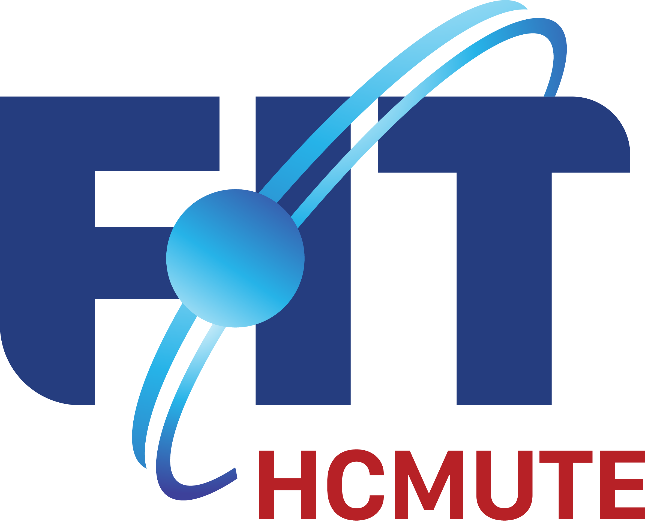
**Trường đại học sư phạm kỹ thuật TP.HCM**

**Khoa Công Nghệ Thông Tin**

**Bộ môn trí tuệ nhân tạo**



**NGUYỄN LÝ HÙNG – 22110337**

**VÕ VĂN TRÍ – 22110444**

**NGUYỄN HOÀNG ANH KHOA – 22110352**

Đề Tài

**NGhiên cứu THUẬT TOÁN Heuristic Alpha - Beta Tree Search   
và ỨNG DỤNG thuật toán vào TRÒ CHƠI TIC-TAC-TOE**

**Giáo viên hướng dẫn**

**ThS. Lê Minh Tân**

**Khóa 2022 – 2026**

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM** **Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Họ và tên Sinh viên 1 : Nguyễn Lý Hùng MSSV 1: 22110337

Họ và tên Sinh viên 2 : Võ Văn Trí MSSV 2: 22110444

Họ và tên Sinh viên 3 : Nguyễn Hoàng Anh Khoa MSSV 3: 22110352

Ngành: Công nghệ Thông tin

Tên đề tài: Nghiên cứu thuật toán Heuristic Alpha - Beta Tree Search và ứng dụng thuật toán vào trò chơi Tic-Tac-Toe

Họ và tên Giáo viên hướng dẫn: Lê Minh Tân.

**NHẬN XÉT**

Về nội dung đề tài & khối lượng thực hiện:

1. Ưu điểm:

1. Khuyết điểm

1. Đánh giá loại :
2. Điểm :

Tp*. Hồ Chí Minh, ngày tháng năm 20*

Giáo viên hướng dẫn

*(Ký & ghi rõ họ tên)*

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM** **Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN**

Họ và tên Sinh viên 1 : Nguyễn Lý Hùng MSSV 1: 22110337

Họ và tên Sinh viên 2 : Võ Văn Trí MSSV 2: 22110444

Họ và tên Sinh viên 3: Nguyễn Hoàng Anh Khoa MSSV 3: 22110352

Ngành: Công nghệ Thông tin

Tên đề tài: Nghiên cứu thuật toán Heuristic Alpha - Beta Tree Search và ứng dụng thuật toán vào trò chơi Tic-Tac-Toe

Họ và tên Giáo viên phản biện: Lê Minh Tân.

**NHẬN XÉT**

Về nội dung đề tài & khối lượng thực hiện:

1. Ưu điểm:

1. Khuyết điểm

1. Đánh giá loại :
2. Điểm :

Tp*. Hồ Chí Minh, ngày tháng năm 20*

Giáo viên phản biện

*(Ký & ghi rõ họ tên)*

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**ĐỀ CƯƠNG LUẬN VĂN TỐT NGHIỆP**

Họ và Tên SV thực hiện 1 : Nguyễn Lý Hùng Mã Số SV : 22110337

Họ và Tên SV thực hiện 2 : Võ Văn Trí Mã Số SV : 22110444

Họ và Tên SV thực hiện 3: Nguyễn Hoàng Anh Khoa Mã Số SV : 22110352

Thời gian làm luận văn từ : 15/04/2024 Đến :19/04/2024

Chuyên ngành : Công nghệ thông tin

Tên luận văn : Nghiên cứu thuật toán Heuristic Alpha - Beta Tree Search và vận dụng thuật toán vào trò chơi Tic-Tac-Toe

GV hướng dẫn : Lê Minh Tân

**Nhiệm Vụ Của Luận Văn :**

1. Tìm hiểu chung về lý thuyết game (Game Theory).
2. Tìm hiểu thuật toán Minimax và đưa ra nhận xét ưu điểm khuyết điểm.
3. Tìm hiểu thuật toán Alpha-Beta Pruning và đưa ra nhận xét ưu điểm khuyết điểm.
4. Tìm hiểu thuật toán Heuristic Alpha-Beta Tree Search và đưa ra nhận xét ưu điểm khuyết điểm.
5. Tìm hiểu về các khái niệm liên quan như Horizon Effect, Forward Pruning,…
6. Ứng dụng thuật toán Heuristic Alpha-Beta Tree Search vào trò chơi Tic-Tac-Toe.
7. Tổng kết.

**KẾ HOẠCH THỰC HIỆN**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Thời gian** | **Công việc** | **Ghi chú** |
| **1** | **15/04/2024** | **Phân công công việc cho từng thành viên trong nhóm** | **Võ Văn Trí phần code, Nguyễn Lý Hùng phần các khái niệm liên quan và Heuristic Alpha-Beta Tree Search, Nguyễn Hoàng Anh Khoa phần Minimax và Alpha-Beta Pruning** |
| **2** | **16/04/2024** | **Các thành viên tìm hiểu và tìm nội dung báo cáo** |  |
| **3** | **17/04/2024-18/04/2024** | **Viết báo cáo và powerpoint** |  |
| **4** | **19/04/2024** | **Tổng hợp nội dung** |  |

Ngày 19 tháng 04 năm 2024

**Người viết đề cương**

**Nguyễn Hoàng Anh Khoa**

MỤC LỤC

[PHẦN 1: MỞ ĐẦU 1](#_Toc165232948)

[1.Tính cấp thiết của đề tài 1](#_Toc165232949)

[2.Đối tượng, phạm vi nghiên cứu 1](#_Toc165232950)

[3.Phân tích hướng nghiên cứu liên quan 2](#_Toc165232951)

[4. Kết quả dự kiến 2](#_Toc165232952)

[PHẦN 2: NỘI DUNG 3](#_Toc165232953)

[1. Tìm hiểu về lý thuyết game 3](#_Toc165232954)

[1.1 Giới thiệu về Zero-sum game 3](#_Toc165232955)

[1.2 Khái niệm về cây trò chơi 4](#_Toc165232956)

[1.3 Thuật toán Minimax 5](#_Toc165232957)

[2. Thuật toán Alpha-Beta Pruning 8](#_Toc165232958)

[2.1 Giới thiệu về thuật toán Alpha-Beta Pruning 8](#_Toc165232959)

[2.2 Thứ tự di chuyển trong Alpha-Beta Pruning 13](#_Toc165232960)

[2.3 Ưu điểm và nhược điểm của thuật toán Alpha-Beta Pruning 13](#_Toc165232961)

[3. Heuristic Alpha-Beta Tree Search 13](#_Toc165232962)

[3.1. Giới thiệu về thuật toán Heuristic Alpha-Beta Tree Search 13](#_Toc165232963)

[3.2. Evaluation Function 14](#_Toc165232964)

[3.3. Cut off Search 15](#_Toc165232965)

[3.4. Mã giả của Heuristic Alpha-Beta Tree Search 16](#_Toc165232966)

[3.5. Ứng dụng thuật toán Heuristic Alpha-Beta Tree Search vào trò chơi Tic-Tac-Toe 18](#_Toc165232967)

[4. Các khái niệm liên quan 23](#_Toc165232968)

[4.1. Horizon Effect 23](#_Toc165232969)

[4.2. Forward Pruning 24](#_Toc165232970)

[4.3. Search và Lookup trong AI 25](#_Toc165232971)

[PHẦN 3: TỔNG KẾT 27](#_Toc165232972)

[PHẦN 4: TÀI LIỆU THAM KHẢO 28](#_Toc165232973)

**PHẦN 1: MỞ ĐẦU**

## 1.Tính cấp thiết của đề tài

Trí tuệ nhân tạo (AI) đang là một đề tài phổ biến trên toàn cầu hiện nay. Thuật toán là một phần rất quan trọng trong mảng trí tuệ nhân tạo. Trong đó thuật toán Heuristic Alpha-Beta Tree Search được xem là một khái niệm quan trọng không chỉ trong ngành trí tuệ nhân tạo mà còn trong việc cải thiện hiệu suất của các thuật toán tìm kiếm trong trò chơi và lập lịch. Việc áp dụng phương pháp này giúp giảm số lượng các nút cần duyệt trong cây tìm kiếm, từ đó sẽ tăng tốc độ và giảm bộ nhớ yêu cầu, đặc biệt là trong các trò chơi có độ phức tạp như cờ vua, cờ vây.

Trong thời đại 4.0 như hiện nay thì việc tạo ra những chương trình trò chơi trong ngành giải trí đang dần phổ biến. Việc áp dụng thuật toán Heuristic Alpha-Beta Tree Search một cách hiệu quả sẽ đem lại cho khách hàng những trải nghiệm mới mẻ và tốt hơn.

Ngoài ra, không chỉ trong lĩnh vực trí tuệ nhân tạo, thuật toán Heuristic Alpha-Beta Tree Search còn được áp dụng vào các lĩnh vực lập lịch và quy hoạch nơi mà việc tìm kiếm giải pháp tối ưu trong không gian lớn của các biến số là thách thức. Sử dụng thuật toán Heuristic Alpha-Beta Tree Search cải tiến đáng kể trong việc giải quyết các vấn đề phức tạp trong thực tế.

Tóm lại, thuật toán Heuristic Alpha-Beta Tree Search rất cần thiết không chỉ nằm ở việc cải thiện hiệu suất của các thuật toán trong lĩnh vực trí tuệ nhân tạo và trò chơi mà còn trong các lĩnh vực khác nhằm đóng góp tích cực vào sự phát triển của công nghệ thông tin và ứng dụng của nó trong thế giới thực.

## 2.Đối tượng, phạm vi nghiên cứu

Đối tượng nghiên cứu: Thuật toán Heuristic Alpha-Beta Tree Search trong lĩnh vực trí tuệ nhân tạo và trò chơi.

Phạm vi nghiên cứu: Khái niệm của thuật toán Heuristic Alpha-Beta Tree Search từ đó tìm hiểu ứng dụng của giải thuật này trong thực tế và đặc biệt trong lĩnh vực trò chơi với trò chơi Tic-Tac-Toe.

## 3.Phân tích hướng nghiên cứu liên quan

Hướng nghiên cứu đi từ tìm hiểu lý thuyết game và thuật toán Minimax, từ đó nghiên cứu cách tối ưu bài toán Minimax bằng thuật toán Alpha-Beta Pruning, từ thuật toán Alpha-Beta Pruning tiếp tục nghiên cứu cách tối ưu bằng Heuristic Alpha-Beta Tree Search. Cuối cùng tìm hiểu về ứng dụng của thuật toán trong việc tạo ra một trò chơi Tic-Tac-Toe.

