**Connect 4 with Artificial Intelligence**

Trần Thị Phương Thảo | 18521422

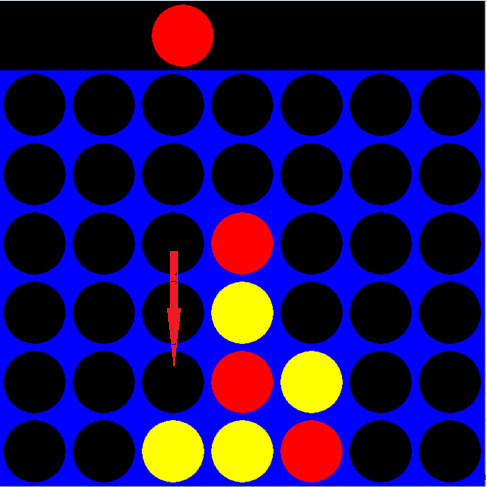
*Artificial intelligence project*

*University Information Technology*

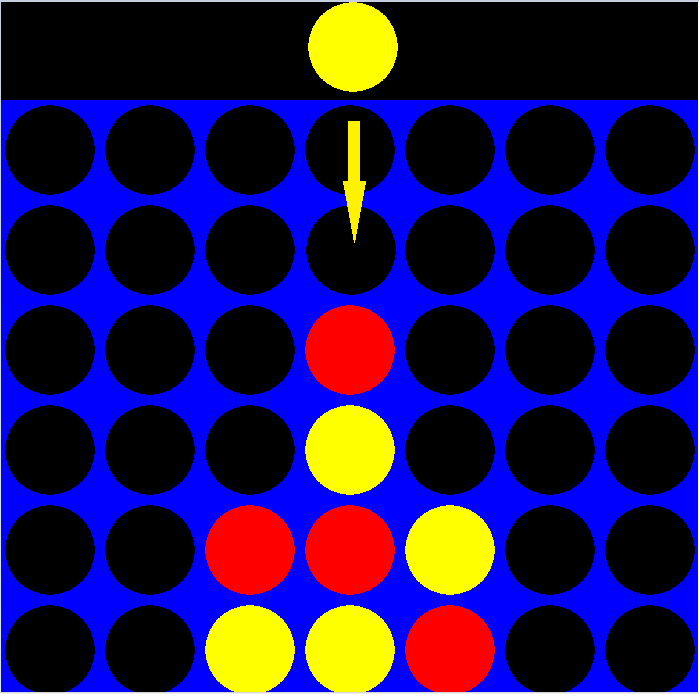
***Tóm tắt*: Connect 4 là một trò chơi hai người chơi với**[**thông tin hoàn hảo**](https://en.wikipedia.org/wiki/Perfect_information)**cho cả hai bên. Tại một thời điểm chơi, người chơi có tất cả thông tin về các di chuyển đã diễn ra và tất cả các di chuyển có thể diễn ra, trong một trạng thái trò chơi nhất định. Connect 4 cũng thuộc về phân loại của một**[**trò chơi có tổng bằng không**](https://en.wikipedia.org/wiki/Zero-sum_game)**, vì lợi thế của người chơi là nhược điểm của đối thủ. Một thước đo độ phức tạp của trò chơi Connect 4 là số lượng vị trí bảng trò chơi có thể có. Đối với Connect 4 cổ điển được chơi trên 6 lưới cao, 7 rộng, có 4.531.985.219.092 vị trí cho tất cả các bảng trò chơi có từ 0 đến 42 miếng. Dưới đây là một số cách tiếp cận của AI với phiên bản Connect 4 cổ điển như các thuật toán minimax, negamax, Expectimax với phương pháp tối ưu hóa cắt tỉa Alpha – Beta và phiên bản biến thể của negamax cùng với hàm đánh giá heuristic. Tìm kiếm sự tối ưu trong từng thuật toán và giải pháp sẽ mang lại cách giải quyết tốt nhất cho bài toán.**

