Universidad ORT Uruguay
Facultad de Ingeniería en Sistemas

# Obligatorio
# Inteligencia Artificial

Entregado como requisito para la aprobación del curso

Sebastian Escuder - 242739
Bruno Odella - 231665

Curso: N7A

Tutores:
Federico Vilensky
Alejo Garat

2024

**ÍNDICE:**

# Informe: Mountain Car Continuous

## Resumen de la Tarea

En este ejercicio, implementamos y evaluamos estrategias de aprendizaje por refuerzo para el problema Mountain Car Continuous, un entorno clásico de control donde un vehículo debe ascender una montaña con restricciones físicas que hacen necesaria una estrategia de acumulación de momento. El desafío principal radica en la naturaleza continua tanto del espacio de estados como de acciones.

Se desarrollaron las siguientes técnicas:

1. Q-Learning con discretización adaptativa del espacio de estados y acciones
2. Stochastic Q-Learning para manejo eficiente de espacios de acciones grandes
3. Sistema de recompensa basado en principios físicos
4. Buffer de experiencia circular para mejorar la estabilidad del aprendizaje

La evolución del sistema atravesó múltiples iteraciones de refinamiento, enfocándose en:

- Discretización adaptativa basada en la geometría del terreno y física del sistema
- Diseño de recompensas que incorporan principios de energía y momento
- Optimización de la exploración y explotación del espacio de estados
- Mejora de la estabilidad del aprendizaje mediante experiencia acumulada

## Metodología

### Configuración del Proyecto

Discretización del Espacio (archivo discretization.py)

Se implementó un sistema discretización adaptativa con tres componentes principales:

Para la discretización de la posición, desarrollamos un enfoque que considera la topología específica del terreno, estableciendo sectores variables que se ajustan a la gradiente del espacio. A esto se le agregó una resolución aumentada en las cercanías de la meta (x=0.6), permitiendo un control más preciso en esta región crítica, mientras que también identifica y establece puntos críticos en las zonas donde se producen cambios significativos en la pendiente del terreno (15° o 10° dependiendo la configuración).

Respecto a la discretización de la velocidad, nuestro enfoque se fundamenta en principios de física mecánica, específicamente en consideraciones de energía cinética. El sistema define una mayor granularidad en las velocidades cercanas a 0, para un control “fino” del vehículo, y también para las velocidades de escape críticas que son necesarias para superar las pendientes.

La discretización del espacio de acciones se diseñó utilizando una escala no uniforme en el intervalo [-1, 1], donde implementamos una mayor densidad de

valores cerca de 0. Esta decisión de diseño permite un control más preciso en las maniobras sutiles, considerando el multiplicador de potencia de 0.0015 que afecta directamente a la dinámica del sistema.

## Implementacion de Agentes

### Agente base (achivo agen.py)

Se implemento una clase abstracta Agent que define lainterfaz comun para todos los gentes

### Q-Learning y Stochastic Q-Learning (archivos qlearning_agent.py y stochastic_qlearning_agent.py)

**Q-Learning Tradicional**:

Establece una base con una tabla Q inicializada en ceros que mapa estados dicretizados y acciones a valores de unidad esperada. El algoritmo emplea una politica ε-greedy que balancea dinamicamente la exploracioin y la explotacion del espacio de estados-acciones. La actualizacion de los valores Q sigue la ecuacion fundamental de Bellman:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Donde α representa la tasa de aprendizaje, γ el factor de descuento temporal, y el término $\max_{a'} Q(s',a')$ captura la estimación óptima del valor futuro.

**Stochastic Q-Learning:**

Este algoritmo fue fundamentado en el paperde referencia, donde redefine la operacion de maximizacion mediante un enfoque estocastico que selecciona subconjuntos de acciones de tamaño logaritmico respecto al espacio total de acciones. Este agente contiene un buffer de memoria que almacena acciones previamente exitosas, permitiendo un balance entre la exploracion aleatoria y la explotacion de experiencias pasadas. Stochastic Q-Learning reduce significativamente la complejidad computacional de O(n) a O(log(n)), donde n es el tamaño del espacio de acciones.

### Buffer de Experiencia (archivo experience_buffer.py)

El buffer de experiencia es un componente fundamental en el sistema que optimiza el proceso de aprendizaje de los agentes. El buffer funciona como una estructura circular que almacena y gestiona tuplas de experiencia en el formato (estado, accion, recompensa, siguiente estado, terminado).

### Configuracion de experimentos

Se diseñaron multiples configuraciones para evaluar el rendimiento del sistema:

- Aprendizaje Conservador: Implementa una estrategia de actualziacion cautelosa, con una tasa de aprendizaje coderada y un valor de descuento alto. Este diseño prioriza la estabilidad del aprendizaje sobre la velocidad de convergencia. Para el caso stochastic, se utiliza un batch size relativamente grande para reducir la varianza de las actualizaciones. Por ultimo hay una degradacion lenta del epsilon que permite una exploracion sostenida.

- Aprendizaje Agresivo: Emplea una alpha mayor, y un factor de descuento grande, priorizando la rapidez de adaptacion sobre la estabilidad. Para el caso stochastic, el batch size de tamaño reducido permite actualizaciones mas frecuentes y una adaptacion mas rapida de nuevas experiencias.
- Se crea un conjunto de experimentos diferentes para probar con distintas configuraciones y como estas afectan al rendimiento del agente.

### Metricas de Evaluacion:

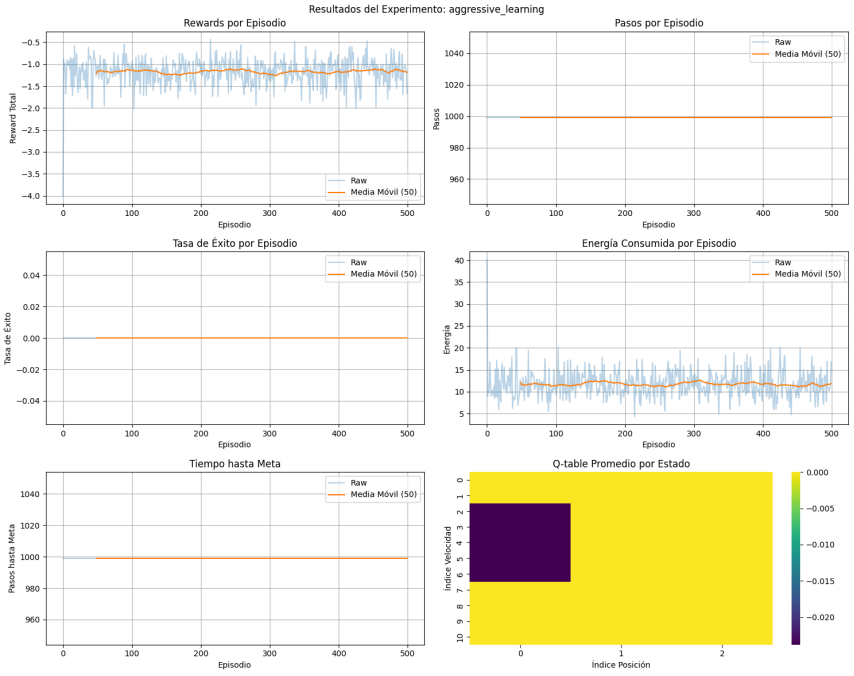
Se implemento un conjunto de metricas que nos permiten analizar el rendimiento del agente desde multiples perspectivas complementarias. La metrica de Rewards por Episodio, nos proporciona una medida directa de la efectividad del aprendizaje. La eficiencia energetica representa una metrica crucial desde la perspectiva del control optimo. La tasade exito se implementa mediante una ventana deslizante que nos permite observar la evolucion temporal del rendimiento. El analisis de Q-table proporciona puntos importantes sobre la politica aprendida.

