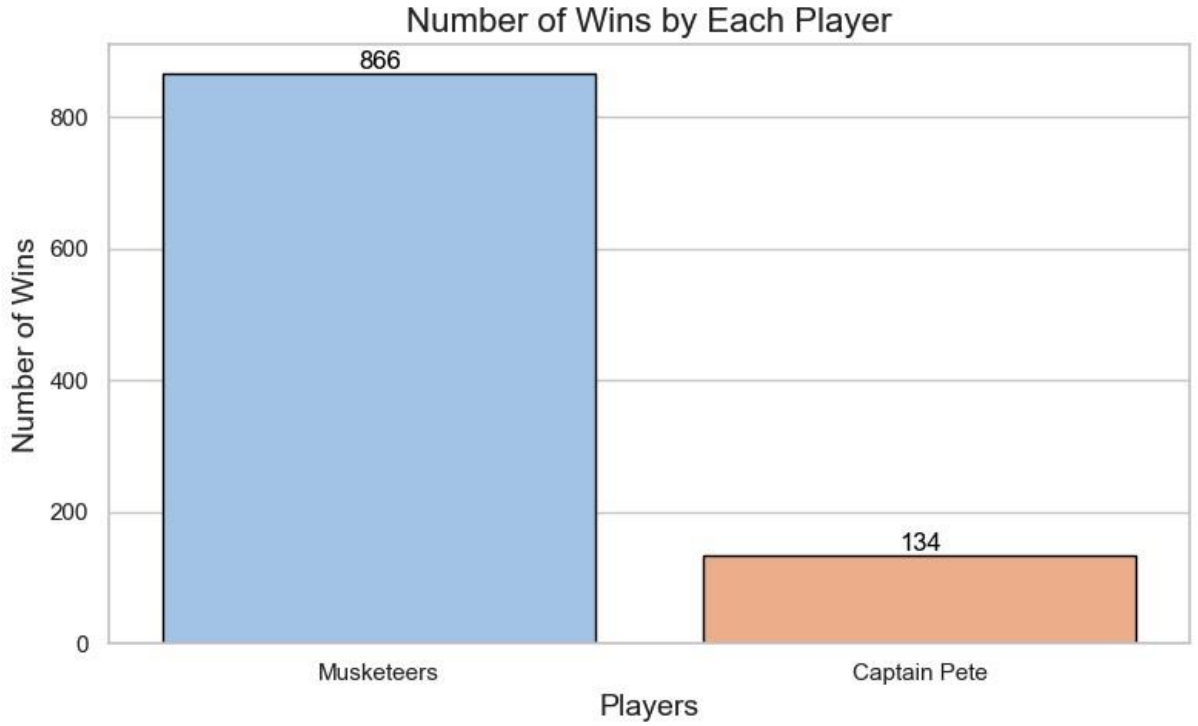
1. **Rendimiento de Mobility:** La heurística `mobility` debería ser reconsiderada o combinada con otras métricas, ya que mostró ser insuficiente en todos los escenarios.
2. **Tiempo de Ejecución:** `Expectimax` es notablemente más lento que `Minimax`. En entornos de tiempo real, esta diferencia puede ser crítica.

