Connect 4

**Tìm kiếm Adversarial: Chơi Connect 4**

# **Giới thiệu và Quy tắc Connect 4**

Connect 4 là một trò chơi bảng kết nối hai người chơi.

* Bàn cờ: Trò chơi diễn ra trên một lưới được treo thẳng đứng, có sáu hàng (ROW = 6) và bảy cột (COLUMN = 7). Kích thước bàn cờ tiêu chuẩn này (6x7) được sử dụng cho việc phân tích độ phức tạp của thuật toán.
* Quân cờ và Lượt chơi:
  + Người chơi luân phiên nhau thả các đĩa màu (quân cờ) vào các cột.
  + Các quân cờ đại diện cho người chơi được ký hiệu bằng số nguyên, ví dụ Người chơi 1 là 1 và Người chơi 2 là 2 (hoặc -1 trong một số cài đặt sau này).
  + Các ô trống được ký hiệu là 0 (EMPTY).
* Nguyên tắc Trọng lực: Các quân cờ luôn rơi thẳng xuống, chiếm không gian trống thấp nhất có sẵn trong cột mà chúng được thả vào. Đây là một ràng buộc quan trọng của trò chơi.
* Mục tiêu Chiến thắng: Mục tiêu của trò chơi là trở thành người đầu tiên tạo ra một hàng gồm bốn quân cờ của mình theo chiều ngang, dọc, hoặc đường chéo. Chiều dài chiến thắng (WIN\_LENGTH) được định nghĩa là 4.

# **Task 1: Định nghĩa Vấn đề Tìm kiếm**

## 2.1. Các thành phần

* Trạng thái ban đầu(Initial state): Ma trận 6x7 toàn 0(EMPTY)
* Hành động (Actions): Thả quân cờ vào cột chưa đầy
* Mô hình chuyển đổi (Results): Trạng thái mới sau khi thả quân
* Trạng thái mục tiêu (Terminal/Goal): Kết thúc khi có người thắng (winning\_move) hoặc Kết thúc khi bàn cờ đầy (Hoà).

## 2.2. Ước tính Không gian Trạng thái

Ước tính thô này là một phép tính "ngây thơ" (naive) nhằm xác định giới hạn trên (upper bound) cho tổng số cấu hình bàn cờ có thể có, bằng cách bỏ qua tất cả các luật chơi của Connect 4.

### a. Phương pháp Ước tính Thô (Naive Upper Bound)

Ước tính thô được xây dựng dựa trên các yếu tố sau:

* Kích thước Bàn cờ: Bàn cờ Connect 4 tiêu chuẩn có 6 hàng (ROW = 6) và 7 cột (COLUMN = 7).
* Tổng số Ô: Tổng số ô trên bàn cờ là 6×7=42 ô (TOTAL\_CELLS = 42).
* Trạng thái mỗi Ô: Mỗi ô trên bàn cờ có 3 khả năng trạng thái có thể có:
  + Trống (EMPTY).
  + Chứa quân cờ của Người chơi 1 (P1).
  + Chứa quân cờ của Người chơi 2 (P2).
* Công thức tính toán: Vì mỗi ô có 3 trạng thái và có 42 ô độc lập, số trạng thái tối đa được tính bằng 3^42.
* Kết quả: Số trạng thái tối đa (Ước tính thô) là: 109,418,989,131,512,359,209. Con số này xấp xỉ 109 nghìn tỷ tỷ.

### b. Sự khác biệt với thực tế

Đây là một sự phóng đại cực lớn bởi vì nó bao gồm vô số trạng thái mà không bao giờ có thể đạt được trong trò chơi thực tế do các ràng buộc luật chơi.

Các ràng buộc bị bỏ qua trong ước tính thô bao gồm:

* Ràng buộc Trọng lực (Gravity Constraint): Đây là ràng buộc quan trọng nhất. Nó quy định rằng một quân cờ không thể "lơ lửng" ở một ô mà ô ngay dưới nó còn trống. Ví dụ, trạng thái [Trống, Player1] trong một cột là không thể, nó phải là [Player1, Trống].
* Ràng buộc Lượt chơi (Turn-based Constraint): Số lượng quân cờ của Người chơi 1 (đi trước) và Người chơi 2 phải tuân theo quy tắc: (Số quân P1) = (Số quân P2) hoặc (Số quân P1) = (Số quân P2) + 1.
* Ràng buộc Trạng thái Kết thúc (Terminal States Constraint): Trò chơi kết thúc ngay khi có người thắng.

Khi áp dụng các ràng buộc này, số lượng trạng thái thực tế của Connect 4 được tính toán chính xác là 4,531,985,219,092 (khoảng 4.5 nghìn tỷ). Con số này cho thấy ước tính thô đã phóng đại không gian trạng thái lên hàng chục tỷ lần.

## 2.3. Ước tính Cây Trò chơi (Game Tree)

Ước tính Cây Trò chơi khác biệt và lớn hơn nhiều so với ước tính Không gian Trạng thái, bởi vì nó không chỉ đếm các trạng thái hợp lệ mà còn đếm các chuỗi nước đi (paths) để đi đến trạng thái đó.

### a. Các Thông số Ước tính Cây Trò chơi

Kích thước của cây trò chơi được ước tính bằng cách sử dụng công thức chuẩn cho độ phức tạp của trò chơi: b^d.