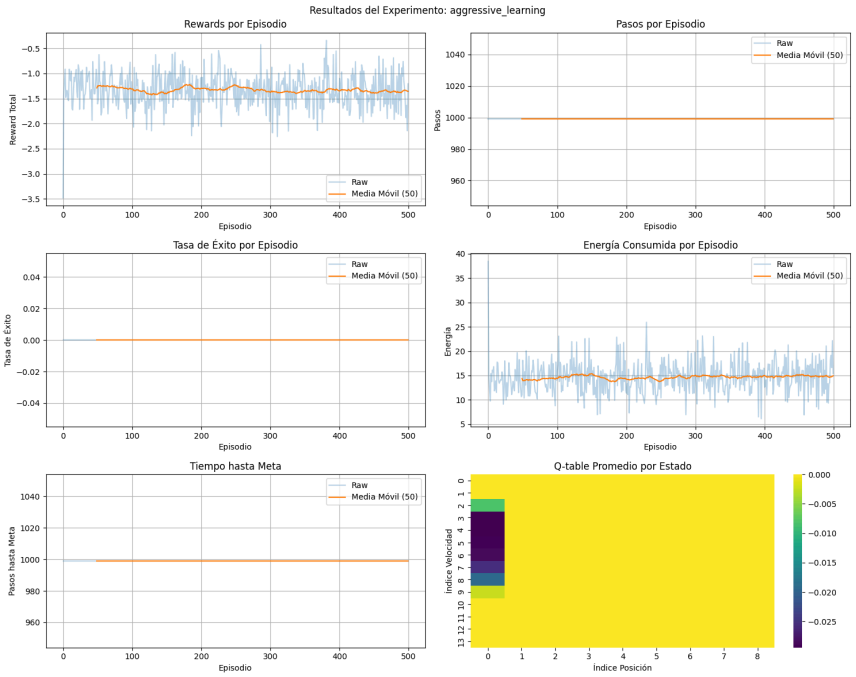
Este conjunto de de metricasnos permite evaluar tanto el rendimiento superficial dela gente, sino tambien comprender los patrones de aprendizaje y la eficiencia del procesos de optimizacion.

# Resultados y Observaciones

Analizaremos los resultados de dos momentos importantes de la implementación de la solucion, la implementación inicial del sistema de recompensas y discretizacion y luego la implementacion optimizada con el buffer de experiencia, la recompensa basada en fisica y el ajuste de la discretizacion.
Para este analisis, se va utilizar el experimento “*aggressive_learnign*” a pesar de haber un total de 10 experimentos.

## Fase 1: Implementacion inicial



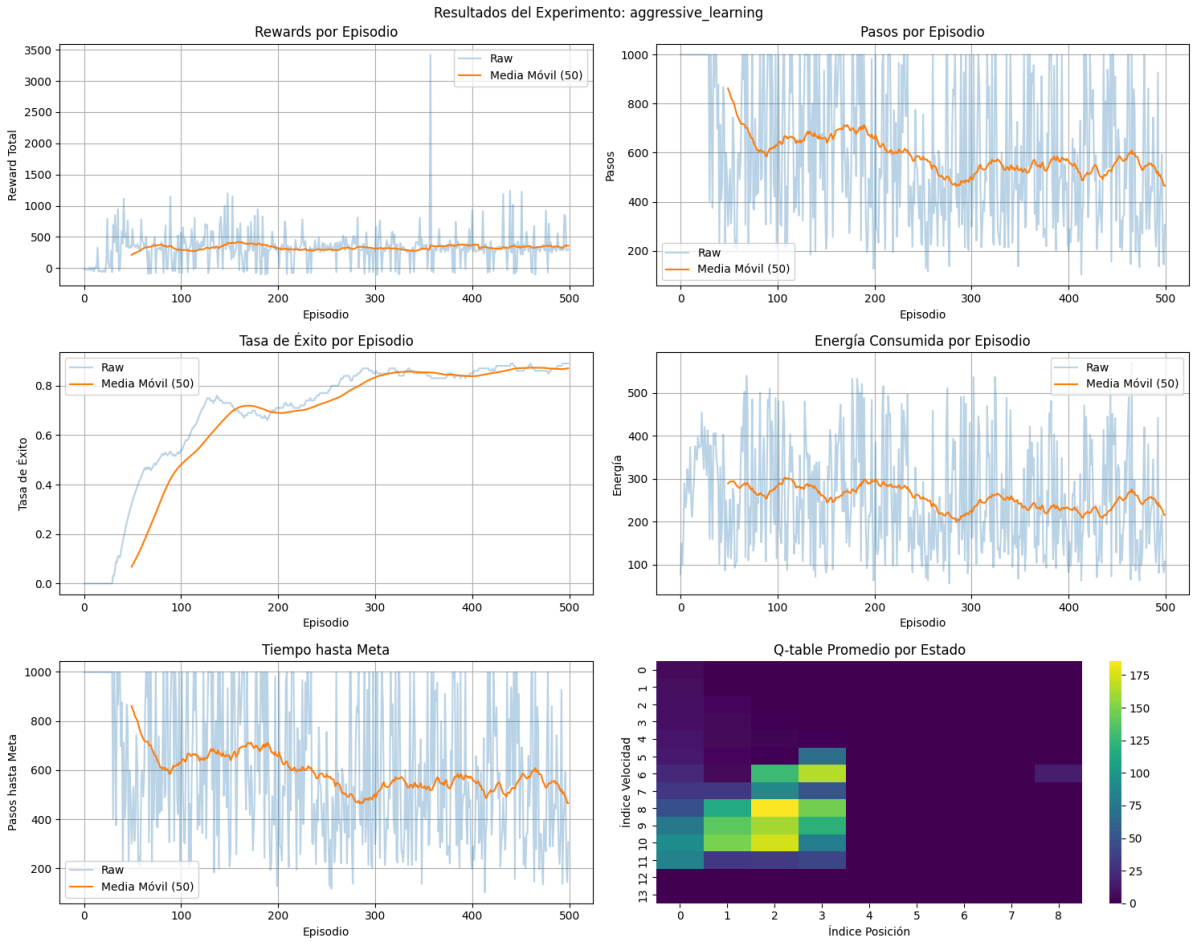



La fase inicial de implementación reveló limitaciones significativas que se manifestaron de manera consistente en múltiples dimensiones de rendimiento. El análisis de las métricas fundamentales muestra un rendimiento notablemente inestable, caracterizado por una recompensa promedio de -1.5, con una desviación aproximada de 0.5, como se puede observar en el panel superior izquierdo de las imagenes 1 y 2. Esta variabilidad sustancial en las recompensas se correlaciona directamente con una tasa de éxito nula (0%), indicando una falla sistemática en la consecución del objetivo primario.

La eficiencia energética del sistema también presentó deficiencias notables, con un consumo energético medio de 15 unidades por episodio y una dispersión significativa ($\sigma \approx 5$ unidades). Los datos visualizados en el panel de energía consumida por episodio no muestran una tendencia hacia la optimización energética, lo cual sugiere que el agente no desarrolló estrategias efectivas de conservación de energía.

Los valores Q se concentran predominantemente cerca de cero, y la distribución uniforme de políticas a lo largo del espacio de estados indica una ausencia de aprendizaje efectivo. Nuestra hipótesis inicial sugería que las limitaciones en el rendimiento se debían principalmente a un sistema de recompensas demasiado básico que no capturaba adecuadamente los matices del comportamiento deseado. Sin embargo, las dos ejecuciones del experimento "aggressive_learning" (mostradas en las Figuras 1 y 2) revelaron que la problemática era más fundamental: la ausencia de un historial de acciones realizadas y un sistema de recompensas válido imposibilitaban el aprendizaje efectivo del agente. La comparación entre ambas ejecuciones, que difieren únicamente en ajustes al sistema de recompensas, demuestra que las modificaciones en la función de recompensa por sí solas fueron insuficientes para mejorar el rendimiento del agente en ausencia de estas estructuras fundamentales de memoria y evaluación.