Recomendación Final

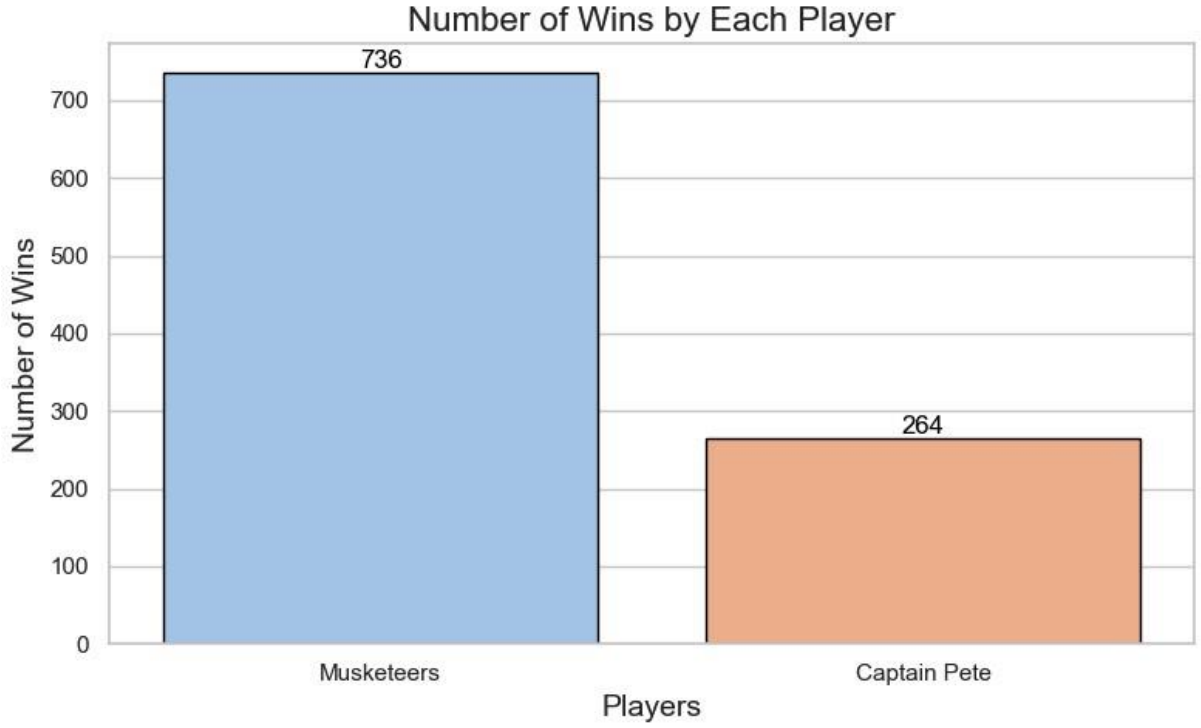
Se recomienda utilizar la configuración **`Expectimax-Default`** para maximizar las probabilidades de éxito en el tablero.

