TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO

**AI CHƠI Ô ĂN QUAN**

Môn: Project I

GVHD:

Phạm Hồng Sơn 20173344

*Hà Nội, ngày 28 tháng 12 năm 2019*

Mục lục

[1 Lời nói đầu 2](#_Toc30161365)

[2 Bài toán đặt ra 3](#_Toc30161366)

[2.1 Bàn cờ 3](#_Toc30161367)

[2.2 Quân cờ 3](#_Toc30161368)

[2.3 Luật chơi 3](#_Toc30161369)

[3 Giải quyết bài toán 5](#_Toc30161370)

[3.1 Minimax 5](#_Toc30161371)

[3.2 Monte Carlo Tree Search (MCST) 7](#_Toc30161372)

[3.2.1 Backpropagation – Lan truyền ngược 10](#_Toc30161373)

[3.3 AlphaGo Zero 14](#_Toc30161374)

[3.3.1 Neural Network 14](#_Toc30161375)

[3.3.2 Monte Carlo Tree Search của AlphaGo Zero 14](#_Toc30161376)

[3.3.3 Học không giám sát thông qua Self-Play 16](#_Toc30161377)

[4 Kết quả 17](#_Toc30161378)

[5 Kết luận 18](#_Toc30161379)

# Lời nói đầu

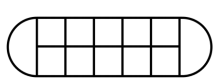
Trí tuệ nhân tạo **là trí tuệ của máy móc được tạo ra bởi con người.**Trí tuệ này có thể nhận biết, học hỏi,... như trí tuệ con người. Xử lý dữ liệu ở mức rộng lớn hơn, quy mô hơn, hệ thống, khoa học và nhanh hơn so với con người. Trong giai đoạn hiện nay, Trí tuệ nhân tạo đang rất được quan tâm và chú trọng phát triển. Trong đề tài lần này, em sẽ áp dụng Trí tuệ nhân tạo để xây dụng một AI chơi Ô ăn quan. Trong quá trình thực hiện đề tài không tránh khỏi những sai sót, em mong sẽ nhận được sự góp ý và đánh giá của thầy.

# Bài toán đặt ra

Ô ăn quan, hay còn gọi tắt là ăn quan hoặc ô quan là một trò chơi dân gian của trẻ em Việt. Đây là trò chơi có tính chất chiến thuật thường dành cho hai người chơi theo lượt (turn-based).

## Bàn cờ

Bàn chơi được kẻ thành một hình chữ nhật rồi chia hình chữ nhật đó thành mười ô vuông, mỗi bên có năm ô đối xứng nhau. Ở hai cạnh ngắn hơn của hình chữ nhật, kẻ hai ô hình bán nguyệt hoặc hình vòng cung hướng ra phía ngoài. Các ô hình vuông gọi là ô dân còn hai ô hình bán nguyệt hoặc vòng cung gọi là ô quan.



## Quân cờ

Quân chơi: trong thuật toán cái đặt sử dụng cùng quân chơi giá trị 1 mỗi quân.

Bố trí quân chơi: quan được đặt trong hai ô hình cánh cung, mỗi ô 10 quân, dân được bố trí vào các ô vuông với số quân đều nhau, mỗi ô 5 dân.

Luật chơi:

Mục tiêu cần đạt được để giành chiến thắng: người thắng cuộc trong trò chơi này là người mà khi cuộc chơi kết thúc có tổng số dân quy đổi nhiều hơn.

Di chuyển quân: Khi đến lượt, người chơi sẽ dùng tất cả số quân trong một ô có quân bất kỳ do người đó chọn trong số 5 ô vuông thuộc quyền kiểm soát của mình để lần lượt rải vào các ô, mỗi ô 1 quân, bắt đầu từ ô gần nhất và có thể rải ngược hay xuôi chiều kim đồng hồ tùy ý. Khi rải hết quân cuối cùng, tùy tình huống mà người chơi sẽ phải xử lý tiếp như sau: Nếu liền sau đó là một ô vuông có chứa quân thì tiếp tục dùng tất cả số quân đó để rải tiếp theo chiều đã chọn. Nếu liền sau đó là một ô trống (không phân biệt ô quan hay ô dân) rồi đến một ô có chứa quân thì người chơi sẽ được ăn tất cả số quân trong ô đó.

Số quân bị ăn sẽ được loại ra khỏi bàn chơi để người chơi tính điểm khi kết thúc. Nếu liền sau ô có quân đã bị ăn lại là một ô trống rồi đến một ô có quân nữa thì người chơi có quyền ăn tiếp cả quân ở ô này... Do đó trong cuộc chơi có thể có phương án rải quân làm cho người chơi ăn hết toàn bộ số quân trên bàn chơi chỉ trong một lượt đi của mình.

Nếu liền sau đó là ô quan có chứa quân hoặc 2 ô trống trở lên hoặc sau khi vừa ăn thì người chơi bị mất lượt và quyền đi tiếp thuộc về đối phương.

Trường hợp đến lượt đi nhưng cả năm ô vuông thuộc quyền kiểm soát của người chơi đều không có dân thì người đó sẽ phải dùng 5 dân đã ăn được của mình để đặt vào mỗi ô 1 dân để có thể thực hiện việc di chuyển quân. Nếu người chơi không đủ 5 dân thì phải vay của đối phương và trả lại khi tính điểm.

