Pregunta 3

Actividad 1: Poda Alpha-Beta

Profundidad	Poda	Promedio J1 (s)	Promedio J2 (s)
1	No	0.0113	0.0110
1	Sí	0.0115	0.0116
2	No	0.3152	0.3065
2	Sí	0.0906	0.1021
3	No	9.8827	9.5250
3	Sí	0.6647	0.7805
4	Sí	9.5037	12.3130
4	No	*No se completó, estimado >500 s*	*No se completó, estimado >500 s*

La implementación de la poda Alfa-Beta mejora significativamente la eficiencia del algoritmo Minimax, ya que permite evitar la exploración de ramas que no pueden influir en la decisión final. Esto se traduce en una reducción importante del número de nodos evaluados, lo que en la práctica implica una mejora sustancial en los tiempos de ejecución. Teóricamente, bajo un buen orden de exploración, la complejidad puede pasar de $\mathcal{O}(b^d)$ a $\mathcal{O}(b^{d/2})$, lo que explica las diferencias observadas empíricamente. En particular, para profundidad 4 sin poda, no fue posible completar la ejecución debido al tiempo excesivo que tomaba: luego de más de 15 minutos por jugador aún no finalizaba, por lo que debimos detener el experimento. Al revisar los tiempos parciales, observamos que algunas jugadas individuales incluso superaban los 1000 segundos, lo cual vuelve inviable su uso práctico. Esta situación contrasta fuertemente con los resultados con poda en la misma profundidad, reafirmando empíricamente el valor de la poda Alfa-Beta en eficiencia sin comprometer la optimalidad de la decisión.

Actividad 2: Comparación de rendimiento

Prof. J1	Prof. J2	Ganó J1	Ganó J2	Empates	Prom. J1 (s)	Prom. J2 (s)
1	1	5	5	0	0.0067	0.0067
1	2	3	7	0	0.0068	0.2125
2	1	7	3	0	0.2406	0.0071
1	3	3	6	1	0.0074	7.4546
3	1	5	4	1	9.4901	0.0095

Los resultados experimentales muestran que el aumento en la profundidad del algoritmo Minimax incrementa significativamente el tiempo promedio por jugada, lo cual es coherente con su complejidad teórica $\mathcal{O}(b^d)$, donde b es el branching factor y d la profundidad. Mientras que profundidades bajas como 1 permiten decisiones inmediatas, profundidades mayores

como 3 generan tiempos promedio superiores a varios segundos por jugada. Esta diferencia computacional se traduce en un desempeño estratégico superior: los agentes con mayor profundidad vencen consistentemente a los de menor profundidad, al anticipar de mejor forma amenazas y oportunidades en el tablero. La ventaja persiste incluso cuando el agente más profundo juega en segundo lugar, lo que evidencia el impacto directo de la profundidad sobre la calidad del juego en entornos adversariales.

Actividad 3: Nueva función de evaluación

Config.	Eval J1	Prof. J1	Eval J2	Prof. J2	Ganó J1	Ganó J2	Empates	Prom. J1 / J2 (s)
1	Defensiva	1	ChatGPT	1	0	10	0	0.0080 / 0.0052
2	ChatGPT	2	Defensiva	2	10	0	0	0.1968 / 0.2998
3	Defensiva	1	ChatGPT	3	0	10	0	0.0081 / 6.0892
4	Defensiva	3	ChatGPT	1	0	10	0	11.7252 / 0.0063

Cuadro 1: Comparación entre funciones de evaluación `defensive_simple_score()` y `chat_gpt_eval()` en diferentes configuraciones de profundidad y rol.

Los resultados indican que la función `chat_gpt_eval()` obtiene un desempeño claramente superior en todas las configuraciones, independiente del orden de turno o la profundidad de su oponente. Incluso cuando enfrenta a un agente que evalúa más profundo con la heurística defensiva, logra ganar todas las partidas, lo que sugiere que captura patrones estratégicos de mayor impacto que la simple prevención de amenazas inmediatas. Sin embargo, esta mejora en la calidad viene acompañada de un incremento sustancial en el tiempo promedio por jugada, en especial cuando se combina con profundidades altas: en la configuración 3, su promedio supera los 6 segundos, y en la configuración 4 el agente defensivo llega a promediar más de 11 segundos por turno. Estos tiempos reflejan que `chat_gpt_eval()` probablemente realiza una evaluación más costosa computacionalmente, pero más informada, lo que resulta en decisiones significativamente más efectivas en contextos competitivos.

Actividad 4: Nueva función de evaluación

N° Partida	Ganador	Prom. J1 (s)	Prom. J2 (s)
1	J1	0.013	0.007
2	J1	0.012	0.007
3	J1	0.012	0.006
4	J1	0.012	0.007
5	J1	0.013	0.006
6	J1	0.012	0.006
7	J1	0.012	0.006
8	J1	0.012	0.007
9	J1	0.012	0.006
10	J2	0.011	0.007
Totales	**9–1**	**0.0122**	**0.0065**

La función de evaluación puntaje_centralizado fue implementada con el objetivo de integrar de forma balanceada componentes ofensivos, defensivos y posicionales en la estimación del valor heurístico de un estado. Esta heurística otorga mayor peso a las fichas ubicadas en las columnas centrales del tablero, bonifica alineaciones propias de dos o más fichas consecutivas y penaliza configuraciones similares del oponente. Con este diseño se busca promover estructuras de juego estables y tácticamente ventajosas, incluso con profundidades de búsqueda acotadas. En el experimento realizado, el agente Minimax que utilizó puntaje_centralizado enfrentó al agente basado en defensive_simple_score en diez partidas, ambas con profundidad 1. El agente con la nueva función ganó nueve de ellas, alcanzando una tasa de victoria del 90 %, lo que cumple con holgura el criterio mínimo del 70 % exigido en el enunciado. Los tiempos promedio de decisión se mantuvieron por debajo de 13 milisegundos, lo que demuestra que el beneficio estratégico obtenido no implica un costo computacional significativo en contextos prácticos de uso.

Actividad 5: Extender el juego a 3 jugadores

El algoritmo `max_n`, implementado en el archivo `algorithms/max_n.py`, extiende correctamente la lógica de Minimax a un entorno de tres jugadores racionales que compiten simultáneamente. A diferencia del caso tradicional de dos jugadores, en que uno maximiza y otro minimiza, en este enfoque cada agente busca maximizar su propia utilidad, evaluada mediante una función heurística vectorial que retorna una estimación de utilidad en $\mathbb{R}^N$, donde $N = 3$. La rotación de turnos se realiza cíclicamente con `next_player = (player_id % player_amount) + 1`, garantizando que todos los jugadores participen de forma equitativa. Para validar el comportamiento del algoritmo, se ejecutó el archivo `main_3_players.py` con un tablero de dimensiones 5×6, utilizando tres instancias de `Max3Player` con la heurística `chat_gpt_eval_3` y profundidad 2. El juego se resolvió correctamente, con turnos alternados entre los tres agentes y sin errores de ejecución. Los tiempos de decisión fueron consistentes con la complejidad esperada del árbol de juego multijugador: los jugadores J1 y J2 promediaron aproximadamente 2 milisegundos por jugada, mientras que J3, con profundidad mayor, alcanzó tiempos de decisión de hasta 0.97 segundos en promedio. El tablero final fue mostrado correctamente y el juego concluyó con un único ganador, validando la compatibilidad completa del algoritmo Max^{N} con la infraestructura provista.