Anexo

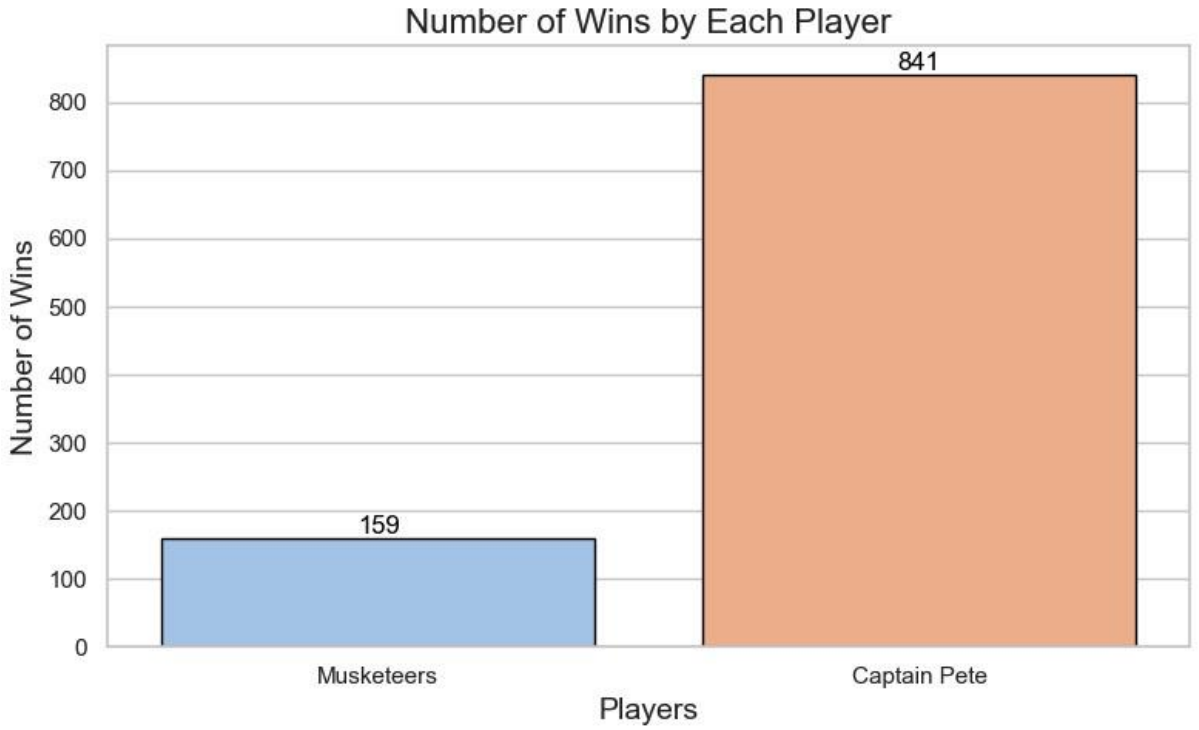
Minimax-default vs Pete



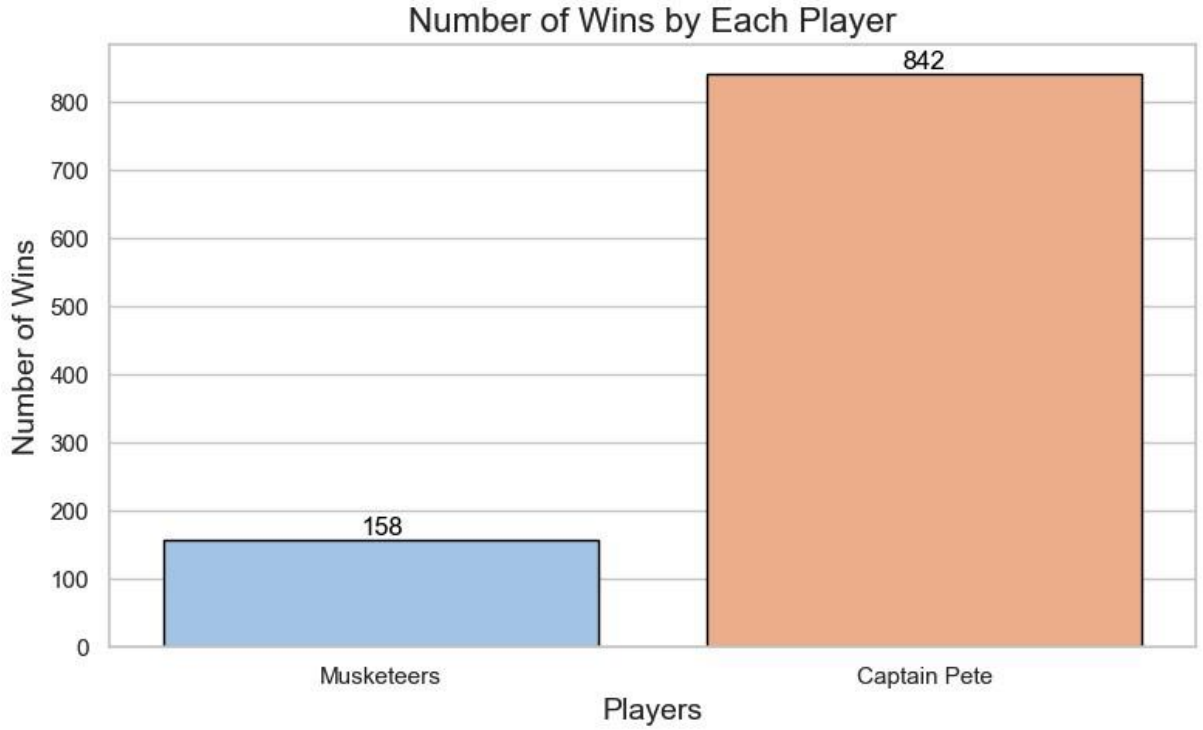