Cuộc chơi sẽ kết thúc khi toàn bộ dân và quan ở hai ô quan đã bị ăn hết. Trường hợp hai ô quan đã bị ăn hết nhưng vẫn còn dân thì quân trong những hình vuông phía bên nào coi như thuộc về người chơi bên ấy. Ô quan có ít dân (có số dân nhỏ hơn 5 phổ biến được coi là ít) gọi là quan non và để cuộc chơi không bị kết thúc sớm cho tăng phần thú vị, luật chơi có thể quy định không được ăn quan non, nếu rơi vào tình huống đó sẽ bị mất lượt.

# Giải quyết bài toán

## Minimax

Từ khi chọn đề tài này, thuật toán đầu tiên mà em có thể nghĩ đến là Minimax. Đây là một phương pháp trong lý thuyết quyết định có mục đích là tối thiểu hóa (minimize) tổn thất vốn được dự tính có thể là "tối đa" (maximize). Có thể hiểu ngược lại là, nó nhằm tối đa hóa lợi ích vốn được dự tính là tối thiểu (maximin).

Thuật toán này rất phù hợp với những trò chơi turn-based hai người có kết quả thắng, thua hoặc hoà. Khi đó, AI với một hàm định giá (hàm phỏng đoán điểm lợi thế - evaluation function) sẽ có nhiệm vụ tối đa hoá điểm lợi thế của bản thân theo hàm lượng giá (đồng nghĩa với tối thiểu hoá điểm lợi thế của đối thủ). Trong qua trình giả lập một số nước đi, AI sẽ đóng cả 2 vai trò là Max (mình) hoặc Min (đối thủ) trên cây trò chơi. Max sẽ tối đa hoá điểm lợi thế còn Min sẽ tối thiểu hoá điểm lợi thế.

Sau khi cài đặt thuật toán, AI đã hoạt động và đã có thể chơi nhưng chưa được “thông minh”. Em nhận ra để AI dựa trên minimax có thể “thông minh” thì cần tăng chiều sâu (số nước đi dự đoán) và xây dựng được hàm định giá tốt (evaluation function). Việc tăng chiều sâu (depth) rất khó khăn bởi số ván cờ giả lập sẽ tăng theo cấp số mũ với mỗi lần tăng chiều sâu. Để xây dựng hàm định giá tốt, người phát triển cần có kiến thức sâu về game đang xây dựng và rất khó đối với những trò chơi có luật phức tạp hơn. Vì vậy em đã tìm một thuật toán khác mà không cần đến hàm định giá.

## Monte Carlo Tree Search (MCST)

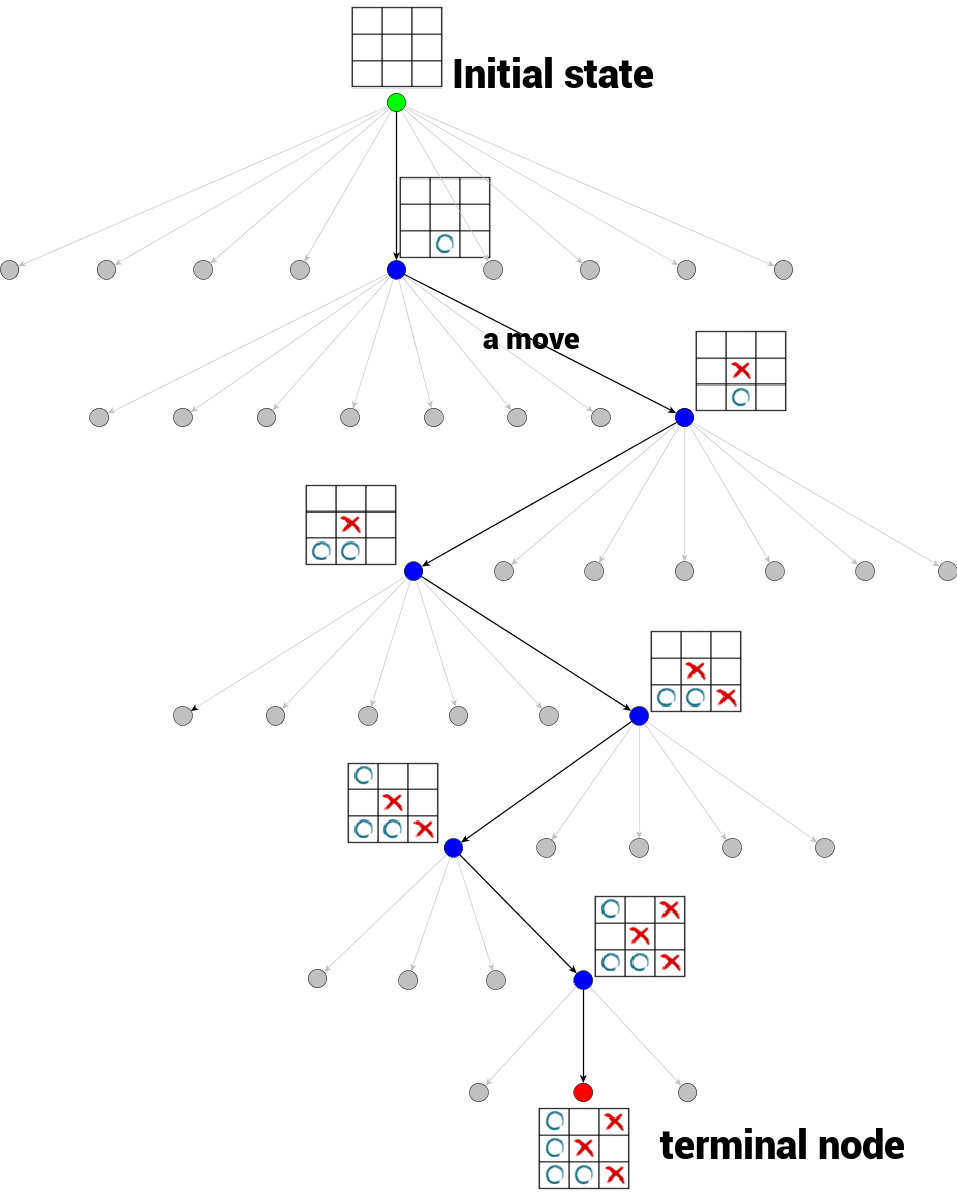
Đây là một thuật toán có tính ứng dụng cao trong nhiều lĩnh vực như AI, thống kê, lý thuyết trò chơi, lý thuyết kinh tế,…

