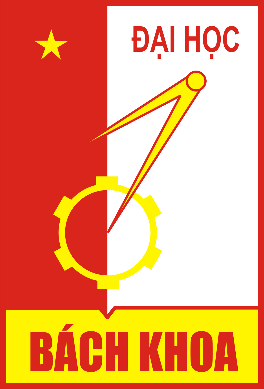
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**Trường Công nghệ thông tin và Truyền thông**



**ĐỀ TÀI: SO SÁNH CÁC THUẬT TOÁN AI TRONG CỜ CARO (TICTACTOE)**

**Môn: Nhập môn trí tuệ nhân tạo**

**GVHD:** TS. Trần Thế Hùng

**Nhóm thực hiện:** Nhóm 11

Nguyễn Văn Phú An 20214982

Phạm Hữu Phúc 20215119

Lương Phúc Quang 20215125

Nguyễn Hoàng Ninh Thuận 20215141

Đặng Minh Chức 20215001

Nguyễn Văn Nam 20215097

*Hà Nội, tháng 6 năm 2024*

**Nội dung**

[I. Tên đề tài, danh sách thành viên 2](#_Toc1746109665)

[1. Tên đề tài : AI chơi cờ caro 2](#_Toc84796069)

[2. Danh sách thành viên, phân công công việc 2](#_Toc2038324995)

[II. Giới thiệu đề tài 3](#_Toc1412128131)

[1. Tổng quan đề tài 3](#_Toc270144130)

[2. Luật chơi Tic-Tac-Toe 3](#_Toc1263064114)

[3. Thiết kế/ Xây dựng logic game theo luật chơi 4](#_Toc801175698)

[3.1. Khởi tạo bàn chơi 4](#_Toc1287910301)

[3.2. Hàm đánh giá 5](#_Toc66582820)

[III. Giới thiệu các thuật toán 6](#_Toc1865243327)

[1. Minimax 6](#_Toc1533151426)

[2. Cắt tỉa Alpha Beta 8](#_Toc1957461494)

[3. Monte Carlo Tree Search 11](#_Toc1439217629)

[IV. Kết quả thử nghiệm 12](#_Toc549333816)

[1.Thuật toán minimax 12](#_Toc1457913469)

[2. Thuật toán cắt tỉa alpha beta 13](#_Toc612284187)

[3.Thuật toán Monte Carlo Tree Search 15](#_Toc1390396670)

[4. Kết luận 16](#_Toc1768570432)

# **Tên đề tài, danh sách thành viên**

## Tên đề tài : **AI chơi cờ caro**

## Danh sách thành viên, phân công công việc

|  |  |  |
| --- | --- | --- |
| Nguyễn Văn Nam | 20215097 | Lập trình logic game |
| Phạm Hữu Phúc | 20215119 |
| Nguyễn Hoàng Ninh Thuận | 20215141 | Lập trình giao diện , thuật toán cắt tỉa Alphabeta |
| Đặng Minh Chức | 20215001 |
| Lương Phúc Quang | 20215125 | Thuật toán minimax |
| Nguyễn Văn Phú An | 20214982 | Thuật toán Monte carlo tree search |

# **Giới thiệu đề tài**

## 1. Tổng quan đề tài

Chủ đề báo cáo của chúng em là trò chơi Tic-Tac-Toe, một trò chơi đơn giản nhưng đầy thách thức, thường được sử dụng làm ví dụ trong lĩnh vực Trí Tuệ Nhân Tạo (AI) để minh họa cách các thuật toán tìm kiếm và ra quyết định hoạt động. Trong báo cáo này, chúng em sẽ triển khai và so sánh ba thuật toán phổ biến: minimax, cắt tỉa alpha-beta, và Monte Carlo.

