Proyecto 2:
BÚSQUEDA MULTIAGENTE EN CONNECT-4

Andres Camilo Orduz Lunar

1. DEFINICIÓN DEL PROBLEMA (ANTECEDENTES)

1.1 Descripción del Juego Connect-4

Connect-4 es un juego de información perfecta, ya que ambos jugadores pueden observar en todo momento el estado completo del tablero sin ningún tipo de información oculta. Además, no contiene elementos aleatorios: cada acción tiene un resultado completamente determinista, pues colocar una ficha en una columna siempre produce la misma transición de estado. Estas características hacen que el juego sea ideal para implementar algoritmos de búsqueda adversaria como Minimax, que asumen entornos completamente observables y deterministas. Aunque el juego base no involucra azar, también se implementa Expectimax para modelar oponentes con comportamientos probabilísticos, permitiendo comparar estrategias deterministas contra agentes estocásticos.

Además, Connect-4 posee un espacio de estados muy grande, lo que lo convierte en un juego interesante desde el punto de vista computacional. Cada casilla del tablero puede estar en tres estados posibles (ficha del jugador 1, ficha del jugador 2 o vacío). Considerando que el tablero tiene 7×6 = 42 casillas, un cálculo inicial arroja un límite superior aproximado de:

$$3^{42} \approx 10^{20}$$

Sin embargo, muchas de estas configuraciones son ilegales, pues en una columna no pueden existir casillas vacías por debajo de una casilla ocupada. Al descartar estas posiciones imposibles, estudios clásicos sobre Connect-4 han estimado un límite superior mucho más ajustado, cercano a:

$$7.1 \times 10^{13} \; Posiciones\ legales$$

Este valor sigue siendo extremadamente grande para explorar de manera exhaustiva, lo que justifica el uso de algoritmos adversariales con profundidad limitada y heurísticas de evaluación.[3]

2. DEFINICIÓN DE ALGORITMOS

2.1 Fundamentos Teóricos

2.1.1 Búsqueda Adversarial

La búsqueda adversarial es un paradigma de resolución de problemas donde múltiples agentes con objetivos conflictivos toman decisiones secuencialmente. En juegos de suma cero con información perfecta, el problema se modela como:

- **Estados:** Configuraciones del tablero
- **Jugadores:** MAX (agente) y MIN (oponente)
- **Función de utilidad:** U(s) $\in \mathbb{R}$ que evalúa estados terminales
- **Árbol de juego:** Representación de todas las secuencias de movimientos posibles

2.1.2 Función de Evaluación Heurística

Para estados no terminales, se diseñó una función de evaluación heurística **h(s)** que estima la utilidad de un estado basándose en características del tablero.

Componentes de la función:

1. **Control del centro (peso: 3 puntos/ficha):**

```
score += count(fichas_propias_en_columna_central) × 3
```

1. **Evaluación de ventanas de 4 casillas:**

- 4 fichas propias: +1000 (victoria)
- 3 fichas propias + 1 vacía: +10 (amenaza fuerte)
- 2 fichas propias + 2 vacías: +5 (potencial)
- 3 fichas oponente + 1 vacía: -80 (amenaza crítica)

1. **Direcciones evaluadas:**

- Horizontales: 24 ventanas (6 filas × 4 posiciones)
- Verticales: 21 ventanas (7 columnas × 3 posiciones)
- Diagonales: 24 ventanas (12 por cada tipo)

Función de evaluación final:

$$E(s) = \begin{cases} +\infty & \text{si MAX ganó} \\ -\infty & \text{si MIN ganó} \\ 0 & \text{si empate} \\ h(s) & \text{en otro caso} \end{cases}$$

2.2 Algoritmo Minimax con Poda Alfa-Beta

2.2.1 Principio Fundamental

Minimax asume que ambos jugadores juegan de manera óptima. MAX busca maximizar la utilidad, mientras MIN busca minimizarla. El valor Minimax de un estado se define recursivamente:

```
            ⎧ max_{a} Minimax(Resultado(s,a))  si es turno de MAX
Minimax(s) =  ⎨
            ⎩ min_{a} Minimax(Resultado(s,a))  si es turno de MIN
```

2.2.2 Poda Alfa-Beta

La poda Alfa-Beta es una optimización que elimina ramas del árbol de búsqueda que no pueden influir en la decisión final, manteniendo los parámetros:

- **α:** Mejor valor para MAX encontrado hasta ahora
- **β:** Mejor valor para MIN encontrado hasta ahora

Condición de poda: Si α ≥ β, se detiene la exploración de esa rama.