Thuật toán Monte Carlo Tree Search (MCST) được giới thiệu bởi [Rémi Coulom](https://en.wikipedia.org/wiki/R%C3%A9mi_Coulom) vào năm 2016, lúc đó nó là thành phần chính của [Crazy Stone](https://en.wikipedia.org/wiki/Crazy_Stone_(software)) - một thuật toán chơi cờ vây khác. Mục đích chính của MCST khá đơn giản: Cho trước một trạng thái trong trò chơi, hãy dựa vào đó tìm nước đi tiếp theo để có khả năng thắng cao nhất. Lý thuyết phát triển thuật toán MCTS khá trừu tượng và phức tạp, nhưng đối với bài toán đặt ra thì ở đây sẽ sử dụng cho các trò chơi 2 người đơn giản.

Cây Trò Chơi – Game Tree: Giống như thuật toán Minimax, MCTS cũng sử dụng cây trò chơi. Một cây trò chơi là một cây mà trong đó mỗi node của nó biểu diễn một trạng thái của trò chơi. Chuyển dịch từ một node đến một node con (child) của nó (nếu có) được gọi là một nước đi. Số lượng node con của một node được gọi là số rẽ nhánh (Branching factor). Node rễ của cây biểu diễn trạng thái ban đầu của trò chơi. Terminal node - nốt cuối - là những nốt mà không có bất cứ node con nào. Những node cuối này là trạng thái của game mà ở đó không thể thực hiện một nước đi nào nữa. Kết quả của trò chơi được đánh giá dựa vào các node cuối này.

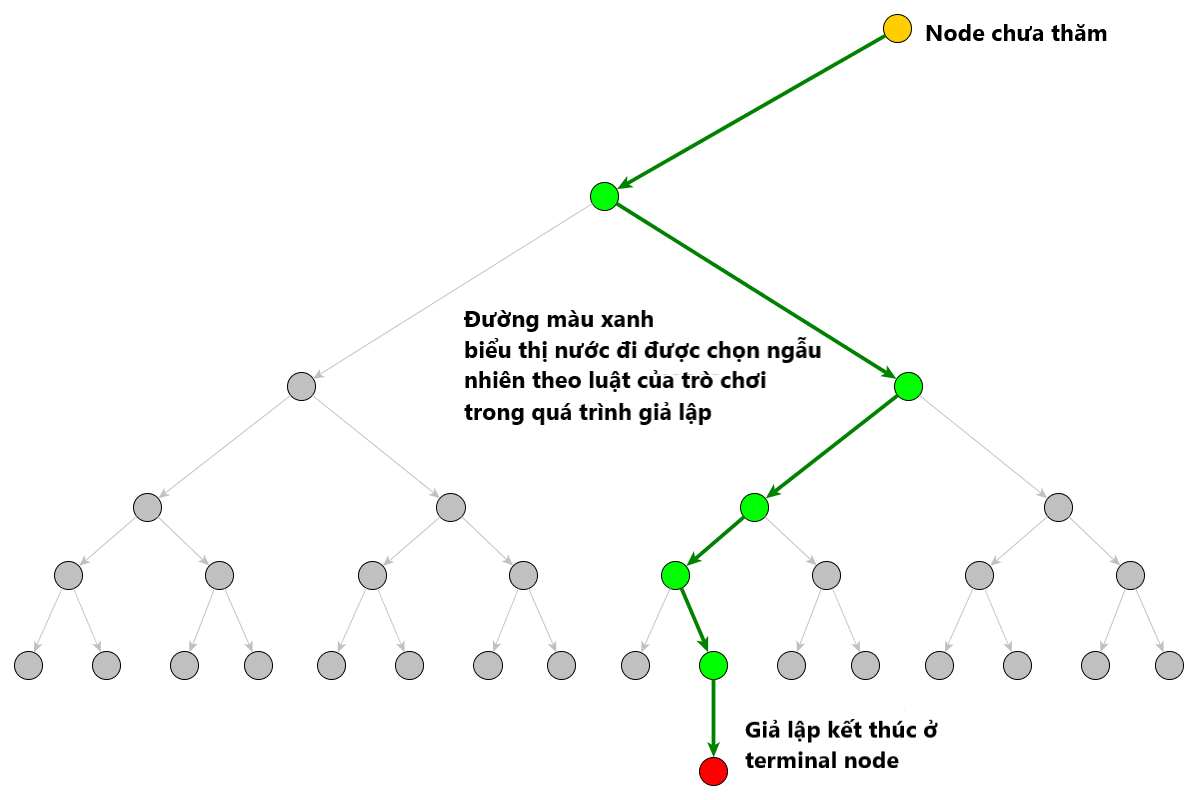
Hình dưới là hình ảnh một cây trò chơi cho game TicTacToe. Có thể thấy node rễ (Initial state) là trạng thái mà bàn cờ chưa có nước nào. Mỗi một node con của node rễ là bàn cờ đã được đi một nước. Số rẽ nhánh phụ thuộc vào độ sâu của cây (depth). Game kết thúc ở các node cuối.