* Yếu tố Rẽ nhánh (b - Branching Factor): Đại diện cho số lượng nước đi trung bình (hoặc tối đa) hợp lệ từ một trạng thái.
  + Connect 4 có 7 cột.
  + Để ước tính giới hạn trên (upper-bound), yếu tố rẽ nhánh tối đa được sử dụng là 7.
* Độ sâu (d - Depth): Đại diện cho độ dài tối đa của một ván cờ tính bằng số lượt đi.
  + Bàn cờ có 6×7=42 ô.
  + Độ sâu tối đa của cây trò chơi là 42 nước đi (khi toàn bộ bàn cờ được lấp đầy).

### b. Ước tính Độ lớn của Cây Trò chơi

Áp dụng công thức b^d:

* Kích thước ước tính số lượng nút lá (leaf nodes), đại diện cho các ván cờ hoàn chỉnh có thể xảy ra, là 7^42.
* Kết quả này là một con số cực kỳ lớn, xấp xỉ 5.5 x 10^35 hay xấp xỉ 550 nghìn tỷ tỷ tỷ.

### c. Sự khác biệt so với Không gian Trạng thái

Có sự khác biệt đáng kể giữa kích thước của Không gian Trạng thái (đã tính toán chính xác là xấp xỉ 4.5 nghìn tỷ) và kích thước của Cây Trò chơi (xấp xỉ 550 nghìn tỷ tỷ tỷ).

* Lý do cốt lõi: Các thuật toán Minimax (nếu không sử dụng bảng hoán vị) phải khám phá chuỗi các con đường (sequences of moves), chứ không phải chỉ các trạng thái (states).
* Cùng một trạng thái bàn cờ có thể xuất hiện nhiều lần ở các độ sâu khác nhau hoặc thông qua các trình tự nước đi khác nhau. Thuật toán Minimax sẽ khám phá tất cả các con đường đó một cách riêng biệt, dẫn đến sự bùng nổ tổ hợp này.

### d. Kết luận cho Triển khai AI

Việc định nghĩa vấn đề tìm kiếm (Task 1) đã cung cấp các hàm nền tảng như initial\_state(), actions(board), và result(board, ...). Tuy nhiên, kết quả ước tính Cây Trò chơi cho thấy rằng, mặc dù định nghĩa là chính xác, việc chạy thuật toán Minimax để duyệt toàn bộ cây trò chơi là không khả thi. Con số 7^42 được gọi là bùng nổ tổ hợp (combinatorial explosion).

Do đó, ta kết luận rằng để AI có thể chơi Connect 4 một cách hiệu quả, cần phải sử dụng các kỹ thuật tiên tiến hơn:

* Tìm kiếm có Giới hạn Độ sâu (Depth-Limited Search): Chỉ nhìn trước một số lượng nước đi giới hạn (ví dụ: d=8 hoặc 10) thay vì d=42. Điều này giảm kích thước cây con cần duyệt xuống mức quản lý được (ví dụ: 7^10 ≈ 282 triệu).
* Hàm Lượng giá Heuristic (Heuristic Evaluation Function): Cần thiết để "chấm điểm" các trạng thái ở độ sâu cắt (cut off depth) vì thuật toán không thể tìm đến trạng thái kết thúc.
* Cắt tỉa Alpha-Beta (Alpha-Beta Pruning): Một kỹ thuật tối ưu hóa giúp cắt bỏ các nhánh của cây tìm kiếm chắc chắn không dẫn đến kết quả tốt nhất, giúp giảm đáng kể thời gian tìm kiếm.

# **Task 2: Môi trường Game và Tác nhân Ngẫu nhiên (Random Agent)**

Task 2 tập trung vào việc thiết lập Môi trường Trò chơi (Game Environment) để kiểm thử các tác nhân AI, bắt đầu với một Tác nhân Ngẫu nhiên (Random Agent), trong bối cảnh rộng hơn của Connect 4 và Tìm kiếm Đối kháng.

Nhiệm vụ này đã định nghĩa lại cách các tác nhân tương tác với trò chơi và cung cấp bằng chứng thực nghiệm về lợi thế của người chơi đi trước.

## 3.1. Định nghĩa Môi trường và Cấu trúc Dữ liệu

Task 2 đã củng cố định nghĩa về Connect 4 và thiết lập một giao diện cho các tác nhân AI:

* Sử dụng NumPy Array: Bàn cờ được triển khai bằng NumPy array (mảng ký tự).
* Ký hiệu Người chơi: Để thuận tiện cho tính toán thuật toán, người chơi được ký hiệu bằng số nguyên, thay vì 1 và 2 như trong Task 1:
  + Người chơi 1 (Đỏ) được ký hiệu là 1 (PLAYER\_1\_PIECE).
  + Người chơi 2 (Vàng) được ký hiệu là -1 (PLAYER\_2\_PIECE).
  + Ô trống vẫn là 0 (EMPTY).
* Chữ ký Hàm Tác nhân: Các nguồn quy định rõ ràng rằng tất cả các hàm tác nhân (Agent functions) phải có chữ ký nhất quán: agent\_type(board, player = 1).
  + board là trạng thái bàn cờ hiện tại (percept).
  + player là quân cờ của người chơi mà tác nhân đang chơi (1 hoặc -1).