## 4. Kết quả dự kiến

Kết quả của việc nghiên cứu về thuật toán Heuristic Alpha-Beta Tree Search sẽ giúp ta hiểu được cách tối ưu một thuật toán trong thực tế. Từ đó, giúp chúng ta có thể áp dụng những giải thuật này trong việc tạo lập một chương trình lớn mang đến cho người dùng những trải nghiệm tuyệt vời.

# **PHẦN 2: NỘI DUNG**

## Tìm hiểu về lý thuyết game

Lý thuyết trò chơi chủ yếu nghiên cứu những tác dụng tương hỗ giữa các kết cấu đã được công thức hóa qua các phương pháp toán học để nghiên cứu hiện tượng có tính chất cạnh tranh. Lý thuyết trò chơi đã trở thành một công cụ phổ biến trong một số ngành như kinh tế học, sinh vật học, khoa học máy tính, chính trị học,… Nguồn gốc của lý thuyết trò chơi hiện đại là do John von Neumann đưa ra ý tưởng và chứng minh điểm cân bằng của chiến lược đối với zero-sum game của hai người.

Các loại game

|  |  |  |
| --- | --- | --- |
|  | Deterministic | Chance |
| Perfect information | Chess, Checkers, Go, Othello | Backgammon, Monopoly |
| Imperfect information |  | Bridge, poker, scrabble nuclear war |

* 1. **Giới thiệu về Zero-sum game**

Trong lý thuyết trò chơi, Zero-sum game là những game có tính chất xác định và nằm trong môi trường quan sát được, trong những game đó thường có hai người chơi, mỗi người chơi đối kháng với nhau một cách luân phiên để giá trị lợi ích luôn ở mức có lợi ở người chơi đó. Zero-sum game không có có tính chất tương hỗ lẫn nhau, nó thực hiện trên ý tưởng tổng số giá trị lợi ích và tổng thiệt hại của người người tham gia được cộng lại sẽ bằng không. Ví dụ như bạn cắt một cái bánh, nếu bạn lấy miếng càng lớn thì lượng bánh của người khác sẽ bị giảm.

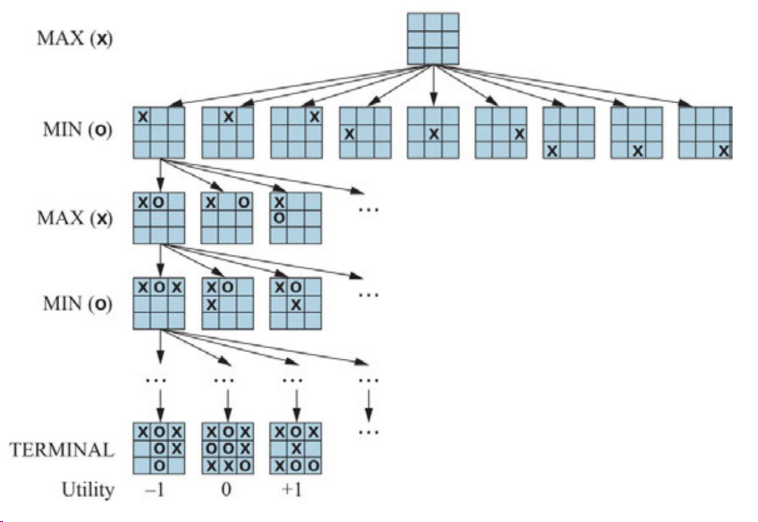
Ý tưởng của Zero-sum game là gọi hai người chơi là MAX và MIN. Người MAX sẽ di chuyển trước, sau đó đến lượt người MIN và họ sẽ luân phiên nhau cho tới khi trò chơi kết thúc. Khi kết thúc trò chơi người chiến thắng sẽ được thưởng còn người thua sẽ bị phạt. Một trò chơi được xác định bởi các yếu tố sau:

* S0 : Trạng thái ban đầu, chỉ định cách thiết lập khi bắt đầu trò chơi.
* TO-MOVE(s): Xác định người chơi di chuyển trong trạng thái s.
* ACTION(s): Tập hợp các hành động phù hợp trong trạng thái s.
* RESULT(s,a): Mô hình chuyển đổi, xác định trạng thái từ việc thực hiện hành động tại trạng thái s.
* IS-TERMINAL(s): Hàm kiểm tra trò chơi đã kết thúc hay chưa, nếu kết thúc sẽ trả

về kết quả đúng, chưa kết thúc sẽ trả về kết quả sai.

* UTILITY(s,p): Hàm tiện ích, xác định giá trị cuối cùng khi trò chơi kết thúc ở trạng thái cuối. Ví dụ trong cờ vua kết quả khi thắng, thua và hòa lần lượt là 1,0,1/2.
  1. **Khái niệm về cây trò chơi**

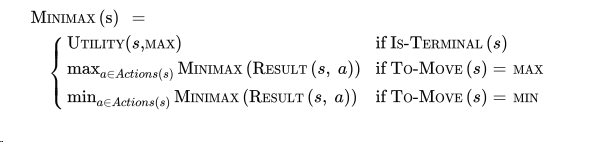
Khái niệm về cây trờ chới (Game Tree) là một cây trong đó các nút của cây là những trạng thái của trò chơi, trong đó các cạnh của cây là các nước đi của người chơi. Cây trò chơi là vô hạn nếu không gian trạng thái không bị giới hạn hoặc những quy tắc của trò chơi cho phép vị trí lặp lại vô hạn.



**Hình 1.1:** Cây trò chơi của trò chơi Tic-Tac-Toe

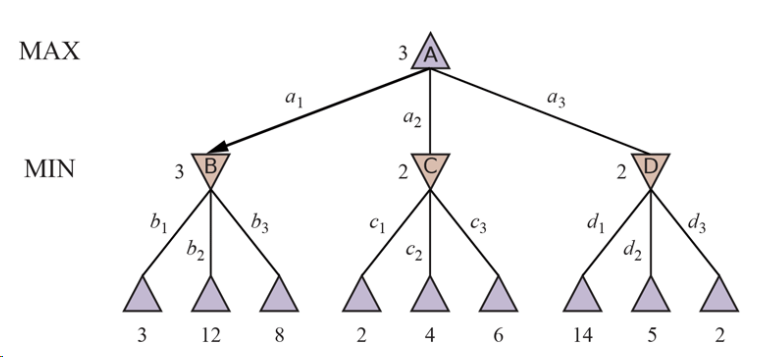
Ví dụ: Ban đầu ta bắt đầu với trạng thái ban đầu, người chơi MAX với giá trị “X” có chín cách đi, sau đó tiếp đến là lượt đi của người chơi MIN với giá trị “O”. Quá trình đó sẽ luân phiên cho đến khi người chơi có ba ô vuông liên tiếp hoặc các ô vuông bị lấp đầy. Giá trị tiện ích xác định trạng thái kết thúc trên mỗi nút lá của cây. Giá trị tiện ích cao thì tốt cho người chơi MAX và xấu cho người chơi MIN. Nút cuối (terminal node) là nút không có bất kỳ nút con nào là vị trí cuối cùng xác định kết quả của trò chơi.

* 1. **Thuật toán Minimax**
     1. **Quyết định tối ưu trò chơi**

Với một cây trò chơi, chiến lược tối ưu được xác định bằng cách tính giá trị minimax của mỗi trạng thái trong cây, hàm tính giá trị minimax được gọi là MINIMAX(s). Giá trị minimax là giá trị tiện ích ở xác định tại trạng thái đó, nếu cả hai người chơi đều chơi một cách tối ưu từ đầu đến cuối. Khi trò chơi chưa kết thúc, người chơi MAX tìm phương án có giá trị tiện ích lớn nhất và người chơi MIN sẽ tìm phương án có giá trị tiện ích nhỏ nhất. Vì vậy ta có công thức MINIMAX sau:

**Hình 1.2:** Công thức Minimax

Ví dụ về cây trò chơi sau:



**Hình 1.3:** Cây trò chơi Minimax

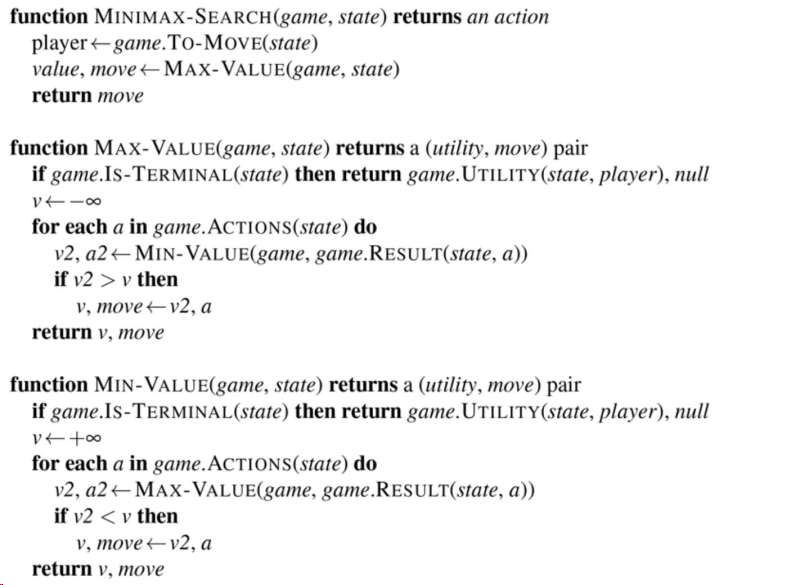
Trong ví dụ trên: MAX sẽ được biểu diễn bằng hình tam giác hướng lên và MIN sẽ được biểu diễn bằng hình tam giác hướng xuống.