Mỗi một quãng đường (traverse) đi từ node rễ đến node cuối là một ván của trò chơi. Cây trò chơi là một dạng cấu trúc dữ liệu đệ quy. Khi chọn được một nước đi tốt nhất rồi thì ta chuyển node rễ của mình xuống node con phía dưới của nó và lại bắt đầu từ đấy.



Đối với thuật toán Monte Carlo Tree Search, cách tính nước đi khả quan nhất được thực hiện khác so với **Minimax**. MCTS sẽ giả lập trò chơi nhiều lần rồi dự đoán nước đi khả quan nhất dựa trên những lần giả lập đó mà không cần hàm định giá nào.

Phần quan trọng nhất của MCTS chính là ***Search***. Search là một tập bao gồm các “quãng đường” trong khắp cây trò chơi từ trên xuống dưới. Một “quãng đường” là một cách di chuyển từ node rễ (trạng thái game hiện tại) đến một node mà chưa được mở rộng hoàn toàn (***not fully expanded***). Một node được gọi là chưa mở rộng hoàn toàn khi có ít nhất một node con của nó chưa được khám phá (***visited***). Khi một ***not fully expanded*** *node* được tìm thấy, node con ***unvisited*** của nó sẽ được chọn để làm node rễ cho một lần chơi giả lập. Kết quả của lần giả lập đó sẽ được truyền ngược lại về node rễ thật (trạng thái hiện tại). Khi quá trình search kết thúc, nước đi tiếp theo sẽ được chọn dựa trên kết quả trả về của nhiều lần giả lập trước đó.



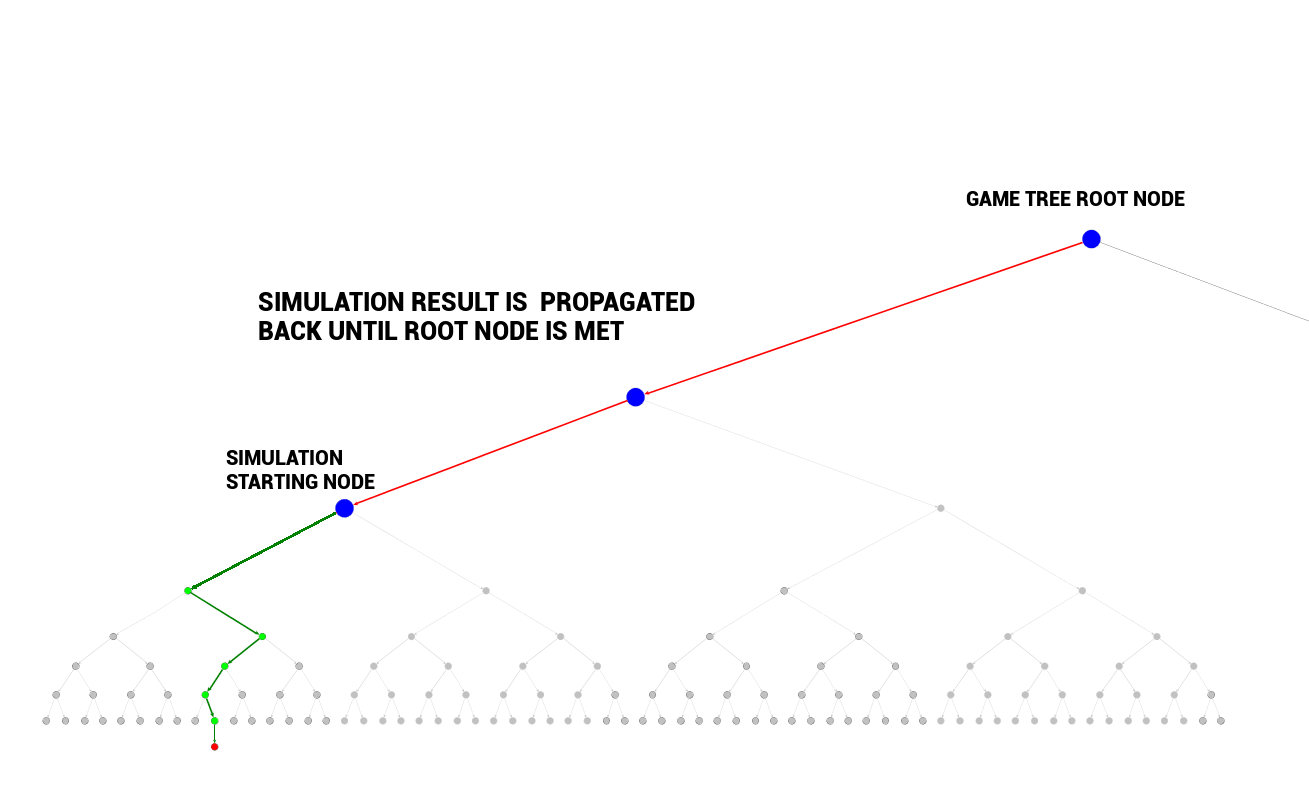
Giả lập ở đây có nghĩa là một ván của trò chơi, gồm một loạt các nước đi liên tiếp theo lượt, bắt đầu từ node hiện tại (trạng thái của trò chơi) và kết thúc ở một node cuối nơi mà kết quả của trò chơi được đánh giá.