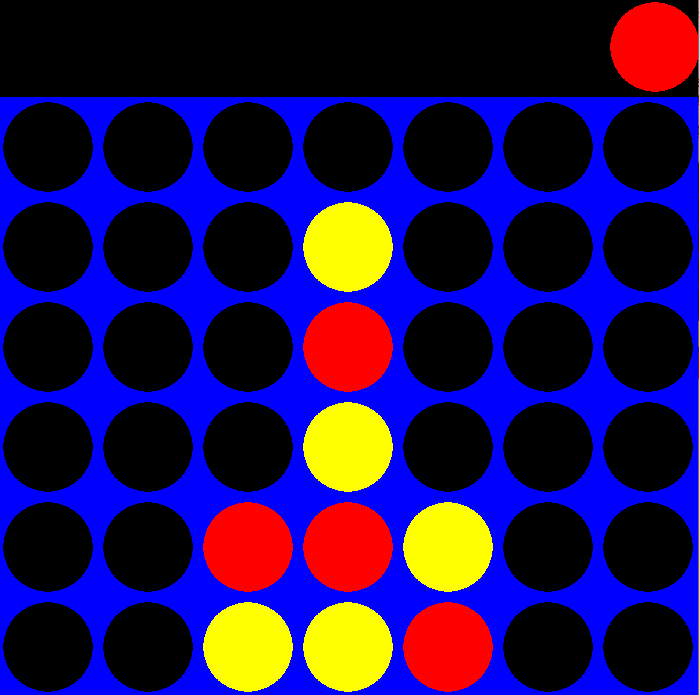
I. TỔNG QUAN VỀ TRÒ CHƠI

Connect 4 được biết đến rộng rãi vào tháng 2 năm 1974 bởi Milton Bradley, lần đầu tiên được giải quyết bởi James Dow Allen (1-10-1988) và độc lập bởi Victor Allis (16-10-1988). Connect 4 là một trò chơi bảng kết nối hai người chơi mà trước tiên người chơi chọn một màu và sau đó lần lượt thả một đĩa màu từ trên xuống vào một lưới treo thẳng đứng bảy cột, sáu hàng. Các đĩa rơi thẳng xuống, chiếm không gian có sẵn thấp nhất trong cột. Mục tiêu của trò chơi là trở thành người đầu tiên tạo thành một đường ngang, dọc hoặc chéo của 4 đĩa riêng. Ví dụ chơi game Connect 4 trên máy tính: đến lượt người chơi đỏ bạn chỉ được chọn 1 trong 7 cột còn hàng sẽ là vị trí trống sâu nhất của bảng tức là từ dưới cùng lên. Người chơi đỏ sẽ di chuyển chuột đến cột mong muốn ở đây là cột thứ 3 sau đó nhấn chuột. Đĩa màu đỏ tương ứng của người chơi lúc này sẽ rơi xuống tới ô còn trống sâu nhất của cột thứ 3 là hàng thứ 2. Sau đó tiếp tục đến lượt của người chơi tiếp theo. Game sẽ kết thúc khi có 1 trong 2 người chơi thắng hoặc bẳng đã hết ô trống (game kết thúc mà không có người thắng).



H1.1





H1.2

H1.3

Hình 1 ví dụ về cách chơi Connect 4 (H1.1->H1.2->H1.3)

II. GIÁ TRỊ TRẠNG THÁI VÀ HÀM ĐÁNH GIÁ (HEURISTIC)

Xây dựng hàm đánh giá để giới hạn độ sâu tìm kiếm (depth) cho các thuật toán AI của tôi.

Với mỗi lần di chuyển cơ bản ở phiên bản Connect 4 cổ điển bạn sẽ có 7 lựa chọn, với mỗi lựa chọn bạn có một giá trị trạng thái. Vậy sau 3 lần di chuyển bạn có 7x7x7 = 343 trường hợp có thể xảy ra tương đương với 343 giá trị trạng thái khác nhau. Số lần di chuyển ở mỗi lần tìm kiếm phụ thuộc vào độ sâu tìm kiếm. Khi chương trình được gọi độ sâu tìm kiếm sẽ được truyền vào và giảm đi sau mỗi lần di chuyển. Khi depth = 0 tất cả các trạng thái trên lớp hiện tại sẽ được ước tính giá trị. Đều cần để ý để tạo heuristic có khả năng sử dụng là:

* Chi phí tính toán: với mỗi di chuyển, chương trình có thể phải tính cả mấy nghìn lần giá trị trạng thái, bạn không thể có độ sâu tìm kiếm tốt nếu như chi phí để tính toán quá lớn.
* Tương ứng với chiến thắng: mục đích của bạn là chiến thắng đối thủ, heuristic cũng phải thể hiện tương thích với chiến thắng nếu không sử dụng nó sẽ không mang lại lợi ích cho bạn.
* Đơn giản, dễ thực hiện: các heuristic càng phúc tạp càng khó cài đặt, dê xảy ra lỗi, tính toán lâu, ảnh hưởng tới depth

# Vì kết quả tôi muốn nhận được ở đây là chiến thắng đối thủ của mình nên trạng thái cho tôi khả năng chiến thắng càng cao thì giá trị sẽ càng lớn. Để chiến thắng, tôi cần tạo thành 4 ô liền kề ngang, dọc hoặc chéo.