Quá trình tiến hành ví dụ trên như sau:

* Đầu tiên gán v là âm vô cùng đối với MAX và v là dương vô cùng đối với MIN.
* Xét tại nút B là vị trí của MIN, đầu tiên ta sẽ lấy min(3,+∞) = 3 và sau đó so sánh tiếp min(12,3)=3 và min(8,3)=3. Nên ta sẽ cập nhật B là v = 3.
* Tương tự ta xét tại nút C, ta lần lượt lấy min (2, +∞)=2, min(2,4) = 2, min(2,6)=2. Nên ta sẽ cập nhật C là v =2.
* Tương tự ta xét tại nút D, ta lần lượt lấy min (14, +∞)=14, min(14,5) = 5, min(5,2)=2. Nên ta sẽ cập nhật D là v =2.
* Tiếp tục, tới lượt nút A lượt của MAX, nó sẽ chọn giá trị lớn nhất trong 3 nút B,C,D lần lượt như sau max(3,2) = 3, max(3,2) = 3. Nên cuối cùng ta sẽ cập nhật A là v = 3.
  + 1. **Giới thiệu về thuật toán Minimax**

Thuật toán Minimax giống như thuật toán Depth First Search (DFS), là một thuật toán đệ quy và quay lui. Thuật toán này sẽ tìm kiếm trên cây trò chơi theo thứ tự sâu dần giống DFS bằng cách đệ quy và quay lui lại khi gặp được trạng thái kết thúc. Trong quá trình quay lui thì thuật toán Minimax sẽ tìm và gán các giá trị minimax cho các node con đến khi các giá trị đó lan ra các nút cha. Mục đích chính của thuật toán là tìm ra nước đi tối ưu nhất cho người chơi bằng cách đánh giá các giá trị của các nút trong cây trò chơi. Thuật toán này được sử dụng rộng rãi trong những trò chơi có hai người chơi như cờ tướng, cờ vua, tic-tac-toe…

Đây là mã giả của thuật toán Minimax:



**Hình 1.4:** Mã giả thuật toán Minimax

Giải thích chi tiết mã giả của thuật toán như sau:

* MINIMAX\_SEARCH(game,state):
* Đây là hàm chính của thuật toán, nhận vào trạng thái của trò chơi hiện tại (state) cùng với đối tượng là (game) và trả về hành động tốt nhất có thể.
* Trong hàm này, ta sẽ xác định lượt đi của người chơi bằng hàm TO\_MOVE(state). Tiếp theo đó, ta gọi hàm MAX\_VALUE(game,state) đề tìm kiếm hành động tốt nhất cho người chơi hiện tại.
* MAX\_VALUE(game,state):
* Hàm này mục đích nhằm tối đa hóa tiện ích trạng thái hiện tại của người chơi.
* Hàm này trả về một cập giá trị tiện ích và nước đi tốt nhất dành cho người chơi.
* Bắt đầu, sẽ có hàm IS\_TERMINAL(state) để kiểm tra trò chơi có kết thúc hay chưa. Nếu trạng thái hiện tại của trò chơi là trạng thái kết thúc thì nó sẽ trả về giá trị tiện ích thông qua hàm UTILITY(state,player)
* Nếu trò chơi chưa kết thúc thì khởi tạo giá trị âm vô cùng và gán nó cho biến “v”. Sau đó đối với mỗi hành động có thể có trong trạng thái hiện tại, bước đầu tiên nó sẽ gọi hàm MIN\_VALUE(game,state) cho các trạng thái kế tiếp tương ứng những hành động đó. Tiếp theo, ta cập nhật giá trị vào biến “v2” và nếu giá trị mới này lớn hơn giá trị lớn nhất trước đó là “v” ta sẽ cập nhật giá trị lớn nhất và hành động tương ứng. Cuối cùng trả về giá trị lớn nhất “v” và hành động liên quan.
* đề tìm kiếm hành động tốt nhất cho người chơi hiện tại.
* MIN\_VALUE(game,state):
* Hàm này mục đích nhằm tối thiểu hóa tiện ích trạng thái hiện tại của người chơi. Nó đại diện cho mục tiêu của đối thủ.
* Bắt đầu, sẽ có hàm IS\_TERMINAL(state) để kiểm tra trò chơi có kết thúc hay chưa. Nếu trạng thái hiện tại của trò chơi là trạng thái kết thúc thì nó sẽ trả về giá trị tiện ích thông qua hàm UTILITY(state,player).
* Nếu trò chơi chưa kết thúc thì khởi tạo giá trị dương vô cùng và gán nó cho biến “v”. Sau đó đối với mỗi hành động có thể có trong trạng thái hiện tại, bước đầu tiên nó sẽ gọi hàm MAX\_VALUE(game,state) cho các trạng thái kế tiếp tương ứng những hành động đó. Tiếp theo, ta cập nhật giá trị vào biến “v2” và nếu giá trị mới này nhỏ hơn giá trị nhỏ nhất trước đó là “v” ta sẽ cập nhật giá trị nhỏ nhất và hành động tương ứng. Cuối cùng trả về giá trị nhỏ nhất “v” và hành động liên quan.
  + 1. **Hạn chế của thuật toán Minimax**

Trong lý thuyết trò chơi, thuật toán Minimax cho phép tìm ra nước đi tối ưu cho người chơi MAX. Tuy nhiên, trong thực tế, việc tính toán nước đi tối ưu này trên toàn bộ cây trò chơi (xem xét tất cả các đỉnh của cây theo kiểu vét cạn) đòi hỏi quá nhiều thời gian và tài nguyên tính toán. Đặc biệt là trong các trò chơi có kích thước của cây trò chơi cực kỳ lớn, việc tính toán này có thể trở nên không khả thi và tốn kém.

## Thuật toán Alpha-Beta Pruning

* 1. **Giới thiệu về thuật toán Alpha-Beta Pruning**

Ta có thể thấy khi sử dụng thuật toán Minimax số lượng trạng thái của trò chơi tăng theo cấp số nhân độ sâu của cây. Điều này sẽ làm tốn chi phí, thời gian thực thi lâu vì thế ta có thể giải quyết bằng phương pháp cắt tỉa phần lớn các nhánh của cây mà không cần kiểm tra từng nút của cây trò chơi cũng như ảnh hưởng đến kết quả. Kỹ thuật cắt tỉa này được gọi là Alpha-Beta Pruning.

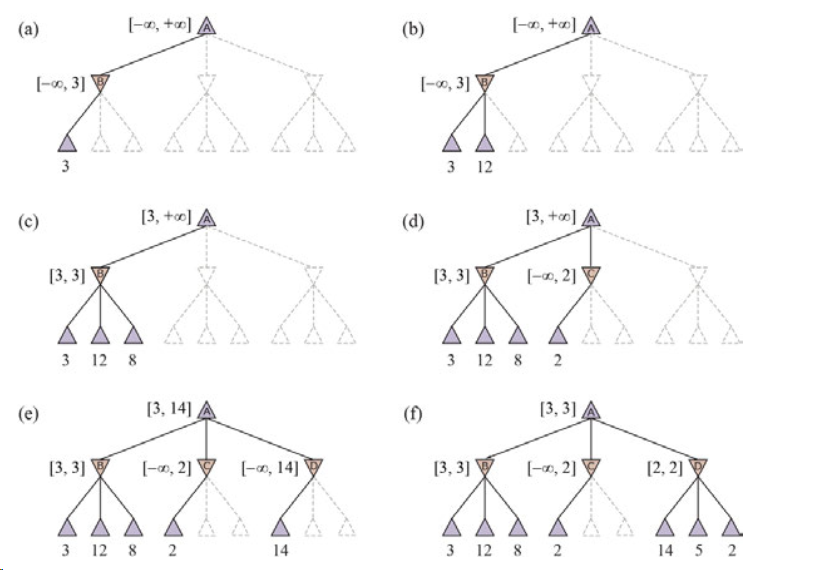
Thuật toán Alpha-Beta Pruning là một thuật toán tìm kiếm nâng cao của Minimax, giải thuật này giúp giảm số lượng các nút cây được đánh giá bởi thuật toán trong cây tìm kiếm. Alpha-Beta Pruning được áp dụng ở bất kỳ độ sâu nào của cây và thậm chí một số tình huống nó không chỉ cắt tỉa nút lá của cây mà còn cắt tỉa toàn bộ cây phụ.

Hai tham số đặc trưng của Alpha-Beta Pruning là:

* Alpha: Lựa chọn tốt nhất (giá trị cao nhất) tại thời điểm hiện tại tại bất kỳ điểm nào trên con đường của Maximizer với giá trị khởi đầu là -∞.
* Beta: Lựa chọn tốt nhất (giá trị thấp nhất) tại thời điểm hiện tại tại bất kỳ điểm nào dọc theo con đường của Minimizer với giá trị khởi đầu là ∞.

Điều kiện để thực hiện Alpha-Beta Pruning là α >= β.

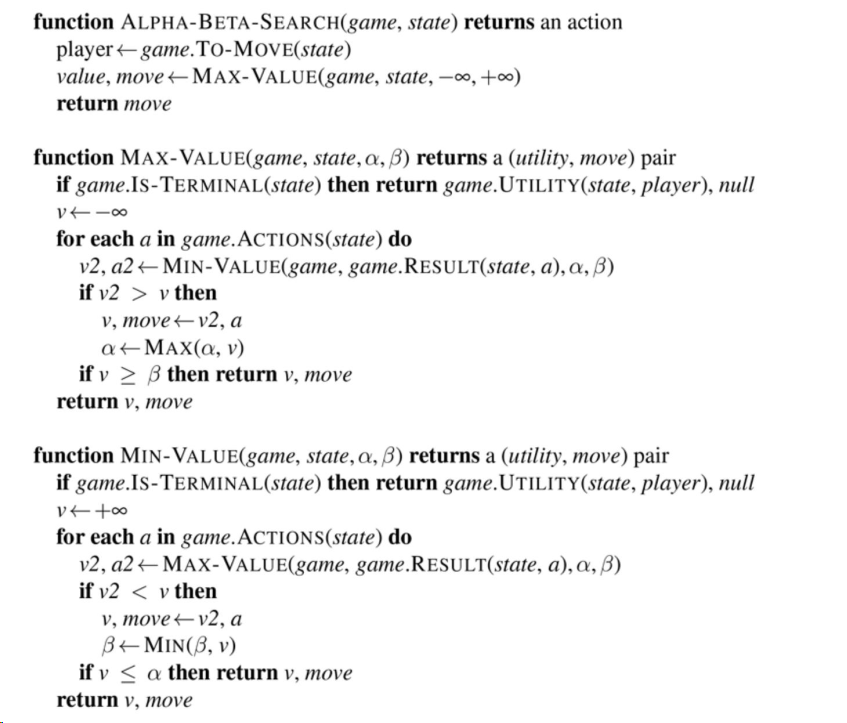
Trong quá trình thực hiện, người chơi MAX sẽ cập nhật giá trị của alpha và người chơi MIN sẽ cập nhật giá trị của beta. Trong khi quay lui lại cây, giá trị của nút sẽ được chuyển đến các nút phía trên thay vì các giá trị alpha và beta. Sau đó, sẽ chuyển các giá trị alpha, beta cho các nút con.



**Hình 2.1:** Cây trò chơi Alpha-Beta Pruning

Trong ví dụ trên, quá trình thực hiện sẽ diễn ra như sau:

* Bước đầu tiên, người chơi MAX sẽ di chuyển đầu tiên từ nút A nơi α = -∞ và β = +∞, những giá trị α và β này sẽ được truyền lại cho nút B.
* Tại nút B, giá trị của β sẽ được tính theo lượt của nó cho MIN. Giá trị của v được so sánh cho với giá trị đầu tiên là 3 nên ta có min(+∞,3) =3 sau đó là min(3,12)=3 và min(3,8)=3 và giá trị của β tại nút B là 3 và giá trị minimax của nút cũng là v=3. Khi này giá trị α cũng sẽ cập nhật thành 3.
* Trả đệ quy lại nút A, giá trị của α sẽ được tính theo lượt MAX. Giá trị α lúc này của nút A là 3 và giá trị của β tại nút A là +∞ đồng thời giá trị v cũng sẽ bằng 3.
* Tiếp tục xét với nút C, với 3 giá trị terminal node của C là 2,4,6. Giá trị α và β tại C lần lượt là 3 và +∞. Vì C là lượt của MIN nên ta gán v = +∞. Giá trị v sẽ được so sánh lần lượt min(+∞,2)=2, do giá trị α>= β (vì 3>2) nên có thể lược bỏ phần bên phải nút C. Cuối cùng cập nhật giá trị β tại nút C là 2.
* Trả đệ quy lại nút A, giá trị của α sẽ được tính theo lượt MAX. Giá trị α lúc này của nút A là 3 và giá trị của β tại nút A là 2 đồng thời giá trị v cũng sẽ bằng 3.
* Tiếp tục xét với nút D, với 3 giá trị terminal node của D là 14,5,2. Giá trị α và β tại D lần lượt là 3 và +∞. Vì D là lượt của MIN nên ta gán v = +∞. Giá trị v sẽ được so sánh lần lượt min(+∞,14)=14, tiếp tục là min(14,5)=5 và min(5,2)=2. Cuối cùng cập nhật giá trị β tại nút D là 2.
* Trả đệ quy lại nút A, giá trị của α sẽ được tính theo lượt MAX. Giá trị α lúc này của nút A là 3 và giá trị của β tại nút A là 2 đồng thời giá trị v cũng sẽ bằng 3.