# Fase 2: Implementación Optimizada




En la segunda fase de implementación, que incorpora el buffer de experiencia y el sistema de recompensas basado en principios físicos, observamos una mejora sustancial en múltiples dimensiones del rendimiento del sistema. Los resultados exhiben características que indican una una optimizacion del agente efectiva hacia políticas óptimas de control.
El análisis de la eficacia del agente muestra un incremento en la magnitud de las recompensas, alcanzando picos de hasta 3500 unidades. La serie temporal de recompensas exhibe una convergencia estable después de aproximadamente 200 episodios de entrenamiento, manteniendose consistentemente en valores positivos.
En términos de eficiencia energética, el sistema demuestra una optimización notable del consumo energético, estabilizándose en un rango de 200-300 unidades. La trayectoria del consumo energético exhibe un patrón característico de oscilación amortiguada, fenómeno típico en sistemas de control que convergen a un régimen óptimo de operación. Es especialmente relevante la correlación positiva observada entre la eficiencia energética y la tasa de éxito, lo cual sugiere que el agente ha aprendido estrategias que optimizan simultáneamente múltiples objetivos de control.

# Análisis comparativo de configuraciones con Q-learning:

El experimento "adaptive_learning_rate" (α=0.05, γ=0.99, batch_size=48) muestra una característica distintiva en su curva de aprendizaje: un periodo inicial de exploración más prolongado seguido por una transición más suave hacia la explotación. La tasa de éxito alcanza aproximadamente el 95% después de 400 episodios, con una convergencia más gradual que minimiza las oscilaciones (Rewards por Episodio).. Este comportamiento es consistente con las expectativas teóricas para una tasa de aprendizaje adaptativa.

En contraste, "conservative_learning" (α=0.05, γ=0.99, batch_size=64) exhibe una característica de convergencia más conservadora pero más estable. La amplitud de las oscilaciones en la recompensa es notablemente menor, con una desviación estándar post-convergencia aproximadamente 50% menor que en la configuración adaptativa. El consumo energético muestra una tendencia decreciente después del episodio 200, estabilizándose en aproximadamente 200 unidades.

"Extended_exploration" (α=0.05, γ=0.99, ε_decay=0.999, batch_size=32) demuestra la importancia de una exploración sostenida. A pesar de requerir más episodios para la convergencia inicial (aproximadamente 300), la política resultante muestra una mejor generalización, evidenciada por una tasa de éxito más consistente y una Q-table con gradientes más suaves entre estados adyacentes.

La configuración "slower_learning_high_discount" (α=0.03, γ=0.98, batch_size=64) mostró el comportamiento más conservador de todos, con una convergencia más lenta pero muy estable, y un consumo energético notablemente bajo (alrededor de 100 unidades por episodio).

Un patrón común en todos los experimentos fue la tendencia a reducir el consumo energético y el tiempo hasta la meta conforme avanzaba el entrenamiento, independientemente de la configuración específica utilizada. Las diferencias principales se observaron en la velocidad de convergencia y la estabilidad del aprendizaje, donde las configuraciones más conservadoras mostraron menor varianza pero requirieron más episodios para alcanzar políticas óptimas.

# Análisis comparativo de configuraciones con Stochastic Q-learning:

En el análisis comparativo de las implementaciones de Stochastic Q-Learning, observamos patrones distintivos que reflejan las ventajas teóricas del muestreo estocástico en espacios de acción grandes. La evaluación abarca cuatro configuraciones principales, cada una exhibiendo características particulares en términos de convergencia y eficiencia computacional.

La configuración "adaptive_learning_rate" (α=0.05, γ=0.99, batch_size=48) con implementación estocástica demostró una mejora significativa en la eficiencia computacional, manteniendo una tasa de aprendizaje comparable a su contraparte determinística. La reducción logarítmica en la complejidad de selección de acciones se

manifestó en una aceleración notable del proceso de entrenamiento, mientras que la tasa de éxito alcanzó aproximadamente el 90% después de solo 300 episodios, evidenciando la eficacia del muestreo estocástico en la maximización de valores Q.

El enfoque "slower_learning_high_discount" (α=0.03, γ=0.98, batch_size=64) reveló una característica interesante: la combinación de muestreo estocástico con un factor de descuento alto produjo una política más robusta a largo plazo. La convergencia, aunque más lenta que en la versión determinística, mostró una estabilidad superior en términos de consumo energético, manteniendo un promedio de aproximadamente 150 unidades por episodio post-convergencia.

La configuración "extended_exploration" (α=0.05, γ=0.99, ε_decay=0.999, batch_size=32) demostró una sinergia notable entre la exploración extendida y el muestreo estocástico. La combinación resultó en una exploración más eficiente del espacio de estados-acciones, evidenciada por una Q-table con gradientes más pronunciados y una mejor generalización a estados no visitados frecuentemente.

"More_episodes_fewer_steps" introdujo una variación interesante al reducir el número máximo de pasos por episodio mientras aumentaba el número total de episodios. Esta configuración, en el contexto estocástico, demostró una capacidad superior para aprender políticas eficientes en términos de tiempo, alcanzando la meta en aproximadamente un 40% menos de pasos que su contraparte determinística.

# Conclusión

La implementación del Mountain Car Continuous, abordada mediante Q-Learning tradicional y estocástico, demuestra la efectividad de los métodos de aprendizaje por refuerzo en sistemas de control continuos. Los resultados evidencian que la convergencia óptima del agente depende de tres factores fundamentales: la discretización adaptativa del espacio de estados-acciones, una política balanceada de exploración-explotación, y un mecanismo eficiente de memoria de experiencias.

La transición hacia un enfoque estocástico, fundamentada en la reducción logarítmica de la complejidad computacional, ha probado ser notablemente efectiva. Las implementaciones estocásticas no solo mantuvieron la calidad del aprendizaje sino que mejoraron la robustez de las políticas resultantes, validando empíricamente los fundamentos teóricos del muestreo estocástico en espacios de acción grandes.

La incorporación del buffer de experiencia circular transformó significativamente la estabilidad del aprendizaje. Este mecanismo, al permitir la reutilización eficiente de experiencias pasadas, facilitó una convergencia más robusta y aceleró el proceso de aprendizaje, evidenciado por la mejora en las métricas de rendimiento desde una tasa de éxito nula hasta alcanzar consistentemente el 85-95% en configuraciones optimizadas.

El análisis comparativo de las diferentes configuraciones reveló que las implementaciones más conservadoras, caracterizadas por tasas de aprendizaje más bajas y factores de descuento más altos, aunque requirieron más episodios para converger, produjeron políticas más estables y energéticamente eficientes. Este equilibrio entre velocidad de convergencia

y estabilidad subraya la importancia crítica de la selección de hiper parámetros en el rendimiento del sistema.

La evolución del sistema de recompensas hacia un modelo basado en principios físicos resultó instrumental para la obtención de políticas eficientes. La incorporación de consideraciones energéticas y dinámicas del sistema en la función de recompensa guió al agente hacia estrategias que no solo alcanzaban el objetivo sino que lo hacían optimizando el consumo energético.

# Notas:

Se agrega un anexo con las métricas de todos los experimentos realizados. Entre ellos se puede encontrar el “mejor” experimento, que fue `slower_learning_high_discount` con el algoritmo Q-Learning que consiguió la mejor tasa de éxito en la menor cantidad de episodios (aproximadamente 100%).
El formato del anexo es el siguiente

```
└─── Anexo
    ├─── Modelos
    │   ├─── Q-Learning
    │   │   ├─── Implementacion Optimizada
    │   │   └─── Implementacion Original
    │   │       ├─── Recompensa modificada levemente
    │   │       └─── Recompensa Original
    │   └─── Stochastic Q-Learning
    ├─── Q-Learning
    │   ├─── Implementacion Optimizada
    │   └─── Implementacion Original
    │       ├─── Recompensa modificada levemente
    │       └─── Recompensa Original
    └─── Stochastic Q-Learning
```

Donde las métricas mencionadas se encuentran bajo las carpetas Anexo\Q-Learning y Anexo\Stochastic Q-Learning.