# 

# H2.

# Hình 2 khung của sổ 4 ô liền kề

Hướng tới mục tiêu dành chiến thắng cho mình: nên nếu tôi thắng, tôi sẽ nhận được phần thưởng rất lớn +1000000; tôi thua (đối thủ thắng) nhận điểm âm lớn tương đương -1000000.

Xét từng khung 4 ô liền kề ngang/dọc/chéo:

Score = giá trị trạng thái

Với mỗi ô của cột thứ 4, ta đều có lớn hơn 5 khả năng chiến thắng được tạo ra, sẽ dễ dàng tìm kiếm sự chiến thắng nếu sở hữu ô trên cột đó – tôi nhận được 3 điểm với mỗi ô của mình trên cột thứ 4 (+3x).

center\_count = số quân của bạn ở cột thứ 4

# Score = Score + center\_count\*3

# Với 4 ô đều có quân của mình, khung mang lại cho tôi chiến thắng (+100)

# Score = Score + 100

# 4 ô liền kề, 3 quân ta với 1 ô trống chúng ta sẽ chiến thắng nếu lượt tiếp theo đánh vào ô đó.Tương tự với 2-2. Tôi sẽ chiến thắng nếu như 2 lượt tiếp theo tôi đánh vào 2 ô trống còn (+5/+2)

# 3-1: Score = Score + 5

# 2-2: Score = Score + 2

# Nếu đối thủ thắng game sẽ kết thúc tôi sẽ phải nhận điểm âm rất lớn. Nên trước khi để thua tôi sẽ cần biết đối thủ của mình sắp thắng khi nào để chặn ô trống đó không cho đối thủ chiến thắng. khung 4 ô với 3 ô đã là quân của đối thủ ô còn lại đang trống (-4)

# Score = Score – 4

# Hàm đánh giá duyệt qua tất cả các trường hợp 4 ô liền kề ngang/dọc/chéo trả về Score mang giá trị của trạng thái lúc này. Hàm heuristic đơn giản đủ để hướng tôi tích cực đến chiến thắng của mình, ngăn chặn hướng đi tới chiến thắng của đối thủ. Heuristic đơn giản với cùng thời gian tính toán tôi có độ sâu thích hợp để chọn lựa được nước đi tốt nhất.

# III. XÂY DỰNG CÁC THUẬT TOÁN

# *A. Thuật toán Minnimax*

# Minnimax là thuật toán đơn giản nhất mà chúng ta sử dụng để giải quyết vấn đề của game Connect 4

Giải quyết Connect 4 có thể được xem là tìm đường dẫn tốt nhất trong cây quyết định trong đó mỗi nút là một Vị trí. Tại mỗi nút, người chơi phải chọn một di chuyển dẫn đến một trong những vị trí tiếp theo có thể. Khi đến lượt của bạn, bạn muốn chọn nước đi tốt nhất có thể sẽ tối đa hóa điểm số của bạn. Nhưng lượt tiếp theo, đối thủ của bạn sẽ cố gắng tối đa hóa điểm số của mình, do đó tối thiểu hóa điểm số của bạn.

Điều này dẫn đến thuật toán Minimax. Dự đoán trước 1 số bước đi tiếp theo của bạn và đối thủ.

* Khi đến lượt của bạn, điểm số là điểm tối đa của bất kỳ vị trí nào có thể tiếp theo (bạn sẽ chơi di chuyển tối đa hóa điểm số của bạn),
* Trong khi đó đến lượt đối thủ của bạn, điểm số là điểm tối thiểu của các vị trí có thể tiếp theo (đối thủ của bạn sẽ chơi chiêu tối thiểu điểm của bạn và tối đa hóa điểm của anh ta).