Thuật toán minimax được sử dụng để tìm nước đi tốt nhất bằng cách đánh giá tất cả các kịch bản có thể xảy ra, trong khi đó cắt tỉa alpha-beta là một cải tiến của minimax, giúp giảm thiểu số lượng trạng thái cần đánh giá bằng cách loại bỏ những nhánh không cần thiết. Cuối cùng, thuật toán Monte Carlo sử dụng phương pháp xác suất để ước lượng giá trị của các nước đi dựa trên các trận đấu mô phỏng. Mục tiêu của chúng tôi là so sánh hiệu suất, độ chính xác và thời gian thực thi của các thuật toán này trong việc giải quyết bài toán Tic-Tac-Toe, từ đó rút ra những kết luận quan trọng về ứng dụng của chúng trong các bài toán AI phức tạp hơn.

## 2. Luật chơi Tic-Tac-Toe

* Bàn cờ: Trò chơi được chơi trên một bảng gồm 3 hàng và 3 cột, tổng cộng có 9 ô vuông.
* Người chơi: Có hai người chơi, thường được gọi là "X" và "O". Một người chơi sẽ chơi với ký hiệu "X" và người kia chơi với ký hiệu "O".
* Cách chơi:
* Người chơi "X" luôn đi trước.
* Người chơi sẽ lần lượt đặt ký hiệu của mình ("X" hoặc "O") vào một ô trống trên bàn cờ.
* Người chơi không được đặt ký hiệu vào ô đã có ký hiệu trước đó.
* Mục tiêu: Mục tiêu của mỗi người chơi là tạo ra một hàng ngang, hàng dọc hoặc đường chéo gồm ba ký hiệu của mình liên tiếp.
* Kết thúc trò chơi:
* Trò chơi kết thúc khi một trong hai người chơi tạo ra được một hàng ngang, hàng dọc hoặc đường chéo gồm ba ký hiệu của mình liên tiếp. Người đó sẽ là người chiến thắng.
* Nếu tất cả các ô trên bàn cờ đều đã được điền kín mà không ai tạo được một hàng ngang, hàng dọc hoặc đường chéo gồm ba ký hiệu của mình, trò chơi sẽ kết thúc với kết quả hòa.

## 3. Thiết kế/ Xây dựng logic game theo luật chơi

### 3.1. Khởi tạo bàn chơi

* Quá trình khởi tạo game Tic Tac Toe bắt đầu với việc thiết lập các thuộc tính cơ bản và trạng thái ban đầu của trò chơi thông qua class Game. Khi một đối tượng của lớp này được tạo, kích thước của bàn cờ được xác định, thường là 3x3. Bàn cờ được khởi tạo dưới dạng một mảng hai chiều trống, đảm bảo tất cả các ô đều chưa có nước đi nào.
* Số lượt chơi được đặt về 0 để theo dõi tiến trình của trò chơi. Lượt chơi đầu tiên được thiết lập cho người chơi 'x', chuẩn bị sẵn sàng cho họ thực hiện nước đi đầu tiên. Quá trình khởi tạo này đảm bảo rằng trò chơi bắt đầu với một bàn cờ trống và đúng lượt người chơi, tạo nền tảng vững chắc cho các bước tiếp theo trong quá trình chơi.
* Mã nguồn :
  + **‘**\_\_init\_\_**’**: Hàm khởi tạo cho class Game, thiết lập kích thước bàn cờ, trạng thái ban đầu của bàn cờ, số lượt đã chơi và lượt hiện tại.
    - self.size: Lưu trữ kích thước của bàn cờ.
    - self.board: Khởi tạo bàn cờ kích thước size x size với các ô trống.
    - self.turn\_count: Khởi tạo số lượt đã chơi là 0.
    - self.player\_turn: Khởi tạo lượt đầu tiên là của người chơi 'x'.
  + In bàn cờ:‘print\_board’- Hàm in ra trạng thái hiện tại của bàn cờ.
    - board\_string: Chuỗi lưu trữ các ô của bàn cờ dưới dạng có thể in ra.
  + Kiểm Tra Điều Kiện Thắng: ‘check\_win’- Hàm kiểm tra toàn bộ bàn cờ để xem có người chơi nào thắng không. Gọi 2 hàm ‘check\_rows’ và ‘check\_diagonals’ để xác định người thắng.
    - ‘check\_rows’: Hàm kiểm tra xem có người chơi nào thắng theo hàng hoặc cột.
    - ‘check\_diagonals’: Hàm kiểm tra xem có người chơi nào thắng theo đường chéo.
  + Thực hiện nước đi:
    - ‘make\_move’: Hàm thực hiện nước đi đặt một quân cờ vào ô trên bàn cờ.
    - ‘undo\_move’: Hàm hoàn tác nước đi ỏ quân cờ khỏi ô trên bàn cờ và chuyển lượt về người chơi trước.
    - ‘make\_human\_move’: Hàm thực hiện nước đi của người chơi
    - ‘make\_random\_move’: Hàm thực hiện nước đi ngẫu nhiên của
  + Bắt Đầu Trò Chơi**:** ‘start\_game’-Hàm vòng lặp chính của trò chơi, kiểm tra điều kiện thắng hoặc hòa sau mỗi lượt.Nó sẽ in bàn cờ,thực hiện lượt chơi,tăng số lượt chơi sau khi vòng lặp kết thúc sẽ in lại bàn cờ lần cuối.Sau đó sẽ kiểm tra người chiến thắng trả về 1 nếu 'x' thắng, 2 nếu 'o' thắng và 0 nếu hòa hoặc chưa có người thắng.