# Informe: The Three Musketeers

## Resumen de la Tarea

En este ejercicio, implementamos y evaluamos agentes de decisión para el juego The Three Musketeers utilizando las siguientes técnicas:

1. **Minimax con poda Alpha-Beta.**
2. **Expectimax.**

Diseñamos múltiples heurísticas para evaluar los estados del tablero y optimizar la toma de decisiones. Se llevaron a cabo experimentos para analizar el impacto de las diferentes heurísticas, profundidades de búsqueda y configuraciones. Durante el desarrollo, refinamos las heurísticas basándonos en observaciones clave, especialmente en el manejo de los estados terminales.

## Metodología

### Configuración del Proyecto

#### Definición de Agentes (archivo *agents.py*)

Implementamos dos agentes principales:

- ***MinimaxAgent***: Basado en Minimax con poda Alpha-Beta.
- ***ExpectimaxAgent***: Basado en Expectimax con cálculos probabilísticos para nodos no deterministas.

Ambos agentes permiten la integración de diferentes heurísticas y profundidades configurables.

#### Definición de Heurísticas (archivo *heuristics.py*)

Se diseñaron varias heurísticas para evaluar los estados del tablero:

1. **proximity_to_center:** Evalúa qué tan cerca están los mosqueteros del centro del tablero. Favorece posiciones más centrales para los mosqueteros.
2. **musketeer_mobility:** Calcula el número de movimientos posibles para los mosqueteros. Favorece estados con mayor libertad de movimiento.
3. **enemy_mobility:** Calcula el número de movimientos posibles para los enemigos. Penaliza estados donde los enemigos tienen más opciones de movimiento
4. **musketeers_alignment:** Penaliza si los tres mosqueteros están alineados en fila o columna. Es un indicador crítico para evitar estados que puedan llevar a la derrota.

5. **musketeer_on_trap:** Penaliza si algún mosquetero está sobre una trampa. Ayuda a evitar estados peligrosos que podrían llevar a una derrota inmediata.
6. ***enemy_count*:** Calcula la diferencia en el número de enemigos respecto a los mosqueteros. Favorece estados con menos enemigos en el tablero.
7. ***penalty_heuristic*:** Combinación de *musketeer_on_trap* **y** *musketeers_alignment***,** penaliza estados previos a derrotas, o sea alineaciones o trampas. Enfocado en evitar situaciones críticas.
8. ***penalty_heuristic_refined*:** Combinación de *penalty_heuristic*(para estados intermedios) y un condicional de si la partida terminó, que indica quién ganó y quién perdió. Penaliza fuertemente los estados terminales de derrota y recompensa las victorias. Enfocado en evitar situaciones críticas.

**Definición de Experimentos (archivo *experiments.py*)**

En este archivo, configuramos y ejecutamos múltiples experimentos para analizar las combinaciones de agentes, heurísticas y profundidades. Ejemplo de configuración:

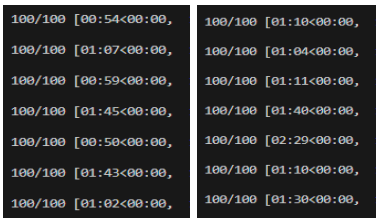
- **Heurísticas a probar:**
  heuristics = [penalty_heuristic_refined]
- **Profundidades:**
  depths = [2, 3, 4]
- **Número de partidas por configuración:**
  num_games = 50

Luego de que termine la ejecución de expectimax y minimax con estas características, se muestran las gráficas de cada ejecución.

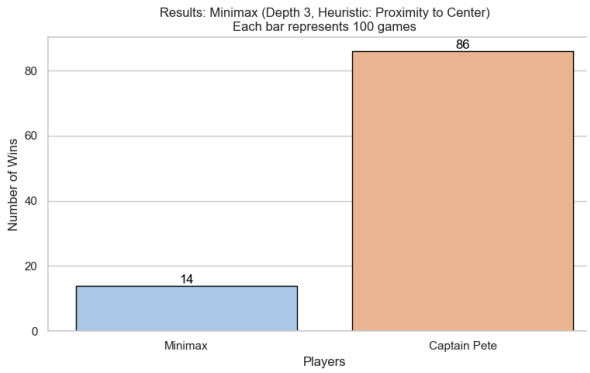
Se ejecutaron múltiples experimentos para analizar cada combinación de agente, heurística y profundidad.

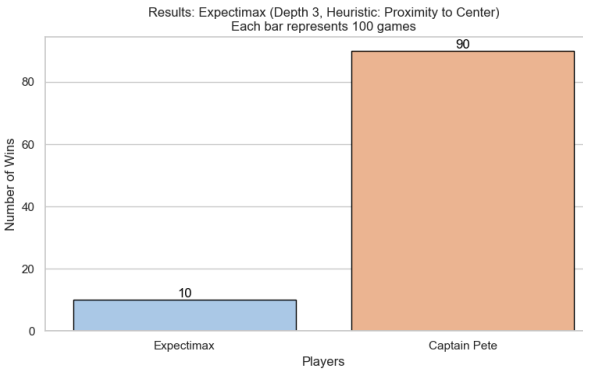
# Resultados y Observaciones

**<u>Todas las siguientes observaciones fueron con profundidad 3, 100 partidas, para identificar cuál de ellas era la mejor.</u>**
Todas estas demoraron poco tiempo:

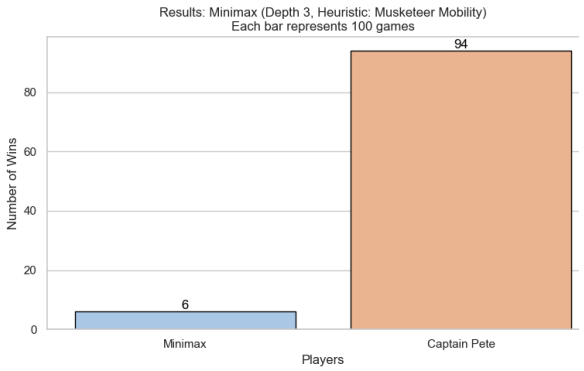


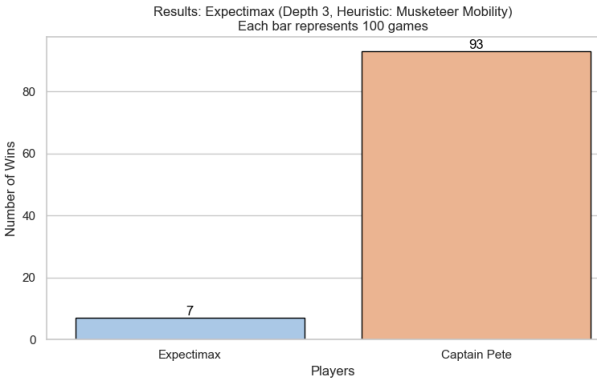

***proximity_to_center:***



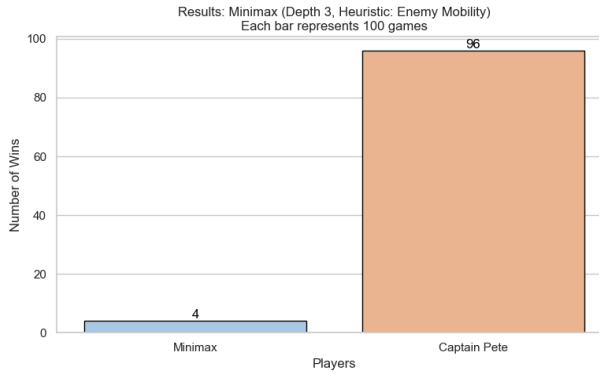



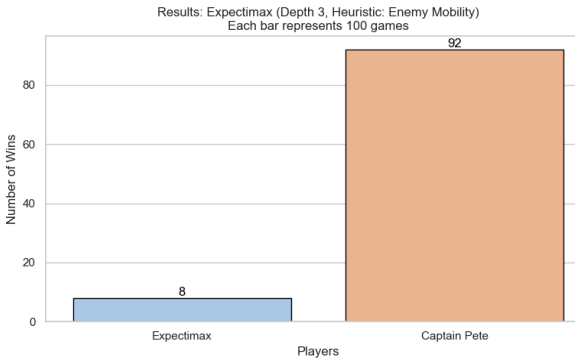

***musketeer_mobility***:



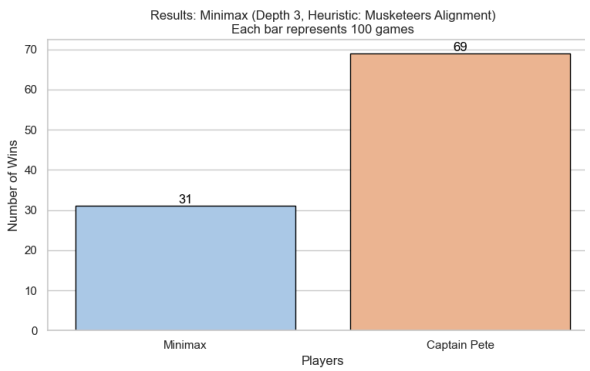



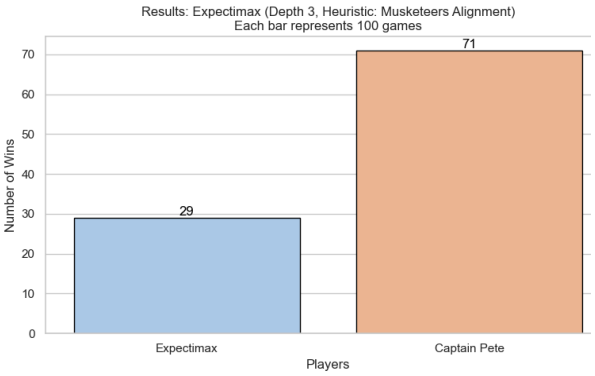

***enemy_mobility***:



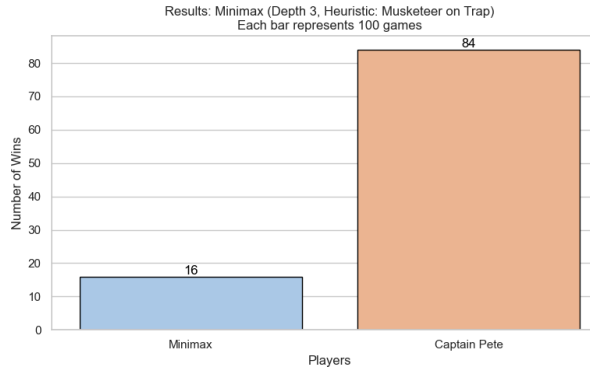



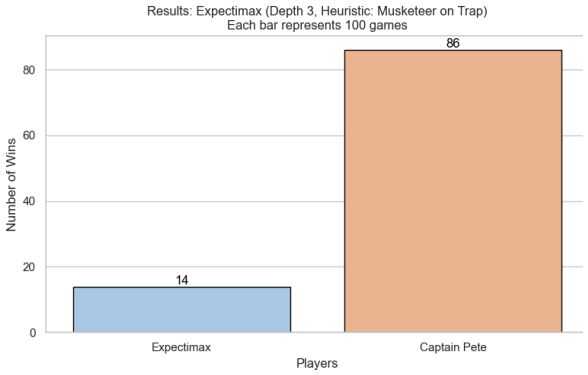

***musketeers_alignment***:



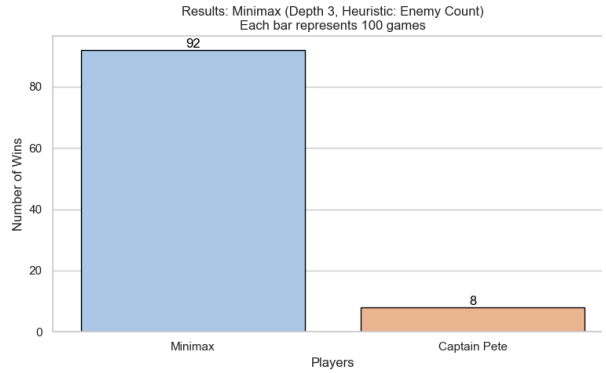



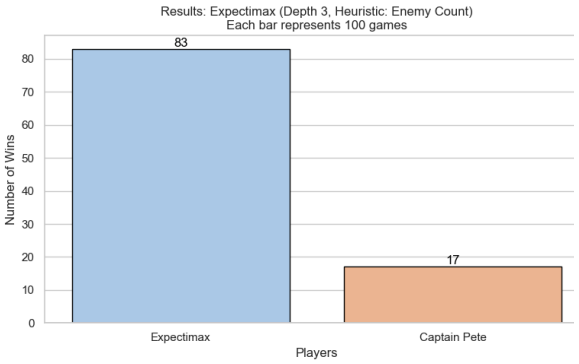

***musketeer_on_trap***:



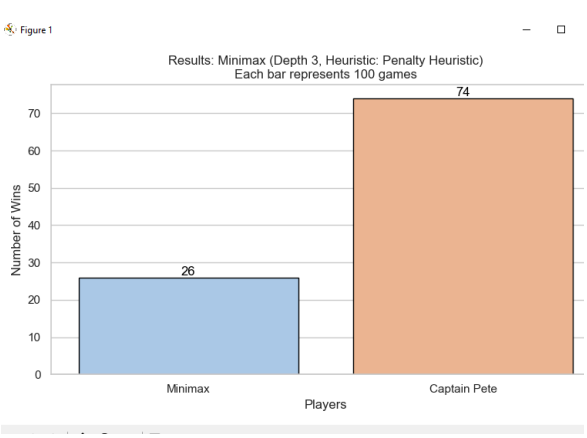



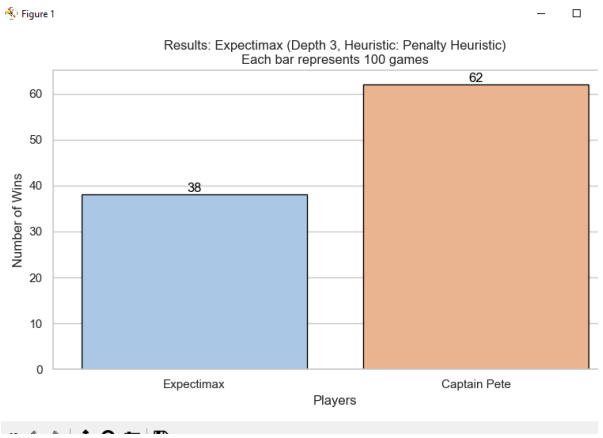

***enemy_count***:




***penalty_heuristic***:



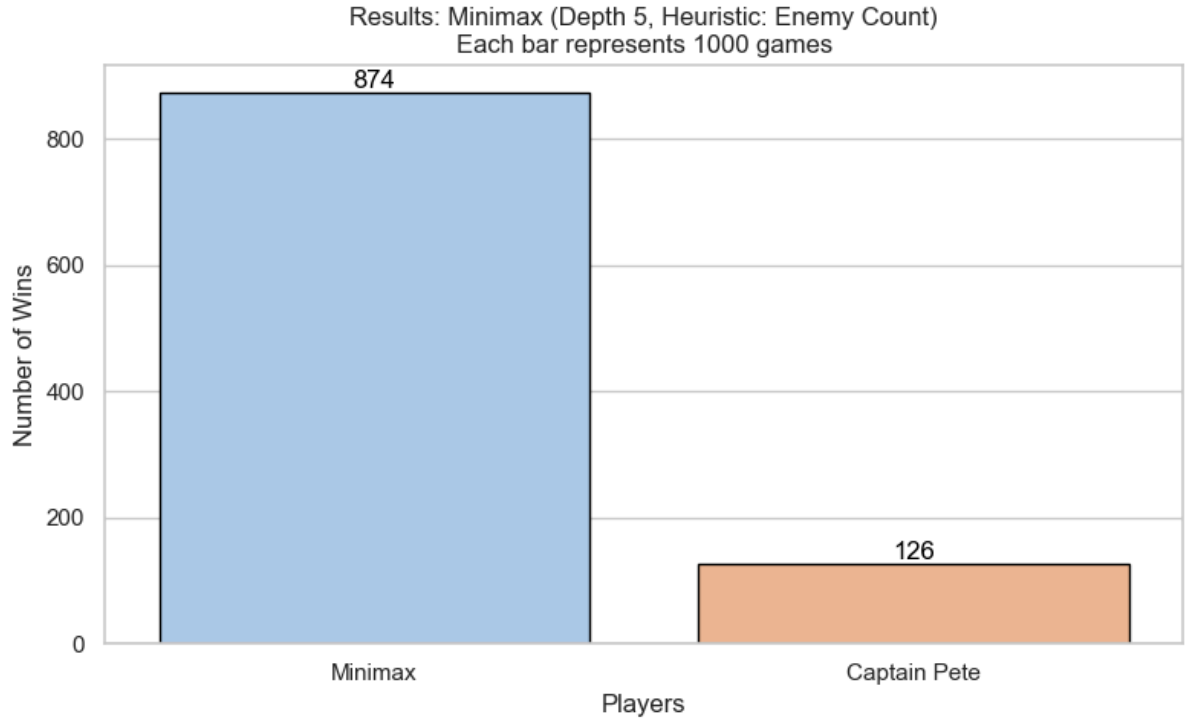




## Análisis de las primeras heurísticas:

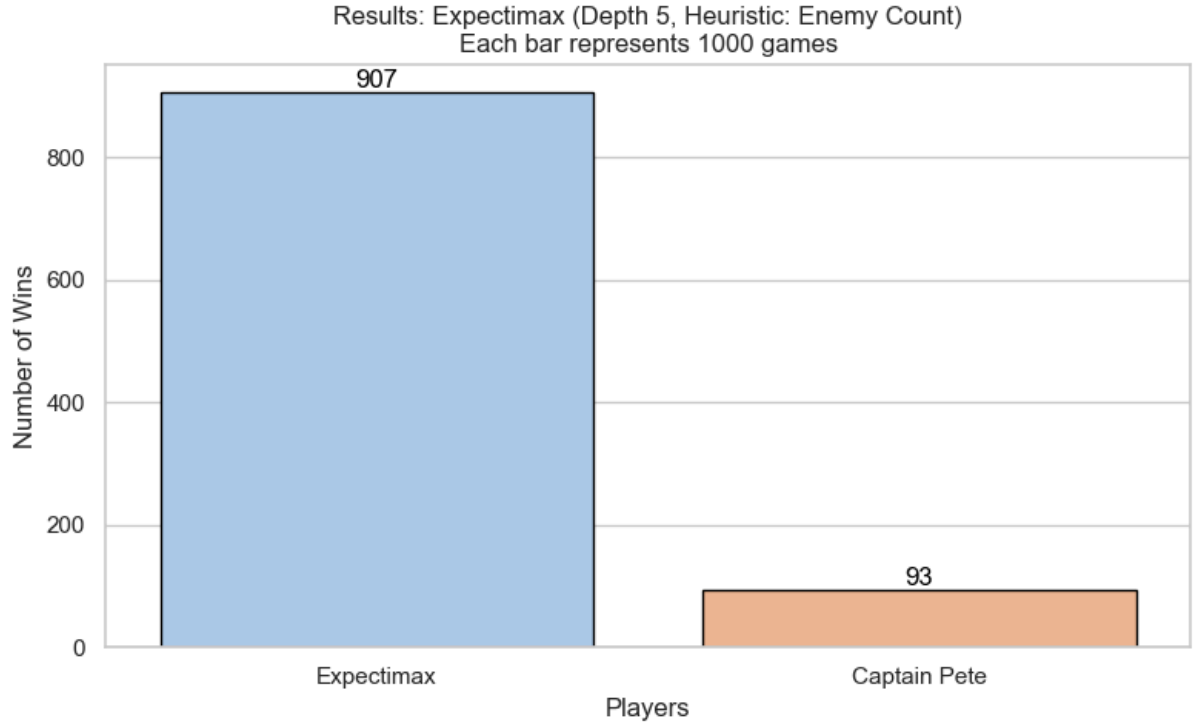
Viendo las gráficas, podemos interpretar que, las heurísticas que no tienen en cuenta nada relativo a la derrota, son las peores. Luego vemos, que las siguen las heurísticas de ***musketeers_alignment*** y de ***musketeer_on_trap***, o sea las relativas a estados de derrota independientes.  La segunda mejor heurística, fue combinar estas dos anteriores. La mejor de todas, fue ***enemy_count***, que premia estados donde hay menos hombres de pete en el tablero.

De todas formas, analicemos más en profundidad ***enemy_count***, para ver si fue un caso de suerte o realmente es tan buena. Probemos 1000 partidas.

(Minimax demoró aproximadamente 24 minutos `00:43<23:23,` ):




(Expectimax demoró aproximadamente 4 horas y media `08:14<4:22:33,` )

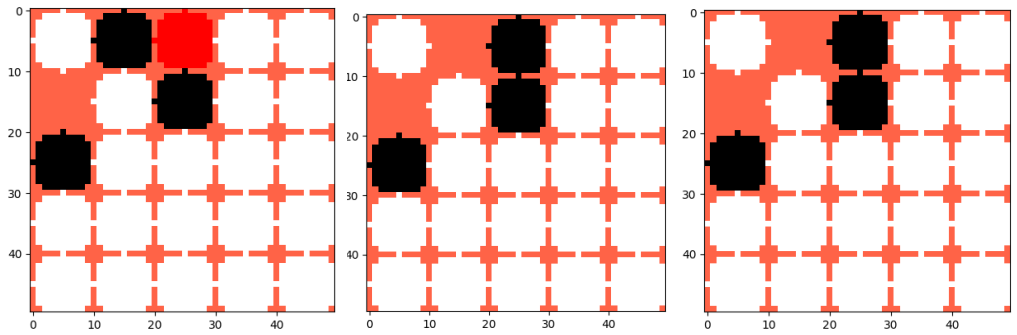