Đây là mã giả của giải thuật Alpha-Beta Pruning.

**Hình 2.2:** Mã giảthuật toán Alpha-Beta Pruning

Giải thích chi tiết mã giả của thuật toán:

* ALPHA\_BETA\_SEARCH(game,state):
* Đây là hàm chính của thuật toán, nhận vào trạng thái của trò chơi hiện tại (state) cùng với đối tượng là (game) và trả về hành động tốt nhất có thể
* Trong hàm này, ta sẽ xác định lượt đi của người chơi bằng hàm TO\_MOVE(state). Sau đó, ta gọi MAX\_VALUE để tìm kiếm hành động tốt nhất cho người chơi hiện tại, sử dụng giá trị alpha và beta để cắt tỉa.

- MAX\_VALUE(game,state, α, β)

* Hàm này đại diện cho quá trình của người chơi cố gắng tối đa hóa giá trị của họ.
* Ta khởi tạo giá trị của v là âm vô cực.
* Ta duyệt qua từng hành động có thể có trong trạng thái hiện tại và gọi MIN\_VALUE cho các trạng thái kế tiếp tương ứng những hành động đó.
* Tiếp theo, ta cập nhật giá trị vào biến “v2” và nếu giá trị mới này lớn hơn giá trị lớn nhất trước đó là “v” ta sẽ cập nhật giá trị lớn nhất và hành động tương ứng.
* Sau đó, ta cập nhật giá trị của “v” là giá trị lớn nhất
* Nếu “v” đã lớn hơn hoặc bằng beta, ta cắt tỉa (có nghĩa là ta không cần xem xét các nút con khác của nút hiện tại), và trả về v.
* Nếu không, ta cập nhật giá trị của alpha là giá trị lớn nhất giữa alpha và v, và tiếp tục với các hành động còn lại.
* Cuối cùng, ta trả về giá trị của v

- MIN\_VALUE(state, α, β):

* Hàm này đại diện cho quá trình của người chơi cố gắng tối thiểu hóa giá trị của họ. Hàm này được xem là đại diện cho đối thủ.
* Ta khởi tạo giá trị của v là dương vô cực.
* Ta duyệt qua từng hành động có thể có trong trạng thái hiện tại và gọi MAX\_VALUE cho các trạng thái kế tiếp tương ứng những hành động đó.
* Tiếp theo, ta cập nhật giá trị vào biến “v2” và nếu giá trị mới này nhỏ hơn giá trị nhỏ nhất trước đó là “v” ta sẽ cập nhật giá trị nhỏ nhất và hành động tương ứng.
* Sau đó, ta cập nhật giá trị của “v” là giá trị nhỏ nhất
* Nếu “v” đã bé hơn hoặc bằng alpha, ta cắt tỉa (có nghĩa là ta không cần xem xét các nút con khác của nút hiện tại), và trả về v.
* Nếu không, ta cập nhật giá trị của beta là giá trị nhỏ nhất giữa beta và v, và tiếp tục với các hành động còn lại.
* Cuối cùng, ta trả về giá trị của v

Khi cả hai hàm MAX\_VALUE và MIN\_VALUE được gọi đệ quy lần lượt, giá trị của alpha và beta được cập nhật và truyền dẫn giữa các cấp độ của cây tìm kiếm. Việc cắt tỉa sẽ giúp giảm số lượng các trạng thái cần xem xét, tăng tốc độ của thuật toán. Cuối cùng, hàm ALPHA\_BETA\_SEARCH sẽ trả về hành động tốt nhất dựa trên kết quả của MAX\_VALUE.

### Thứ tự di chuyển trong Alpha-Beta Pruning

Thuật toán Alpha-Beta Pruning được áp dụng hiệu quả là do thứ tự mà mỗi nút được kiểm tra. Có hai loại thứ tự như sau:

* Worst ordering: Trong vài trường hợp, thuật toán Alpha-Beta Pruning hoạt động như thuật toán Minimax, nó không cắt tỉa bất kỳ lá nào của cây. Điều này dẫn đến tiêu tốn chi phí và thời gian lâu, do đó độ phức tạp về thời gian của giải thuật trên sẽ là O(bm).
* Ideal ordering: Đây là điều kiện lý tưởng để Alpha-Beta Pruning chỉ duyệt phía bên trái cây và cắt tỉa được diễn ra nhiều lần trên cây. Do đó độ phức tạp trong điều kiện lý tưởng này là O(bm/2).

### Ưu điểm và nhược điểm của thuật toán Alpha-Beta Pruning

* + 1. **Ưu điểm của Alpha-Beta Pruning so với Minimax**

Ưu điểm của thuật toán Alpha-Beta Pruning so với thuật toán Minimax là về hiệu suất sẽ cao hơn so với Minimax vì Alpha-Beta Pruning sẽ loại bỏ việc duyệt qua các nút không cần thiết, giảm nhiều thời gian hơn so với Minimax. Ngoài ra, thuật toán Alpha-Beta Pruning không cần nhiều bộ nhớ bổ sung vì thế có thể dễ dàng triển khai trong những trò chơi lớn.

* + 1. **Nhược điểm của Alpha-Beta Pruning**

Do hiệu suất của thuật toán Alpha-Beta Pruning phụ thuộc nhiều vào thứ tự duyệt. Nếu thứ tự này không tốt sẽ không thể cắt bỏ được nhiều nút như mong muốn.

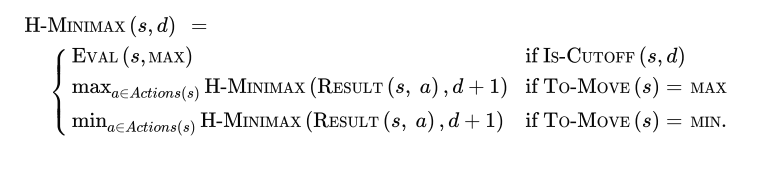
Việc phải duyệt tới tận nút lá để có thể đánh giá trạng thái đích trong các bài toán có độ phức tạp lớn vẫn chưa phải là một giải pháp tối ưu

Thuật toán Alpha-Beta Pruning đòi hiểu phải hiểu sâu để có thể triển khai và sắp xếp các nút một cách hiệu quả nhất.

## Heuristic Alpha-Beta Tree Search

### Giới thiệu về thuật toán Heuristic Alpha-Beta Tree Search

Heuristic Alpha-Beta Tree Search là một cải tiến của Minimax và Alpha-Beta Pruning. Cả hai thuật toán này yêu cầu duyệt qua một số lượng lớn các nút để đánh giá, điều này có thể trở nên không thực tế trong một khoảng thời gian hợp lý. Để tận dụng thời gian tính toán hạn chế, chúng ta có thể cắt bỏ tìm kiếm sớm và áp dụng một hàm đánh giá heuristics cho các trạng thái, hiệu quả như việc xem các nonterminal nodes (trạng thái chưa kết thúc) như là terminal nodes (trạng thái đã kết thúc). Nói cách khác, chúng ta thay thế hàm UTILITY bằng EVAL để ước lượng giá trị heuristic của một trạng thái. Đồng thời thay thế IS\_TERMINAL bằng IS\_CUTOFF để kiểm tra xem node hiện tại có phải là điểm cut-off (điểm cắt tỉa) hay không. Điều đó đưa chúng ta đến công thức H-MINIMAX(s, d) cho giá trị minimax heuristics của trạng thái tại độ sâu tìm kiếm d:

***Hình 3.1:*** *Công thức H-Minimax*

### Evaluation Function

Hàm EVAL (s, p) là một hàm đánh giá heuristics trả về ước lượng về tiện ích kỳ vọng của trạng thái s đối với người chơi p:

+ Đối với terminal nodes thì Eval(s, p) = Utility (s, p)

+ Đối với nonterminal nodes thì UTILITY(loss, p) ≤ EVAL(s, p) ≤ UTILITY(win, p)

Hàm đánh giá là một thành phần quan trọng trong các chương trình trò chơi, đặc biệt là trò đối kháng. Nó có nhiệm vụ đánh giá một thế cờ cụ thể, dự đoán bên nào có ưu thế hơn.

Một hàm đánh giá tốt ngoài những yêu cầu cơ bản đó thì cần phải:

• Thời gian tính toán không quá lâu: Điều này quan trọng vì mục tiêu của việc sử dụng hàm đánh giá là để tìm kiếm nhanh hơn. Nếu quá trình tính toán mất quá nhiều thời gian, thuật toán tìm kiếm sẽ trở nên không hiệu quả.

• Phải có mối tương quan mạnh mẽ với khả năng thực sự của việc chiến thắng: Hàm đánh giá nên phản ánh chính xác khả năng của mỗi nước đi đến việc chiến thắng trong trò chơi. Điều này đảm bảo rằng thuật toán tìm kiếm sẽ ưu tiên các nước đi có khả năng chiến thắng cao hơn.

Khi thuật toán tìm kiếm bị cắt tại các trạng thái không không phải phải terminal nodes, thuật toán sẽ không biết chính xác kết quả cuối cùng của những trạng thái đó.

Một số trạng thái có thể dẫn đến chiến thắng, một số dẫn đến hòa và một số dẫn đến thua. Trong trường hợp này, hàm đánh giá nên phản ánh khả năng chiến thắng, hòa hoặc thua của trạng thái một cách chính xác để thuật toán tìm kiếm có thể đưa ra quyết định tốt hơn.

### Cut off Search

Vấn đề thời gian buộc chúng ta phải cut off quá trình tìm kiếm sớm hơn, thay vì phải duyệt toàn bộ các nút trạng thái. Đối thủ sẽ không đủ kiên nhẫn cho việc bạn mất quá nhiều thời gian để đưa ra quyết định và nó sẽ ảnh hưởng tệ tới trải nghiệm chơi của hai bên.

Vậy khi nào nên dừng trong một thuật toán tìm kiếm? Có một số phương pháp để quyết định điều này. Cách đơn giản nhất là đặt một giới hạn độ sâu cố định, thường được ký hiệu là d. Độ sâu này được chọn sao cho thuật toán có thể chọn được một nước đi trong khoảng thời gian cụ thể. Một cách tiếp cận mạnh mẽ hơn là sử dụng phương pháp sâu rộng lặp đi lặp lại. Khi hết thời gian, thuật toán sẽ trả về nước đi được chọn thông qua tìm kiếm sâu nhất đã hoàn tất.