Minimax-default vs Random



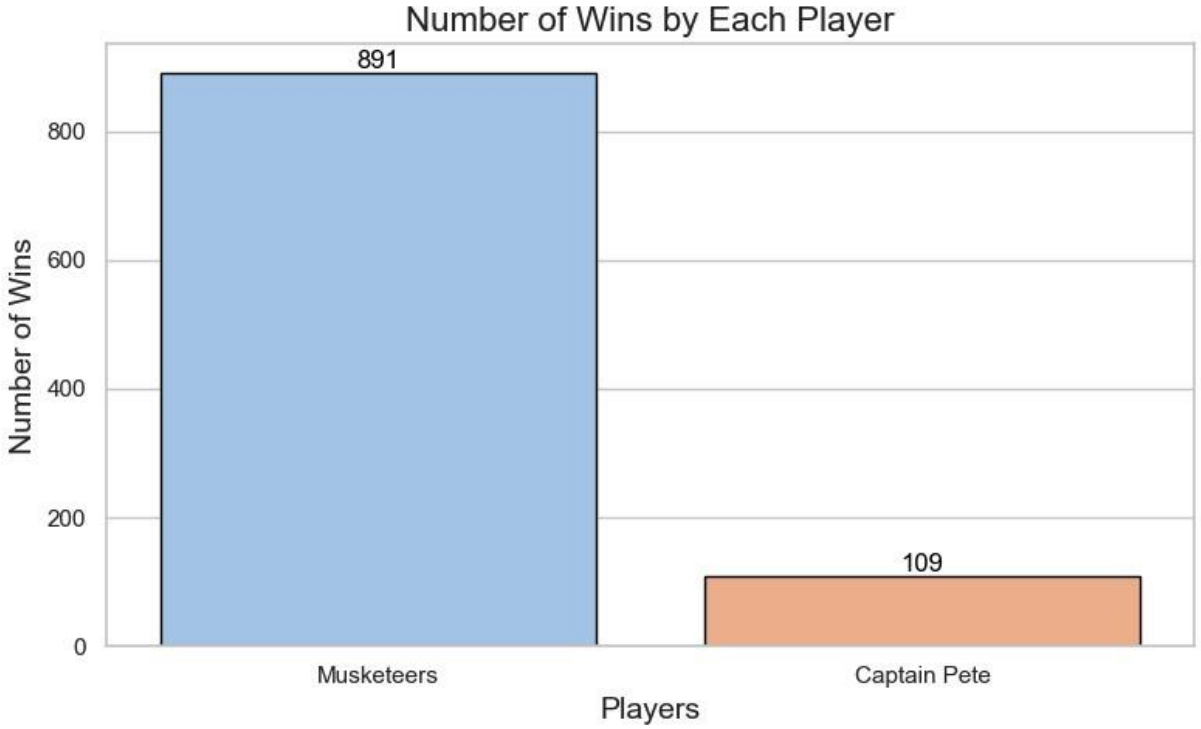
Minimax-mobility vs Pete



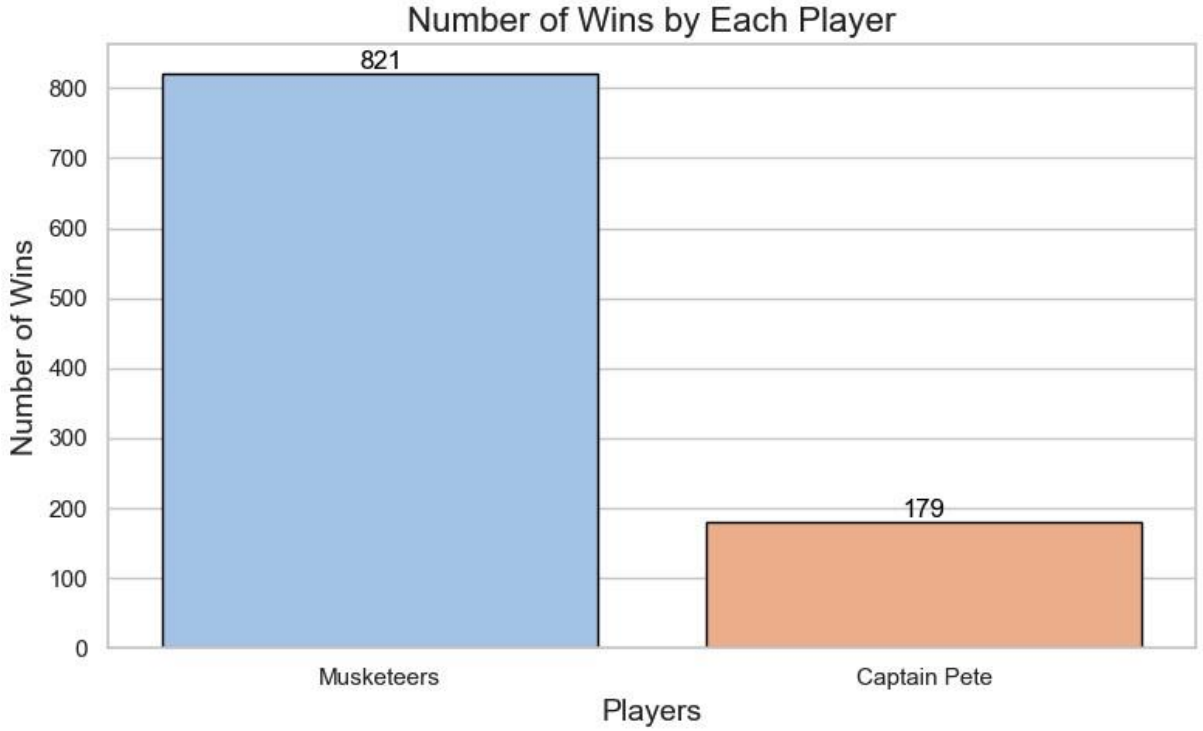