2.2.3 Pseudocódigo Implementado

```
function MINIMAX(estado, profundidad, α, β, esMaximizador):
    si profundidad = 0 o es_terminal(estado):
        return evaluar(estado)

    movimientos ← obtener_movimientos_válidos(estado)

    si esMaximizador:
        valor ← -∞
        para cada movimiento en movimientos:
            hijo ← aplicar_movimiento(estado, movimiento, MAX)
            valor ← max(valor, MINIMAX(hijo, profundidad-1, α, β, falso))
            α ← max(α, valor)
            si α ≥ β:
                break  // Poda Beta
        return valor
    sino:
        valor ← +∞
        para cada movimiento en movimientos:
            hijo ← aplicar_movimiento(estado, movimiento, MIN)
            valor ← min(valor, MINIMAX(hijo, profundidad-1, α, β, verdadero))
            β ← min(β, valor)
            si α ≥ β:
```

```
            break  // Poda Alfa
    return valor
```

2.3 Algoritmo Expectimax

2.3.1 Principio Fundamental

Expectimax modifica Minimax para manejar oponentes que no juegan óptimamente. En lugar de asumir que MIN siempre elige el peor movimiento para MAX, calcula el **valor esperado** asumiendo una distribución de probabilidad sobre las acciones del oponente.

Diferencia clave:

```
Minimax:    Valor_MIN = min_{a} Valor(hijo_a)
Expectimax: Valor_MIN = Σ P(a) × Valor(hijo_a)
```

En la implementación, asumimos distribución uniforme: P(a) = 1/n para n movimientos válidos.

2.3.2 Casos de Uso

- **Minimax:** Torneos, juego contra IA, análisis teórico
- **Expectimax:** Juego contra humanos, oponentes impredecibles, entornos con incertidumbre

2.3.3 Pseudocódigo Implementado

```
function EXPECTIMAX(estado, profundidad, esMaximizador):
    si profundidad = 0 o es_terminal(estado):
        return evaluar(estado)

    movimientos ← obtener_movimientos_válidos(estado)

    si esMaximizador:
        valor ← -∞
        para cada movimiento en movimientos:
            hijo ← aplicar_movimiento(estado, movimiento, MAX)
            valor ← max(valor, EXPECTIMAX(hijo, profundidad-1, falso))
        return valor
    sino:
        // Nodo de azar: calcular valor esperado
        total ← 0
        para cada movimiento en movimientos:
            hijo ← aplicar_movimiento(estado, movimiento, MIN)
```

```
        total ← total + EXPECTIMAX(hijo, profundidad-1, verdadero)
    return total / |movimientos|
```

Nota: A diferencia de Minimax, Expectimax no permite aplicar poda Alfa-Beta. Esto se debe a que los nodos de Expectimax calculan valores esperados mediante promedios ponderados, no máximos o mínimos estrictos. Como cada hijo contribuye al valor esperado, no es posible descartar ramas sin alterar el resultado final, lo que impide realizar cortes garantizados como en la poda Alfa-Beta.

3. RESULTADOS (EJECUCIONES DEL ALGORITMO)

Con el fin de evaluar el desempeño de los algoritmos **Minimax** y **Expectimax** en el entorno del juego Connect-4, se realizaron **10 ejecuciones completas** del módulo de experimentos.

Cada ejecución consiste en **10 partidas por algoritmo**, enfrentando:

- **MAX:** Minimax / Expectimax
- **MIN:** agente completamente aleatorio (Random)

En total, cada algoritmo jugó **100 partidas**.

3.1. Desempeño del algoritmo Minimax

Los resultados del agente Minimax fueron completamente consistentes en todas las ejecuciones:

- En las 10 ejecuciones obtuvo **10 victorias cada vez**.

Totales globales (100 partidas):

- **100 victorias**
- **0 derrotas**
- **0 empates**

Esto significa un rendimiento **perfecto (100%)** frente a un oponente aleatorio.

La estabilidad de Minimax era esperada, ya que su naturaleza adversarial y su poda alfa-beta le permiten evitar jugadas peligrosas y diseñar una estrategia sólida incluso con profundidad limitada.