# Điểm số cho từng nước đi ở đây chúng ta sẽ nhận được thông qua hàm đánh giá đó là lý do vì sao chúng ta cần xây dựng hàm đánh giá thuật tốt trước đó.

# Biểu đồ dưới đây là ví dụ về cây tìm kiếm bằng thuật toán Minimax

# 

Hình 3 Ví dụ về cây tìm kiếm của thuật toán minimax

Thuật toán được thực hiện với tìm kiếm giới hạn độ sâu để có thể xác định giá trị của nút lá cuối cùng thông qua hàm heuristic. Dưới đây là mã giả của thuật toán minimax sử dụng tìm kiếm giới hạn độ sâu.

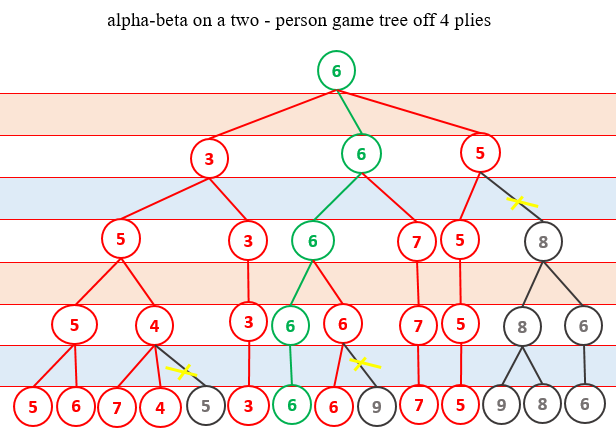
Mã giả của thuật toán minimax. Trong mã bên dưới, người chơi chúng tôi muốn tối đa hóa mức tăng là người chơi 1

|  |
| --- |
| **function** minimax(node, depth, maximizingPlayer) **is**  **if** depth = 0 **or** node is a terminal node **then**  **return** the heuristic value of node  **if** maximizingPlayer **then**  value := −∞  **for each** child of node **do**  value := max(value, minimax(child, depth − 1, FALSE))  **return** value  **else** *(\* minimizing player \*)*  value := +∞  **for each** child of node **do**  value := min(value, minimax(child, depth − 1, TRUE))  **return** value |

**Pseudocode 1** Minimax algorithm

*B. Thuật toán Minimax với Cắt tỉa Alpha-Beta*

Cắt tỉa alpha-beta được sử dụng để cắt số lượng nút trong cây tìm kiếm được đánh giá bằng thuật toán minimax. Nó dừng hoàn toàn đánh giá một động thái khi có ít nhất một khả năng đã được tìm thấy chứng minh rằng hành động đó tồi tệ hơn một động thái được kiểm tra trước đó, vì vậy chúng tôi không cần phải đánh giá thêm.



Hình 4 Ví dụ về cây tìm kiếm của thuật toán alpha-beta

Mã giả của việc cắt tỉa alpha-beta.

|  |
| --- |
| **function** alphabeta(node, depth, α, β, maximizingPlayer) **is**  **if** depth = 0 **or** node is a terminal node **then**  **return** the heuristic value of node  **if** maximizingPlayer **then**  value := −∞  **for each** child of node **do**  value := max(value, alphabeta(child, depth − 1, α, β, FALSE))  α := max(α, value)  **if** α ≥ β **then**  **break** *(\* β cut-off \*)*  **return** value  **else**  value := +∞  **for each** child of node **do**  value :=min(value, alphabeta(child, depth − 1, α, β, TRUE))  β := min(β, value)  **if** β ≤ α **then**  **break** *(\* α cut-off \*)*  **return** value |

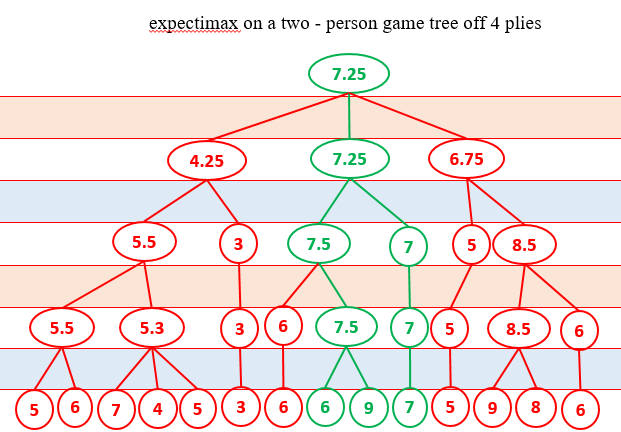
**Pseudocode 2** Alpha-Beta pruning algorithm