Các nước đi được thực hiện bởi một hàm số tên là ***Rollout Policy***. Đầu vào của nó là một trạng thái của trò chơi còn đầu ra là nước đi tiếp theo cho trạng thái đó. Dĩ nhiên, các nước đi bắt buộc phải tuân theo luật của trò chơi, nhưng cách mà chúng được chọn thì rất ngẫu nhiên. Hàm ngẫu nhiên này thường là uniform random - phân phối đều.

Ở dạng đơn giản nhất, giả lập là tập các nước đi ngẫu nhiên liên tiếp bắt đầu từ một node rễ và kết thúc ở một nốt cuối. Giả lập luôn kết thúc với một kết quả đánh giá, ví dụ là thắng, thua, hòa hoặc một chỉ số nào đó.

Khi một giả lập bắt đầu ở một node nào đó, node đó được gọi là ***visited***. Các node được chọn bởi ***Rollout Policy*** thì không được tính là ***visited***.

### **Backpropagation – Lan truyền ngược**

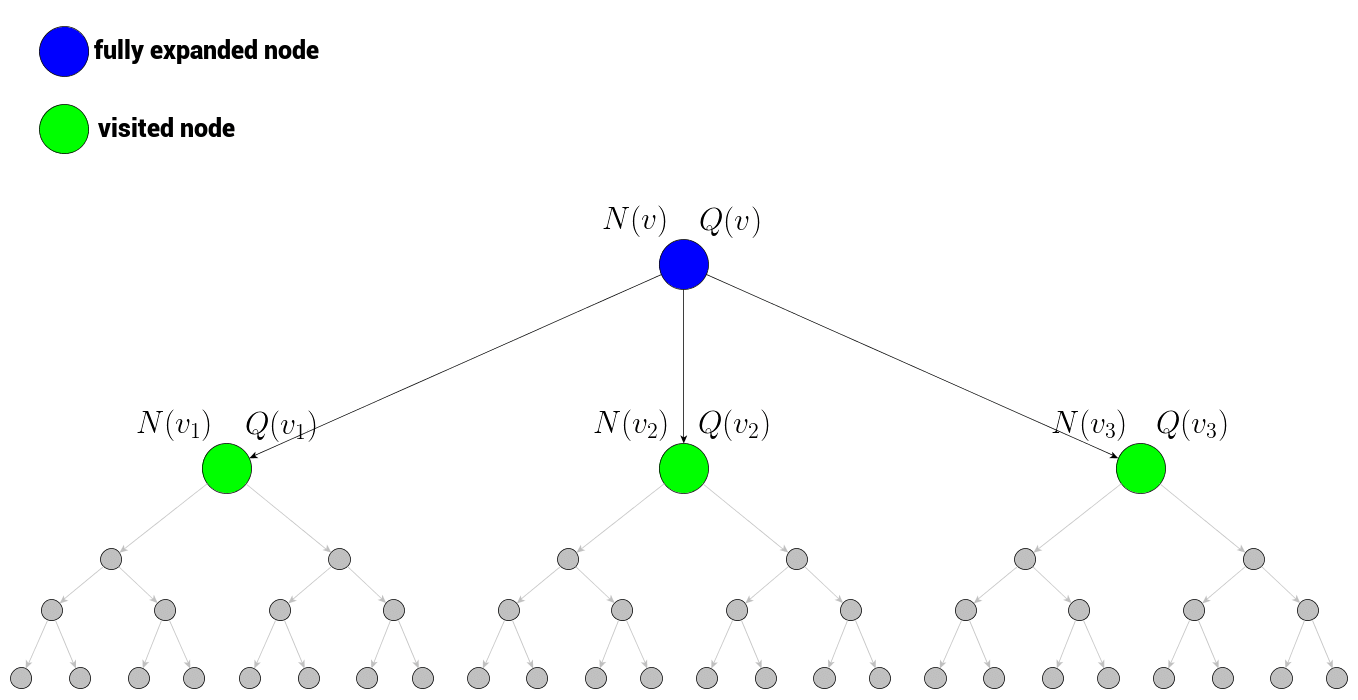


Sau khi mỗi giả lập kết thúc, kết quả được gửi trở về node rễ ban đầu (node trạng thái hiện tại của trò chơi, nó là node cha của node rễ của giả lập). Các kết quả được thu về để thực hiện tính toán. Cụ thể có 2 giá trị sau:

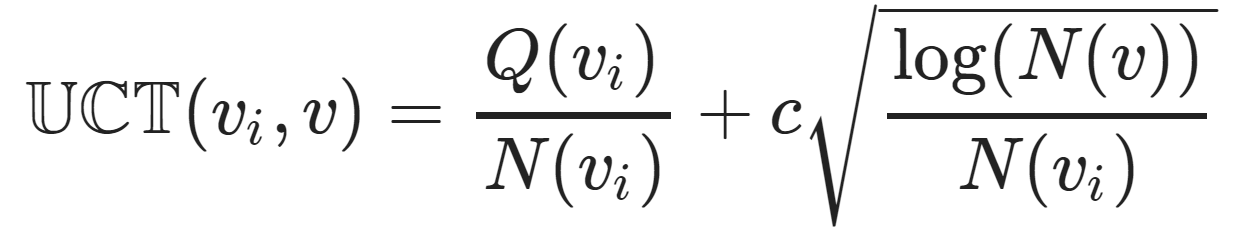
* Q():**Chỉ số giả lập khả quan (Total Simulation Reward**), là một thuộc tính của của node , là tổng của tất cả các kết quả của những lần giả lập mà **đi qua** node đó, không nhất thiết phải là node rễ của giả lập.
* N(): Chỉ số visit, là số lần mà các con đường ***backpropagation*** đi qua node đó (tức là nó đóng góp bao nhiêu lần cho cho chỉ số giả lập khả quan).