Các hàm hỗ trợ được định nghĩa trong Task 1 (actions, result, terminal, utility) đã được sử dụng lại và tinh chỉnh để làm việc với môi trường game này.

## 3.2. Triển khai Tác nhân Ngẫu nhiên (Random Agent)

Tác nhân Ngẫu nhiên (random\_player) là tác nhân cơ sở đầu tiên được triển khai.

* Mục tiêu: Tác nhân này đơn giản là chọn một hành động hợp lệ một cách ngẫu nhiên từ trạng thái bàn cờ hiện tại.
* Logic: Hàm này trước hết gọi hàm actions(board) để lấy danh sách tất cả các cột có thể thả quân cờ. Sau đó, nó sử dụng thư viện random để chọn ngẫu nhiên một chỉ số cột từ danh sách các hành động hợp lệ đó.
* Đầu ra: Nó trả về một chỉ số cột (hành động) hợp lệ.

## 3.3. Mô phỏng Môi trường (Game Loop)

Để cho hai tác nhân chơi với nhau, một hàm play\_game đã được tạo ra, hoạt động như một môi trường (environment):

* Quản lý lượt chơi: Hàm play\_game khởi tạo bàn cờ trống và bắt đầu với Người chơi 1 (PLAYER\_1\_PIECE).
* Vòng lặp: Vòng lặp trò chơi tiếp diễn chừng nào trạng thái chưa phải là trạng thái kết thúc (while not terminal(board)).
* Chuyển đổi Trạng thái: Sau khi nhận được nước đi (move) từ tác nhân tương ứng, hàm cập nhật trạng thái bàn cờ bằng cách gọi hàm result(board, current\_player, move).
* Đổi lượt: Lượt chơi được chuyển bằng cách nhân quân cờ hiện tại với -1 (current\_player = -current\_player).
* Kết quả: Hàm trả về người thắng (1 hoặc -1) hoặc 0 nếu hòa (EMPTY).

## 3.4. Kết quả Thực nghiệm (Hai Tác nhân Ngẫu nhiên)

Phần cuối của Task 2 yêu cầu chạy 1000 ván đấu giữa hai tác nhân ngẫu nhiên (random\_player vs random\_player) để kiểm tra phân phối thắng thua.

Kết quả mô phỏng (1000 ván):

| Kết quả | Số ván thắng | Tỷ lệ thắng |
| --- | --- | --- |
| Người chơi 1 (Đỏ) | 561 | 56.1% |
| Người chơi 2 (Vàng) | 437 | 43.7% |
| Hòa | 2 | 0.2% |

Phân tích Kết quả:

* Tác nhân Ngẫu nhiên không hoàn toàn ngẫu nhiên về kết quả: Mặc dù cả hai tác nhân đều chọn nước đi ngẫu nhiên, Người chơi 1 thắng áp đảo (56.1%).
* Kết quả có thể dự đoán được (Expected Result): Kết quả này được dự đoán trước (expected) do lợi thế của người chơi đi trước (first-move advantage) trong Connect 4. Người chơi đi trước có thể bắt đầu bằng cách chiếm các vị trí trung tâm chiến lược, thường dẫn đến tỷ lệ thắng cao hơn so với người chơi đi sau trong các ván đấu ngẫu nhiên.

# **Task 3: Tìm kiếm Minimax với Alpha-Beta Pruning**

Task 3 trong bối cảnh Tìm kiếm Đối kháng (Adversarial Search): Chơi Connect 4 tập trung vào việc triển khai và tối ưu hóa thuật toán Tìm kiếm Minimax với Kỹ thuật Cắt tỉa Alpha-Beta (Minimax Search with Alpha-Beta Pruning) để tạo ra một tác nhân AI hiệu quả hơn Tác nhân Ngẫu nhiên (Random Agent).

## 4.1. Triển khai Thuật toán Minimax với Alpha-Beta Pruning

Task 3 yêu cầu triển khai Minimax, một thuật toán đệ quy tìm kiếm giá trị tối ưu (value) và nước đi tốt nhất (column) cho người chơi hiện tại, giả định rằng đối thủ sẽ chơi tối ưu để giảm thiểu điểm số của mình.

#### a. Cấu trúc Tác nhân (MinimaxAgent Class)

Tác nhân được triển khai dưới dạng một lớp (MinimaxAgent) để lưu trữ thông tin trạng thái nội tại, bao gồm quân cờ (piece) và độ sâu tìm kiếm (search\_depth).

#### b. Thuật toán Minimax (\_minimax function)

Thuật toán đệ quy này bao gồm logic cắt tỉa Alpha-Beta.

* Alpha (α): Giá trị tối thiểu mà người chơi tối đa hóa (Maximizer) đã đảm bảo có thể đạt được.
* Beta (β): Giá trị tối đa mà người chơi tối thiểu hóa (Minimizer) đã đảm bảo có thể đạt được.
* Cắt tỉa: Nếu alpha lớn hơn hoặc bằng beta (alpha >= beta), nhánh tìm kiếm hiện tại sẽ bị cắt bỏ (pruned), vì người chơi Maximizer đã tìm thấy một nước đi tốt hơn ở các nhánh khác, hoặc Minimizer đã tìm thấy một nước đi khiến Maximizer không bao giờ đạt được giá trị mong muốn ở nhánh này.