### 3.2. Hàm đánh giá

- Mục đích : tính toán điểm số cho mỗi người chơi dựa trên trạng thái hiện tại của bàn cờ

- Ý tưởng đánh giá:

* Điểm số cho hàng ngang và hàng dọc
  + Nếu có 2 quân của người chơi liên tiếp và không bị chặn, cộng một lượng điểm lớn (ví dụ: 3 điểm) và thêm số ô trống còn lại trên bàn cờ.
  + Nếu có 1 quân của người chơi trong một hàng hoặc cột mà không có quân đối thủ, cộng một lượng điểm nhỏ (ví dụ: 1 điểm).
* Điểm số cho đường chéo
  + Tương tự như hàng ngang và hàng dọc, nhưng áp dụng cho các đường chéo từ trái sang phải và từ phải sang trái.
* Hiện thực hóa ý tưởng thành mã nguồn
  + Hàm evaluate\_player:
    - Gọi các hàm con để đánh giá hàng ngang, hàng dọc và đường chéo.
    - Tổng hợp điểm số từ các hàm con và trả về điểm cuối cùng cho người chơi.
  + Hàm evaluate\_horizontal:
    - Duyệt qua từng hàng của bàn cờ và tính điểm dựa trên các tiêu chí đánh giá.
    - Chuyển đổi hàng thành cột bằng cách sử dụng phép chuyển vị (transpose) và thực hiện đánh giá tương tự.
  + Hàm evaluate\_diagonal:
    - Tạo mảng chứa các phần tử trên đường chéo từ trái sang phải và từ phải sang trái.
    - Tính điểm cho các đường chéo dựa trên các tiêu chí đánh giá.

# **Giới thiệu các thuật toán**

## Minimax

Tổng quan:

* Thuật toán "Minimax" là một kỹ thuật tìm kiếm thường được sử dụng trong các trò chơi đối kháng như Tic Tac Toe, Cờ Vua, và Cờ Đam.
* Tên gọi "Minimax" xuất phát từ hai mục tiêu chính của thuật toán:
* Minimization (Tối thiểu hóa): Thuật toán cố gắng giảm thiểu mức độ thất bại có thể gặp phải. Người chơi sẽ chọn các nước đi làm cho đối thủ có kết quả xấu nhất.
* Maximization (Tối đa hóa): Thuật toán cố gắng tối đa hóa lợi ích có thể đạt được. Người chơi sẽ chọn các nước đi làm cho mình có kết quả tốt nhất.

Ý tưởng thuật toán:

* Đệ quy:
* Thuật toán Minimax sử dụng đệ quy để duyệt qua tất cả các trạng thái có thể của trò chơi.
* Mỗi lần đệ quy tương ứng với một lượt đi của một người chơi.
* Đánh giá trạng thái:
* Khi đạt đến một trạng thái kết thúc của trò chơi (ví dụ: thắng, thua hoặc hòa), thuật toán đánh giá trạng thái đó và trả về một điểm số.
* Lựa chọn nước đi:
* Khi đệ quy trở về từ một lượt đi, thuật toán chọn nước đi tốt nhất dựa trên các điểm số đã đánh giá.
* Nếu là lượt của người chơi Maximizer, nước đi tốt nhất là nước đi có điểm số cao nhất.
* Nếu là lượt của người chơi Minimizer, nước đi tốt nhất là nước đi có điểm số thấp nhất.