Thuật ngữ α là giới hạn dưới của mức tăng và β là giới hạn trên. Ở lượt người chơi 1, chúng tôi muốn tối đa hóa mức tăng bằng cách đánh giá tất cả các nước đi có thể. Giá trị của α được cập nhật mỗi lần. Khi tại một thời điểm nào đó chúng ta thấy rằng không lớn hơn α hoặc nói cách khác, giới hạn dưới không lớn hơn giới hạn trên, thì chúng ta có thể dừng lại ở đây một cách an toàn, bởi vì chúng ta đã chứng minh rằng nước đi cũ sẽ tốt hơn những bước đi đó. Điều này được gọi là β cắt. Nếu điểm cắt α ở độ sâu quá lớn hay không có điểm cắt thì thuật toán cắt tỉa Alpha-Beta lại trở về giống với Minimax.

*C. Thuật toán Expectimax*

Trong thuật toán minimax, chúng tôi sẽ chọn di chuyển dựa trên mức tăng tối đa (hoặc tối thiểu). Nó có nghĩa là chúng ta chỉ cần giá trị tối thiểu hoặc tối đa của các nút con.

Trong thuật toán kỳ vọng, khi chúng ta đánh giá nút đối thủ, chúng ta sẽ tính toán tất cả các chuyển động có thể, được tính theo xác suất xảy ra. Nói cách khác, chúng tôi sẽ tính giá trị kỳ vọng của mức tăng trong tất cả các trường hợp có thể.



Hình 5 Ví dụ về cây tìm kiếm của thuật toán expectimax

Để hiểu rõ hơn, dưới đây là mã giả của thuật toán expectimax.

|  |
| --- |
| **function** expectiminimax(node, depth)  **if** node is a terminal node **or** depth = 0  **return** the heuristic value of node    **if** we are to play at node  // Return value of maximum-valued child node  **let** α := -∞  **foreach** child of node  α := max(α, expectiminimax(child, depth-1))  **else** random event at node  // Return weighted average of all child nodes' values  **let** α := 0  **foreach** child of node  α := α + (Probability[child] × expectiminimax(child, depth-1))  **return** α |

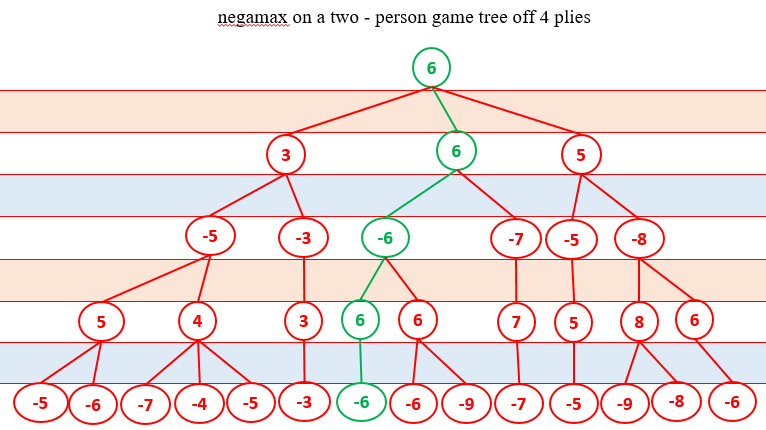
**Pseudocode 3** Expectimax algorithm

Trong hầu hết các trường hợp, xác suất của mỗi cột sẽ giống nhau. Trong trường hợp của chúng tôi, Connect 4, xác suất của các cột sẽ giống nhau. Giá trị của nút cơ hội lúc này sẽ là trung bình của tất cả các giá trị của các trạng thái có thể xảy ra tiếp theo.

*D. Thuật toán Negamax*

Negamax là một dạng biến thể của [minimax](https://en.wikipedia.org/wiki/Minimax) dựa trên thuộc [tính tổng bằng không](https://en.wikipedia.org/wiki/Zero-sum_(Game_theory)) của [trò chơi hai người chơi](https://en.wikipedia.org/wiki/Two-player_game) .