## 4.2. Sự Cần thiết của Giới hạn Độ sâu và Heuristic

Do sự bùng nổ tổ hợp (combinatorial explosion) của Cây Trò chơi Connect 4 nên việc duyệt toàn bộ cây là không khả thi.

Vì vậy, Minimax phải được triển khai như một thuật toán Tìm kiếm có Giới hạn Độ sâu (Depth-Limited Search).

#### a. Cắt tìm kiếm (Cutoff Search)

Tìm kiếm sẽ dừng lại khi đạt đến search\_depth được định nghĩa trước (ví dụ: d=4 hoặc d=5) hoặc khi đạt đến một trạng thái kết thúc (is\_terminal\_node).

#### b. Hàm Lượng giá (Heuristic Evaluation Function)

Khi tìm kiếm dừng lại ở độ sâu cắt (depth == 0) mà không phải là trạng thái kết thúc, thuật toán phải sử dụng Hàm Lượng giá Heuristic (\_score\_position) để "chấm điểm" trạng thái bàn cờ hiện tại.

Hàm heuristic được thiết kế để:

* Thưởng điểm Trung tâm: Thêm điểm thưởng cho các quân cờ ở cột giữa, vì cột giữa là vị trí chiến lược quan trọng.
* Đánh giá Cửa sổ: Chấm điểm các "cửa sổ" (window) 4 ô dựa trên số lượng quân cờ của người chơi và đối thủ trong đó.
* Hệ số Điểm: Các mối đe dọa trực tiếp được gán điểm cao:
  + 4 quân liên tiếp (Thắng): 1,000,000 (SCORE\_4\_IN\_ROW) (hoặc 10,000,000 khi trả về từ Minimax).
  + 3 quân liên tiếp + 1 ô trống: 100 (SCORE\_3\_IN\_ROW).
  + Chặn 3 quân liên tiếp của đối thủ: -80 (SCORE\_OPPONENT\_3).

## 4.3. Tối ưu hóa Hiệu suất: Sắp xếp Nước đi (Move Ordering)

Để tăng cường hiệu quả của cắt tỉa Alpha-Beta, Task 3 đề xuất sử dụng chiến lược Sắp xếp Nước đi (Move Ordering).

* Mục đích: Cắt tỉa Alpha-Beta hoạt động hiệu quả nhất khi nước đi tốt nhất được đánh giá sớm nhất.
* Chiến lược: Chiến lược đơn giản được mô tả là Center First (Ưu tiên Cột giữa).
  + Danh sách nước đi hợp lệ (valid\_actions\_list) được sắp xếp theo thứ tự ưu tiên (ưu tiên cột 3, sau đó 2 và 4, v.v.).
* Thực nghiệm: Các thử nghiệm so sánh thời gian thực hiện nước đi (trên bàn cờ 6x7) ở các độ sâu khác nhau cho thấy việc sắp xếp nước đi có thể cải thiện đáng kể hiệu suất (ví dụ: cải thiện 52.88% ở độ sâu 4 và 40.09% ở độ sâu 6), mặc dù kết quả có thể không nhất quán do tính ngẫu nhiên của trạng thái "giữa ván" được tạo ra.

## 4.4. Giải quyết Trạng thái Ban đầu: Sách Mở đầu (Opening Book)

Task 3 cũng đề cập đến vấn đề về việc tìm kiếm Minimax ở trạng thái bàn cờ trống là trường hợp tồi tệ nhất.

* Giải pháp: Sử dụng Sách Mở đầu (Opening Book), nơi các nước đi tối ưu cho các trạng thái đầu tiên đã được tính toán trước (offline) và lưu trữ.
* Triển khai: Tác nhân kiểm tra xem trạng thái bàn cờ hiện tại có nằm trong sách mở đầu hay không. Nếu có, nó sẽ trả về nước đi đã được tính toán trước (cột 3) ngay lập tức mà không cần chạy thuật toán Minimax, giúp tiết kiệm thời gian đáng kể.

## 4.5. Kết quả và Độ phức tạp Tính toán

Các thử nghiệm trong Task 3 minh họa rõ ràng độ phức tạp tính toán:

Thời gian tìm kiếm: Thời gian để tác nhân Minimax đưa ra một nước đi tăng nhanh chóng khi tăng độ sâu tìm kiếm hoặc tăng kích thước bàn cờ (yếu tố rẽ nhánh b).

* 1. Ví dụ, ở độ sâu 5, thời gian cho nước đi tăng từ 0.03 giây (bàn cờ 4x4) lên 0.63 giây (bàn cờ 6x7).
  2. Việc tăng độ sâu từ 6 lên 7 trên bàn cờ 6x7 cũng làm tăng thời gian tìm kiếm đáng kể.

Kết luận về Tính khả thi: Do độ phức tạp, việc sử dụng Minimax trên bàn cờ 6x7 là không khả thi nếu không có giới hạn độ sâu (ví dụ, chỉ tìm kiếm tối đa 8-10 nước đi) và các kỹ thuật tối ưu hóa (Alpha-Beta, Heuristic, Move Ordering).