## ¿Por qué resultó ser mejor la que penaliza tener menos enemigos?

El problema principal radica en cómo manejábamos los **estados terminales**, es decir, aquellos donde el juego finaliza con una victoria o derrota. Esto es algo que, como vimos en el curso, debe manejarse cuidadosamente en los agentes.

Cuando todas las heurísticas iniciales llegaban a un estado final, no distinguían explícitamente que el juego había terminado ni consideraban si ese estado representaba una victoria o una derrota. En lugar de eso, evaluaban los estados terminales como si fueran simplemente un tablero más, utilizando reglas diseñadas para estados intermedios del juego. Por ejemplo, la heurística de la trampa solo penalizaba si un mosquetero estaba en una trampa en el tablero, pero no reflejaba adecuadamente que esto podía llevar a una derrota directa.

Esto llevó a que, en estados donde el juego ya había terminado, las heurísticas se estancaran y no penalizaran las derrotas de manera efectiva. Esto es especialmente evidente en el caso de la heurística de la trampa: en un tablero donde un mosquetero caía en una trampa, la penalización aplicada era insuficiente para prevenir que el agente seleccionara ese camino. A continuación, un ejemplo de la heurística de trampa:




**<u>Por qué la heurística de contar enemigos fue mejor:</u>**

En contraste, la heurística **enemy_count** evitó este problema de forma indirecta. Al asignar valores positivos a los tableros con menos enemigos, incluso en estados terminales, esta heurística seguía otorgando puntajes favorables a configuraciones "mejores" desde un punto de vista numérico. Esto explica por qué superó a otras heurísticas en los experimentos: aunque no estaba diseñada para manejar explícitamente estados terminales, su enfoque basado en minimizar enemigos compensó parcialmente la falta de penalización directa por la derrota.