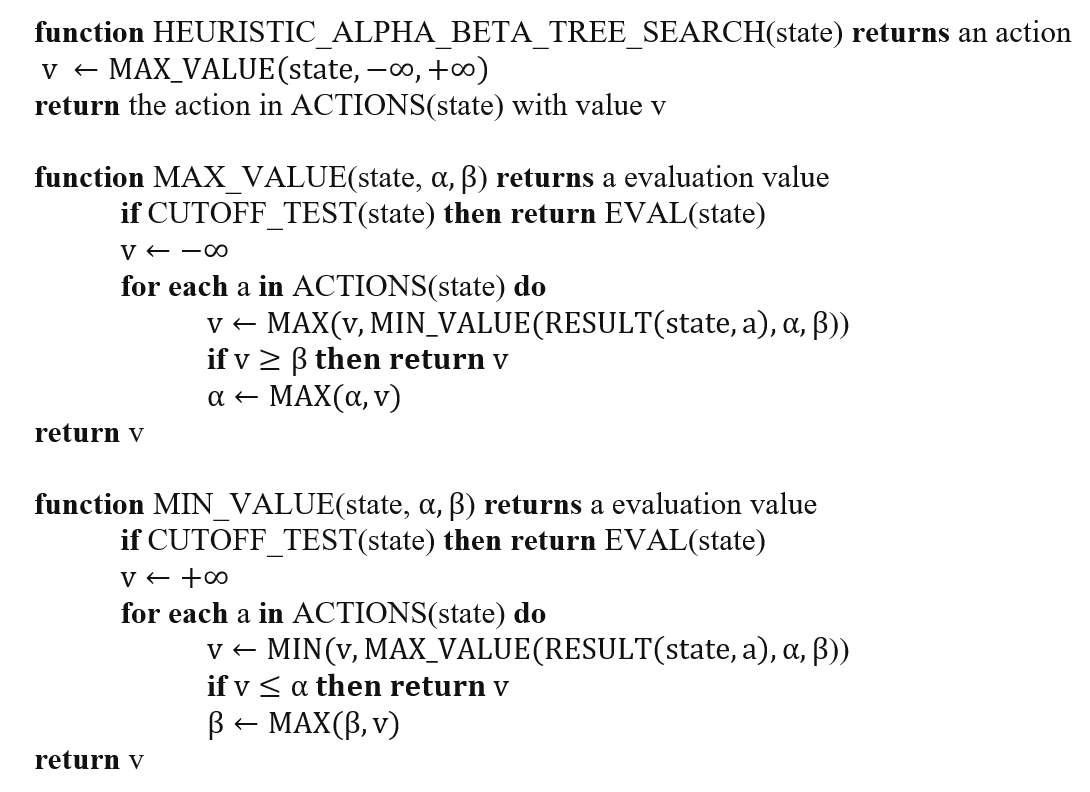
Tuy nhiên, các phương pháp này có thể gây ra lỗi do tính gần đúng của hàm đánh giá. Ví dụ, trong trường hợp của cờ vua, một nước đi có thể dẫn đến tình huống không lường trước được, khiến cho hàm đánh giá không còn đúng đắn.

Có một khái niệm quan trọng gọi là các vị trí ổn định. Ở các vị trí này, không có nước đi tiếp theo quan trọng. Ngược lại các vị trí không ổn định thường là những nước đi bắt quân đối phương có thể thay đổi giá trị đánh giá một cách mạnh mẽ. Điều này có nghĩa là các nước đi bắt quân đối phương có thể thay đổi đáng kể giá trị của vị trí và ảnh hưởng đến kết quả của hàm đánh giá. Do đó, chúng ta cần kiểm tra xem một vị trí có phải là vị trí ổn định hay không trước khi áp dụng hàm đánh giá.

Sách cờ vua giới thiệu cung cấp một tài liệu gần đúng giá trị đánh giá cho mỗi quân cờ: mỗi quân Tốt trị giá là 1, quân Tượng trị giá là 3, quân Xe trị giá là 5 và quân Hậu trị giá là 9. Các giá trị tính năng này sau đó được cộng lại đơn giản để có được đánh giá về vị trí. Khi cây trò chơi duyệt tới trường hợp có thể ăn được quân Hậu - đây là quân mạnh nhất trên bàn cờ và quan trọng thứ hai sau quân Vua. Quân Hậu trong trường hợp này có giá trị tương đương với chín quân Tốt, tức là hơn một chút so với một quân Xe và quân Tượng hợp lại, Việc trao đổi quân Hậu lấy một quân không phải quân Hậu của đối phương thường dẫn tới bất lợi lớn trong ván đấu. Do đó, ta không nên áp dụng hàm đánh giá tại các vị trí này

Trong trường hợp các vị trí không ổn định, thuật toán cắt tỉa sẽ trả về giá trị False và tiếp tục tìm kiếm cho đến khi đạt được một vị trí ổn định. Điều này giúp tránh việc đánh giá quá nhiều các vị trí không ổn định và đảm bảo rằng chỉ có các vị trí quan trọng nhất được đánh giá kỹ lưỡng.

### Mã giả của Heuristic Alpha-Beta Tree Search

Mã giả của Heuristic Alpha-Beta Tree Search khá tương đồng với Alpha-Beta Pruning. Chúng ta chỉ thay thế hàm UTILITY bằng hàm EVAL, cái sẽ được dùng để đánh giá utility của một trạng thái. Đồng thời thay thế TERMINAL\_TEST bằng CUTOFF\_TEST, hàm được dùng để kiểm tra một trạng thái có phải điểm cut-off hay không. Một trạng thái được gọi là điểm cut-off nếu nó đạt tới độ sâu d hoặc trạng thái đang xét là terminal nodes

**Hình 3.2:** Mã giả thuật toán Heuristic Alpha-Beta Tree Search

Giải thích chi tiết mã giả của thuật toán:

- HEURISTIC\_ALPHA\_BETA\_TREE\_SEARCH(state):

+ Đây là hàm chính của thuật toán, nhận vào trạng thái hiện tại (state) của trò chơi và trả về hành động tốt nhất có thể.

+ Trong hàm này, ta gọi MAX\_VALUE để tìm kiếm hành động tốt nhất cho người chơi hiện tại, sử dụng giá trị alpha và beta để cắt tỉa.

- MAX\_VALUE(state, α, β)

* Hàm này đại diện cho quá trình của người chơi cố gắng tối đa hóa giá trị của họ.
* Nếu trạng thái hiện tại là một điểm cut-off, ta trả về giá trị đánh giá của trạng thái đó bằng hàm EVAL
* Ta khởi tạo giá trị của v là âm vô cực.
* Ta duyệt qua từng hành động có thể có trong trạng thái hiện tại và gọi MIN\_VALUE cho các trạng thái kế tiếp tương ứng những hành động đó.
* Ta cập nhật giá trị của v là giá trị lớn nhất giữa v và giá trị trả về từ MIN\_VALUE.
* Nếu v đã lớn hơn hoặc bằng beta, ta cắt tỉa (có nghĩa là ta không cần xem xét các nút con khác của nút hiện tại), và trả về v.
* Nếu không, ta cập nhật giá trị của alpha là giá trị lớn nhất giữa alpha và v, và tiếp tục với các hành động còn lại.
* Cuối cùng, ta trả về giá trị của v

- MIN\_VALUE(state, α, β):

* Hàm này tương tự như MAX\_VALUE, nhưng đại diện cho quá trình của đối thủ cố gắng tối thiểu hóa giá trị của bạn.
* Nếu trạng thái hiện tại là một điểm cut-off, ta trả về giá trị đánh giá của trạng thái đó bằng hàm EVAL.
* Ta khởi tạo giá trị của v là dương vô cực.
* Ta duyệt qua từng hành động có thể có trong trạng thái hiện tại và gọi MAX\_VALUE cho các trạng thái kế tiếp tương ứng với những hành động đó.
* Ta cập nhật giá trị của v là giá trị nhỏ nhất giữa v và giá trị trả về từ MAX\_VALUE.
* Nếu v đã nhỏ hơn hoặc bằng alpha, ta cắt tỉa (có nghĩa là ta không cần xem xét các nút con khác của nút hiện tại), và trả về v.
* Nếu không, ta cập nhật giá trị của beta là giá trị nhỏ nhất giữa beta và v, và tiếp tục với các hành động còn lại.
* Cuối cùng, ta trả về giá trị của v.

Khi cả hai hàm MAX\_VALUE và MIN\_VALUE được gọi đệ quy lần lượt, giá trị của alpha và beta được cập nhật và truyền dẫn giữa các cấp độ của cây tìm kiếm. Việc cắt tỉa sẽ giúp giảm số lượng các trạng thái cần xem xét, tăng tốc độ của thuật toán. Cuối cùng, hàm HEURISTIC\_ALPHA\_BETA\_TREE\_SEARCH sẽ trả về hành động tốt nhất dựa trên kết quả của MAX\_VALUE.

### Ứng dụng thuật toán Heuristic Alpha-Beta Tree Search vào trò chơi Tic-Tac-Toe

* + 1. **Giới thiệu trò chơi Tic-Tac-Toe**

Tic-tac-toe là một trò chơi bảng phổ biến được viết trên bàn cờ 3x3 ô. Hai người chơi, một người sử dụng ký hiệu 'X' và người kia sử dụng ký hiệu 'O', thay phiên nhau điền vào các ô bằng ký hiệu của họ. Người chiến thắng là người có thể tạo 3 chuỗi biểu tượng duy nhất đầu tiên, theo chiều ngang, chiều dọc hoặc đường chéo.

* + 1. **Phân tích bài toán**

Dựa trên trạng thái ban đầu của bàn cờ, ta xây dựng một cây trò chơi với mỗi nút biểu diễn một trạng thái có thể xảy ra và các nút con tương ứng với các trạng thái có thể đạt được từ trạng thái hiện tại thông qua các nước đi hợp lệ. Các nút lá đại diện cho trạng thái cuối cùng của ván cờ, với ba kết quả là thắng, thua và hòa.

Thuật toán Tìm kiếm Cây Heuristic Alpha-Beta được áp dụng để duyệt qua cây trò chơi, nhằm xác định nước đi tối ưu nhất cho người chơi, giúp tăng cường khả năng chiến thắng. Trong bài toán này, giả định rằng thời gian suy nghĩ có hạn, do đó chỉ cho phép duyệt cây đến độ sâu tối đa là 2, thay vì tiếp tục duyệt đến các nút lá. Tại các điểm cắt (cut-off), một hàm heuristic được sử dụng để đánh giá trạng thái của nút đang xét, hỗ trợ việc đưa ra quyết định hợp lý cho nước đi tiếp theo.