Thuật toán này dựa trên thực tế là {\ displaystyle \ max (a, b) = - \ min (-a, -b)}max(a, b) = - min(-a, -b) để đơn giản hóa việc thực hiện thuật toán [minimax](https://en.wikipedia.org/wiki/Minimax). Ví dụ max(2, 3) = - min(-2, -3) = 3. Chính xác hơn, giá trị của một vị trí đối với người chơi A trong một trò chơi như vậy là sự phủ định giá trị đối với người chơi B. Do đó, người chơi khi di chuyển tìm kiếm một động thái tối đa hóa sự phủ định của giá trị do di chuyển: vị trí kế vị này phải theo định nghĩa đã được đối thủ coi trọng. Lý do của câu trước hoạt động bất kể A hay B đang di chuyển. Điều này có nghĩa là một thủ tục duy nhất có thể được sử dụng để định giá cả hai vị trí. Đây là một đơn giản hóa mã hóa trên minimax, yêu cầu A chọn di chuyển với người kế thừa có giá trị tối đa trong khi B chọn di chuyển với người kế thừa có giá trị tối thiểu.

Hình 6 Ví dụ về cây tìm kiếm của thuật toán negamax

Mã giả của thuật toán

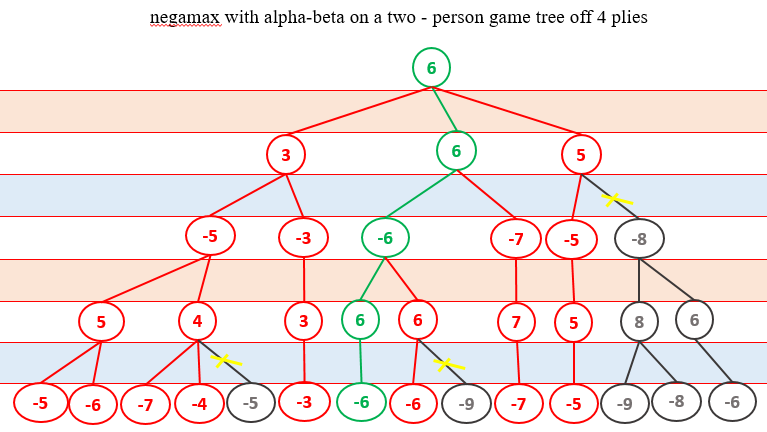
|  |
| --- |
| **function** negamax(node, depth, color) **is**  **if** depth = 0 **or** node is a terminal node **then**  **return** color × the heuristic value of node  value := −∞  **for each** child of node **do**  value := max(value, −negamax(child, depth − 1, −color))  **return** value |

**Pseudocode 4** Negamax algorithm

Điểm số Negamax khớp với điểm số minimax cho các nút nơi người chơi A sắp chơi và trong đó người chơi A là người chơi tối đa hóa trong tương đương minimax. Negamax luôn tìm kiếm giá trị tối đa cho tất cả các nút của nó. Do đó đối với các nút của người chơi B, điểm minimax là một phủ định của điểm negamax của nó. Người chơi B là người chơi thu nhỏ trong tương đương minimax. Nên thuật toán negamax chỉ tối ưu cho việc cài đặt còn giá trị tìm kiếm vẫn giống như thuật toán minimax.

* *Negamax với cắt tỉa Alpha-Beta*

Tối ưu hóa thuật toán cho [minimax](https://en.wikipedia.org/wiki/Minimax) cũng được áp dụng tương tự cho Negamax. [Việc cắt tỉa alpha-beta](https://en.wikipedia.org/wiki/Alpha-beta_pruning) có thể làm giảm số lượng nút mà thuật toán negamax đánh giá trong cây tìm kiếm theo cách tương tự với việc sử dụng nó với thuật toán minimax.