Chi tiết thực hiện:

* Khởi tạo:
* Depth: xác định số lần đệ quy sẽ thực hiện để tìm kiếm nước đi tối ưu.
* Player: xác định người chơi hiện tại là Maximizer hoặc Minimizer.
* curr\_max: Lưu trữ điểm số tốt nhất cho người chơi hiện tại.
* curr\_best\_move: Lưu trữ nước đi tốt nhất tìm được cho mỗi lượt.
* Kiểm tra trạng thái kết thúc:
* Thắng cuộc:
  + Nếu game.check\_win() == self.maximizer, điều này có nghĩa là người chơi Maximizer đã thắng.
  + Trong trường hợp này, hàm trả về 1000 điểm.
* Thua cuộc:
  + Nếu game.check\_win() != self.maximizer và game.check\_win() != 0, điều này có nghĩa là người chơi Maximizer đã thua (Minimizer thắng).
  + Hàm trả về -1000 điểm.
* Hòa:
  + Nếu game.get\_empty\_squares\_count() == 0, tức là không còn ô trống nào trên bàn cờ và không có người thắng.
  + Hàm trả về 0 điểm.
* Đạt độ sâu tối đa:
  + Nếu depth == 0, nghĩa là đã đạt đến độ sâu tối đa được chỉ định cho thuật toán.
  + Trong trường hợp này, hàm tính điểm dựa trên hàm đánh giá của lớp game (game.evaluate\_player(player)) và trả về kết quả này.
* Duyệt qua các ô trống:
* Hàm duyệt qua từng ô trống trên bàn cờ và thử nước đi đó.
* Đối với mỗi nước đi, hàm đệ quy để tính điểm số cho trạng thái mới và cập nhật curr\_max và curr\_best\_move nếu cần.
* Trả về kết quả:
* Hàm trả về điểm số tốt nhất (curr\_max) và nước đi tốt nhất (curr\_best\_move) cho người chơi hiện tại.

## 2. Cắt tỉa Alpha Beta

Thuật toán Cắt tỉa Alpha Beta (Alpha Beta Prunning) là thuật toán tìm kiếm cải tiến của thuật toán Minimax, được sử dụng để giảm số lượng trạng thái cần xem xét trong quá trình tìm kiếm nước đi tối ưu.

Ý Tưởng Chính của Thuật Toán Alpha-Beta Pruning

* Cắt Tỉa:
  + Nếu một nhánh tìm kiếm nào đó không thể cải thiện đối với giá trị mà chúng ta đã có, thì không cần xét đến nhánh tìm kiếm đó nữa!
  + Việc cắt tỉa các nhánh tìm kiếm tồi sẽ không ảnh hưởng đến kết quả cuối cùng.
  + Sử dụng hai biến α (alpha) và β (beta) để theo dõi giới hạn của giá trị tối ưu mà Maximizer và Minimizer có thể đạt được.
* Alpha (α):
  + Đại diện cho điểm số tối ưu nhất mà Maximizer có thể đảm bảo được tại bất kỳ điểm nào trong cây tìm kiếm.
* Beta (β):
  + Đại diện cho điểm số tối ưu nhất mà Minimizer có thể đảm bảo được tại bất kỳ điểm nào trong cây tìm kiếm.
* Cắt Tỉa Nhánh:
  + Nếu tại bất kỳ điểm nào, beta nhỏ hơn hoặc bằng alpha, có nghĩa là một trạng thái tối con ưu hơn đã được duyệt qua trước đó ở nút cha trên cây tìm kiếm, khi đó thuật toán sẽ cắt tỉa nhánh đó và không tiếp tục đánh giá các trạng thái con của nó.