3.2. Desempeño del algoritmo Expectimax

Los resultados muestran una mayor variabilidad, coherente con su naturaleza probabilística. A continuación se listan los resultados por ejecución:

Ejecución	Victorias	Derrotas	Empates
1	7	3	0
2	10	0	0
3	9	1	0
4	10	0	0
5	8	2	0
6	9	1	0
7	8	2	0
8	9	1	0
9	9	1	0
10	8	2	0

Totales globales (100 partidas):

- **87 victorias**
- **13 derrotas**
- **0 empates**

Porcentajes:

- **87% victorias**
- **13% derrotas**

Expectimax, al basarse en valores esperados en lugar del peor caso, puede subestimar amenazas del oponente. Esto explica la aparición de derrotas ocasionales incluso frente a un agente no estratégico.

Aun así, mantiene un rendimiento elevado, muy por encima del 80%.

3.3. Comparación general

Algoritmo	Partidas jugadas	Victorias	Derrotas	Empates	Rendimiento
Minimax	100	100	0	0	**100%**
Expectimax	100	87	13	0	**87%**

Conclusiones de los resultados

- **Minimax** demuestra ser mucho más estable y seguro. Su estrategia basada en el peor caso evita por completo derrotas frente a un oponente aleatorio.

- **Expectimax** logra muy buen rendimiento, pero su falta de defensa frente a movimientos adversos genera derrotas que Minimax nunca comete.
- La diferencia entre ambos algoritmos evidencia la importancia del modelo del oponente:
  - Minimax asume un **oponente perfecto**, es decir, nunca se expone.
  - Expectimax asume un **oponente aleatorio**, es decir, puede tomar riesgos que terminan en derrotas.

4. Conclusiones y enlace al repositorio

El desarrollo del proyecto permitió analizar el funcionamiento y desempeño de dos algoritmos de búsqueda multiagente ampliamente utilizados en juegos deterministas con información perfecta: **Minimax** y **Expectimax**. A partir de su implementación y evaluación en el juego Connect-4, se pueden destacar las siguientes conclusiones:

1. Minimax es más estable y robusto que Expectimax

El algoritmo Minimax obtuvo un rendimiento **perfecto** en las 100 partidas ejecutadas, sin presentar derrotas ni empates.

Esto confirma su capacidad para evitar escenarios peligrosos mediante una evaluación basada en el **peor caso**, lo cual es ideal en entornos competitivos y adversariales.

2. Expectimax tiene buen desempeño, pero es más vulnerable

Aunque Expectimax logró un **87% de victorias**, presentó 13 derrotas, todas frente a un oponente completamente aleatorio.

Esto demuestra que su supuesto central (que el oponente actúa aleatoriamente) puede conducir a movimientos arriesgados o sobreoptimistas en posiciones críticas.

3. La naturaleza del modelo del oponente cambia drásticamente la estrategia

- Minimax asume un oponente óptimo → genera jugadas más defensivas, seguras y conservadoras.
- Expectimax asume un oponente aleatorio → juega de forma más flexible, pero puede subestimar amenazas.

Esta diferencia conceptual explica los resultados observados.

4. El dominio Connect-4 es adecuado para comparar algoritmos adversariales

El juego ofrece:

- decisiones secuenciales,

- un espacio de estados grande,
- juego determinista,
- información perfecta.

Esto permite observar claramente diferencias entre estrategias deterministas y probabilísticas.

5. La función de evaluación es fundamental

Ambos algoritmos dependen directamente de la calidad de la heurística, especialmente porque se usan profundidades limitadas.

Incluso una heurística relativamente simple fue suficiente para lograr altos niveles de desempeño, especialmente en el caso de Minimax.

4.1 Enlace al repositorio

https://github.com/ACorduz/Connect-4---Proyecto2IA

5. Referencias

[1] Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. Capítulos 5 y 6: Búsqueda Adversarial y Juegos.

[2] UC Berkeley – CS188: Introduction to Artificial Intelligence. *Lecture 6: Adversarial Search*.

Disponible en: https://inst.eecs.berkeley.edu/~cs188/fa25/

[3] Allis, L. V. (1988). *Solving Connect Four*. MIT Seminar Slides.

Disponible en: https://web.mit.edu/sp.268/www/2010/connectFourSlides.pdf

[4] Gamesolver.org. *Solving Connect Four – Introduction*.

Disponible en: http://blog.gamesolver.org/solving-connect-four/01-introduction/

[5] GeeksforGeeks. *Adversarial Search Algorithms*.

Disponible en: https://www.geeksforgeeks.org/artificial-intelligence/adversarial-search-algorithms/

6. Anexos

Video Funcionamiento: https://drive.google.com/file/d/16KOjGe0imeag-HwlGBtjfkTz3pr5yXrp/view?usp=sharing