Mỗi node đều có 2 giá trị này, nếu nhìn vào một node, có thể biết được mức độ chiến thắng nếu đi qua node đó. (***Q()*** cao hoặc ***N()*** thấp).

Khi một quá trình search bắt đầu, do chưa có bất kì giả lập nào được thực hiện nên ***unvisited*** node sẽ được bắt đầu đầu tiên.



Nhìn vào hình trên, ta thấy node màu xanh dương là node rễ, đã mở rộng hoàn toàn (***fully expanded***) bởi các node con của nó đều đã được đi qua (***visited***) - tức là node rễ của 1 giả lập. Để có thể chọn một node trong số các node con này làm nước di chuyển tiếp theo thì dùng ***UCT (Upper Confidence bound applied for Tree)***, một đại lượng quan trọng để quyết định node nào sẽ được sử dụng để chọn nước đi tiếp theo. Đây chính làm hàm số quan trọng, cốt lõi của **MCTS**. Nó có công thức như sau:

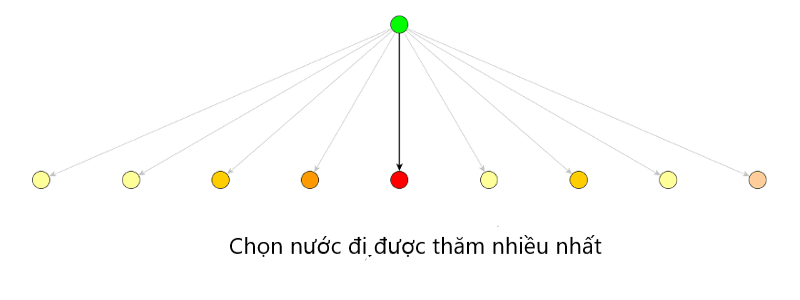


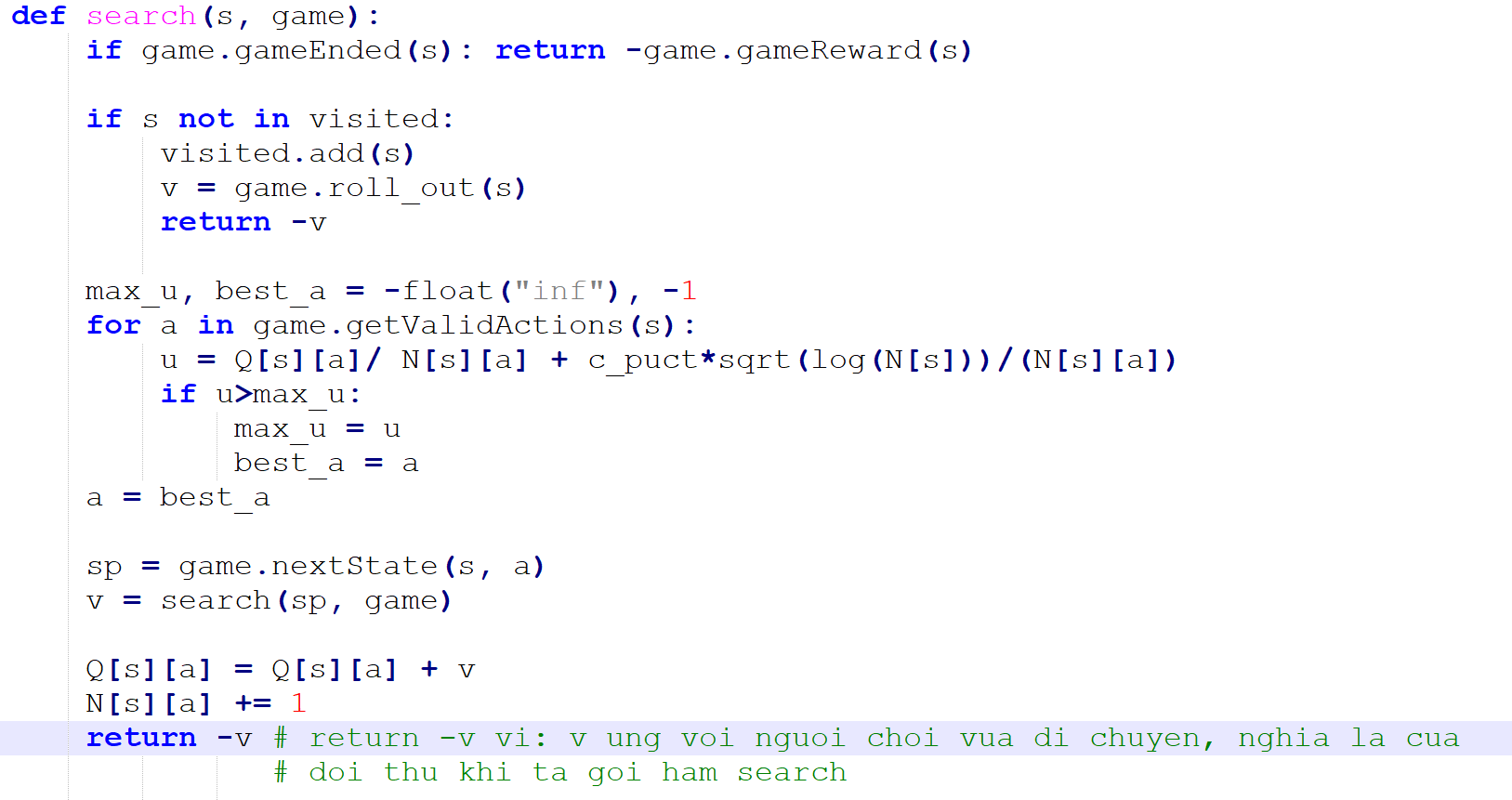
Node được chọn để làm nước đi tiếp theo cho MCTS chính là node  có chỉ số ***UCT*** cao nhất. Hàm số này gồm có 2 thành phần. Thành phần thứ nhất,  được gọi là ***exploitation component (thành phần khai thác)*** có thể được hiểu như là tỉ lệ ***thắng/thua***. Tử số càng cao và mẫu số càng bé thì chỉ cố này càng lớn. Chỉ số này khá quan trọng và hiển nhiên để quyết định một node có được chọn hay không, nhưng sẽ không chỉ dùng mỗi nó. Bởi vì rất có thể sẽ rơi vào trường hợp của thuật toán tham lam, tức là quá thiên vị những kết quả tốt nhất đầu tiên. Các nước đi ngẫu nhiên trong mỗi giả lập được thực hiện bởi hàm số ***Rollout Policy***, và hàm số này là một hàm số ngẫu nhiên, nên những kết quả tốt nhất đầu tiên chưa chắc (dù có khả năng cao hơn chút) đã là những nước đi tốt nhất. Cụ thể hơn, ngay trong lần giả lập đầu tiên, node con nào là nước đi để dành chiến thắng sẽ được thiên vị, những node con khác kém may mắn hơn sẽ bị loại ngay mà không có cơ hội cải thiện chỉ số ở những lần giả lập tiếp theo.