Hiệu suất: Minimax Agent (d=5) đã đánh bại Random Agent khi chơi P1 và P2 trên bàn cờ 6x7 trong các thử nghiệm đơn lẻ. Trong thử nghiệm với các trạng thái chiến thuật cố định, tác nhân Minimax đã xác định chính xác nước đi chiến thắng (tạo 4 quân) hoặc nước đi chặn đối thủ (chặn 3 quân).

# **Task 4: Tìm kiếm Heuristic Alpha-Beta**

Task 4: Tìm kiếm Heuristic Alpha-Beta là đỉnh cao của việc áp dụng các kỹ thuật Tìm kiếm Đối kháng (Adversarial Search) cho Connect 4, kết hợp thuật toán Minimax được tối ưu hóa bằng Cắt tỉa Alpha-Beta với Giới hạn Độ sâu (Cutoff Search) và Hàm Lượng giá Heuristic (Heuristic Evaluation Function).

Việc này là cần thiết do kích thước khổng lồ của cây trò chơi khiến việc tìm kiếm toàn bộ là bất khả thi.

### 5.1. Hàm Lượng giá Heuristic (\_score\_position)

Task 4 bắt đầu bằng việc định nghĩa và triển khai một hàm heuristic, được sử dụng để ước tính giá trị của một trạng thái khi tìm kiếm bị cắt ở độ sâu giới hạn.

#### a. Định nghĩa Điểm số

Các điểm số được định nghĩa để ưu tiên các nước đi tấn công và phòng thủ:

Chiến thắng (SCORE\_WIN): 1,000,000. Giá trị này phải lớn hơn nhiều so với bất kỳ tổ hợp điểm heuristic nào khác để đảm bảo AI ưu tiên thắng tuyệt đối.

3 quân liên tiếp + 1 ô trống (SCORE\_3\_IN\_ROW): 100. Đây là mối đe dọa tấn công lớn.

2 quân liên tiếp + 2 ô trống (SCORE\_2\_IN\_ROW): 10.

Thưởng điểm Trung tâm (SCORE\_CENTER\_BONUS): 3 điểm cho mỗi quân cờ ở cột giữa.

Chặn 3 quân của đối thủ (SCORE\_BLOCK\_3): -80. Điểm tiêu cực này cho thấy việc chặn mối đe dọa của đối thủ là quan trọng hơn việc tạo 2 quân cho mình, nhưng vẫn thấp hơn điểm thắng tuyệt đối.

Chặn 2 quân của đối thủ (SCORE\_BLOCK\_2): -5.

#### b. Cơ chế Đánh giá (Evaluate Window)

Hàm heuristic hoạt động bằng cách quét toàn bộ bàn cờ (ngang, dọc, chéo chính, chéo phụ) và chấm điểm cho mỗi "cửa sổ" 4 ô (\_evaluate\_window). Tổng điểm của tất cả các cửa sổ, cộng với điểm thưởng trung tâm, tạo nên điểm heuristic cuối cùng (\_score\_position).

### 5.2. Cắt tìm kiếm và Minimax (Cutting Off Search)

Task 4 sửa đổi thuật toán Minimax của Task 3 để chính thức kết hợp việc cắt tìm kiếm.

Điều kiện Dừng: Hàm đệ quy Minimax (\_minimax) dừng lại khi:

Nó đạt đến trạng thái kết thúc (is\_terminal\_node = True), trong trường hợp này, nó trả về điểm tiện ích tuyệt đối.

Nó đạt đến độ sâu cắt (depth == 0), trong trường hợp này, nó trả về điểm do Hàm Heuristic (\_score\_position) cung cấp.

Thử nghiệm Độ sâu Cắt: Việc thử nghiệm với các độ sâu khác nhau (từ 1 đến 6) cho thấy:

Hiệu suất: Thời gian tìm kiếm tăng nhanh theo cấp số nhân khi độ sâu tăng. Độ sâu 1 chỉ mất 0.0000 giây, trong khi độ sâu 6 mất 0.3912 giây trên một trạng thái giữa ván cụ thể.

Chất lượng Nước đi: Các độ sâu khác nhau có thể dẫn đến các nước đi được chọn khác nhau do chất lượng của hàm heuristic ở các điểm cắt khác nhau. Ví dụ, ở độ sâu 5, AI chọn cột 2 (Score: 171), nhưng ở độ sâu 6, nó chọn cột 3 (Score: 134).

### 5.3. Tối ưu hóa: Sắp xếp Nước đi (Move Ordering)

Cũng như trong Task 3, việc sử dụng chiến lược Sắp xếp Nước đi (Center First) được tiếp tục sử dụng để tăng hiệu suất cắt tỉa Alpha-Beta.

Thử nghiệm Mở rộng: Việc đo thời gian Minimax khi tăng số lượng cột (yếu tố rẽ nhánh b) với độ sâu cố định (d=6) cho thấy:

Độ phức tạp: Thời gian tính toán tăng đáng kể khi số cột tăng (từ b=4 mất 0.0840 giây lên b=7 mất 0.7955 giây). Sự tăng trưởng này phản ánh tác động của yếu tố rẽ nhánh b lên độ phức tạp của tìm kiếm O(b^d).