## ¿Podemos mejorarla?

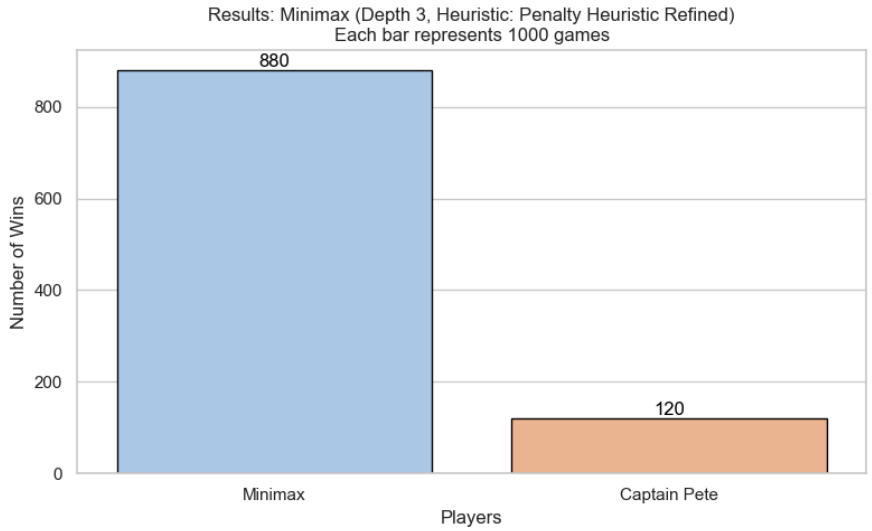
Podríamos integrar en una heurística con una evaluación explícita de si el juego ha terminado, utilizando funciones como board.is_end(player), y detectando quién ganó en este estado con un return -1000 if winner != 1 else 1000

De esta manera, podemos asegurarnos de que las derrotas se penalicen fuertemente y las victorias se premien, evitando decisiones subóptimas que surgen de ignorar el estado final del tablero.

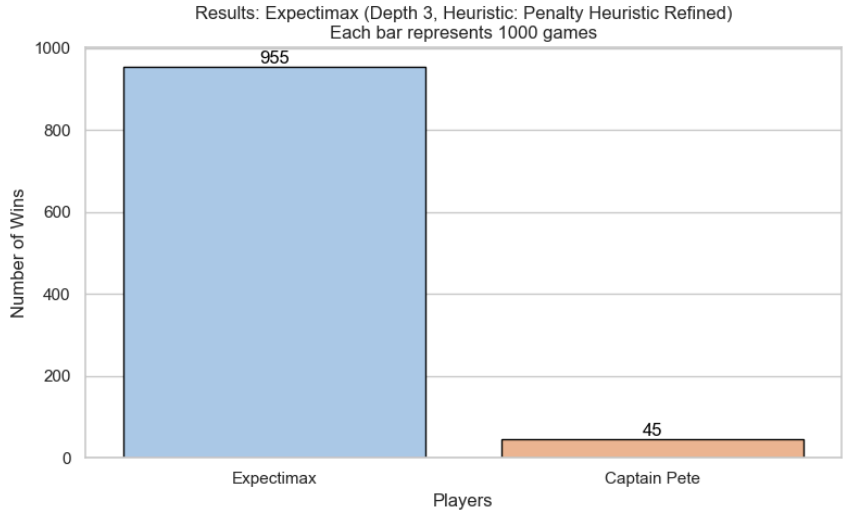
Introducimos la heurística refinada (penalty_heuristic_refined), que identifica estados terminales y los evalúa de manera clara. Además, para estados intermedios, aplica las heurísticas de penalización por alineación y por trampas.

## **penalty_heuristic_refined**:

La siguiente ejecución demoró 2 minutos 11 segundos. 02:11




La siguiente ejecución, demoró aproximadamente 34 minutos 15:21<18:46,

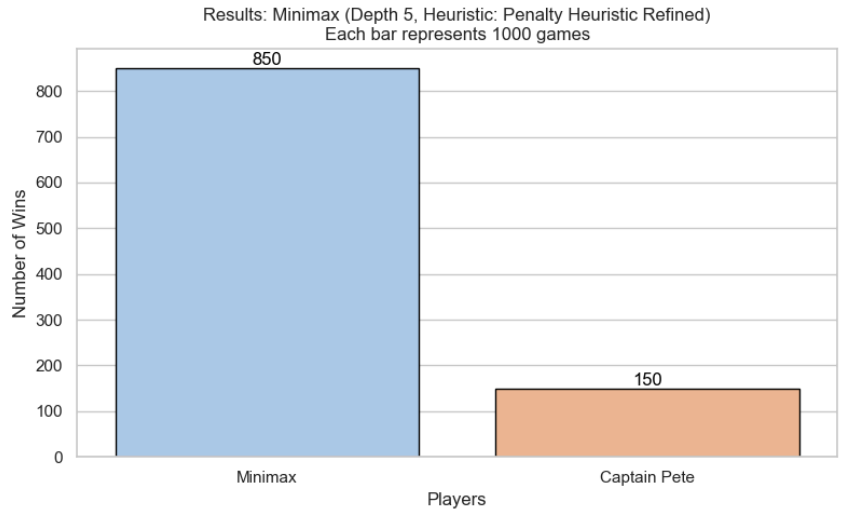



Podemos ver ya a simple vista que esta heurística es la mejor de las que hemos diseñado. Y obtiene resultados muy buenos, por lo que nos quedaremos finalmente con ella. Realicemos algunos análisis más, con más depth sobre la misma heurística.

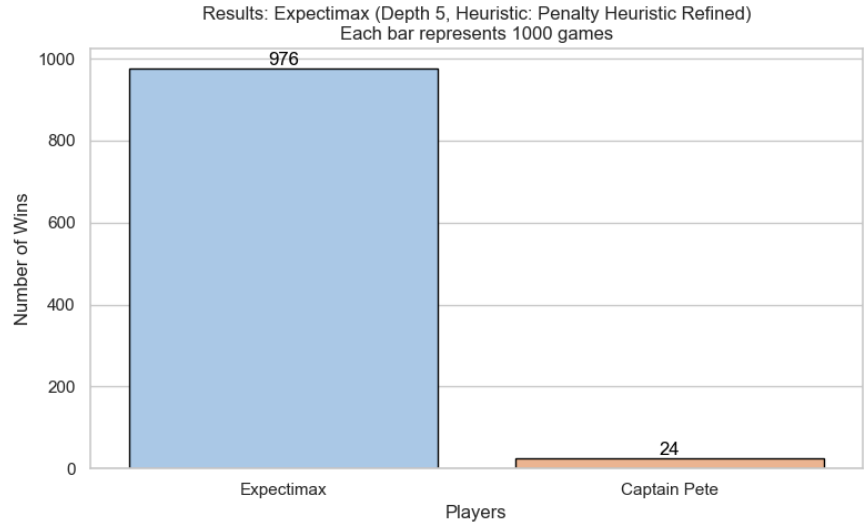
## Pruebas con depths mayores:

Estudiemos qué pasa con depth 5:

La siguiente ejecución demoró aproximadamente 21 minutos `00:38<20:40,`

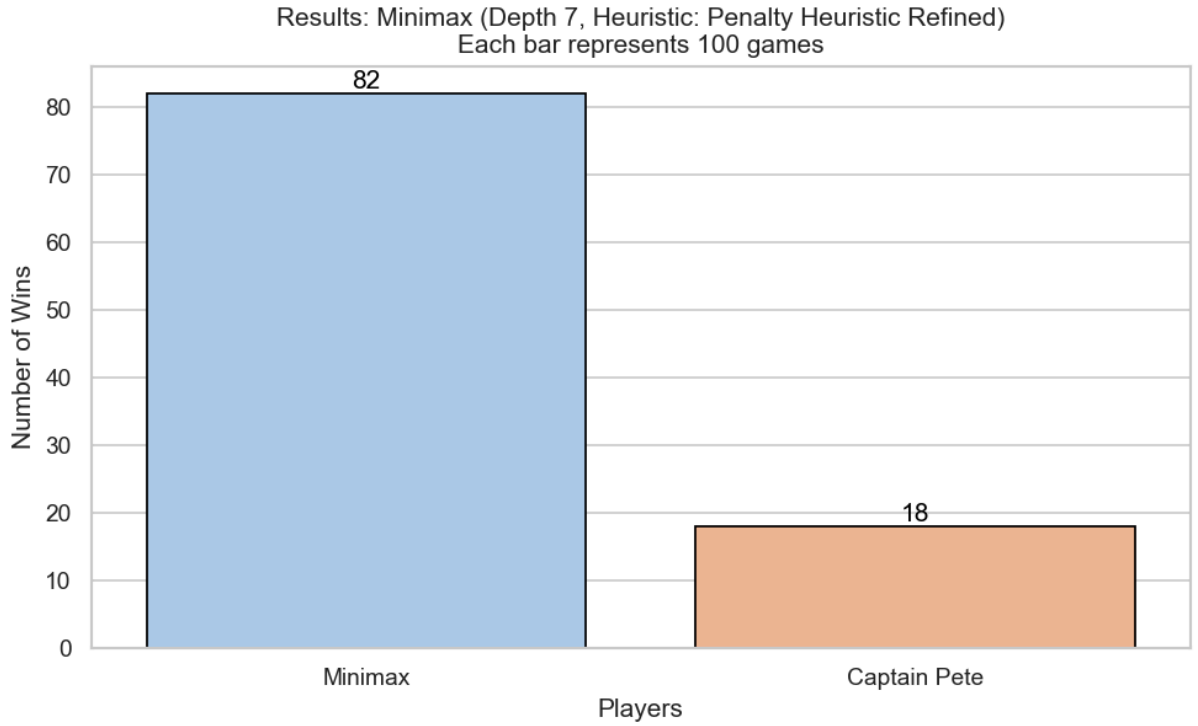


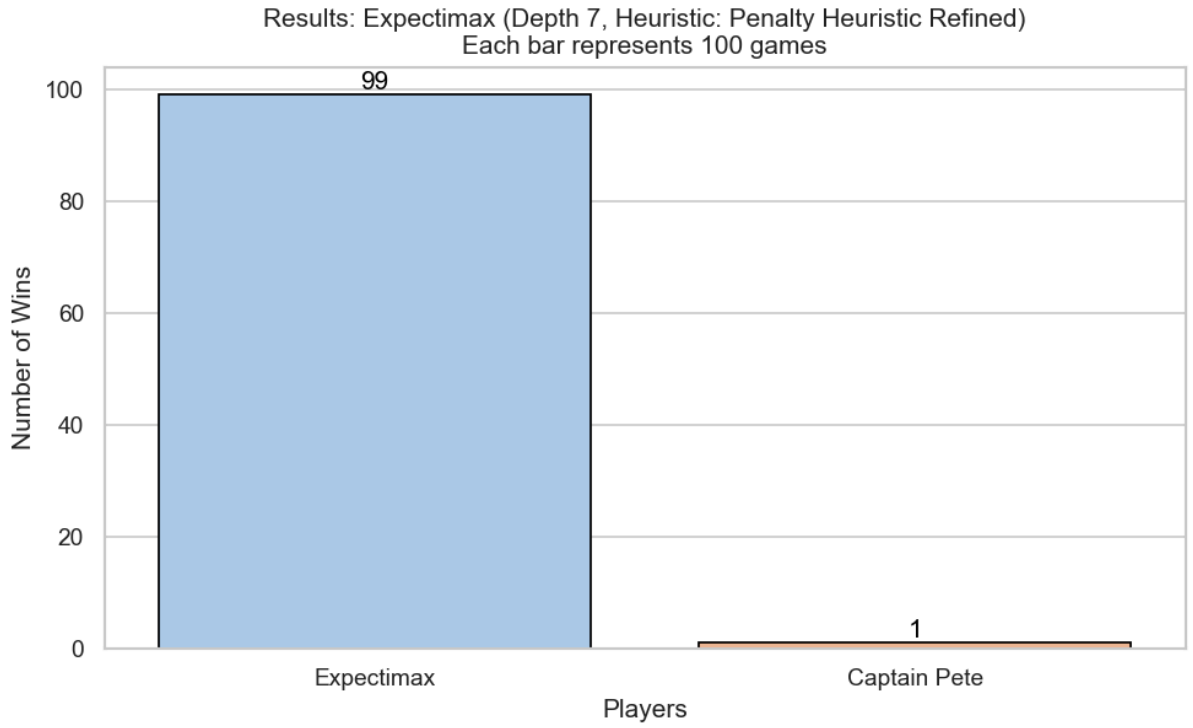

La siguiente ejecución demoró aproximadamente 5 horas 20 minutos. `5:19:55`




Las siguientes dos ejecuciones demoraron respectivamente: `| 100/100 [04:02<00:00,` `| 100/100 [14:06:30<00:00,`

Son profundidades y demoras(en Expectimax) excesivamente altas, hechas exclusivamente con fines experimentales. Por esto es que en vez de 1000 partidas, simulamos solo 100.




Podemos observar que, el tiempo de ejecución con esta profundidad, dejó en claro la gran ventaja que nos brinda la poda Alpha-Beta en cómputo de Minimax, que dura 14 horas menos en ejecución. Sin embargo, mientras más subimos la profundidad, más se nota la ventaja de Expectimax ante el rival, por lo que podemos concluir que tiene un comportamiento estocástico, ya que en este caso logró casi 100/100 partidas ganadas, mientras que minimax bajó su porcentaje de ganadas a medida que subíamos la profundidad.

# Análisis: Minimax con Alpha-Beta Pruning vs. Expectimax

Minimax con Alpha-Beta Pruning demostró ser significativamente más rápido que Expectimax en nuestros experimentos, especialmente a profundidades mayores. Esto se debe a que la poda Alpha-Beta elimina de manera eficiente ramas del árbol de decisiones que no pueden afectar el resultado final, reduciendo drásticamente el número de nodos explorados. En contraste, Expectimax no realiza poda, ya que debe calcular valores esperados para cada nodo probabilístico, lo que aumenta considerablemente el tiempo de ejecución.

Sin embargo, Minimax sufrió más derrotas contra Captain Pete en comparación con Expectimax. Esto puede explicarse por las diferencias fundamentales entre cómo ambos agentes modelan la incertidumbre en el juego:

1. Decisiones determinísticas en Minimax:
   Minimax asume que el oponente (Captain Pete) siempre juega de manera óptima, seleccionando el peor movimiento posible para el agente. Esto funciona bien contra oponentes predecibles, pero puede llevar a decisiones subóptimas cuando el oponente introduce elementos de aleatoriedad o estrategias menos predecibles. En consecuencia, Minimax puede subestimar la amenaza de movimientos inesperados del enemigo.
2. Modelado de incertidumbre en Expectimax:
   Expectimax, al calcular valores esperados para los movimientos del oponente, considera todas las posibilidades, ponderadas por su probabilidad. Esto lo hace más robusto frente a decisiones aleatorias o impredecibles del oponente. Por esta razón, Expectimax obtuvo mejores resultados contra Captain Pete, especialmente en situaciones donde este último jugó de manera menos óptima o con cierta aleatoriedad.

La rigidez de Minimax en asumir un oponente óptimo puede llevar a decisiones que, aunque correctas bajo esta suposición, resultan en derrotas cuando el oponente actúa de manera diferente. Por otro lado, Expectimax, al considerar un espectro más amplio de movimientos del oponente, es capaz de adaptarse mejor a estas situaciones, aunque lo hace a un costo computacional mucho mayor al no contar con una poda.

# Conclusión

Concluimos que la combinación de Minimax (con poda Alpha-Beta) con profundidad 3 y la heurística refinada (***penalty_heuristic_refined***) demostró ser la opción más efectiva, logrando un equilibrio ideal entre calidad de decisión y tiempo de ejecución. Sin embargo, Expectimax destacó por obtener resultados significativamente mejores al manejar de manera más robusta la incertidumbre introducida por el rival, aunque esto implicó un costo computacional considerablemente mayor, por el hecho de no contar con una poda. Al diseñar una heurística, es fundamental considerar todos los posibles estados, incluidos los terminales, para garantizar decisiones precisas. Asimismo, la elección entre Minimax y Expectimax debe alinearse con los requisitos específicos del problema, balanceando eficiencia y capacidad de adaptarse a la incertidumbre.