* + 1. **Hàm đánh giá EVAL của trò chơi Tic-Tac-Toe**

Hàm này sẽ cố gắng ước lượng giá trị heuristic cho trạng thái của node hiện tại bằng cách:

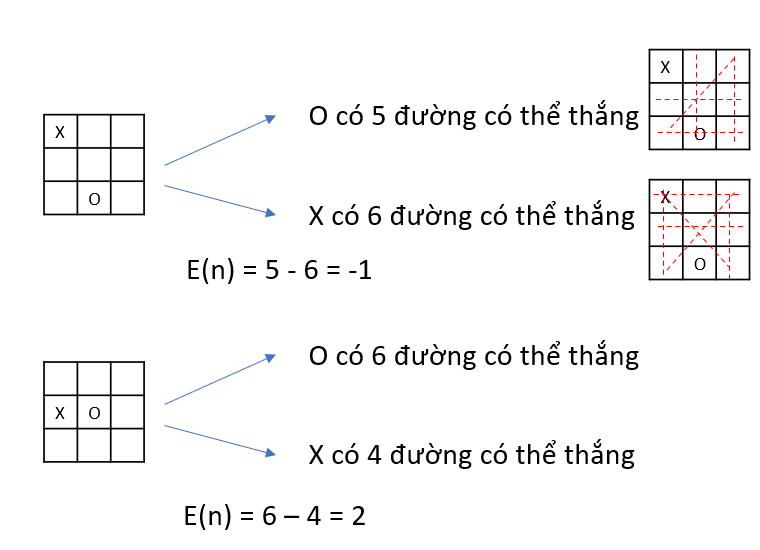
- Đối với terminal nodes:

+ Nếu MAX thắng thì giá trị heuristic = + ∞

+ Nếu MIN thắng thì giá trị heuristic = - ∞

- Đối với nonterminal nodes:

+ Giá trị heuristic được tính bằng cách đếm tất cả các đường chiến thắng mà MAX có thể mở trừ đi tất cả các dòng chiến thắng mà MIN có thể mở. Giá trị heuristic càng cao thì khả năng chiến thắng của MAX càng lớn



***Hình 3.3:*** *Minh họa cách hàm EVAL đánh giá một trạng thái của trò chơi Tic-Tac-Toe*

Hàm Heuristic: E(n) = M(n) – O(n)

Trong đó:

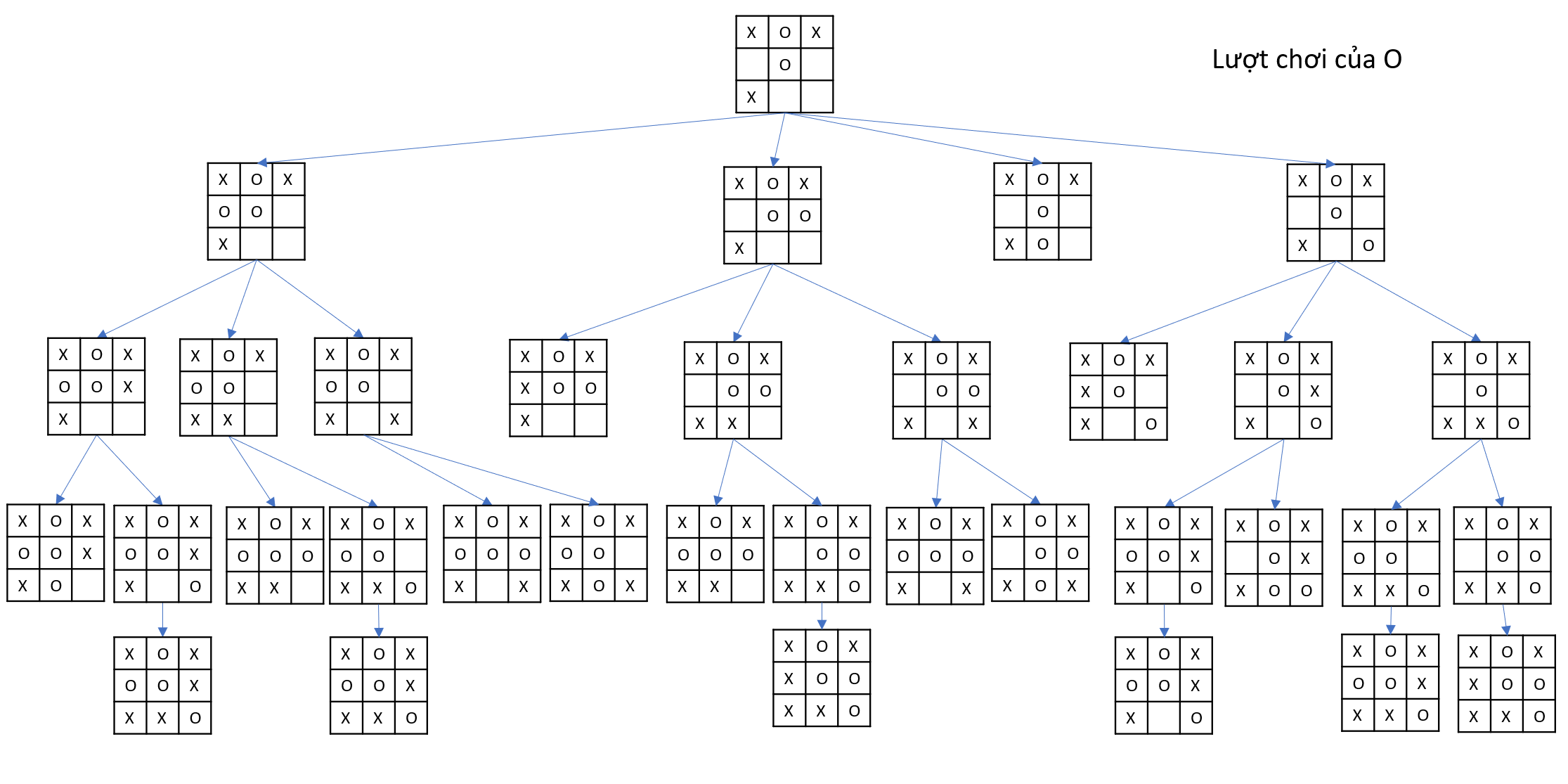
M(n) là tổng số đường thắng có thể của tôi

O(n) là tổng số đường thắng có thể của đối thủ

E(n) là trị số đánh giá tổng cộng cho trạng thái n

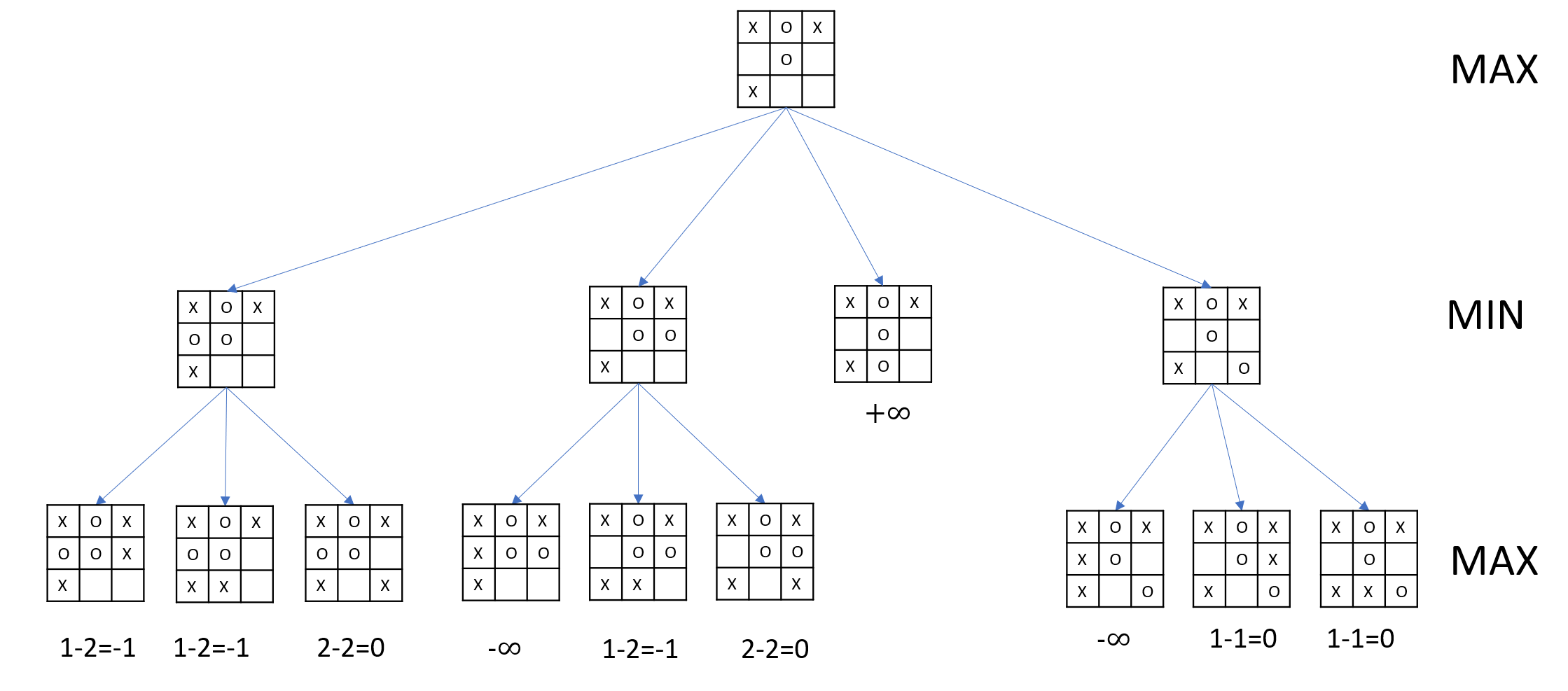
* + 1. **Demo xử lý bài toán**

Giả sử lượt chơi tiếp theo là của O, và O mong muốn tìm được nước đi tốt nhất cho mình.



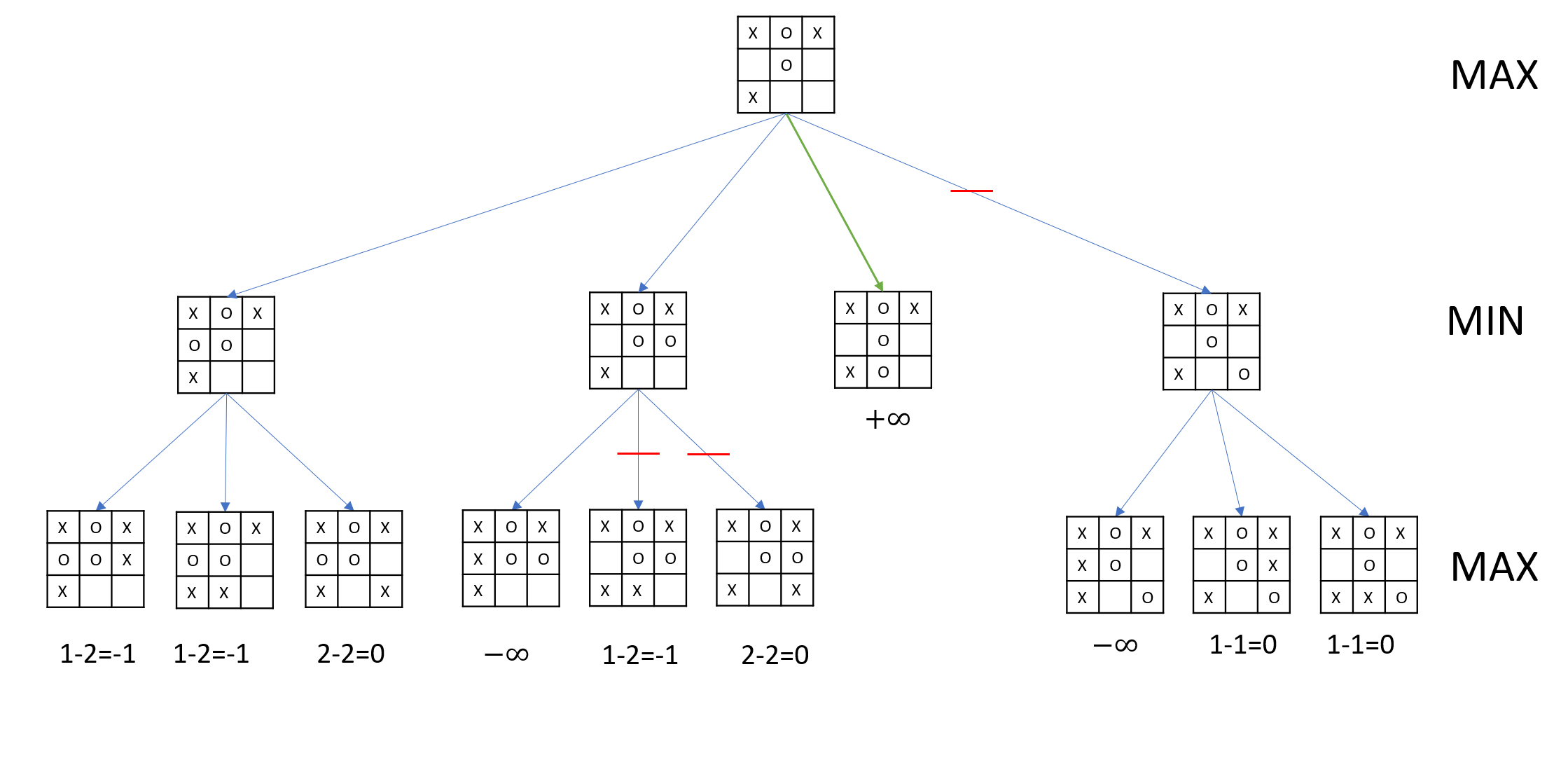
***Hình 3.4:*** *Cây trò chơi trong trường hợp duyệt tất cả khả năng có thể xảy ra*