### 5.4. Kết quả Thực nghiệm (Minimax Heuristic vs. Minimax Heuristic)

Các nguồn thực hiện một ván đấu giữa hai tác nhân Minimax Heuristic có độ sâu khác nhau trên bàn cờ 6x7:

P1: Minimax (Độ sâu = 5).

P2: Minimax (Độ sâu = 3).

Kết quả:

Người chiến thắng: P1 (Minimax Sâu 5) đã thắng trận đấu.

Phân tích: Kết quả này cho thấy rằng việc nhìn xa hơn (Độ sâu 5) mang lại lợi thế chiến lược rõ rệt so với tác nhân nhìn nông hơn (Độ sâu 3), vì P1 có thể dự đoán và thiết lập các mối đe dọa chiến thắng mà P2 không nhìn thấy.

### 5.5. Khả năng phát hiện chiến thuật (Heuristic Test)

Các thử nghiệm thủ công được thiết lập để kiểm tra xem hàm heuristic có đang đưa ra quyết định đúng đắn dựa trên thang điểm không:

Kiểm tra Thành công (Passed): Tác nhân đã thành công trong Test 2 (Phải chặn 3-hàng của đối thủ), chứng minh rằng điểm phạt -80 đủ mạnh để vượt qua các cơ hội tấn công nhỏ hơn của chính AI.

Kiểm tra Thất bại (Failed): Tác nhân đã thất bại trong các tình huống phức tạp hơn (Test 1, 3, 4), cho thấy:

Sự chưa hoàn hảo của Heuristic: Hàm heuristic, mặc dù chi tiết, có thể không đánh giá chính xác các mối đe dọa chiến lược phức tạp hoặc các vị trí "thiên đường/địa ngục" (winning/losing position).

Giới hạn Độ sâu: Với độ sâu tìm kiếm cố định là 4 (search\_depth=4), AI không thể nhìn đủ xa để phát hiện ra các mối đe dọa không thể chặn (unblockable threats) hoặc chuỗi chiến thắng dài hơn 4 nước đi.

# **Task Nâng cao: Pure Monte Carlo Search (PMC)**

Task Nâng cao (Advanced Task) yêu cầu triển khai Tìm kiếm Monte Carlo Thuần túy (Pure Monte Carlo Search - PMC) như một chiến lược đối kháng khác, so với Minimax/Alpha-Beta, để chơi Connect 4. Phương pháp này dựa vào mô phỏng ngẫu nhiên (rollouts) thay vì tìm kiếm toàn bộ cây trò chơi theo chiều sâu.

### 6.1. Cơ chế hoạt động của Pure Monte Carlo Search (PMC)

PMC hoạt động bằng cách ước tính tiềm năng của một nước đi bằng cách chơi thử (mô phỏng) một số lượng lớn các ván đấu ngẫu nhiên từ trạng thái sau nước đi đó.

#### a. Cấu trúc Tác nhân và Số lần Mô phỏng

Tác nhân PMC (PureMonteCarloAgent) được định nghĩa để lưu trữ quân cờ (piece) và số lần mô phỏng (num\_simulations) cần thực hiện cho MỖI nước đi hợp lệ.

#### b. Rollout (Chơi thử Ngẫu nhiên)

Chức năng cốt lõi là \_rollout:

* Hàm này chạy một ván cờ hoàn toàn ngẫu nhiên từ trạng thái hiện tại (sim\_board) cho đến khi đạt đến trạng thái kết thúc (terminal).
* Trong mỗi lượt, quân cờ được thả vào một cột được chọn ngẫu nhiên từ danh sách các hành động hợp lệ (valid\_moves).
* Rollout trả về người thắng (1 hoặc -1) hoặc 0 nếu hòa (EMPTY).

#### c. Quyết định nước đi (find\_move)

Tác nhân PMC đưa ra quyết định dựa trên thống kê về các lần mô phỏng:

* Lặp qua các Hành động: Tác nhân duyệt qua tất cả các hành động hợp lệ từ bàn cờ hiện tại.
* Kiểm tra Thắng Ngay lập tức: Nếu một hành động dẫn đến trạng thái thắng ngay lập tức, tác nhân sẽ chọn nước đi đó và trả về 1.0 (100% thắng), bỏ qua các mô phỏng còn lại.
* Chạy Mô phỏng: Đối với mỗi nước đi không thắng ngay lập tức, tác nhân chạy num\_simulations ván đấu ngẫu nhiên, bắt đầu từ lượt của đối thủ.
* Tính điểm: Điểm số (tỷ lệ thắng ước tính) cho mỗi hành động được tính bằng công thức: Điểm = (Số ván thắng + 0.5 x Số ván hòa) / Tổng số lần mô phỏng
* Chọn nước đi: Tác nhân chọn nước đi có tỷ lệ điểm cao nhất (tức là nước đi tối đa hóa khả năng thắng hoặc hòa trong các ván ngẫu nhiên).