Đó là lý do tại sao có thêm một đại lượng phía sau, gọi là ***exploration component*** *(thành phần tiềm năng):* . Nhìn vào công thức và quan sát, đại lượng tiềm năng “ưa” những node mà có số lần được giả lập đi qua ít hơn.

Đại lượng ***c***có trong phần ***exploration*** là một tham số dùng để cân bằng 2 đại lượng ***exploitation*** và ***exploration***.

Thực tế cho thấy, khi kết thúc ***MCTS***, những node được chọn thường là những node có chỉ số ***exploration*** cao nhất. Điều này cũng dễ hiểu bởi nếu nó quan trọng và có khả năng thắng cao, nó sẽ thường được đi qua nhiều hơn cả. Sau khi đi một nước, tức chuyển node rễ ban đầu xuống một node con của chính nó, ta lại bắt đầu một trạng thái trò chơi mới, và thay vì chạy ***MCTS*** lại từ đầu, ta có thể sử dụng các chỉ số đã chạy từ lần trước đó để quyết định nước đi.





Như ta thấy, trong hàm *search()* có gọi hàm *roll\_out()* để giả lập đến cuối ván cờ. Việc *rollout* này tốn khá nhiều thời gian khi numSims (số lần giả lập – giả lập càng nhiều cây càng thông minh do học được nhiều ván cờ hơn) lớn.

Cây MCST cũng dễ bị đi vào lối mòn khi quá ưu tiên những node có *Q(vi)* cao từ sớm, cần rất nhiều vòng lặp để có thể khai thác những node mới – điều này cũng rất mạo hiểm khi các node ít được thăm chưa chắc đã là node tốt do không có tri thức về trò chơi dẫn đến mất thời gian và tài nguyên.

Do đó để cải tiến cây MCST, em đã tìm thấy AlphaGo Zero của Google DeepMind. Ở AlphaGo Zero, chúng ta dùng một Neural network (mạng nơron nhân tạo) để cắt giảm bước roll\_out bằng cách dự đoán *reward* – *v* của nước đi *s*. Đồng thời, mạng này cũng đóng góp tri thức cho hàm *UCT* để chọn nước đi tốt hơn.

## AlphaGo Zero

### **Neural Network**

Mạng nơron nhân tạo, Artificial Neural Network (ANN) gọi tắt là neural network (mạng nơron), là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ neural sinh học. Nó được tạo lên từ một số lượng lớn các phần tử (gọi là perceptron hay neural) kết nối với nhau thông qua các liên kết (gọi là trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể nào đó thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học chính là quá trình hiệu chỉnh trọng số liên kết giữa các neural.

Và ở trong mô hình AlphaGo Zero sẽ sử dụng mô hình Convolutional Neural Network (CNN) với:

**Input**: Trạng thái bàn cờ *s*.

**Output**: + : Xác suất chọn các nước đi.

+ *v:* Điểm lợi thế của người chơi trong trạng thái bàn cờ: một số thực trong khoảng (-1;1). *v* càng gần 1 thì lợi thế càng lớn (Trong cây MCST gốc, *v* chính là kết quả trận đấu (1-win, 0-draw, -1-lost) trả về khi roll-out).

Khi *train* (huấn luyện), Neural Network sẽ được cung cấp *examples* với ba thông tin:

+ *st*: Trạng thái bàn cờ (input)

+ : Từ cây MCST sẽ được trình bày ở sau.