Hình 7 Ví dụ về cây tìm kiếm của thuật toán negamax với cắt tỉa alpha-beta

Mã giả cho thuật toán negamax với [việc cắt tỉa alpha-beta](https://en.wikipedia.org/wiki/Alpha-beta_pruning)

|  |
| --- |
| **function** negamax(node, depth, α, β, color) **is**  **if** depth = 0 **or** node is a terminal node **then**  **return** color × the heuristic value of node  childNodes := generateMoves(node)  childNodes := orderMoves(childNodes)  value := −∞  **foreach** child in childNodes **do**  value := max(value, −negamax(child, depth − 1, −β, −α, −color))  α := max(α, value)  **if** α ≥ β **then**  **break** *(\* cut-off \*)*  **return** value |

**Pseudocode 5** Negamax algorithm with alpha-beta

*E. Thuật toán Minmin*

Trong khi đang cài đặt thuật toán Negamax, tôi nhận ra mình đã đi sai hướng. Tôi đã sử dụng lộn chiến thuật ở đây, nhưng thật bất ngờ kết quả mà thuật toán này mang lại cho tôi lại tốt hơn thuật toán Negamax. Vì vậy tôi quyết định giới thiệu cho các bạn cách cài đặt này. Bởi vì chưa xác định được tên của thuật toán này nên tôi tạm gọi nó là minmin.

Ở thuật toán minimax, ở cả 2 lượt đều là cố gắng tối ưu hóa hay là tối thiểu hóa điểm của cùng một đối tượng. Còn với minmin sẽ là thao tác trên điểm của 2 đối tượng khác nhau. Lượt đi của bạn, sẽ cố gắng tối thiểu hóa số điểm của đối thủ (chọn nước đi khiến cho đối thủ có giá trị trạng thái là thấp nhất), tương tự với lượt đi của đối thủ của bạn là cố tối thiểu hóa số điểm của bạn. của người chơi sẽ là tố đa hóa điểm số chính bạn, lượt đi của đố thủ là tối thiểu hóa điểm số của bạn. Sự khác biệt giữa 2 thuật toán là ở nước đi của bạn, minimax sẽ tối đa hóa điểm của bạn còn minmin sẽ là tối thiểu hóa điểm số của đối thủ. Ở thuật toán minmin hàm Heuristic sẽ trả về luân phiên giá trị trạng thái của bạn và của đối thủ. Lượt đi của bạn hàm heuristic trả về giá trị trạng thái của đối thủ còn lượt của đối thủ sẽ trả về giá trị trạng thái của bạn. Điểm này khác với các thuật toán phía trước vì trước đó hàm Heuristic luôn trả về giá trị trạng thái giá trị của bạn.

Tôi cũng đã tiến hành việc cắt tỉa Alpha-Beta với minmin để giảm thiểu thời giạn tính toán không cần thiết.

Mã giả cho thuật toán minmin của tôi được chỉnh sửa một chút từ mã giả của thuật toán negamax.

|  |
| --- |
| **function** minmin(node, depth, α, β, color, play1) **is**  **if** depth = 0 **or** node is a terminal node **then**  **return** the heuristic value of node  childNodes := generateMoves(node)  childNodes := orderMoves(childNodes)  value := −∞  **foreach** child in childNodes **do**  value := max(value, −minmin(child, depth − 1, −β, −α, −color, play2))  α := max(α, value)  **if** α ≥ β **then**  **break** *(\* cut-off \*)*  **return** value |

**Pseudocode 6** minmin algorithm with alpha-beta

IV. KẾT QUẢ MANG LẠI CỦA CÁC THUẬT TOÁN VỚI VIỆC GIẢI QUYẾT GAME CONNECT 4

*A.Đánh giá kết quả từng thuật toán*

Tìm kiếm thời gian chạy, khả năng chiến thắng đối thủ, số nước đi trung bình cho mỗi đối thủ AI.

\* Người chơi A là một trong những thuật toán AI phía trên còn người chơi B sẽ là lựa chọn ngẫu nhiên nước đi (B đi nước đi ngẫu nhiên).

Bảng 1 kết quả chạy các thuật toán (minimax, alpha-beta, negamax, minmin, expectimax) với đối thủ random.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Avr\_time | Count\_s | Win |
| minimax | 1.41 | 6.5 | 100/100 |
| alpha-beta | 0.33 | 6.8 | 100/100 |
| negamax | 0.3 | 6.58 | 100/100 |
| minmin | 0.31 | 6.32 | 100/100 |
| expectimax | 6.07 | 5.6 | 100/100 |

Bảng 1.1 A - B with depth = 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Avr\_time | Count\_s | Win |
| minimax | 12.38 | 5.86 | 100/100 |
| alpha-beta | 1.42 | 5.97 | 100/100 |
| negamax | 1.21 | 5.8 | 100/100 |
| minmin | 1.41 | 6.04 | 100/100 |
| expectimax | 27.98 | 4.5 | 100/100 |

Bảng 1.2 A - B with depth = 5

\* Người chơi A là một trong những thuật toán AI phía trên còn người chơi C sẽ là lựa chọn nước đi có giá trị trạng thái tốt nhất.