### 6.2. Hiệu suất PMC trên các Trạng thái Thử nghiệm

Tác nhân PMC (với 200 lần mô phỏng) được thử nghiệm trên các trạng thái bàn cờ cố định:

* Thắng Ngay lập tức (Test 1 & 2): PMC ĐẠT các thử nghiệm này. Điều này là do logic kiểm tra thắng ngay lập tức được thêm vào hàm find\_move, chứ không phải nhờ các mô phỏng (rollouts).
* Chặn Mối đe dọa (Test 5 & 6): PMC cũng ĐẠT các thử nghiệm này (chặn 3-hàng ngang và chặn 3-hàng dọc của đối thủ).
  + Mặc dù PMC được biết là yếu trong việc phòng thủ phức tạp so với Minimax, trong các trường hợp đơn giản này, nước đi chặn là nước đi duy nhất có tỷ lệ thắng/hòa hợp lý, dẫn đến tỷ lệ thắng ước tính khoảng 42%–46% và được chọn là nước đi tốt nhất.

### 6.3. Xác định Nước đi Đầu tiên Tốt nhất

PMC được sử dụng để xác định nước đi đầu tiên tối ưu cho Người chơi 1 (P1) trên bàn cờ 6x7 trống.

#### a. Phương pháp

* Thử nghiệm chạy 2000 lần mô phỏng cho mỗi trong 7 cột có thể đi.
* Mục tiêu là tìm ra cột mang lại tỷ lệ thắng/hòa cao nhất cho P1, dựa trên giả định rằng tất cả các nước đi tiếp theo (của cả P1 và P2) đều là ngẫu nhiên.

#### b. Kết quả (Tỷ lệ Thắng/Hòa của P1)

Sau khi chạy tổng cộng 14,000 ván đấu (7 cột \* 2000 mô phỏng), kết quả cho thấy:

* Cột 3: 64.00%
* Cột 4: 58.23%
* Cột 2: 55.17%
* Các cột biên (0 và 6) có tỷ lệ thấp nhất (53.37% và 50.30%).

#### c. Kết luận và Giả định

* Nước đi Đầu tiên Tốt nhất: CỘT 3 là nước đi đầu tiên tốt nhất theo tiêu chí Monte Carlo Thuần túy.
* Giả định quan trọng: Nước đi này được coi là "tốt nhất" theo giả định rằng đối thủ (và bản thân tác nhân trong phần còn lại của trò chơi) chơi hoàn toàn ngẫu nhiên. Kết quả này chỉ ra lợi thế vị trí trung tâm khi chơi ngẫu nhiên, chứ không phải lợi thế đối kháng chiến lược (như Minimax, vốn giả định đối thủ chơi tối ưu).

Mean Connect 4

# **5 Bài Học Bất Ngờ Khi Dạy AI Chơi Cờ Connect 4**

Chắc hẳn ai trong chúng ta cũng từng chơi Connect 4, trò chơi thả cờ đơn giản với mục tiêu xếp được 4 quân thẳng hàng. Nhưng sẽ ra sao nếu chúng ta xây dựng một trí tuệ nhân tạo (AI) để chơi trò này? Hành trình dạy AI chơi một phiên bản đặc biệt của Connect 4 đã khám phá ra những sự thật đáng ngạc nhiên và phản trực giác về logic, chiến lược, và cả sự phức tạp ẩn sau những quy tắc tưởng chừng đơn giản.

--------------------------------------------------------------------------------

### **1. Một Thay Đổi Nhỏ Tạo Ra Ván Cờ Vô Hạn**

Trong dự án này, AI không chơi Connect 4 thông thường mà là "Mean" Connect 4. Luật chơi bổ sung rất đơn giản: thay vì thả một quân cờ mới, người chơi có thể di chuyển một quân cờ ở hàng dưới cùng của đối thủ sang một cột khác.

Hệ quả của thay đổi này lại vô cùng lớn. Trong Connect 4 thông thường, trò chơi chắc chắn kết thúc sau tối đa 42 nước đi vì bàn cờ sẽ đầy. Tuy nhiên, với luật "mean", số lượng quân cờ trên bàn không tăng lên, dẫn đến khả năng tạo ra các vòng lặp bất tận.

Do khả năng xảy ra các vòng lặp vô hạn, cây trò chơi về mặt lý thuyết là vô hạn. Trong thực tế, điều này đòi hỏi phải sử dụng **tìm kiếm giới hạn độ sâu (depth-limited search)**.

Điều này đi ngược lại với suy nghĩ thông thường: chỉ một biến thể nhỏ đã thay đổi bản chất của vấn đề. Nó buộc AI phải chuyển đổi tư duy từ "giải một câu đố hữu hạn" sang "điều hướng một không gian chiến lược vô hạn". Thay vì tìm kiếm một lời giải hoàn hảo, mục tiêu giờ đây là tìm ra nước đi tối ưu nhất trong giới hạn thực tế—một thách thức cốt lõi của AI trong thế giới thực.

--------------------------------------------------------------------------------

### **2. Lợi Thế Vô Hình: Đi Trước Thắng 55% Ngay Cả Khi Chơi Ngẫu Nhiên**

Để kiểm tra luật chơi, một mô phỏng đã được thực hiện cho hai AI chơi hoàn toàn ngẫu nhiên đấu với nhau 1000 ván. Kết quả thật đáng kinh ngạc:

* **Người chơi 1 (đi trước):** Thắng 552 ván (55.2%)
* **Người chơi 2 (đi sau):** Thắng 443 ván (44.3%)
* **Hòa:** 5 ván (0.5%)

Kết quả này được xem là hoàn toàn có thể dự đoán được. Người đi trước luôn có một lợi thế nhỏ vì họ thực hiện nước đi đầu tiên và luôn có khả năng đi nhiều nước hơn hoặc bằng đối thủ trong suốt ván cờ.