Chi Tiết Thực Hiện của Hàm Alpha-Beta Pruning trong chương trình

* Khởi Tạo:
  + Tạo một instance của lớp AlphaBeta, với các thông tin về trò chơi và người chơi hiện tại.
  + self.best\_move: Lưu trữ nước đi tốt nhất tìm được.
  + self.node\_count: Đếm số lượng nút được duyệt qua trong cây tìm kiếm.
  + self.maximizer: Xác định người chơi Maximizer dựa trên lượt đi của người chơi ban đầu. Trong chương trình nhóm em đã đặt mặc định người chơi là Maximizer, còn AI sẽ là Minimizer.
* Cắt Tỉa Alpha-Beta:
  + Hàm alpha\_beta\_pruning được gọi đệ quy để duyệt qua các trạng thái có thể của trò chơi.
  + depth: Độ sâu tối đa của cây tìm kiếm .
  + player: Người chơi hiện tại (Maximizer hoặc Minimizer).
  + alpha và beta: Các giá trị cắt tỉa ban đầu, được khởi tạo lần lượt là - ∞ và + ∞.
* Kiểm Tra Trạng Thái Kết Thúc:
  + Nếu game.check\_win() == self.maximizer, trả về 1000 điểm (Maximizer thắng).
  + Nếu game.check\_win() != self.maximizer và game.check\_win() != 0, trả về -1000 điểm (Maximizer thua).
  + Nếu không còn ô trống (game.get\_empty\_squares\_count() == 0), trả về 0 điểm (hòa).
  + Nếu đạt độ sâu tối đa (depth == 0), trả về điểm số dựa trên hàm đánh giá (game.evaluate\_player(player)).
* Thực hiện các nước đi mô phỏng:
  + Với mỗi ô trống, thực hiện nước đi và gọi đệ quy hàm alpha\_beta\_pruning với trạng thái mới.
  + Maximizer: Cập nhật curr\_max và alpha với giá trị lớn nhất giữa alpha và curr\_max.
  + Minimizer: Cập nhật curr\_max và beta với giá trị nhỏ nhất giữa beta và curr\_max.
  + Nếu α ≤ β, break khỏi vòng while để cắt tỉa nhánh hiện tại và không duyệt tiếp.
* Cập Nhật và Trả Về:
  + Cập nhật self.best\_move với nước đi tốt nhất tìm được.
  + Trả về curr\_max là điểm số tốt nhất cho người chơi hiện tại.

## 3. Monte Carlo Tree Search

Thuật toán Monte Carlo Tree Search (MCTS) là một thuật toán tìm kiếm ngẫu nhiên được sử dụng rộng rãi trong trí tuệ nhân tạo (AI), đặc biệt là trong các trò chơi chiến lược theo lượt như cờ vây, cờ tướng, cờ caro, ...

MCTS hoạt động bằng cách xây dựng dần dần một "cây trò chơi" mô tả các trạng thái có thể xảy ra của trò chơi. Nó sử dụng mô phỏng ngẫu nhiên để đánh giá các nước đi tiềm năng và sau đó tập trung nỗ lực tìm kiếm vào các nước đi có vẻ hứa hẹn nhất.

**MCTS gồm có 4 bước chính như sau:**

* + **Chọn (Selection):**
  + Bắt đầu từ nút gốc, sử dụng một chính sách lựa chọn (ví dụ: Upper Confidence Bound Applied to Trees - UCT) để chọn một nút con.
  + Chính sách này cân bằng giữa việc khám phá các nhánh mới của cây (ít được khám phá) và khai thác các nhánh có giá trị cao (đã được khám phá nhiều).
  + Lặp lại bước này cho đến khi đến được nút lá (nút không có con).
  + Trong dự án của nhóm chúng em, hàm UCT được tính theo công thức như sau:

: Điểm của nút hiện tại

: Số lần nút cha được mô phỏng

: Số lần nút hiện tại được mô phỏng

* + **Mở rộng (Expansion):**
  + Khi nó không thể áp dụng UCT để tìm nút con tiếp theo, MCTS sẽ mở rộng cây trò chơi bằng cách thêm mọi trạng thái có thể đạt được từ nút hiện tại vào cây trò chơi
  + **Mô phỏng (Simulation):**
  + Sau khi mở rộng, MCTS chọn một nút lá ngẫu nhiên và chơi một ván cờ ngẫu nhiên từ nút lá cho đến khi kết thúc.
  + Gán giá trị cho nút lá dựa trên kết quả ván cờ mô phỏng (ví dụ: +1 cho thắng, -1 cho thua).
  + Trong dự án của nhóm chúng em, giá trị của nút lá sẽ được tính như sau: nếu thắng

nếu thua

empty squares là số ô còn trống còn lại trên bàn cờ để thuật toán có thể ưu tiên những nước đi mang lại chiến thắng sớm nhất

* + **Cập nhật (Backpropagation):**
  + Truyền ngược giá trị từ nút lá lên đến nút gốc, cập nhật giá trị trung bình của mỗi nút trên đường đi.
  + Điều này giúp các nút "học hỏi" từ kết quả của các ván cờ mô phỏng.

**Lặp lại các bước 2, 3 và 4 cho đến khi hết thời gian hoặc tìm được nước đi tối ưu.**

# **Kết quả thử nghiệm**

## 1.Thuật toán minimax

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 97 | 1 | 2 | 167.89 | 919 |
| Minimax (depth = 3) | 0 | 0 | 100 | 119.79 | 936 |
| Minimax (depth = 5) | 0 | 0 | 100 | 92.00 | 936 |
| Alpha Beta (depth = 3) | 0 | 0 | 100 | 100.93 | 936 |
| Alpha Beta (depth = 5) | 0 | 0 | 100 | 116.84 | 936 |
| MCTS  (time = 0.25s) | 56 | 16 | 28 | 115.74 | 931 |
| MCTS  (time = 0.5s) | 42 | 15 | 43 | 114.32 | 931 |

Bảng 1: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán Minimax

(depth = 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 99 | 0 | 1 | 4039.37 | 21940 |
| Minimax (depth = 3) | 100 | 0 | 0 | 3204.46 | 21984 |
| Minimax (depth = 5) | 0 | 0 | 100 | 1868.34 | 21908 |
| Alpha Beta (depth = 3) | 100 | 0 | 0 | 4397.51 | 21984 |
| Alpha Beta (depth = 5) | 0 | 0 | 100 | 2141.81 | 21908 |
| MCTS  (time = 0.25s) | 69 | 0 | 31 | 2572.58 | 21953 |
| MCTS  (time = 0.5s) | 63 | 0 | 37 | 2585.60 | 21958 |

Bảng 2: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán Minimax

(depth = 5)

## 2. Thuật toán cắt tỉa alpha beta

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 97 | 0 | 3 | 49.89 | 332 |
| Minimax (depth = 3) | 0 | 0 | 100 | 29.49 | 387 |
| Minimax (depth = 5) | 0 | 0 | 100 | 26.45 | 324 |
| Alpha Beta (depth = 3) | 0 | 0 | 100 | 31.02 | 387 |
| Alpha Beta (depth = 5) | 0 | 0 | 100 | 23.86 | 324 |
| MCTS  (time = 0.25s) | 59 | 14 | 27 | 42.98 | 351 |
| MCTS  (time = 0.5s) | 52 | 14 | 34 | 69.94 | 352 |

Bảng 3: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán cắt tỉa

Alpha Beta (depth = 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 99 | 0 | 1 | 446.44 | 2958 |
| Minimax (depth = 3) | 100 | 0 | 0 | 430.61 | 3315 |
| Minimax (depth = 5) | 0 | 0 | 100 | 244.50 | 3393 |
| Alpha Beta (depth = 3) | 100 | 0 | 0 | 364.17 | 3315 |
| Alpha Beta (depth = 5) | 0 | 0 | 100 | 268.04 | 3393 |
| MCTS  (time = 0.25s) | 0 | 74 | 26 | 535.53 | 3101 |
| MCTS  (time = 0.5s) | 0 | 64 | 36 | 307.52 | 3153 |