Minimax-mobility vs Random



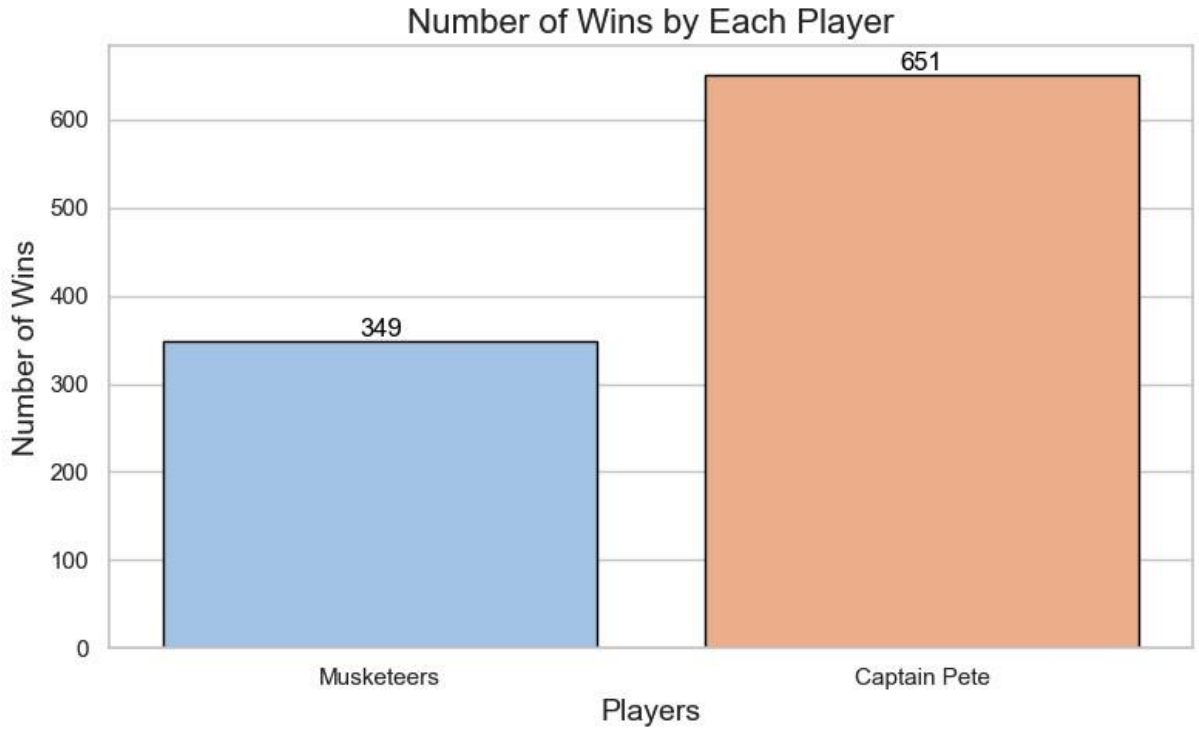
Expectimax-default vs Pete



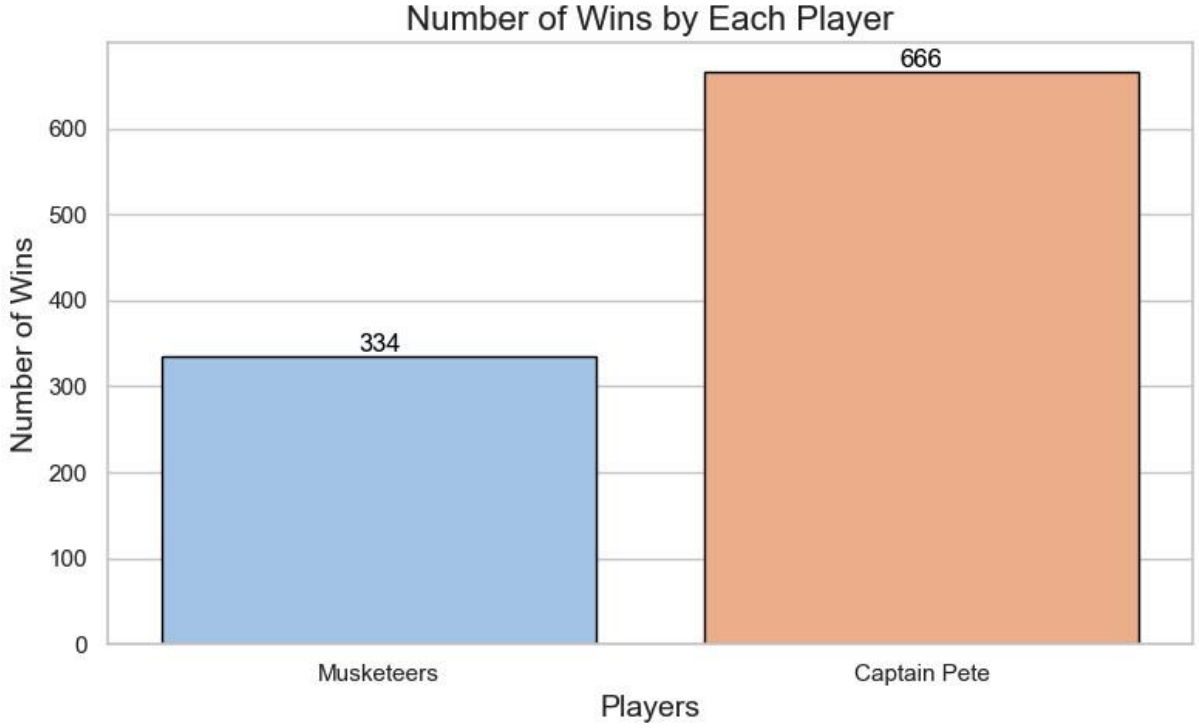
Expectimax-default vs Random



Expectimax-mobility vs Pete



Expectimax-mobility vs Random



Juegos

Example games