Đây là một bài học đắt giá về lợi thế cấu trúc. Sức mạnh của việc "đi trước" đã được định sẵn trong luật chơi, mạnh đến mức nó thể hiện rõ ràng ngay cả khi không có bất kỳ kỹ năng hay chiến lược nào được áp dụng, chỉ đơn thuần là các nước đi ngẫu nhiên.

--------------------------------------------------------------------------------

### **3. Cái Giá Của Một Cột Cờ: Thời Gian Tính Toán Tăng Theo Cấp Số Nhân**

Độ phức tạp của trò chơi đối với AI tăng lên như thế nào khi bàn cờ lớn hơn? Một thử nghiệm đã được thực hiện để đo thời gian AI ra quyết định khi số cột tăng dần.

| Kích thước Bảng | Thời gian (giây) |
| --- | --- |
| 6x4 | 0.0442 |
| 6x5 | 0.1081 |
| 6x6 | 0.2819 |
| 6x7 | 0.5757 |

Dữ liệu cho thấy thời gian ra quyết định của AI tăng theo cấp số nhân khi số cột (hay còn gọi là hệ số rẽ nhánh - branching factor) tăng lên. Đây là biểu hiện kinh điển của độ phức tạp thời gian O(b^d) trong các thuật toán tìm kiếm.

Để dễ hình dung: việc thêm chỉ **MỘT** cột vào bàn cờ từ 6x6 lên 6x7 đã khiến AI phải làm việc vất vả hơn gần gấp đôi. Điều này minh họa một cách trực quan về "lời nguyền của số chiều" (curse of dimensionality), một trong những thách thức lớn nhất trong lĩnh vực AI.

--------------------------------------------------------------------------------

### **4. Mẹo Nhỏ, Hiệu Quả Lớn: Sắp Xếp Nước Đi Giúp AI Nhanh Hơn 72%**

Để tối ưu hóa AI, một chiến lược đơn giản gọi là "Move Ordering" (Sắp xếp nước đi) đã được áp dụng. Thay vì duyệt các nước đi một cách ngẫu nhiên, AI được lập trình để ưu tiên xét các nước đi ở cột giữa trước các cột ở rìa, vì các nước đi ở trung tâm có nhiều khả năng tạo ra đường thắng hơn.

Kết quả của mẹo nhỏ này thật sự ấn tượng. Trong trường hợp thử nghiệm với độ sâu tìm kiếm là 7, chiến lược này đã giảm 72.33% số nút (trạng thái) mà AI cần duyệt, từ 54,119 xuống chỉ còn 14,975.

Đây là một ví dụ hoàn hảo về sự đối lập giữa "sức mạnh tính toán" và "trí thông minh thuật toán". Thay vì chờ đợi một máy tính mạnh hơn, một "mẹo" logic đơn giản dựa trên hiểu biết về trò chơi lại hiệu quả hơn. Nó chứng minh rằng sự tinh tế trong thuật toán đôi khi có thể chiến thắng sức mạnh xử lý thô.

--------------------------------------------------------------------------------

### **5. Nghịch Lý Nước Đi Đầu Tiên: Bàn Cờ Trống Là Bài Toán Khó Nhất**

Một trong những phát hiện phản trực giác nhất là: đối với thuật toán Minimax với cắt tỉa Alpha-Beta—một phương pháp tìm kiếm nước đi tối ưu bằng cách giả định đối thủ cũng sẽ chơi một cách hoàn hảo—bàn cờ trống là trường hợp tệ nhất và tốn nhiều thời gian xử lý nhất.

Lý do là vì trên một bàn cờ trống, không có mối đe dọa nào, nên thuật toán không thể "cắt tỉa" các nhánh tìm kiếm một cách hiệu quả. Nó không thể sớm xác định các nước đi rõ ràng là xấu để loại bỏ.

Bắt đầu với một bảng trống là trường hợp tệ nhất cho Minimax với Alpha-Beta... Nó buộc phải duyệt gần như toàn bộ cây tìm kiếm (tới độ sâu giới hạn).

Giải pháp cho vấn đề này không phải là tính toán nhiều hơn, mà là sử dụng kiến thức có sẵn. Một kỹ thuật gọi là **"Sách lược mở đầu" (Opening Book)** được áp dụng. Thay vì tính toán từ đầu, AI chỉ cần tra cứu nước đi tốt nhất đã được xác định trước cho bàn cờ trống, đó là **cột giữa (cột 3)**.

Đây là một nghịch lý thú vị: bài toán trông có vẻ đơn giản nhất lại là bài toán tốn nhiều công sức nhất cho AI, và giải pháp tối ưu lại là "gian lận" bằng cách dựa vào kiến thức đã biết trước.

--------------------------------------------------------------------------------

### **Kết luận**

Hành trình này cho thấy ngay cả trong những quy tắc đơn giản nhất cũng có thể ẩn chứa một độ phức tạp bùng nổ. Điều đó đặt ra một câu hỏi thú vị: những hệ thống đơn giản nào khác trong cuộc sống của chúng ta cũng đang che giấu những lớp sâu sắc đáng kinh ngạc như vậy?