Bảng 4: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán cắt tỉa

Alpha Beta (depth = 5)

## 3.Thuật toán Monte Carlo Tree Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 96 | 2 | 2 | 252.42 | 426 |
| Minimax (depth = 3) | 43 | 14 | 43 | 256.96 | 520 |
| Minimax (depth = 5) | 0 | 16 | 84 | 257.45 | 613 |
| Alpha Beta (depth = 3) | 33 | 15 | 52 | 257.84 | 674 |
| Alpha Beta (depth = 5) | 0 | 18 | 82 | 255.38 | 635 |
| MCTS  (time = 0.25s) | 64 | 7 | 29 | 254.32 | 684 |
| MCTS  (time = 0.5s) | 62 | 8 | 30 | 255.93 | 461 |

Bảng 5: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán Monte Carlo Tree Search (time = 0.25s)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đối thủ | Thắng | Thua | Hoà | Thời gian trung bình (ms) | Số nút được thăm trung bình |
| Nước đi ngẫu nhiên | 95 | 1 | 4 | 507.14 | 572 |
| Minimax (depth = 3) | 33 | 18 | 49 | 505.64 | 1125 |
| Minimax (depth = 5) | 0 | 12 | 88 | 505.78 | 1176 |
| Alpha Beta (depth = 3) | 37 | 14 | 49 | 505.63 | 1229 |
| Alpha Beta (depth = 5) | 0 | 12 | 88 | 504.12 | 1189 |
| MCTS  (time = 0.25s) | 51 | 14 | 25 | 511.51 | 632 |
| MCTS  (time = 0.5s) | 56 | 11 | 33 | 507.19 | 923 |

Bảng 6: Kết quả 100 ván cờ với người chơi đi trước là AI sử dụng thuật toán Monte Carlo Tree Search (time = 0.5s)

## 4. Kết luận

Từ kết quả thử nghiệm, nhóm chúng em đã rút ra được những kết luận về các thuật toán như sau:

* Thuật toán Minimax và cắt tỉa Alpha Beta
  + Do cây trò chơi của tictactoe có kích thước nhỏ với chiều sâu tối đa là 9, thuật toán Minimax và cắt tỉa Alpha Beta có thể tìm được nước đi tốt nhất với độ sâu đủ lớn. Độ sâu thiết lập càng lớn thì khả năng tìm được nước đi tốt nhất càng cao.
  + Ở các nước đi đầu tiên, AI sẽ mất khá nhiều thời gian để tìm được đi tốt nhất. Độ sâu thiết lập càng lớn thì thời gian tính toán càng lâu.
  + Do không có yếu tố ngẫu nhiên nên đối với các thuật toán Minimax và Alpha Beta, nếu người chơi còn lại đi những nước giống nhau trong các ván đấu khác nhau thì thuật toán cũng sẽ chọn những nước đi giống nhau trong các ván đấu đó.
* Thuật toán Monte Carlo Tree Search
  + Do thuật toán Monte Carlo Tree Search có yếu tố ngẫu nhiên khi mô phỏng 1 ván chơi và khi chọn nút con để mô phỏng nên sẽ có trường hợp thuật toán không tìm được nước đi tối ưu nhất.
  + Tuỳ thuộc vào hàm UCT và cách tính giá trị của playout cho các nút lá mà thuật toán có thể dẫn đến tình trạng “nạn đói”: một số nút sẽ hầu như không được thăm, một số nút khác lại được thuật ưu tiên và giá trị UCT của nút đó lại tiếp tục tăng nếu mô phỏng ngẫu nhiên trả về kết quả thắng, mặc dù nước đi đó là nước đi tồi.
  + Do cây trò chơi của game tictactoe có kích thước nhỏ nên thuật toán MCTS sẽ gặp khó khăn khi phải đối đầu với thuật toán tìm được nước đi tối ưu như minimax. Đối với những trò chơi có cây trò chơi lớn hơn và các thuật toán như minimax và alpha beta không thể tìm được nước đi tối ưu, khi đó MCTS sẽ có thể thể hiện được những điểm mạnh của mình