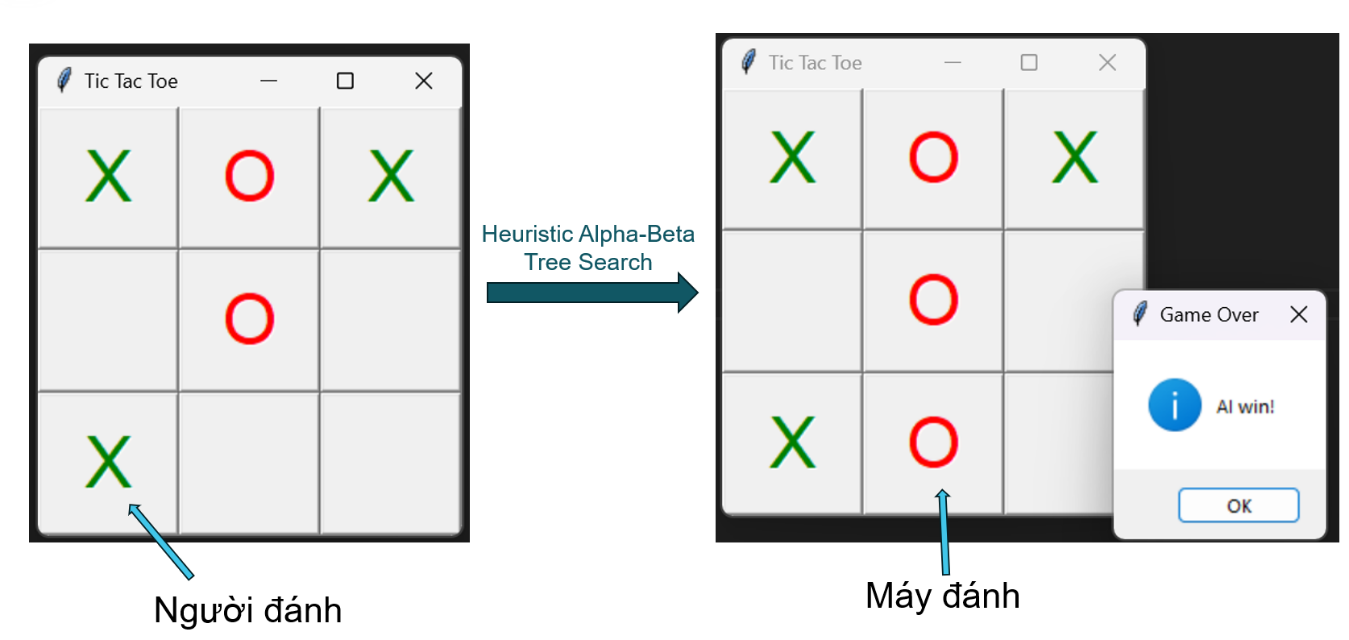
Có thể thấy có rất nhiều trường hợp cần phải xét, nhưng bằng thuật toán Heuristic Alpha Beta Tree Search, tôi sẽ giới hạn độ sâu tìm kiếm xuống còn 2. Tại những điểm được cut-off ta sẽ áp dụng hàm Heuristic đánh giá trạng thái tại các điểm đó.



***Hình 3.5:*** *Cây trò chơi khi được giới hạn độ sâu bằng 2*

Việc còn lại là áp dụng thuật toán Alpha-Beta Pruning đã đề cập mục 2 và ta sẽ thu được kết quả.



***Hình 3.6:*** *Cây trò chơi sau khi tiến cắt tỉa và kết quả tìm kiếm*

***Hình 3.7:*** *Kết quả thực tế khi thực hiện trên ngôn ngữ Python*

* + 1. **Ưu điểm và nhược điểm của Heuristic Alpha-Beta Tree Search**

Heuristic Alpha-Beta Tree Search là một phương pháp tìm kiếm cây quyết định trong trò chơi với các thuật toán Alpha-Beta pruning kết hợp với các hàm heuristic để đánh giá các trạng thái của trò chơi. Dưới đây là một số ưu điểm và nhược điểm của phương pháp này:

- Ưu điểm:

+ Hiệu quả về mặt thời gian: Heuristic Alpha-Beta Tree Search có khả năng cắt tỉa các nhánh không quan trọng trong cây tìm kiếm, giảm đáng kể số lượng nút phải kiểm tra. Điều này làm tăng hiệu suất của thuật toán, giúp tìm kiếm nước đi tốt nhất trong thời gian ít hơn.

+ Khả năng xử lý cây tìm kiếm lớn: Bằng cách sử dụng Alpha-Beta pruning, thuật toán có thể xử lý các cây tìm kiếm lớn mà không cần kiểm tra mọi nút, điều này làm cho nó phù hợp với các trò chơi có không gian trạng thái lớn

+ Sử dụng thông tin đánh giá ở mỗi nút: Bằng cách sử dụng hàm heuristic để đánh giá các trạng thái của trò chơi, thuật toán có thể dự đoán giá trị của các nút mà không cần phải duyệt hết các nút lá. Điều này giúp tăng tốc độ của thuật toán và làm cho nó trở nên thông minh hơn trong việc chọn các nước đi tiếp theo.

- Nhược điểm:

+ Độ chính xác của hàm heuristic: Sự hiệu quả của Heuristic Alpha-Beta Tree Search phụ thuộc rất nhiều vào chất lượng của hàm heuristic được sử dụng. Nếu hàm heuristic không đánh giá các trạng thái của trò chơi một cách chính xác, thuật toán có thể đưa ra các quyết định không tốt.

+ Khả năng mắc kẹt ở các trạng thái không thể cắt tỉa: Trong một số trường hợp, có thể xảy ra tình huống mà các trạng thái không thể cắt tỉa, dẫn đến việc thuật toán phải duyệt qua một số lượng lớn các nút, làm giảm hiệu suất của thuật toán.

+ Khó cài đặt hàm đánh giá: quá trình cài đặt có thể gặp khó khăn, đặc biệt là khi tích hợp vào các hệ thống phức tạp hoặc yêu cầu sự tương tác nhiều với các phần mềm khác. Sự phức tạp trong việc tích hợp này có thể dẫn đến tăng chi phí và thời gian triển khai, đồng thời tăng nguy cơ phát sinh lỗi trong quá trình triển khai và sử dụng.

## Các khái niệm liên quan

### Horizon Effect

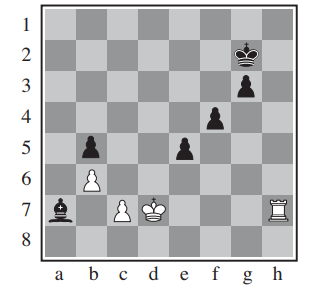
**Horizon Effect** hay hiệu ứng đường chân trời là một vấn đề phổ biến trong các thuật toán tìm kiếm được sử dụng trong trí tuệ nhân tạo, đặc biệt là trong các trò chơi đối kháng như cờ vua, cờ vây, caro ...

Hiệu ứng đường chân trời (Horizon Effect) trong trí tuệ nhân tạo (AI) là hiện tượng mà một thuật toán tìm kiếm có thể gặp phải khi đánh giá một trạng thái trò chơi chỉ dựa trên một số bước dựa trên độ sâu nhất định mà không xem xét đến tất cả các kết quả tiềm năng ở xa. Điều này dẫn đến việc thuật toán có thể đánh giá một trạng thái trò chơi là tốt nhưng thực tế nó không phải là tốt vì có nước đi trong tương lai có thể sẽ không theo mong muốn vì chỉ nhìn thấy một phần trên cây trò chơi.

Ví dụ: Giả sử bạn đang chơi cờ vua với máy tính. Bạn sắp chiếu vua máy tính, nhưng máy tính lại bỏ qua nước đi để thoát khỏi chiếu vua và tập trung vào các nước đi khác. Lý do là máy tính chỉ nhìn thấy các nước đi trước mắt và không tính đến nước đi thoát khỏi chiếu vua.

**\*Hướng giải quyết:**

Để giảm thiểu hiệu ứng Horizon, chiến lược "singular extensions" cho phép mở rộng những nước đi được xem xét nếu chúng được xác định là rõ ràng tốt hơn so với tất cả các nước đi khác trong tình huống đó. Bằng cách này, các nhánh quan trọng được mở rộng và cung cấp thông tin hữu ích hơn cho quá trình đánh giá và ra quyết định của chương trình. Điều này có thể giúp cải thiện khả năng dự đoán và tránh được các tình huống không lợi trong trò chơi.

Ví dụ:

Trong nước đi này chúng ta có thể thấy quân tượng đen đang bị ép vào thế phải bị đánh bại nhưng quân đen có thể trì hoãn việc mất tượng bằng cách chiếu vua trắng bằng các quân tốt, buộc vua trắng phải ăn các quân tốt điều này đẩy việc mất tượng xảy ra xa hơn trong tương lai tạo cơ hội cho quân đen có thêm thời gian để phản công. Do đó, thuật toán tìm kiếm coi việc hy sinh tốt là nước đi tốt thay vì nước đi xấu.

**Hình 4.1:** Ví dụ về Horizon Effect

**Kết luận:**

Hiệu ứng Horizon là một thách thức cho các thuật toán trí tuệ nhân tạo trong trò chơi đối kháng. Chiến thuật "singular extensions" là một cách để giảm thiểu hiệu ứng này và giúp trí tuệ nhân tạo đưa ra những quyết định tốt hơn trong những tình huống khó khăn.