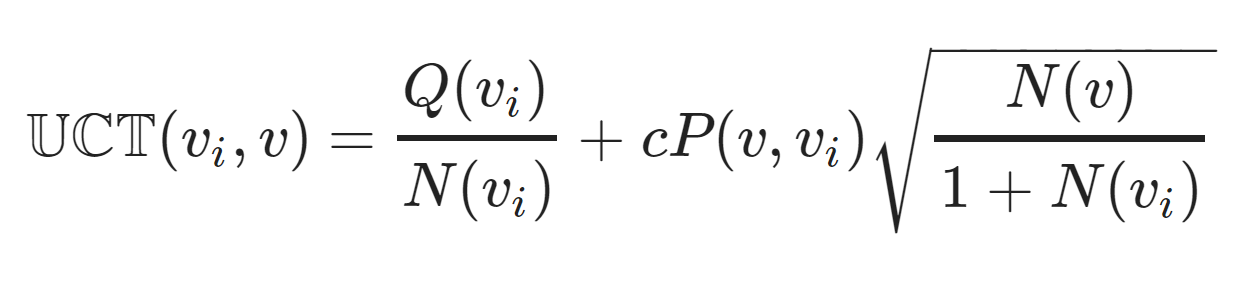
+ *zt*: Kết quả ván đấu thu được sau khi kết thúc 1 ván self-play

Sau đó mạng sẽ được điều chỉnh để cực tiểu hoá *loss function* dưới đây:

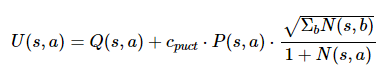
### **Monte Carlo Tree Search của AlphaGo Zero**

Trong bài toán này, chúng ta sử dụng CNN để cải tiến MCTS. Thay vì thực hiện *rollout* mỗi khi gặp một node mới trên MCTS thì với mỗi trạng thái đầu vào s của bàn cờ sẽ đi qua mạng CNN và cho ra *policy* (*s*) và điểm lợi thế *v*, *v* được lan truyền ngược lại đến node root.

Hàm *UCT* mới sẽ được tính lại theo công thức dưới đây:



*Tương tự với:*



+ *Q(s,a)*: phần thưởng cho nước đi a của trạng thái *s* (đã chia cho N).

+ *N(s,a)*: số lần nước đi *a* được *Backpropagation* qua.

+ P(*s*,⋅) = (*s*): dự đoán nước đi từ trạng thái *s* – Output của neural network.

Công thức update *Q(s,a):*

Một vòng giả lập sẽ được thực hiện như sau: Bắt đầu từ trạng thái *s*, ta chọn (hàm *search*) được nước đi *a* tốt nhất theo *policy* hiện tại theo công thức *U(s,a)*. Nếu trạng thái *s’* (thu được khi thực hiện *a* tại *s*) đã tồn tại trên cây MCTS, ta đệ quy *search* trạng thái *s’.* Nếu *s’* chưa có trên cấy MCTS, ta thêm trạng thái này vào cây, tính toán các giá trị (*s*), *v(s’)* bằng neural network hiện tại và khởi tạo các giá trị *Q(s’,a)* và *N(s’,a)* bằng 0 với mọi *a*. Sau đó lan truyền ngược lại *v(s’)* và update tất cả *Q(s,a)*. Nếu *s’* là node terminal, *v* sẽ nhận giá trị 1 (thắng) hoặc -1 (thua) hoặc 0 (hòa).

Sau một số vòng giả lập, giá trị *N(s,a)* ở node gốc được cải thiện. Chính sách cải thiện ngẫu nhiên (improved stochastic policy ) dùng là train neural network là .

### **Học không giám sát thông qua Self-Play**

Ban đầu chúng ta khởi tạo neural network với trọng số ngẫu nhiên. Với mỗi vòng lặp, một số lượng self-play game sẽ được chơi. Trong mỗi lượt của self-play game, một số lượng giả lập MCST sẽ được thực hiện từ trạng thái . Chúng ta tính *policy* đã được cải thiện và thu được *example (mẫu)* ( Kết quả ván đấu sẽ được thêm vào *example* cuối cùng khi ván game kết thúc (+1, -1, 0). Cây MCTS được duy trì suốt game đấu. Sau mỗi vòng lặp, neural network sẽ được *train* theo mẫu với *example* thu được. Mạng cũ và mạng vừa được *train* sẽ đấu với nhau một số vòng, nếu tỉ lệ thắng của mạng mới lớn hơn một ngưỡng nào đó cho trước (0.6 với bản demo), mạng mới sẽ được chấp nhận thay thế cho mạng cũ.

# Kết quả

Sau 1 tuần training thì AI đã có thể thắng 100% khi thi đấu với các máy chọn nước đi ngẫu nhiên (thử nghiệm với 100 ván), thắng áp đảo em và một số bạn với thời gian suy nghĩ mỗi nước đi trung bình 4s và thắng AI của một số game Ô ăn quan trên mạng.

AI được training trên Google Colaboratory.

# Kết luận

Thông qua việc tìm hiểu và nghiên cứu đề tài này giúp em có cái nhìn toàn diện hơn trong việc ứng dụng trí tuệ nhân tạo vào giải quyết các bài toán. Hiểu rõ hơn về Neural network và học thêm được về Monte Carlo Tree Search. Do thời gian có hạn nên đề tài không tránh khỏi những sai sót, mong thầy góp ý, đánh giá giúp em hoàn thiện đề tài.

Em xin cảm ơn!

**Tài liệu tham khảo**

1. Monte Carlo Tree Search – beginners guide của Kamil Czarnogórski

<https://int8.io/monte-carlo-tree-search-beginners-guide/>

2. A Simple Alpha(Go) Zero Tutorial của Surag Nair

<https://web.stanford.edu/~surag/posts/alphazero.html>/

3. Ô ăn quan – Wikipedia

<https://vi.wikipedia.org/wiki/%C3%94_%C4%83n_quan>/