Bảng 2 kết quả chạy các thuật toán (minimax, alpha-beta, negamax, minmin, expectimax) với đối thủ tốt nhất

|  |  |  |  |
| --- | --- | --- | --- |
|  | Avr\_time | Count\_s | Win |
| minimax | 1.0 | 13 | 100/100 |
| alpha-beta | 0.22 | 13 | 100/100 |
| negamax | 0.2 | 13 | 100/100 |
| minmin | 0.21 | 13 | 100/100 |
| expectimax | 5.89 | 7 | 100/100 |

Bảng 2.1 A - C with depth = 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Avr\_time | Count\_s | Win |
| minimax | 8.5 | 10 | 100/100 |
| alpha-beta | 1.33 | 10 | 100/100 |
| negamax | 1.26 | 10 | 100/100 |
| minmin | 1.73 | 6 | 100/100 |
| expectimax | 28.55 | 4 | 100/100 |

Bảng 2.1 A - C with depth = 5

Depth: độ sâu tìm kiếm

Avr\_time: thời gian trung bình cho cần thiết cho 1 lần tính toán

Count\_s: số nước đi trung bình cho 1 ván game

Win: tỉ lệ chiến thắng

Các thuật toán minimax, alpha-beta, negamax cho khả năng chiến thắng đối thủ với trung bình nước đi xấp xỉ nhau do có ý tưởng thuật toán tương tự nhau. Nhưng do alpha-beta, negamax đã áp dụng cắt tỉa phần tính toán không cần thiết nên cho thời gian tính toán nhanh hơn.

Minmin mặc dù có ý tưởng thuật toán khác alpha-beta nhưng số lượng công việc tính toán tương tự nên cho thời gian tính toán nước đi xấp xỉ với alpha-beta và negamax. Với depth = 5 minmin cho kết quả về nước đi cần cho chiến thắng tốt hơn so minimax, alpha-beta, negamax trước đối thủ luôn chọn nước đi có giá trị trạng thái tốt nhất.

Expectimax có trung bình nước đi là thấp nhất nhưng tốn kém quá nhiều thời gian cho việc tính toán.

*B.So sáng các thuật toán*

Tiến hành so sánh các thuật toán alpha-beta (A) , minmin (M), expectimax (E) với nhau.

Cho từng cặp thuật toán trở thành đối thủ đi tìm chiến thắng trong game Connect 4. Tiến hành thi đấu 2 ván. A vs M, ván 1 A đi trước, ván 2 M đi trước. Mỗi ván đối tượng chiến thắng sẽ dành được 1 điểm.

Kết quả được thể hiện qua bảng sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| depth | 1 | 2 | 3 | 4 | 5 | 6 | sum |
| A vs M | 0/2 | 1/1 | 1/0 | 1/1 | 0/2 | 1/1 | 4/7 |
| A vs E | 1/1 | 1/1 | 2/0 | 2/0 | 2/0 | 2/0 | 10/2 |
| M vs E | 2/0 | 1/1 | 1/1 | 2/0 | 2/0 | 2/0 | 10/2 |

Bảng 3 kết quả

Từ kết quả bảng trên và kết quả đáng giá trong phần A ta có:  
 Xét về khả năng chiến thắng trò chơi: thuật toán cho khả năng thấp nhất là expectimax, cao nhất là minmin. Các thuật toán minimax, alpha-beta, negamax cho khả năng tương tự nhau.

Xét về thời gian tính toán nước đi: các thuật toán alpha-beta, negamax, minmin cho kết quả tối ưu nhất, expectimax cần quá nhiều thời gian tính toán.

Expectimax sẽ tốt nếu như đối thủ của bạn ít khôn ngoan và chi phí cho mỗi nước đi là rất lớn. Vì expectimax sẽ cho bạn số nước đi ít nhất so với các thuật toán ở đây để chiến thắng đối thủ.

Vậy kết quả tốt nhất chúng ta nhận với game Connect 4 cho mọi trường hợp ở đây là thuật toán minmin. Cho kết quả tối ưu cả về thời gian tính toán và khả năng chiến thắng đối thủ.