### Forward Pruning

**So sánh Forward Pruning và Alpha-Beta Pruning :**

* **Forward Pruning** (Cắt tỉa hướng trước)**:** Là kỹ thuật cắt tỉa các nước đi có vẻ bất lợi nhưng vẫn có khả năng là nước đi tốt. Mục đích của nó là tiết kiệm thời gian tính toán nhưng có rủi ro là đưa ra quyết định sai lầm.
* **Alpha-Beta Pruning:** Là kỹ thuật cắt tỉa các nhánh cây không tiềm năng trong quá trình tìm kiếm nước đi tốt nhất cho trò chơi đối kháng. Nó loại bỏ các nhánh cây mà chắc chắn không thể ảnh hưởng đến đánh giá cuối cùng.

**Một vài cách tiếp cận khác của Forward Pruning:**

**Beam Search:**

* Đây là một cách tiếp cận cụ thể để cắt tỉa hướng trước, trong đó ở mỗi cấp độ của cây tìm kiếm (ply), chỉ có một số lượng giới hạn (chùm) các nước đi hứa hẹn nhất (dựa trên hàm đánh giá) được xem xét.
* Phương pháp này có thể giảm đáng kể số nhánh được khám phá nhưng có nguy cơ loại bỏ nước đi tốt nhất thực sự nếu nó không nằm trong số "n" nước đi được đánh giá hàng đầu.

**Lưu ý:** không có gì đảm bảo nước đi tốt nhất sẽ không bị cắt tỉa.

**Probcut:**

* Thuật toán này kết hợp cắt tỉa hướng trước với cách tiếp cận xác suất để giảm thiểu nguy cơ loại bỏ nước đi tốt nhất.
* Nó tận dụng kinh nghiệm trước đây (thống kê) để ước tính khả năng một nước đi dẫn đến điểm số ngoài phạm vi alpha-beta (phạm vi điểm số có thể sử dụng để cắt tỉa trong một số thuật toán tìm kiếm).
* Tương tự như cắt tỉa alpha-beta, ProbCut loại bỏ các nước đi nằm chắc chắn ngoài phạm vi alpha-beta.
* Tuy nhiên, đối với các nước đi có kết quả không chắc chắn (trong phạm vi alpha-beta nhưng điểm số cuối cùng không xác định), ProbCut sử dụng xác suất ước tính (dựa trên kinh nghiệm trước đây) để quyết định có cắt tỉa hoặc giữ lại nước đi để khám phá thêm hay không.

**Kết luận:**

Lựa chọn kỹ thuật cắt tỉa phù hợp phụ thuộc vào mục tiêu cụ thể của thuật toán và đặc điểm của trò chơi. Việc cân bằng giữa hiệu quả và độ chính xác là yếu tố quan trọng để tối ưu hóa hiệu suất của AI chơi game.

### Search và Lookup trong AI

**\* Lấy ví dụ về việc chơi cờ vua:**

**Vấn đề:**

* Chương trình cờ vua cần đưa ra quyết định sáng suốt nhưng phải chịu giới hạn về sức mạnh xử lý.
* Việc áp dụng brute-force cho mọi chuỗi di chuyển có thể tốn kém về mặt tính toán và không cần thiết.

**Khái niệm về Search và Lookup :**

* **Search** (Tìm kiếm) : Phương pháp này liên quan đến việc phân tích cây các nước đi tiềm năng trong tương lai,liên quan đến việc xây dựng "cây" trạng thái trò chơi, nơi mỗi nhánh đại diện cho một nước đi tiềm năng. Đánh giá mỗi nước đi và chọn lựa chọn hứa hẹn nhất. Tuy nhiên, điều này có thể tốn thời gian cho các tình huống phức tạp.
* **Lookup** (Tra cứu) : Phương pháp này liên quan đến việc tra cứu một cơ sở dữ liệu (bảng) được tính toán trước về các vị trí và nước đi tối ưu của chúng. Điều này nhanh hơn nhiều so với tìm kiếm nhưng có thể không chính xác cho các tình huống không phổ biến.

**Cách tiếp cận lý tưởng:**

* **Khai cuộc:** Vì có tương đối ít nước đi hợp lệ trong khai cuộc, chương trình có thể sử dụng hiệu quả bảng tra cứu để chọn một nước đi khai cuộc mạnh
* **Trung cuộc:** Đây là giai đoạn phức tạp nhất với số lượng nước đi khả thi cao nhất. Ở đây, chương trình có thể chuyển sang thuật toán tìm kiếm để phân tích các nước đi tiềm năng và đưa chiến lược tốt nhất.
* **Hậu cuộc:** Hậu cuộc thường liên quan đến ít quân cờ hơn và nhiều mô hình lặp lại hơn, khiến cho bảng tra cứu có thể hữu ích trở lại cho các đường chiến thắng đã biết hoặc chiến lược phòng thủ.

**Con người so Máy tính:**

* **Lợi thế tìm kiếm:** Máy tính có thể phân tích nhiều khả năng hơn con người, cho phép chúng xác định kỹ lưỡng bước đi và chiến lược đột phá mà con người có thể bỏ lỡ.
* **Chuyên môn của con người:** Con người xuất sắc trong việc hiểu các khái niệm vị trí và lập kế hoạch dài hạn, những điều mà các chương trình cờ vua máy tính hiện tại không nắm bắt đầy đủ. Điều này đặc biệt đúng đối với các thế cờ tàn phức tạp với các kết hợp quân cờ cụ thể như “Vua, tượng và mã so với Vua”

**Kết luận :**

* Chương trình cờ vua sử dụng kết hợp các chiến lược tìm kiếm và tra cứu để có hiệu suất tối ưu.
* Lựa chọn giữa tìm kiếm và tra cứu phụ thuộc vào độ phức tạp của vị trí.
* Ngay cả với khả năng tìm kiếm mạnh mẽ, máy tính cũng khó có thể sở hữu hiểu biết chiến lược giống như kỳ thủ cờ vua trong những tình huống cụ thể.

# **PHẦN 3: TỔNG KẾT**

Heuristic Alpha-Beta Tree Search là một thuật toán tìm kiếm hiệu quả được sử dụng rộng rãi trong các trò chơi đối kháng như cờ vua, cờ vây và cờ caro. Nó hoạt động bằng cách duyệt qua cây trò chơi, đánh giá các trạng thái tiềm năng và loại bỏ các nhánh không hứa hẹn để tiết kiệm thời gian tính toán.

**Ứng dụng của thuật toán trong trò chơi:**

Heuristic Alpha-Beta Tree Search được sử dụng phổ biến trong các chương trình chơi cờ vua, cờ vây, cờ caro và các trò chơi đối kháng khác để tìm ra nước đi tối ưu cho máy tính. Nhờ hiệu quả và độ chính xác cao, nó đóng vai trò quan trọng trong việc phát triển các chương trình chơi game trí tuệ nhân tạo (AI) mạnh mẽ, có khả năng cạnh tranh với con người ở mức độ cao.

**Tóm lại,** Heuristic Alpha-Beta Tree Search là một thuật toán tìm kiếm mạnh mẽ và linh hoạt với nhiều ứng dụng trong trí tuệ nhân tạo và khoa học máy tính. Nó đóng vai trò quan trọng trong việc phát triển các chương trình chơi game mạnh mẽ, hệ thống ra quyết định thông minh và giải pháp tối ưu hóa hiệu quả. Tuy nhiên, cần lưu ý đến độ phức tạp của thuật toán và chất lượng của hàm đánh giá khi sử dụng Heuristic Alpha-Beta Tree Search.

# **PHẦN 4: TÀI LIỆU THAM KHẢO**

[1] Tác giả: Sammar Saleh. Tên bài viết: Heuristic Alpha-Beta Tree Search Algorithm (Tic Tac Toe). Đường dẫn: <https://medium.com/@sammar.saleh00/heuristic-alpha-beta-tree-search-algorithm-tic-tac-toe-4338807b1ac7> . Thời gian truy cập: 18/04/2024.

[2] Tác giả: Stuart Russell and Peter Norvig. Tên sách: "Artificial Intelligence: A Modern Approach. Thời gian truy cập: 18/04/2024.

[3] Tác giả: Nguyễn Đình Cường. Đường dẫn: <https://www.researchgate.net/profile/Nguyen-Dinh-Cuong-2/publication/331310387_Artificial_Intelligence_TRI_TUE_NHAN_TAO_Bai_giang_Nha_trang-2012_NGUYEN_DINH_CUONG_Bo_mon_Cong_Nghe_Phan_Mem_Khoa_Cong_Nghe_Thong_Tin/links/5c71f2a7299bf1268d1fdb86/Artificial-Intelligence-TRI-TUE-NHAN-TAO-Bai-giang-Nha-trang-2012-NGUYEN-DINH-CUONG-Bo-mon-Cong-Nghe-Phan-Mem-Khoa-Cong-Nghe-Thong-Tin.pdf>. Thời gian truy cập: 16/04/2024.

[4] Tác giả: Wikipedia. Tên bài viết: Tic-Tac-Toe.

Đường dẫn: <https://vi.wikipedia.org/wiki/Tic-tac-toe> . Thời gian truy cập: 17/04/2024.

[5] Tác giả: Dr.Zuhair Zafar. Tên bài viết: Artificial Intelligence. Đường dẫn: <https://fr.scribd.com/document/544005225/Lecture-11> . Thời gian truy cập: 18/04/2024.

[6] Tác giả: Dục Đoàn Trình. Tên bài viết: Thuật toán Alpha-Beta Pruning. Đường dẫn: <https://websitehcm.com/thuat-toan-alpha-beta-pruning/> . Thời gian truy cập: 18/04/2024.

[7] Tác giả: Wikipedia. Tên bài viết: Minimax. Đường dẫn: <https://vi.wikipedia.org/wiki/Minimax> . Thời gian truy cập: 18/04/2024.

[8] Tác giả: Tom Lenaerts. Tên bài viết: Artificial Intelligence 1: Game playing. Đường dẫn: <https://fr.slideshare.net/MhdSb/6-games-53858104?from_search=9> . Thời gian truy cập: 17/04/2024.

[9] Tác giả: Dheerendra. Tên bài viết: Adversarial Search. Đường dẫn: <https://fr.slideshare.net/DheerendraKumar15/adversarial-search-83425734> . Thời gian truy cập: 17/04/2024.

[10] Tác giả: Dục Đoàn Trình. Tên bài viết: Thuật toán Mini-Max trong Trí tuệ nhân tạo. Đường dẫn: <https://websitehcm.com/thuat-toan-mini-max-trong-tri-tue-nhan-tao/> . Thời gian truy cập: 18/04/2024.

[11] Tác giả: SHICHIKI LÊ. Tên bài viết: Giải thuật tìm kiếm Minimax. Đường dẫn: <https://www.iostream.co/article/giai-thuat-tim-kiem-minimax-s1EVnH> . Thời gian truy cập: 18/04/2024.

[12] Tác giả: Bro. Tên bài viết: Thuật toán Minimax (AI trong Game). Đường dẫn: <https://viblo.asia/p/thuat-toan-minimax-ai-trong-game-APqzeaVVzVe> . Thời gian truy cập: