BỘ CÔNG AN

HỌC VIỆN AN NINH NHÂN DÂN

BÀI TIỂU LUẬN

"SỬ DỤNG CHIẾN LƯỢC MINIMAX VÀ PHƯƠNG PHÁP CẮT TỈA ALPHA BETA TRONG VIỆC XÂY DỰNG TRÒ CHƠI NIM"

Họ và tên: HUỲNH NGỌC KHANH

Trung đội: B15D52

SBD: 13

Học phần: NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Khoa giảng dạy: NV7

Hà Nội – 2023

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

STT	Từ viết tắt	Từ gốc	Tiếng Việt
1	AI	Artificial Intelligence	Trí tuệ nhân tạo
2	ABP	Alpha-beta pruning	Kỹ thuật cắt tỉa Alpha-beta
3	IDS	Iterative deepening	Tìm kiếm sâu dần
4	UCT	Upper Confidence Bounds for Tree	Thuật toán tìm kiếm UCT
5	MCTS	Monte Carlo tree search	Cây tìm kiếm Monte Carlo
6	ML	Machine Learning	Học máy
7	RL	Reinforcement Learning	Mô hình học tăng cường
8	NN	Neural Network	Mạng nơ ron
9	MTD	Minimax with Transposition Tables	Bảng chuyển vị trạng thái với Minimax

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

MỞ ĐẦU

1. Giới thiệu đề tài

Lý thuyết trò chơi bắt nguồn từ toán học. Lý thuyết đó vô cùng phong phú đa dạng và được vô số học giả thuộc mọi tầng lớp xã hội áp dụng ở các lĩnh vực khác nhau. Lý thuyết trò chơi (game theory) thường được coi là một nhánh của toán học ứng dụng và kinh tế học ứng dụng nhằm nghiên cứu về các tình huống trong đó các bên tham gia trò chơi áp dụng những chiến lược ra quyết định nhằm tối ưu hóa kết quả mình nhận được.

Trí tuệ nhân tạo đã vận dụng lý thuyết trò chơi để nghiên cứu về các trò chơi đối kháng và thiết kế chương trình chơi cờ giữa Người và Máy tính. Do bùng nổ tổ hợp quá lớn của cây trò chơi mà cả người và máy không thể (và không bao giờ) có thể tìm kiếm vét cạn (hết mọi khả năng). Do đó phương pháp tìm kiếm duy nhất là chỉ tìm kiếm đến một độ sâu giới hạn nào đó và chọn nước đi dẫn đến một thế cờ có lợi nhất cho mình. Do phải tính cả khả năng chống trả của đối phương nên ta không dùng được các thuật toán tìm kiếm thông thường mà phải dùng một thuật toán tìm kiếm riêng cho cây trò chơi. Có nhiều chiến lược tìm kiểu được sử dụng phổ biến như chiến thuật Minimax, chiến thuật Monte Carlo, sử dụng học tăng cường, ... Các chiến lược đều có những ưu, nhược điểm riêng, hiệu quả áp dụng trong các trò chơi khác nhau. Chúng ta đều biết, nhiều tình huống trong thực tế đặc biệt là trong lĩnh vực Kinh tế, Chính trị có thể nhìn nhận như một trò chơi có tổng bằng không. Do đó việc nghiên cứu chiến lược tìm kiếm nước đi cho dạng trò chơi này có thể mang lại những ý nghĩa thực tiễn nhất định.

2. Phạm vi nghiên cứu

Phạm vi nghiên cứu của bài tiểu luận tập trung chủ yếu vào lý thuyết giải thuật tìm kiếm Minimax trong lý thuyết trò chơi, đặc biệt là các trò chơi cổ điển có tổng bằng 0. Dựa trên các kiến thức này, bài tiểu luận thực hiện xây dựng một chương trình mô phỏng trò chơi giữa người và máy tính để xem xét đánh giá giải thuật trên trong trò chơi cổ điển NIM. Ngoài ra, bài tiểu luận cũng đề cập đến các phương pháp cải tiến, tối ưu hóa giải thuật trong bài toán trên.

3. Mục tiêu đề tài

Mục tiêu của bài tiêu luận dựa trên các yêu cầu sau:

- Tìm hiểu, nghiên cứu chiến lược tìm kiếm Minimax và phương pháp cắt tỉa Alpha-Beta.

+ Tổng quan về thuật toán

+ Các ưu điểm nhược điểm, thuận lợi khó khăn của thuật toán

+ Các ứng dụng của thuật toán Minimax trong công nghệ hiện nay

+ Đánh giá thuật toán

- Áp dụng giải thuật trong bài toán NIM

+ Giới thiệu bài toán

+ Áp dụng thuật toán để giải quyết bài toán

+ Đề xuất giải pháp tối ưu hơn để giải quyết bài toán

+ Demo

4. Nội dung đề tài

Nội dung của tiểu luận là tìm hiểu và nghiên cứu về thuật toán tìm kiếm đối kháng Minimax, phương pháp cắt tỉa Alpha-Beta, các cải tiến của nó và ứng dụng trong trò chơi có tổng bằng không. Nội dung được tìm hiểu nghiên cứu dựa trên các giáo trình trí tuệ nhân tạo hiện hành trên các trường đại học [1]–[6]. Nội dung được chia làm 2 phần chính với 5 chương tương ứng:

Phần 1 trình bày lý thuyết về trò chơi và các thuật toán được chọn để tìm hiểu, nghiên cứu, bao gồm 03 chương với nội dung sau:

+ Chương 1 trình bày một vấn đề cơ bản, tổng quan về các khái niệm: bài toán tìm kiếm, biểu diễn vấn đề bằng không gian trạng thái và các kỹ thuật tìm kiếm cơ bản.

+ Chương 2 trình bày giải thuật tìm kiếm Minimax, cách xây dựng giải thuật, thủ tục thực hiện giải thuật và đánh giá giải thuật.

+ Chương 3 trình bày về phương pháp cắt tỉa Alpha-Beta, cách xây dựng giải thuật, đánh giá và so sánh với giải thuật Minimax truyền thống.

Phần 2 trình bày về việc xây dựng chương trình trò chơi bằng cách sử dụng chiến thuật Minimax và phương pháp cắt tỉa Alpha-Beta, bao gồm 02 chương:

+ Chương 4 giới thiệu bài toán NIM, các hình thức trò chơi cũng như hướng giải quyết bài toán NIM sử dụng chiến lược Minimax với phương pháp cắt tỉa Alpha-Beta.

+ Chương 5 giới thiệu chương trình đơn giản mô tả hướng giải quyết bài toán bằng cach xây dựng trò chơi giữa người và máy. Các phương pháp cải tiến cũng được đưa ra trong chương này.

PHẦN 1: CHIẾN LƯỢC MINIMAX VÀ PHƯƠNG PHÁP CẮT TỈA ALPHA-BETA

CHƯƠNG 1. MỘT SỐ VẤN ĐỀ LIÊN QUAN

1.1. Bài toán tìm kiếm

Trong lý thuyết tính toán, một bài toán tìm kiếm là một loại bài toán tính toán được biểu diễn bởi các mối quan hệ nhị phân. Nếu R là một quan hệ nhị phân sao cho $field(R) \subseteq T^+$ và T là một máy tìm kiếm Turning, thì T tính f nếu:

- Nếu mỗi x có một số giá trị y mà $R(x, y)$ thì T truy nhập vào với đầu ra z mà $R(x, z)$ (có thể có nhiều y, và T chỉ cần một trong số chúng)

- Nếu với giá trị x mà không có giá trị y thỏa mãn $R(x, y)$ thì T loại bỏ x.

Một quan hệ R có thể được biểu diễn như một bài toán tìm kiếm, và một máy Turning tính R còn được gọi để giải quyết nó. Mọi bài toán tìm kiếm đều tương ứng với một bài toán quyết định, cụ thể:

$$L(R) = \{x | yR(x, y)\}$$

Có thể hiểu đơn giản như sau: "Cho một dãy gồm n đối tượng $a_1, a_2, \dots, a_n$. Mỗi đối tượng a_i có một khóa key $(1 \leq i \leq n)$ gọi là khóa tìm kiếm. Cần tìm kiếm đối tượng có khóa bằng một tham số k cho trước, tức là $a_i.key = k$". Quá trình tìm kiếm sẽ hoàn thành nếu như có một trong hai trường hợp sau đây xảy ra:

+ Tìm được đối tượng có khóa tương ứng bằng k, khi đó phép tìm kiếm thành công.

+ Không tìm được đối tượng nào có khóa tương ứng bằng k, khi đó phép tìm kiếm thất bại.

1.2. Không gian trạng thái

Không gian trạng thái bao gồm tất cả các trạng thái của các đối tượng có thể xảy ra mà bài toán cần có để tìm ra được đối tượng được yêu cầu phù hợp với bài toán. Không gian trạng thái có thể được biểu diễn bằng đồ thị có hướng hoặc vô hướng tùy thuộc vào muc tiêu và hướng giải quyết của bài toán.

Để có thể biểu diễn được không gian trạng thái một cách tốt nhất, ta cần chú ý đến các yếu tố khi xây dựng không gian trạng thái như: trạng thái ban đầu, tập hợp các phép chuyển trạng thái, tập hợp các trạng tháu có thể đạt đến từ trạng thái ban đầu và tập hợp các trạng thái kết thúc.

1.3. Kỹ thuật tìm kiếm đối kháng

Kỹ thuật tìm kiếm đối kháng còn gọi là tìm kiếm có đối thủ là chiến lược tìm kiếm được áp dụng để tìm ra nước đi cho người chơi trong các trò chơi đối kháng (2 người với nhau). Điển hình cho các trò chơi này là cờ vua, cờ tướng, … trong các trò chơi này, có một cây trò chơi bao gồm tất cả các nước đi có thể của cả hai đấu thủ và các cấu hình bàn cờ là kết quả của các nước đi đó. Ta có thể tìm kiếm trên cây này để có được một chiến lược chơi hiệu quả.

Dạng bài toán này có đặc trưng duy nhất là ta phải tính đến mọi nước đi mà đối thủ của ta có thể sử dụng. Để làm điều này, các chương trình máy tính, cũng như các dạng khác của trí tuệ nhân tạo như lập kế hoạch tự động (machine planning) thường sử dụng các thuật toán tìm kiếm như thuật toán minimax, tỉa cây tìm kiếm, và tỉa cây alpha-beta (alpha-beta pruning) để giải quyết vấn đề trên.

1.4. Cây trò chơi

Cây trò chơi (Game tree) là một sơ đồ hình cây thể hiện từng trạng thái, từng trường hợp của trò chơi theo từng nước đi. Cây trò chơi dùng các node để thể hiện trạng thái của trò chơi. Cây này là một dạng của cây ngữ nghĩa, có các nhánh ứng với việc chuyển cấu hình sau một nước đi. Có thể xem hai nhánh xuất phát từ một node là hai quyết định của hai đấu thủ [5]. Gọi p là số mức của cây thì độ sâu của cây là $d = p - 1$. Mỗi lựa chọn hay bước chuyển là một nước đi.

1.5. Trò chơi có tổng bằng 0

Trò chơi có tổng bằng không là trò chơi có tổng giá trị kết quả (mà người thắng được hưởng) là cố định. Bất cứ bên nào thắng (+1) cũng làm cho bên kia thua cuộc (-1), tương ứng với tình huống ganh đua thuần tuý, cuối cùng dẫn tới tổng (+1- 1) = 0.

Cờ vua là một trò chơi có tổng bằng không bởi không thể có trường hợp cả hai bên đều thắng hoặc đều thua. Nếu một bên thắng thì bên kia nhất định là thua và ngược lại [1]. Thể thao là những ví dụ điển hình nhất của trò chơi có tổng bằng không. Nhà vô địch chỉ có thể đạt được vinh quang khi toàn bộ các đối thủ khác đều thua cuộc. Trong một giải bóng đá tổng số trận thắng luôn bằng tổng số trận thua cũng là bởi cái tính chất tổng bằng không ấy [3].

CHƯƠNG 2. THUẬT TOÁN MINMAX

2.1. Tổng quan về thuật toán Minimax

2.1.1. Khái niệm

Thuật toán Minimax là một thuật toán tìm kiếm đệ quy lựa chọn bước đi kế tiếp, bằng cách định giá trị cho các bước trên cây trò chơi, sau đó tìm ra bước đi kế tiếp có giá trị phù hợp. Thuật toán thường được sử dụng phổ biến trong các trò chơi đối kháng, trong đó hai người thay phiên đi nước đi của mình như: cờ vua, tic-tac-toe, cờ vây, … [7]

Minimax (hay còn gọi là minmax) 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 cũng được mở rộng cho nhiều trò chơi phức tạp hơn và giúp đưa ra các quyết định chung khi có sự hiện diện của sự không chắc chắn [8].

Trò chơi có một trạng thái bắt đầu và mỗi nước đi sẽ biến đổi trạng thái hiện tại thành một trạng thái mới. Trò chơi sẽ kết thúc theo một quy định nào đó, theo đó thì cuộc chơi sẽ dẫn đến một trạng thái phản ánh có một người thắng cuộc hoặc một trạng thái mà cả hai đấu thủ không thể phát triển được nước đi của mình, được gọi là trạng thái hòa cờ. Do đó, để có thể thắng trò chơi, ta phải tính toán, phân tích xem từ một trạng thái nào đó sẽ dẫn đến phản ứng và nước đi của đối thủ như thế nào với điều kiện cả hai đều có trình độ như nhau, tức là, ta và đối thủ cùng sử dụng một không gian trạng thái như nhau.

Giải thuật Minimax sẽ giúp tìm ra nước đi tốt nhất, bằng cách đi ngược từ cuối trò chơi trở về đầu. Tại mỗi bước, nó sẽ ước định rằng người A đang cố gắng tối đa hóa cơ hội thắng của A khi đến lượt của mình (tìm kiếm bước đi có tỷ lệ thắng cao và phù hợp nhất), còn ở nước đi kế tiếp thì người chơi B cố gắng để tối thiểu hóa cơ hội thắng của người A (cố gắng giảm thiểu tỷ lệ thắng của A) [8].

2.1.2. Lịch sử hình thành [9]

Trong những cột mốc đánh dấu sự hình thành và phát triển của trí tuệ nhân tạo, sự hình thành của thuật toán Minimax luôn được gọi tên như sự đóng góp rất lớn cho trí tuệ nhân tạo nói chung và thuật toán ứng dụng trong trò chơi nói riêng. Claude Shannon, được coi là cha đẻ của lý thuyết thông tin, đã đóng góp quan trọng cho sự phát triển của thuật toán Minimax. Năm 1950, ông đã viết bài báo "Programming a Computer for Playing Chess" (Lập trình một máy tính chơi cờ vua) - bài báo đầu tiên về phát triển một phần mềm máy tính chơi cờ vua. Bài báo trình bày phương pháp sử dụng thuật toán Minimax để lập kế hoạch cho một chương trình chơi cờ vua trên máy tính. Ông đã đề xuất sử dụng một bảng đánh giá vị trí của các quân cờ để tính toán giá trị của các trạng thái trong trò chơi.

Năm 1956, cụm từ "Artificial Intelligence" (trí tuệ nhân tạo) được đề xuất bởi nhà khoa học máy tính John McCarthy tại hội nghị Dartmouth, đánh dấu mốc chính thức ra đời của trí tuệ nhân tạo. McCarthy vừa là cha đẻ của trí tuệ nhân tạo, còn là cha đẻ của ngôn ngữ lập trình Lisp. Trong thời gian ấy, bài báo "The Dartmouth Conference" (hội nghị Dartmouth) của John McCarthy đã đề cập đến việc sử dụng thuật toán Minimax để tạo ra một chương trình chơi trò chơi tic-tac-toe trên máy tính.

Từ đó đến nay, đã có rất nhiều nghiên cứu và bài báo về thuật toán Minimax, có thể kể đến như: "Dynamic programming and minimax algorithms" (Quy hoạch động và thuật toán Minimax) – Bellman, R. (1957); "Analysis of a simple game using minimax" (Phân tích một trò chơi đơn giản sử dụng Minimax) – Knuth, D.E. (1975); "Artificial intelligence: A modern approach" (Trí tuệ nhân tạo – Cách tiếp cận hiện đại) - Russell, S., & Norvig, P. (2010). Và kể từ ấy, thuật toán Minimax đã được sử dụng rộng rãi trong các trò chơi hai người với lượt đi xen kẽ, và được coi là một trong những thuật toán cơ bản nhất của trí tuệ nhân tạo. Các biến thể của thuật toán này như Alpha-Beta pruning và Monte Carlo Tree Search cũng đã được phát triển để tăng tốc độ tính toán và cải thiện hiệu quả của thuật toán.

2.2. Xây dựng giải thuật Minimax

2.2.1. Ý tưởng [7], [8]

Hai đối thủ trong trò chơi được gọi là MIN và MAX luân phiên thay thế lượt đi. MAX đại diện cho người quyết dành thắng lợi và cố gắng tối đa hóa ưu thế của mình. Ngược lại người chơi đại diện cho MIN lại cố gắng giảm điểm số của MAX và cố gắng làm cho điểm số của mình càng âm càng tốt. Giả thiết đưa ra MIN và MAX có kiến thức như nhau về không gian trạng thái trò chơi, cả hai đối thủ đều cố gắng như nhau và sử dụng những thông tin là như nhau.

Một trò chơi như vậy có thể được biểu diễn bởi một cây trò chơi. Mỗi một Node của cây biểu diễn cho một trạng thái. Node gốc biểu diễn cho trạng thái bắt đầu của cuộc chơi. Mỗi node lá biểu diễn cho một trạng thái kết thúc của trò chơi (trạng thái thắng, thua hoặc hòa). Nếu trạng thái x được biểu diễn bởi node n thì các con của n biểu diễn cho tất cả các trạng thái kết quả của các nước đi có thể xuất phát từ trạng thái x. Do hai đấu thủ luân phiên nhau đi nước của mình nên các mức (lớp) trên cây trò chơi cũng luân phiên nhau là MAX và MIN. Trên cây trò chơi các node ứng với trạng thái mà từ đó người chơi MAX chọn nước đi sẽ thuộc lớp MAX, các node ứng với trạng thái mà từ đó người chơi MIN chọn nước đi sẽ thuộc lớp MIN.

Chiến lược minimax được thể hiện qua quy tắc định trị cho các node trên cây trò chơi, được biểu diễn như sau:

- Nếu node là node lá gán thì cho node đó một giá trị để phản ánh trạng thái thắng, thua hay hòa của các đấu thủ.

- Sử dụng giá trị của các node lá để xác định giá trị của các node ở các mức trên trong cây trò chơi theo quy tắc:

    + Node thuộc lớp MAX thì gán cho nó giá trị lớn nhất của các node con của node đó.

    + Node thuộc lớp MIN thì gán cho nó giá trị nhỏ nhất của các node con của node đó.

Giá trị được gán cho từng trạng thái theo quy tắc trên chỉ rõ giá trị của trạng thái tốt nhất mà mỗi đối thủ có thể hy vọng đạt được. Người chơi sẽ sử dụng các giá trị này để lựa chọn các nước đi cho mình một cách hợp lý nhất. Đối với người chơi MAX khi đến lượt đi, người chơi này sẽ chọn nước đi ứng với trạng thái có giá trị cao nhất trong các trạng thái con, còn với người chơi MIN khi đến lượt sẽ chọn nước đi ứng với trạng thái có giá trị nhỏ nhất trong các trạng thái con.

Cụ thể, một giải thuật minimax sẽ thực hiện theo các bước như sau:

+ Nếu như đạt đến giới hạn tìm kiếm (đến tầng dưới cùng của cây tìm kiếm tức là trạng thái kết thúc của trò chơi).

+ Tính giá trị của thế cờ hiện tại ứng với người chơi ở đó. Ghi nhớ kết quả.

+ Nếu như mức đang xét là của người chơi cực tiểu (node MIN), áp dụng thủ tục Minimax này cho các con của nó. Ghi nhớ kết quả nhỏ nhất.

+ Nếu như mức đang xét là của người chơi cực đại (node MAX), áp dụng thủ tục Minimax này cho các con của nó. Ghi nhớ kết quả lớn nhất.

2.2.2. Xây dựng mã giả

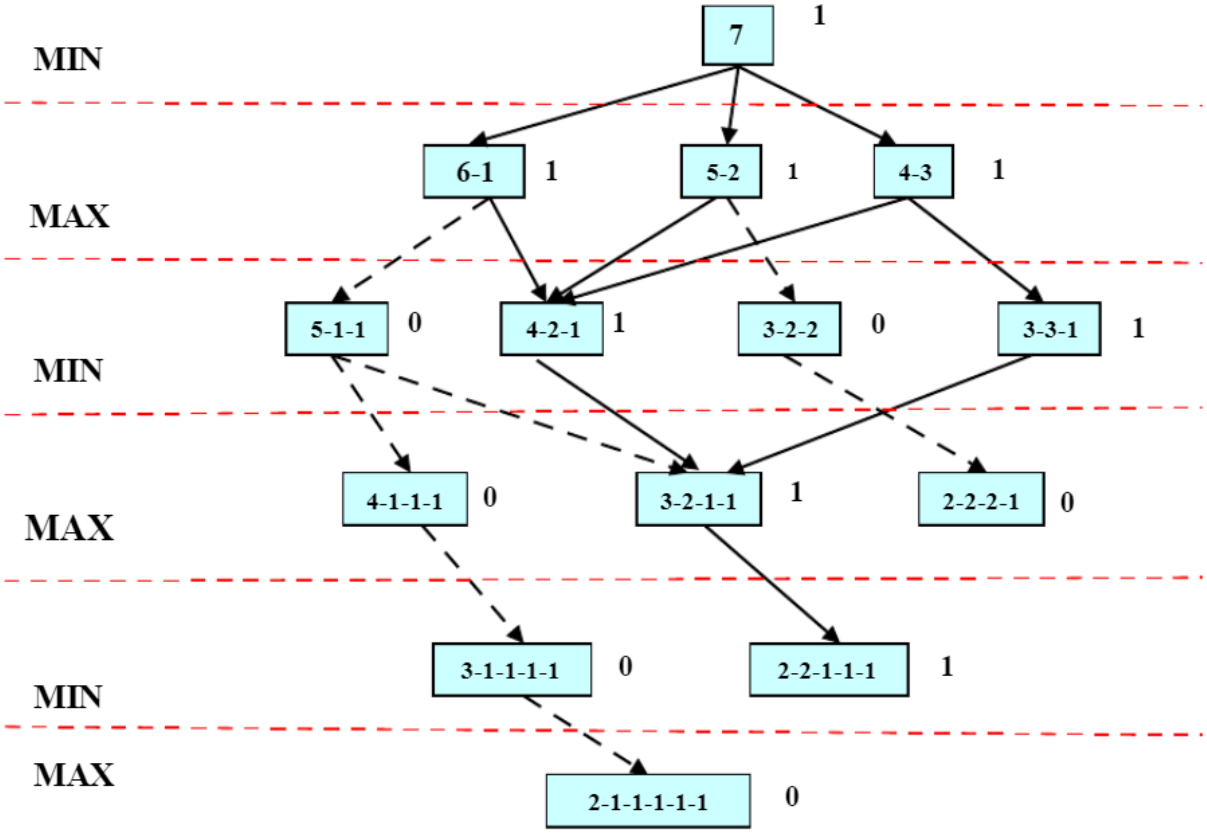
Để có thể hiểu rõ hơn, ta xem xét mã giả mô tả hàm Minimax dưới đây:

```
function minimaxFunction(node, depth, maximumPlayer=True){
    if node is node_leaf or depth == 0:
        return value(node);
    if maximumPlayer:
        maxx = -INF
        for child of node:
            maxx = max(maxx, minimaxFunction(child, depth-1, False))
        return maxx
    else:
        minn = INF
        for child of node:
            minn = min(minn, minimaxFunction(child, depth-1, True))
        return maxx
}
```

Phương thức ***minimaxFunction*** được truyền vào 03 tham số với: *node* đại diện cho node muốn xác định lượng giá; *depth* là độ sâu của node hiện tại, với

$depth = 0$ đại diện node gốc; $maximumPlayer$ đại diện cho mức mà người chơi đang thực hiện, tham số nhận giá trị $True$ nếu người chơi là mức MAX, ngược lại, giá trị là $False$ nếu người chơi là mức MIN. Phương thức sẽ kết thúc và trả về giá trị cho node nếu node là node gốc hoặc là node lá. Nếu không, phương thức sẽ tính giá trị theo từng trường hợp MAX, MIN tùy theo yêu cầu người dùng. Nếu hiện tại, node cần xác định đang ở mức MAX, phương thức sẽ tính gán giá trị lớn nhất của tất cả các node con MIN cho node cha MAX này. Ngược lại, nếu node cần xác định đang ở mức MIN, phương thức sẽ tính gán giá trị nhỏ nhất của tất cả các node con MAX cho node cha MIN chứa tất cả node con đó.

2.2.3. Ví dụ trên trò chơi Nim



Hình 2.1: Ví dụ về không gian trạng thái của trò chơi NIM đơn giản

Để đơn giản việc mô tả không gian trạng thái, ta xét một biến thể (cách chơi khác) của trò chơi Nim với đề bài như sau: Ta có một số token (vật biểu hiện như đồng xu, lá bài, mảnh gỗ…) được đặt trên bàn giữa hai đối thủ. Ở mỗi nước đi, người chơi phải chia đống token thành hai đống nhỏ có số lượng khác nhau. Người chơi nào đến lượt mà không chia được là thua cuộc. Ứng với một lượng token vừa

phải, không gian trạng thái này có thể triển khai đến cùng. Hình 2.1 biểu diễn không gian trạng thái của trò chơi có 7 token.

Khi chơi các trò chơi khó có thể triển khai hết không gian trạng thái, khó khăn chủ yếu là phải tính toán phản ứng của đối thủ. Một cách xử lý đơn giản nhất là giả sử đối thủ của chúng ta cũng sử dụng kiến thức về không gian trạng thái giống như chúng ta và áp dụng kiến thức đó kiên định để thắng cuộc. Mặc dù giả thiết này có những hạn chế của nó nhưng nó cũng cho chúng ta một cơ sở hợp lý để dự đoán hành vi của đối thủ. Giải thuật Minimax sẽ tìm kiếm không gian của trò chơi này theo giả thiết đó.

Áp dụng chiến lược Minimax, chúng ta đánh dấu luân phiên từng mức trong không gian tìm kiếm phù hợp với đối thủ có nước đi ở mức đó. Trong ví dụ trên, MIN được quyền đi trước, từng node lá được gán giá trị 1 hay 0 tùy theo kết quả đó là thắng cuộc đối với MAX hay MIN. Kết quả của việc áp dụng Minimax vào đồ thị không gian trạng thái đối với trò chơi Nim được thể hiện như hình trên. Vì tất cả các nước đi đầu tiên có thể xảy ra cho MIN sẽ dẫn đến các node có giá trị 1 nên đối thủ MAX luôn có thể bắt trò chơi giành thắng lợi cho mình bất kể nước đi đầu tiên của MIN là như thế nào (đường đi thắng lợi của MAX được cho theo mũi tên đậm).

2.3. Tiêu chuẩn Minimax khi gặp sự không chắc chắn [10]

2.3.1. Tiêu chuẩn Minimax trong lý thuyết quyết định thống kê

Trong lý thuyết quyết định thống kê cổ điển, một đánh giá δ thông thường được sử dụng để đánh giá một tham số $\theta \in \Theta$. Giả sử ta có một hàm rủi ro $R(\theta, \delta)$, thường được cho như là một tích phân của một hàm mất mát. Trong cấu trúc này, $\tilde{\delta}$ được gọi là *minimax* nếu như nó thỏa mãn phương trình sau:

$$\sup_{\theta} R(\theta, \tilde{\delta}) = \inf_{\delta} \sup_{\theta} R(\theta, \delta)$$

Trong lý thuyết quyết định, một tiêu chuẩn Minimax được sử dụng được gọi là đánh giá Bayes, tiêu chuẩn này có một phân bổ cho trước Π. Do đó, một tiêu chuẩn được gọi là đánh giá Bayes nếu như nó là hàm tối thiểu rủi ro trung bình, tức là giá trị của tiêu chuẩn tương đương với:

$$\int_{\Theta} R(\theta, \delta) \, . \, d\Pi(\theta)$$

2.3.2. Chiến thuật Minimax khi gặp sự không chắc chắn [7]

Lý thuyết minimax đã được mở rộng ra các quyết định khi mà không có người chơi khác, nhưng các hậu quả của các quyết định dựa trên những sự kiện không biết trước. Chẳng hạn, quyết định tương lai phát đạt của một mỏ khoáng chất kèm theo đuôi một giá phải trả nếu như không có khoáng sản ở nơi muốn thăm dò, nhưng sẽ đem lại mối lợi lớn nếu có. Một phương pháp tiếp cận được sử dụng sẽ tương tự với ý tưởng sử dụng minimax cho bài toán có tổng bằng không, và sử dụng một suy nghĩa giống như là luật Murphy, theo một tiếp cận làm tối thiểu các tổn thất dự định cực đại (*maximum expected loss*), sử dụng các kỹ thuật giống như trong những trò chơi hai người với tổng bằng không.

Nói cách khác, việc quyết định được xem xét như một trò chơi không có người chơi đối diện, trong đó sự không chắc chắn được xem xét thông qua việc tối thiểu hóa các tổn thất dự định cực đại, tương tự như cách Minimax tối ưu hóa lợi ích và tổn thất trong trò chơi hai người với tổng bằng không. Tuy nhiên, trong trường hợp này, sự không chắc chắn không chỉ đến từ hành động của người chơi mà còn đến từ các biến không biết trước được trong tình huống tương lai, và việc áp dụng các kỹ thuật tối ưu này giúp tối thiểu hóa rủi ro hoặc tổn thất trong quyết định.

Một số kỹ thuật được sử dụng để xử lý khi thuật toán Minimax gặp phải trường hợp này, có thể kể đến như:

- *Thống kê và xác suất*: Khi sự không chắc chắn xuất hiện, Minimax có thể sử dụng các phân phối xác suất hoặc thông tin thống kê để ước tính xác suất của các sự kiện không chắc chắn.

- *Cắt cành (Prunning)*: Trong trò chơi hoặc quyết định phân tách thành các nhánh, kỹ thuật cắt cành (pruning) như alpha-beta pruning có thể được áp dụng để giảm bớt số lượng trạng thái phải duyệt. Điều này giúp tối ưu hóa hiệu suất tính toán của thuật toán Minimax, đặc biệt trong các trường hợp lớn và phức tạp.

- *Ước tính (Estimation techniques)*: Trong một số trường hợp, Minimax có thể sử dụng các kỹ thuật ước tính như Monte Carlo để ước tính giá trị của các trạng thái không chắc chắn.

- *Mô hình hóa không chắc chắn*: Minimax có thể sử dụng các mô hình không chắc chắn như mô hình Markov Decision để mô phỏng và xử lý sự không chắc chắn trong quyết định

2.4. Giải thuật Minimax với độ sâu được giới hạn

Đối với các trò chơi có tính phức tạp cao, không gian trạng trái của chúng thường lớn và mở rộng hơn, khi áp dụng chiến thuật Minimax cho các trò chơi này thì hiếm khi ta có khả năng mở rộng đồ thị không gian trạng thái đến các node lá. Thay vào đó không gian trạng thái này chỉ có thể được triển khai đến một số mức xác định phụ thuộc tiềm năng về thời gian tính toán và bộ nhớ chẳng hạn. Do đó, thông thường các nhà phân tích thường sử dụng một chiến thuật tốt hơn, chiến thuật này được gọi là tính trước n nước đi (n – move lookahead) và chúng sử dụng thêm một hàm đánh giá heuristic.

Vì giá trị các node trong đồ thị con này không phải là trạng thái kết thúc của trò chơi nên chúng không phản ánh giá trị thắng cuộc hay thua cuộc. Chúng chỉ có thể được gán một giá trị phù hợp với một hàm đánh giá heuristic nào đó. Giá trị được truyền ngược về node gốc không cung cấp thông tin thắng cuộc hay thua cuộc mà chỉ là giá trị heuristic của trạng thái tốt nhất có thể tiếp cận sau n nước đi kể từ node xuất phát. Việc tính trước này sẽ làm tăng hiệu quả của heuristic vì nó được áp dụng vào một phạm vi lớn hơn trong không gian trạng thái. Minimax sẽ hợp nhất tất cả các giá trị của các node con cháu của một trạng thái thành một giá trị duy nhất cho trạng thái đó.

Trong các cây trò chơi được tìm kiếm bằng mức hay lớp, MAX và MIN luân phiên nhau chọn các nước đi. Mỗi nước đi của một đối thủ sẽ xác định một lớp mới trên cây. Các chương trình trò chơi nói chung đều dự tính trước một độ sâu lớp cố định (thường được xác định bằng các giới hạn về không gian hoặc thời

gian của máy tính). Các trạng thái trên mức đó được đánh giá theo một hàm heuristic và các giá trị này sẽ được truyền ngược lên bằng thủ tục Minimax, sau đó thuật toán tìm kiếm sẽ dùng các giá trị vừa nhận được để chọn lựa một nước trong số các nước đi kế tiếp. Bằng cách tối đa hóa cho các node cha MAX và tối thiểu hóa cho các node cha MIN, những giá trị này đi lùi theo đồ thị đến node con của trạng thái hiện hành. Sau đó trạng thái hiện hành dùng chúng để tiến hành lựa chọn trong các node con của nó để lấy giá trị cho mình [8].

Ở đây, chiến lược sử dụng một hàm heuristic phức tạp hơn, nó cố gắng đo mức độ tranh chấp trong trò chơi. Heuristic chọn một trạng thái cần đo, tính tất cả các đường thắng mở ra cho MAX, rồi trừ đi tổng số các đường thắng mở ra cho MIN. Giải thuật tìm kiếm sẽ cố gắng tối đa hóa sự chênh lệch (hiệu số) đó. Nếu có một trạng thái bắt buộc thắng cuộc cho MAX, nó sẽ được đánh giá là $+\infty$, còn với trạng thái bắt buộc thắng cuộc cho MIN thì được đánh giá là $-\infty$.

2.5. Ưu điểm và nhược điểm của thuật toán Minimax [7], [9]

2.5.1. Ưu điểm

Ưu điểm của chiến lược Minimax có thể kể đến như:

- Bản chất, chiến lược minimax có tính chất vét cạn nên không bỏ sót trạng thái nào, mọi bước đi tiếp theo được tìm kiếm và đánh giá được. Do đó, chiến lược đảm bảo tìm kiếm được bước đi tối ưu cho một người chơi trong trò chơi.

- Dễ hiểu và áp dụng: Minimax có cách tiếp cận đơn giản, dễ hiểu và thực hiện, phù hợp với nhiều loại trò chơi và tình huống quyết định.

- Tính toán độ chính xác cao nếu không gian trạng thái trò chơi không quá lớn. Vì vậy, chiến lược phù hợp với các trò chơi có thông tin hoàn toàn, qua đó tìm được chiến lược tối ưu nhất.

- Ngoài ra, chiến lược có thể được mở rộng để giải quyết các trò chơi phức tạp hơn bằng cách sử dụng các kỹ thuật như Alpha-Beta pruning hoặc Monte Carlo Tree Search.

2.5.2. Nhược điểm

Tuy nhiên, chiến lược Minimax cũng tồn tại một số hạn chế có thể kể đến như sau:

- Không hiệu quả khi có sự phức tạp và không chắc chắn: Như đã trình bày, nếu chiến lược gặp phải không gian trạng thái lớn hoặc không chắc chắn, chiến lược khó có thể tìm ra lời giải tối ưu vì thời gian tính toán lớn và bộ nhớ lớn. Do đó, Minimax có thể trở nên không hiệu quả vì đòi hỏi quá nhiều tài nguyên tính toán và không gian lưu trữ các trạng thái. Một số kỹ thuật được áp dụng để khắc phục, tuy nhiên chúng không triệt để.

- Không linh hoạt với các chiến lược không truyền thống: Đôi khi, Minimax không linh hoạt đối với các chiến lược không truyền thống hoặc các tình huống không thể mô hình hóa theo cách tối ưu.

- Yêu cầu thông tin hoàn toàn: Minimax hiệu quả trong các trò chơi có thông tin hoàn toàn, nhưng trong thực tế, thông tin thường không hoàn toàn và điều này có thể làm giảm hiệu suất của phương pháp này. Do đó, thuật toán Minimax không thể xử lý các trò chơi có yếu tố ngẫu nhiên, ví dụ như poker hoặc blackjack.

Tóm lại, thuật toán Minimax là một thuật toán tìm kiếm cây trò chơi phổ biến và mạnh mẽ cho các trò chơi hai người với lượt đi xen kẽ. Tuy nhiên, nó có những giới hạn về khả năng áp dụng cho các trò chơi lớn và phức tạp hơn. Vì vậy, việc sử dụng chiến lược này trong các bài toán đòi hỏi nhiều vấn đề cũng như kỹ thuật xử lý để có thể tìm ra lời giải một cách tối ưu nhất.

2.6. Đánh giá thuật toán [6]–[8]

Về bản chất, chiến lược Minimax sử dụng kỹ thuật vét cạn và thực hiện thăm toàn bộ cây trò chơi bằng việc dùng chiến lược tìm kiếm theo chiều sâu. Do đó, chiến lược Minimax sẽ có đầy đủ các tính chất của các giải thuật trên. Để cụ thể, giả sử cây tìm kiếm có hệ số phân nhánh là b và tham số độ sâu của cây là d.

a. Tính đầy đủ

Chiến lược Minimax sẽ tìm kiếm tất cả các giá trị của từng bước đi, do đó chiến lược sẽ tìm kiếm được tất cả các lời giải của bài toán nếu thời gian tính toán và không gian bộ nhớ đủ lớn và cho phép.

b. Tính tối ưu

Như được giải thích ở trên, vì tìm ra tất cả các lời giải của bài toán, do đó chiến lược có thể tìm ra được lời giải tối ưu nhất. Tùy thuộc vào các điều kiện ràng buộc mà nó sẽ tìm ra được các lời giải tối ưu phù hợp nhất.

c. Độ phức tạp không gian

Bản chất của thuật toán là tìm kiếm theo chiều sâu, vì vậy việc đòi hỏi không gian bộ nhớ của nó chỉ tuyến tính với d và b. Do đó, độ phức tạp không gian là $O(bd)$ [8][13].

d. Độ phức tạp tính toán

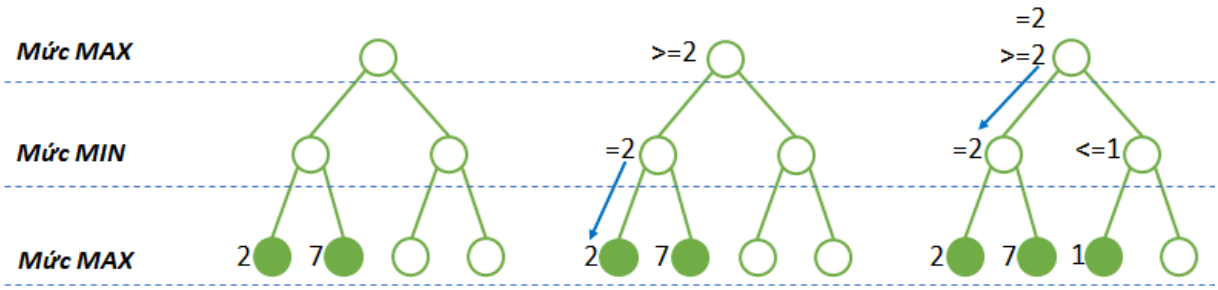
Độ phức tạp tính toán của chiến lược này tương ứng trực tiếp với kích thước không gian tìm kiếm b^d. Thuật toán sẽ thăm tất cả các node không chỉ là các node lá vì vậy số lượng các node được thăm sẽ là $\frac{b(b^d-1)}{b-1}$. Nhưng hàm đánh giá sẽ là phương thức chi phối hầu hết thời gian và chỉ làm việc trên các node lá, vì vậy việc thăm các node không phải các node lá có thể được bỏ qua [8]. Do đó, chiến lược sẽ độ phức tạp thời gian là $O(b^d)$.

CHƯƠNG 3. PHƯƠNG PHÁP CẮT TỈA ALPHA-BETA

Như đã trình bày tại Chương 2, thuật toán Minimax có một số hạn chế đáng lưu ý khi áp dụng trong các trò chơi có trạng thái lớn và không chắc chắn. Một trong những vấn đề lớn nhất là độ phức tạp tính toán và không gian. Với một cây trạng thái lớn, Minimax có thể phải duyệt hàng ngàn hoặc thậm chí hàng triệu trạng thái, dẫn đến thời gian tính toán lâu và yêu cầu bộ nhớ lớn.

Để giải quyết vấn đề này, nhiều kỹ thuật đã được nghiên cứu, cải tiến để khắc phục các nhược điểm của chiến lược Minimax. Trong đó, 02 kỹ thuật được sử dụng phổ biến là kỹ thuật cắt tỉa Alpha-Beta và kỹ thuật cây tìm kiếm Monte Carlo (Monte Carlo Tree Search – MCTS). Do mục tiêu cũng như yêu cầu của đề bài, tiểu luận sẽ trình bày về kỹ thuật cắt tỉa Alpha-Beta, đây là một kỹ thuật cải tiến khá quan trọng và áp dụng nhiều trong các trò chơi hiện nay.

Thuật toán Alpha-Beta là một cải tiến của chiến lược Minimax nhằm bỏ bớt các nhánh trong cây trò chơi, giảm số lượng node cần phải xét tới trong chiến lược Minimax. Từ đó làm giảm thời gian cần thiết của việc tìm kiếm, và không làm lãng phí thời gian tìm kiếm những bước đi gây bất lợi rõ rệt cho người chơi. Việc thực thi Alpha-Beta ghi lại nước đi tốt nhất cho mỗi bên khi nó duyệt cây.



Hình 3.1: Ví dụ về ý tưởng của giải thuật cắt tỉa Alpha-Beta

Xem xét một ví dụ được thể hiện bởi hình 3.1, giả sử hai node ở mức MAX lớp 3 đã được gán giá trị 2 và 7. Nếu thực hiện giải thuật Minimax đối với các node đó sẽ cho thấy node MAX trên cùng sẽ có giá trị được bảo toàn là 2 nếu đi theo nhánh bên trái dù các giá trị của các node mức MAX lớp 3 có giá trị thế nào đi nữa. Nếu node MIN bên phải được xác định giá trị là 1. Nếu đi vào nhánh này,

đối phương sẽ đảm bảo làm điểm của người chơi cực đại không vượt quá 1, dù các giá trị của các node khác có thế nào đi nữa. Do đó đến đây, nước đi tốt nhất là chọn nước đi bên trái với đảm bảo là ít nhất đạt được 2 điểm. Do đó, không cần phải xác định giá trị các node còn lại.

3.1. Ý tưởng giải thuật cắt tỉa ALPHA-BETA

Ý tưởng của tìm kiếm Alpha-beta rất đơn giản: Thay vì nếu như tìm kiếm toàn bộ không gian đến một độ sâu lớp cố định, tìm kiếm Alpha-Beta thực hiện theo kiểu tìm kiếm sâu. Có hai giá trị, gọi là $alpha$ và $beta$ được tạo ra trong quá trình tìm kiếm:

- Giá trị $alpha$ liên quan với các node MAX và có khuynh hướng không bao giờ giảm.

- Ngược lại, giá trị $beta$ liên quan đến các node MIN và có khuynh hướng không bao giờ tăng.

Giả sử có giá trị $alpha$ của một node MAX là 6, MAX không cần phải xem xét giá trị truyền ngược nào nhỏ hơn hoặc bằng 6 có liên quan với một node MIN nào đó bên dưới. Giá trị $alpha$ là giá trị thấp nhất mà MAX có thể nhận được sau khi cho rằng MIN cũng sẽ nhận giá trị tốt nhất của nó. Tương tự nếu MIN có giá trị $beta$ là 6 nó cũng không cần xem xét các node nằm dưới nó có giá trị lớn hơn hoặc bằng 6.

Để bắt đầu thuật toán tìm kiếm Alpha-beta, thuật toán sẽ đi xuống hết độ sâu lớp theo kiểu tìm kiếm sâu, đồng thời áp dụng đánh giá heuristic cho một trạng thái và tất cả các trạng thái anh em của nó. Giả sử tất cả đều là node MIN. Giá trị tối đa của các node MIN này sẽ được truyền ngược lên cho node cha (là một node MAX). Sau đó giá trị này được gán cho ông của các node MIN như là một giá trị $beta$ kết thúc tốt nhất. Tiếp theo thuật toán này sẽ đi xuống các node cháu khác và kết thúc việc tìm kiếm đối với node cha của chúng nếu gặp bất kỳ một giá trị nào lớn hơn hoặc bằng giá trị $beta$ này. Quá trình này gọi là cắt tỉa Beta ($\beta - cut$). Cách làm tương tự cũng được thực hiện cho việc cắt tỉa Alpha ($\alpha - cut$) đối với các node cháu của một node MAX

Hai quy luật cắt tỉa dựa trên các giá trị $alpha$ và $beta$ bao gồm:

- Quá trình tìm kiếm có thể có kết thúc bên dưới một node MIN nào có giá trị $beta$ nhỏ hơn hoặc bằng giá trị $alpha$ của một node cha MAX bất kỳ của nó.

- Quá trình tìm kiếm có thể kết thúc bên dưới một node MAX nào có giá trị $alpha$ lớn hơn hoặc bằng giá trị $beta$ của một node cha MIN bất kỳ của nó.

Việc cắt tỉa Alpha-beta như vậy thể hiện quan hệ giữa các node ở lớp n và các node ở lớp $n + 2$ và do quan hệ đó toàn bộ các cây con bắt nguồn ở lớp $n + 1$ đều có thể loại khỏi việc xem xét.

Chú ý rằng giá trị truyền ngược thu được hoàn toàn giống như kết quả Minimax, đồng thời tiết kiệm được các bước tìm kiếm một cách đáng kể.

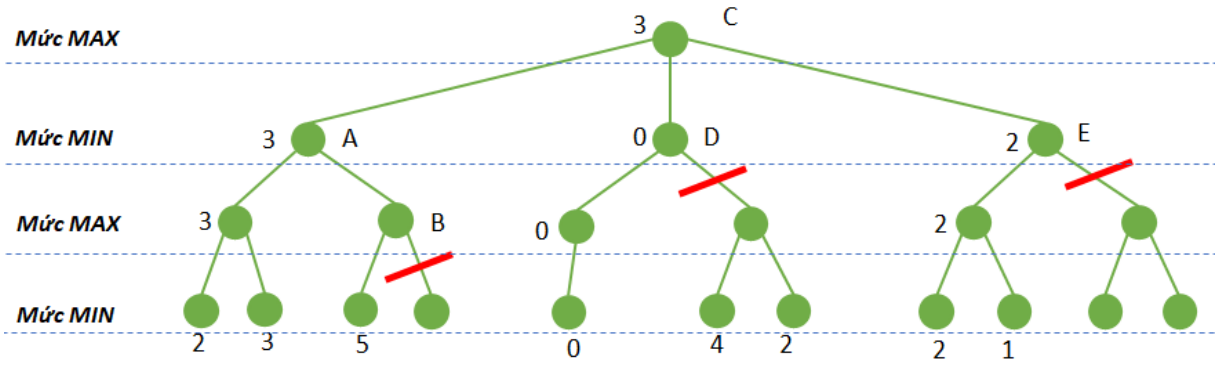
3.2. Xây dựng giải thuật

Từ ý tưởng xây giải thuật Alpha-Beta trên, ta sẽ xây dựng giải thuật với các bước như sau:

- Nếu mức đang xét là đỉnh (gốc của cây), đặt giá trị $alpha = -\infty$ và $beta = +\infty$.
- Nếu như đạt đến giới hạn tìm kiếm (đến tầng dưới cùng của cây tìm kiếm hoặc là node lá), tính giá trị tĩnh của thế cờ hiện tại ứng với người chơi ở đó. Ghi lại kết quả.
- Nếu như mức đang xét là của người chơi cực tiểu (MIN), thực hiện các công việc sau cho đến khi tất cả các con của nó đã được xét với thủ tục Alpha-Beta hoặc cho đến khi $alpha \geq beta$:
    + Áp dụng giải thuật Alpha-Beta với giá trị $alpha$ và $beta$ hiện tại cho một node con. Ghi lại kết quả.
    + So sánh giá trị ghi nhớ với giá trị $beta$, nếu giá trị đó nhỏ hơn thì gán giá trị đó cho $beta$. Ghi nhớ lại giá trị $beta$.
- Nếu như mức đang xét là của người chơi cực đại (MAX), thực hiện các công việc sau cho đến khi tất cả các con của nó đã được xét với thủ tục

Alpha-Beta hoặc cho đến khi $alpha \geq beta$:

+ Áp dụng giải thuật Alpha-Beta với giá trị $alpha$ và $beta$ hiện tại cho một node con. Ghi lại kết quả.
+ So sánh giá trị ghi nhớ với giá trị $alpha$, nếu giá trị đó nhỏ hơn thì gán giá trị đó cho $alpha$. Ghi nhớ lại giá trị $alpha$.



Hình 3.2: Ví dụ giải thuật cắt tỉa Alpha-Beta

3.3. Mã giả thuật toán cắt tỉa Alpha-Beta

Dựa trên các bước của thuật toán, ta xây dựng hàm cắt tỉa Alpha-Beta với mã giả với các tham số $depth$ (là độ sâu tìm kiếm), $alpha$ và $beta$:

```
function minimaxFunction(node, depth, maximumPlayer=True){
    return alphabetaFunction(node, depth, -INF, INF, maximumPlayer=True)
}
def alphabetaFunction(node, depth, a, b, maximumPlayer=True){
    if node is node_leaf or depth == 0:
        return value(node);
    if maximumPlayer:
        for child of node:
            a = max(a, minimaxFunction(child, depth-1, a, b, False))
            if a>=b:
                break
        return a
    else:
        for child of node:
            b = min(b, minimaxFunction(child, depth-1, a, b, True))
            if a>=b:
                break
        return b
}
```

So với thuật toán Minimax thì trong thuật toán AlphaBeta đã đưa thêm hai biến $alpha$, $beta$ làm hai mức ngưỡng. Ta thấy cứ mỗi khi $alpha \geq beta$ thì

thuật toán không thực hiện tiếp vòng lặp, có nghĩa là nó không chịu mở rộng tiếp những nhánh còn lại nữa. Các nhánh đó đã bị cắt bỏ và do đó ta sẽ tiết kiệm được thời gian. Việc cắt bỏ này hoàn toàn an toàn với những lí do ta đã xét ở trên. Ta thấy rằng mỗi lần hàm này được gọi thì chỉ có tham số $beta$ được dùng để so sánh cắt bỏ, còn tham số $alpha$ không được dùng. Tuy nhiên khi áp dụng cùng thuật toán cho cây con thì ta đã hoán vị hai giá trị $alpha$, $beta$ cho nhau (và đảo cả dấu), do đó $alpha$ sẽ có tác dụng trong độ sâu sau, rồi độ sâu sau nữa lại đến lượt $beta$, ... Nói cách khác, một giá trị chỉ luôn ảnh hưởng đến người chơi cực đại, còn giá trị kia lại luôn ảnh hưởng đến người chơi cực tiểu. Chúng là các ngưỡng giữa các nước đi được chấp nhận và không chấp nhận. Những nước đi cần quan tâm phải nằm lọt giữa hai giá trị này. Dần dần khoảng cách giữa hai giá trị $alpha$ và $beta$ càng ngày càng thu hẹp và dẫn đến các nhánh cây có giá trị nằm ngoài khoảng này nhanh chóng bị cắt bỏ.

3.3. Đánh giá thuật toán

a. Tính đầy đủ

Phương pháp cắt tỉa Alpha-Beta về bản chất tương tự như chiến lược Minimax truyền thống nên bài toán sẽ tìm được tất cả lời giải, miễn thời gian tìm kiếm là không hạn chế.

b. Tính tối ưu

Phương pháp có thể tìm ra tất cả lời giải, do đó có thể tìm ra lời giải tối ưu nhất cho bài toán, hơn nữa phương pháp này về bản chất chỉ làm giảm sự bùng nổ nhánh trong không gian tìm kiếm, do đó về bản chất phương pháp vẫn mang các tính chất của chiến lược alpha-beta.

c. Độ phức tạp không gian

Bản chất của thuật toán là tìm kiếm theo chiều sâu, vì vậy việc đòi hỏi không gian bộ nhớ của nó chỉ tuyến tính với d và b. Do đó, độ phức tạp không gian là $O(bd)$ [8][13].

d. Độ phức tạp tính toán

Hiệu quả của việc cắt nhánh Alpha-beta phụ thuộc nhiều vào thứ tự các nước đi kế tiếp được thực hiện. Nếu các nước đi kế tiếp được thực hiện có thứ tự ngẫu nhiên thì tổng số nút được thực hiện sẽ khoảng $O\left(b^{\frac{3}{4}d}\right)$ [8]. Trong điều kiện lý tưởng, giải thuật cắt tỉa Alpha-Beta chỉ phải xét số node theo công thức:

$$\begin{cases} 2b^{\frac{d}{2}-1} & với\ d\ chẵn \\ b^{\frac{d+1}{2}} + b^{\frac{d}{2}} - 1 & với\ d\ lẻ \end{cases}$$

Trong trường hợp tốt nhất, thuật toán cắt tỉa Alpha-Beta chỉ cần thực hiện khoảng $O\left(b^{\frac{d}{2}}\right)$ node để chọn ra nước đi tốt nhất, thay vì $O(b^d)$. Hệ số phân nhánh hiệu quả trở thành $\sqrt{b}$ thay vì b như trong giải thuật Minimax truyền thống.

3.4. So sánh giải thuật Minimax sử dụng cắt tỉa Alpha-Beta với giải thuật Minimax truyền thống

Với cùng một cây trò chơi với độ sâu là 4 và hệ số phân nhánh là 40. Đối với chiến lược minimax truyền thống, ta có số node cần xét là $40^4 = 2560000$, còn đối với giải thuật cắt tỉa Alpha-Beta, số node cần phải xem xét trong điều kiện lý tưởng nhất là $2 \times 40^2 - 1 = 3199$ (node), ít hơn thuật toán Minimax truyền thống khoảng $\frac{256000}{3199} \approx 800$ lần. Để có thể nhìn rõ hơn về độ chênh lệch này, ta có thể tham khảo bảng dữ liệu sau với độ sâu của cây lần lượt tăng dần:

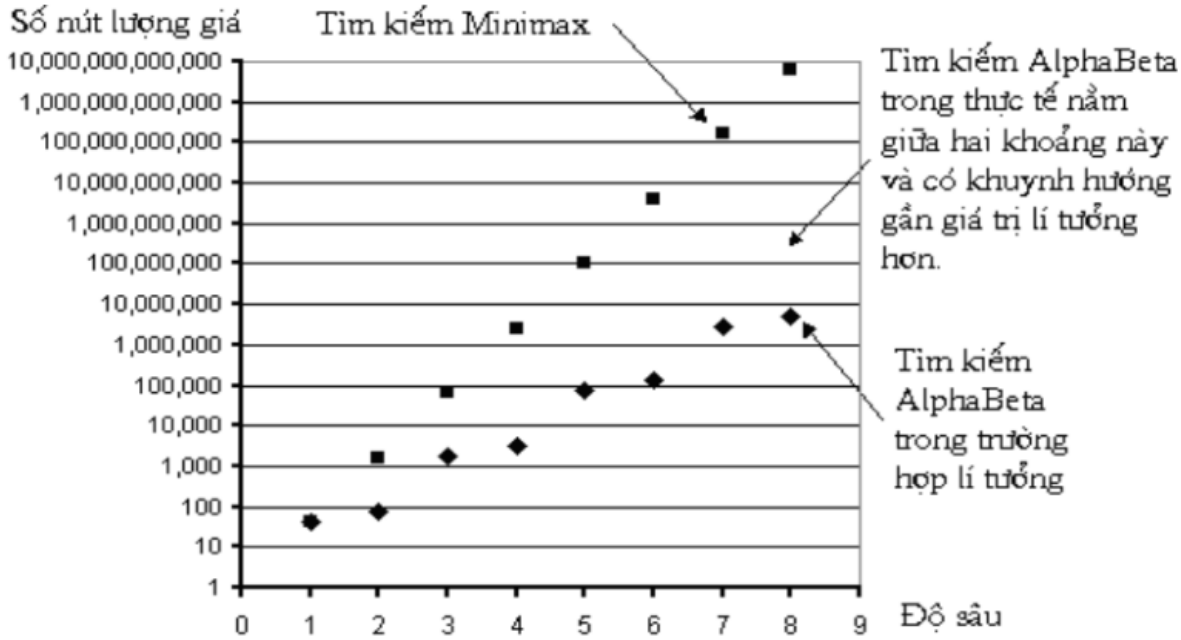
Bảng 3.1: Bảng ví dụ về sự bùng nổ không gian trạng thái với hệ số phân nhánh b = 40 và độ sâu lần lượt tăng dần

Độ sâu	Minimax		Alpha-Beta		Tỷ lệ số node
	Số node	Số lần tăng	Số node	Số lần tăng	
1	40		40		
2	1600	40	79	1.7	20
3	64000	40	1852	23.2	34
4	2560000	40	3199	1.7	800
5	102400000	40	74118	23.2	1381
6	4096000000	40	127999	1.7	32000
7	163840000000	40	2964770	23.2	55262
8	6553600000000	40	5120000	1.7	1280000

Một số nhận xét được rút ra như sau:

- Số lần tăng số node khi tăng độ sâu của Minimax luôn là hệ số phân nhánh b, trong trường hợp này là 40. Số lần tăng của Alpha-beta ít hơn nhiều: chỉ từ 1.7 lần khi tăng từ d lẻ sang d chẵn và 23.2 lần khi từ d chẵn sang lẻ, trung bình chỉ tăng khoảng hơn 6 lần khi tăng d.

- Số node của Alpha-beta tăng chậm hơn rất nhiều lần so với Minimax. Tỉ số node phải xét giữa hai giải thuật này càng cao khi d càng lớn. Công thức tính số node cho thấy số node phải xét khi dùng Alpha-beta ít hơn nhiều so với Minimax nhưng vẫn là hàm số mũ và vẫn dẫn tới bùng nổ tổ hợp. Thuật toán Alpha-beta hoàn toàn không chống được bùng nổ tổ hợp mà chỉ làm giảm tốc độ bùng nổ tổ hợp. Tuy trong thực tế số node phải xét (lượng giá) thường nhiều hơn trong điều kiện lý tưởng nhưng nó vẫn đủ để tiết kiệm khá nhiều thời gian. Trong cùng một khoảng thời gian, thuật toán Alpha-beta có thể tìm đến độ sâu gấp hai lần độ sâu tìm kiếm bằng Minimax. Hình sau đây là đồ thị so sánh giữa hai thuật toán này [5].



Hình 3.3: Biểu đồ mô tả sự bùng nổ trạng thái của hai giải thuật

Như vậy, thuật toán Alpha-beta chỉ làm giảm sự bùng nổ tổ hợp chứ không chống được nó [11].

Tóm lại: Do bùng nổ tổ hợp quá lớn của cây trò chơi mà cả hai người chơi không thể (và không bao giờ) có thể tìm kiếm vét cạn (hết mọi khả năng). Do đó phương pháp tìm kiếm duy nhất là chỉ tìm kiếm đến một độ sâu giới hạn nào đó và chọn nước đi dẫn đến một thế cờ có lợi nhất cho mình. Do phải tính cả khả năng chống trả của đối phương nên ta không dùng được các thuật toán tìm kiếm thông thường. Phải dùng một thuật toán tìm kiếm riêng cho cây trò chơi. Đó là thuật toán Minimax và cải tiến của nó là thuật toán Alpha-beta. Tuy cả hai thuật toán đều không tránh được bùng nổ tổ hợp nhưng thuật toán Alpha-beta làm chậm bùng nổ tổ hợp hơn nên được dùng nhiều trong các trò chơi cờ.

3.6. Các ứng dụng của thuật toán

Thuật toán Minimax và phương pháp cắt tỉa Alpha-Beta là hai phương pháp quan trọng trong lĩnh vực trò chơi và trí tuệ nhân tạo. Dưới đây là một số ứng dụng của hai phương pháp này trong thực tế:

- *Giải quyết các bài toán trò chơi*: Chiến lược Minimax và phương pháp cắt tỉa Alpha-Beta được sử dụng rộng rãi trong các bài toán trò chơi, đặc biệt là các trò chơi đối kháng với không gian trạng thái cụ thể để đưa ra những quyết định tối ưu và tính toán các nước đi hay hành động dựa trên hành vi của người chơi hoặc đối thủ ảo. Các giải thuật trên được áp dụng mạnh mẽ trong lĩnh vực này để tạo ra các trò chơi AI thú vị, giúp người chơi có thể rèn luyện trí tuệ, khả năng phán đoán.

- *Tối ưu hóa và tìm kiếm*: Về bản chất, chiến lược Minimax và phương pháp cắt tỉa Alpha-Beta duyệt qua tất cả các phương pháp, tìm ra được phương pháp tốt nhất, do đó, chúng được sử dụng để tìm kiếm lựa chọn được các phương pháp tối ưu nhất áp dụng trong các vấn đề như lập lịch sản xuất, quản lý công việc, giúp tối ưu hóa quy trình, thời gian, nâng cao hiệu suất làm việc.

- *Tìm kiếm và xử lý thông tin*: Trong việc tìm kiếm thông tin trên internet hoặc trong cơ sở dữ liệu lớn, các phương pháp tối ưu như Alpha-Beta cắt tỉa có thể giúp loại bỏ các lựa chọn không cần thiết, giảm thời gian tìm kiếm.

- *Trí tuệ nhân tạo và học máy*: Thuật toán Minimax và phương pháp cắt cụt Alpha-Beta cung cấp một cách tiếp cận phù hợp để đào tạo mô hình trí tuệ nhân tạo và học máy. Chúng có thể được sử dụng để huấn luyện và cải tiến các mô hình thông qua việc tìm kiếm và đánh giá các chiến lược tốt nhất trong một không gian quyết định lớn.

Ngoài ra, chiến lược Minimax và phương pháp cắt tỉa Alpha-Beta có thể được sử dụng để tìm kiếm chiến lược tối ưu trong quản lý tài sản và quản lý rủi ro. Thuật toán minimax cũng có thể được sử dụng trong các bài toán tìm kiếm con đường tối ưu trong đường đi tối ưu hoá, tối ưu hóa mạng và tối ưu hóa vận hành của hệ thống. Tuy nhiên, các lĩnh vực này thường ít khi sử dụng vì không gian trạng thái của chúng rất phức tạp và nhiều biến đổi theo thời gian thực.

PHẦN 2: XÂY DỰNG CHƯƠNG TRÌNH TRÒ CHƠI NIM

CHƯƠNG 4. BÀI TOÁN NIM VÀ PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN

4.1. Phân tích bài toán

4.1.1. Giới thiệu bài toán

Trò chơi Nim được xuất phát từ Trung quốc, là một trò chơi mang tính chiến thuật. Nội dung của trò chơi này như sau:

Có n đống sỏi, mỗi đống có một số lượng sỏi nhất định. Hai người chơi luận phiên nhau, lần lượt lấy một số sỏi bất kỳ (khác 0) từ một trong n đống sỏi đó (tức là, mỗi lần chơi chỉ lấy sỏi từ một đống bất kỳ). Trò chơi kết thúc khi không còn viên sỏi nào trên bàn và người thua cuộc là người bốc số sỏi cuối cùng. Rõ ràng ở đây không có sự may mắn, ta có thể tìm ra nước đi tốt nhất bằng việc dự đoán kết quả của nước đi trước đó một cách khéo léo.

Thực tế, bài toán NIM có 02 thể thức chơi tùy theo kết quả mà quyết định người chiến thắng, bao gồm: 1. *Normal game*: Là thể thức phổ biến hơn, thể thức này quy định người bốc viên sỏi cuối cùng là người chiến thắng; 2. *Misère game*: Thể thức này ít phổ biến hơn, quy định người bốc viên sỏi cuối cùng là người thua cuộc. Trong quá trình tìm hiểu, nghiên cứu, để thuận tiện cho việc kiểm tra, bài tiểu luận sẽ thực hiện theo thể thức Misère game, tức người bốc viên sỏi cuối cùng là người thua cuộc [12].

Như vậy, yêu cầu đặt ra ở đây là có thể xây dụng một chương trình trò chơi giữa người và máy sao cho máy luôn thắng trò chơi này hay không? Để làm được điều này, chúng ta cần phải xây dựng một giải thuật tối ưu để có thể giúp máy tính chọn được các nước đi một cách tối ưu nhất để chiến thắng trò chơi sau mỗi nước đi của con người.

4.1.2. Cơ sở lý thuyết

Dựa trên luật chơi được mô tả ở trên, có thể thấy trò chơi Nim thuộc lớp

các trò chơi đối kháng giữa hai người chơi. Cụ thể, trò chơi này thuộc dạng trò chơi có tổng bằng không với hai người chơi (Two players, Zero-sum-game). Với yêu cầu của bài tiểu luận đề ra, ta sẽ áp dụng chiến thuật tìm kiếm Minimax và giải thuật cắt tỉa Alpha-beta trong trò chơi này [13].

Một số nhận xét và những quy tắc thường được áp dụng để chiến thắng trò chơi này có thể liệt kê như [14]:

+ Nếu sau một số lượt đi, chỉ còn lại hai hàng với số đồng xu bằng nhau và đến lượt người chơi thứ hai thì người thứ hai thua cuộc.

+ Sau một số bước đi, nếu số đồng xu trong ba hàng còn lại là (a,b,c) với $a = b \neq 1$ và đến lượt người chơi thứ hai thì người thứ hai thắng [15].

Các quy tắc này đã được chứng minh và sử dụng nhiều trong quá tình đối kháng giữa hai người, tuy nhiên, ở đây ta sử dụng giải thuật Minimax với phương pháp cắt tỉa Alpha-Beta nên các quy tắc này không được sử dụng trong chương trình. Tuy nhiên, ta có thể sử dụng các quy tắc này để có thể dự đoán trước ai sẽ chiến thắng và sử dụng để chọn có nên đi trước hay không.

Máy tính có lợi thế là khả năng tính toán nhanh, khả năng nhớ tốt gấp nhiều lần so với con người, vậy để máy tính thắng thì thật dễ, vấn đề là làm sao để máy có thể nghĩ được như người? Và có thể tính trước được ít nhất là 4 đến 5 nước.Vậy phải có thuật toán để máy có thể quét qua các phương án đi, và chọn phương án cuối cùng là tốt nhất sao cho máy tính có thể thắng. Trong ứng dụng này ta chọn thuật toán Minimax và thuật toán cải tiến Alpha-beta để cài đặt cho máy tính chơi.

4.2. Xây dựng hướng giải quyết

Chương trình trò chơi NIM giữa người và máy được xây dựng theo quy trình sau đây:

B1: Nhập dữ liệu đầu vào: Số đống sỏi và số lượng sỏi có trong mỗi đống (có thể sử dụng các hàm sinh ngẫu nhiên tùy người dùng).

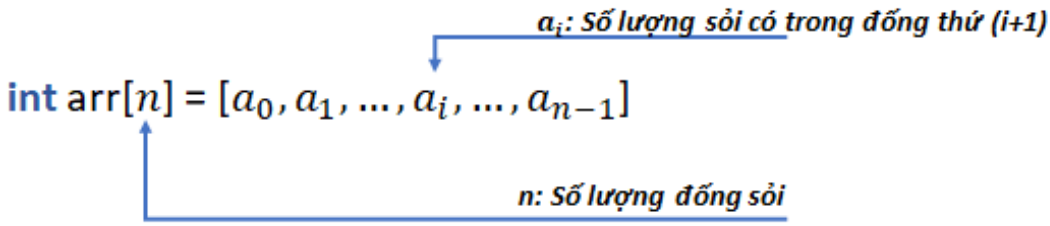
B2: Thực hiện xác nhận người nào sẽ đi trước, thực hiện nhận đầu vào của ngườ chơi theo mỗi lượt đi và thông báo số lượng còn lại của mỗi đống trước và sau mỗi lượt đi.

B3: Thực hiện tính toán đối với mỗi bước đi, thông báo kết quả chiến thắng khi kết thúc trò chơi.

4.2.1. Xây dựng dữ liệu đầu vào

Để có thể giải quyết bất kỳ một bài toán nào, ta cần phải xác định dữ liệu đầu vào và cấu trúc dữ liệu được sử dụng để tối ưu hóa vùng nhớ, dảm bảo cho chương trình có thể thực hiện một cách tối ưu và tốt nhất. Theo yêu cầu và các điều kiện của bài toán, có thể thấy dữ liệu đầu vào là số lượng đống sỏi và số lượng sỏi có trong mỗi đống, sau mỗi lượt chơi thì ta phải cập nhật lại số lượng sỏi của mỗi đống.

Như vậy, phương pháp biểu diễn tối ưu nhất là sử dụng mảng một chiều có độ dài là n để lưu thông tin số lượng của các đống sỏi. Để dễ hiểu ta có thể biểu diễn thông qua sơ đồ sau:



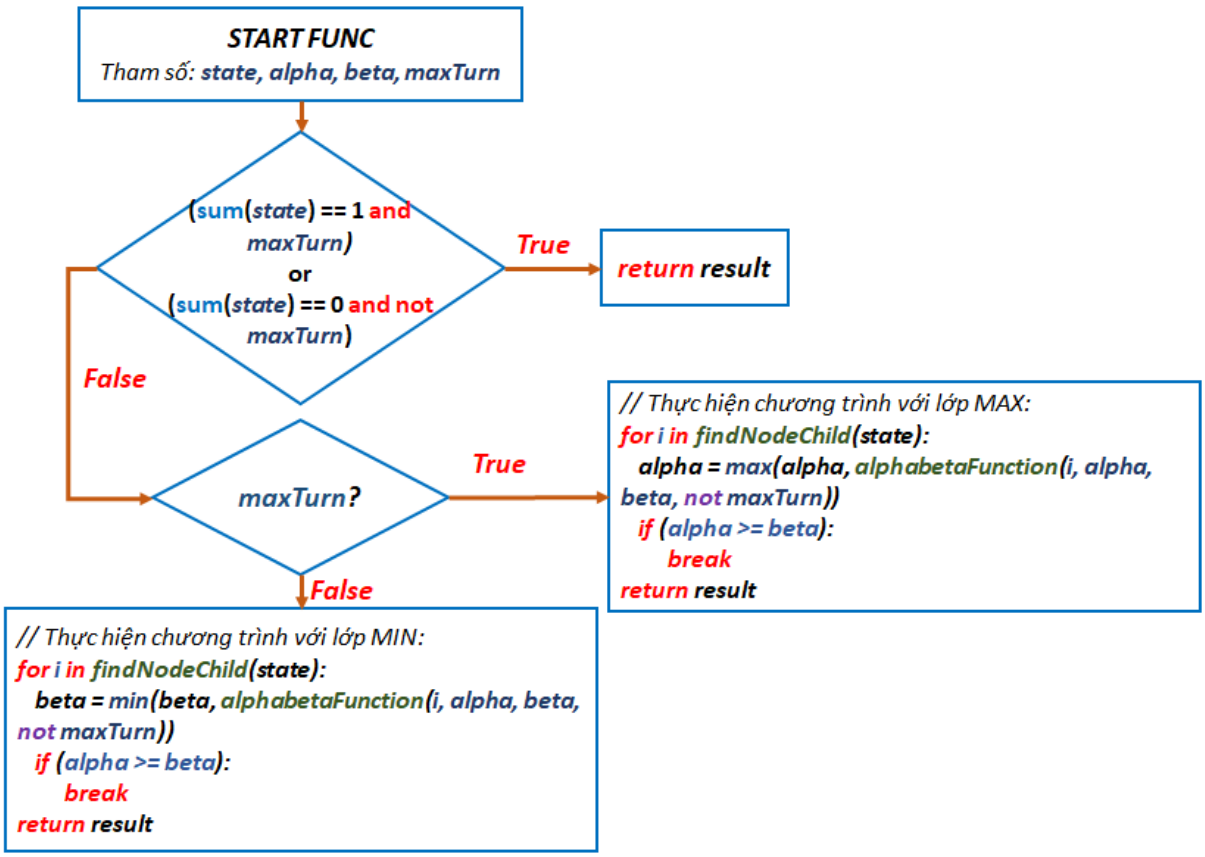
Hình 4.1: Mảng dữ liệu đầu vào cho chương trình

Ngoài ra, để thực hiện một lượt chơi, người dùng sẽ được yêu cầu nhập vào 2 tham số với tham số đầu tiên là vị trí của đống sỏi được chọn và tham số thứ hai là số lượng sỏi cần lấy. Một số phương thức được sinh ra để kiểm tra tính chính xác của dữ liệu nhập vào này.

4.2.2. Xây dựng chức năng thuật toán cắt tỉa Alpha-Beta

Quy trình của hàm cắt tỉa Alpha-Beta nhận vào 04 đối số bao gồm: $state$, $alpha$, $beta$, $maxTurn$. Quy trình kết thúc khi tổng số sỏi còn lại là 1 hoặc 0, tuy từng trường hợp sẽ cho kết quả cụ thể người chiến thắng. Nếu mức hiện tại là MAX, quy trình sẽ thực hiện xét lần lượt các node con lớp MIN thuộc node con đó và tìm giá trị lớn nhất, sau đó gán giá trị lớn nhất giữa $alpha$ và giá trị node đang xét cho $alpha$, nếu giá trị $alpha \geq beta$ thì dừng và trả về kết quả. Ngược lại, nếu mức hiện tại là MIN thì quy trình sẽ thực hiện xét lần lượt các node con MAX của node này, sau đó gán giá trị nhỏ nhất giữa $beta$ và giá trị node con đó

cho $beta$. Nếu $alpha \geq beta$ thì dừng và đưa ra giá trị. Để có thể hiểu rõ hơn, ta có sơ đồ quy trình được thể hiện tại Hình 4.2.

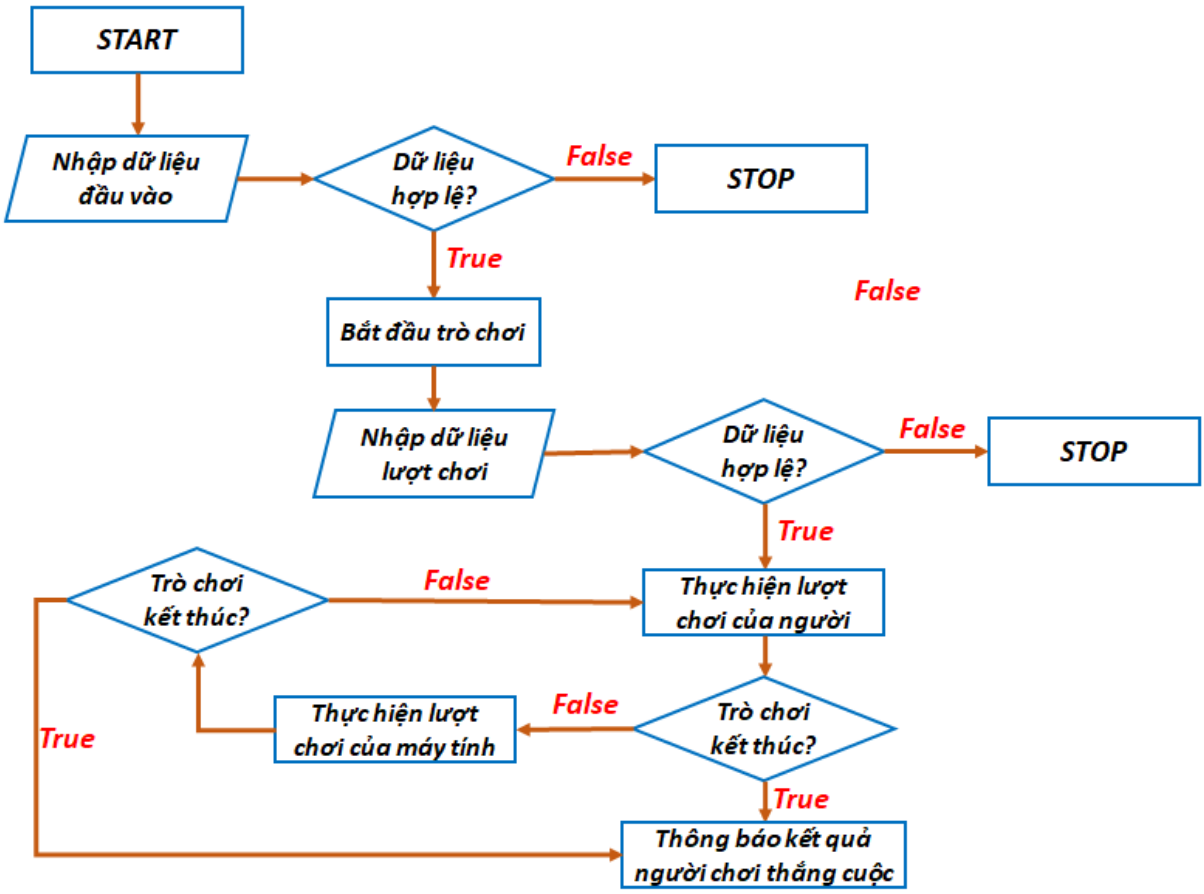


Hình 4.2: Sơ đồ chức năng Minimax sử dụng cắt tỉa alpha-beta

Ngoài ra, phương thức sử dụng thêm một phương thức ***findNodeChild()*** thực hiện tìm các node con của node hiện tại. Để tối ưu hóa lượng tính toán, phương thức sắp xếp lại số lượng trong các node để có được số lượng node con phân biệt ít nhất.

4.2.3. Xây dựng quy trình giải quyết bài toán

Luồng quy trình giải quyết bài toán sẽ thực hiện nhận dữ liệu đầu vào từ người dùng để làm các dữ liệu bài toán. Bài toán bắt đầu, người chơi sẽ đi trước, người chơi nhập dữ liệu cho bước đi của mình, sau khi chương trình xử lý xong bước đi của người chơi, chương trình sẽ sử dụng thuật toán Minimax với phương pháp cắt tỉa Alpha-beta để tính toán bước đi tối ưu nhất cho máy tính từ trạng thái hiện tại. Nếu không thể tìm được bước đi tốt nhất cho máy tính, thuật toán sẽ chọn bước đi đầu tiên cho máy tính. Trò chơi kết thúc khi lượng sỏi còn 1 hoặc chỉ còn 1 đống sỏi hoặc lượng sỏi còn lại bằng 0, sau đó chương trình sẽ thông báo kết quả. Sơ đồ về quy trình giải quyết bài toán được thể hiện tại Hình 4.3.



Hình 4.3: Sơ đồ quy trình xử lý, giải quyết bài toán

CHƯƠNG 5. VIẾT CHƯƠNG TRÌNH DEMO

Chương trình sử dụng ngôn ngữ lập trình Python phiên bản 3.11.6 và được viết trên ứng dụng VSCode. Ngoài ra, chương trình sử dụng một số thư viện Python như NumPy, Pillow, ... Vì thời gian tìm hiểu cũng như kiến thức về giao diện ứng dụng còn hạn chế, do đó, chương trình này được khởi chạy trực tiếp trên terminal và không có giao diện, người dùng sẽ sử dụng chương trình thông qua việc truyền vào các giá trị cần có trong mỗi lượt chơi và chương trình sẽ kết thúc và đưa ra kết quả thắng thua.

5.1. Mã nguồn chương trình

Mã nguồn của chương trình được đưa lên GitHub với đường dẫn: https://github.com/Huynhngockhanh52/Minimax_Alpha_beta_Demo.

Cụ thể, chương trình bao gồm hai file Python:

+ ***main.py*** là file chạy chính của chương trình trò chơi, thực hiện quy trình, đưa là các lượt chơi cụ thể. Luồng trò chơi được xác định trong phương thức **`if __name__ == "__main__"`**, bao gồm:

- Xác định dữ liệu đầu vào.
- Lặp lại, cập nhật lại dữ liệu với các bước thực hiện của người chơi và máy tính
- Nắm bắt các điều kiện chiến thắng, kết thúc và đưa ra kết quả.

+ ***alphabeta_solution.py*** là file chứa các hàm phục vụ giải thuật minimax sử dụng phương pháp cắt tỉa Alpha-Beta.

- Tìm kiếm tất cả các bước đi tiếp theo có thể có từ dữ liệu hiện tại.
- Sử dụng thuật toán minimax để chọn nước đi tốt nhất bằng cách tìm kiếm cây trò chơi bằng cách cắt tỉa

5.1.1. Mã nguồn file main.py:

```
# ================== Chương trình trò chơi NIM ======================
# Thư viện sử dụng:
import numpy
from alphabeta_solution import alphabetaFunction
```

```python
class Board(object):
    """
    Lớp Board là lớp chương trình chính, bao gồm các phương thức khởi tạo,
phương thức cập nhật trạng thái tiếp theo và phương thức tính toán của máy.
    """
    # Phương thức khởi tạo
    def __init__(self, board):
        self.board = board

    # Cập nhật lại trạng thái sau mỗi lượt chơi
    def update(self, piles, num):
        self.board[piles] -= num

    # Phương thức tính toán của người chơi là máy tính
    def computerUpdate(self):
        self.board = alphabetaFunction(self.board, -float('inf'),
float('inf'), True)[1][1]

def isValid(remove, board):
    """
    Phương thức isValid xác định dữ liệu đầu vào cho mỗi lượt đi có chính
xác hay không.
    Tham số:
        remove: Dữ liệu đầu vào cho mỗi lượt chơi
        board: Dữ liệu về số lượng sỏi hiện tại
    Trả về:
        True or False: True nếu dữ liệu phù hợp, ngược lại là False
    """
    if not remove or len(remove) != 2: return False
    if remove[0] > 0 and remove[1] > 0 and remove[1] <= len(board) and
remove[0] <= board[int(remove[1]-1)]:
        return True
    return False

# Phương thức main, xử lý chính trò chơi:
if __name__ == "__main__":
    print("================ Starting Nim Game with AI! ================")

    # Khởi tạo dữ liệu đầu vào
    ele = int(input("Số lượng đống sỏi: "))
    lis = []

    print("Số lượng sỏi có trong mỗi đống (sử dụng ENTER để ngăn cách mỗi
giá trị):")
    for _ in range(ele):
        lis.append(int(input()))
    game = Board(lis)
    print("***CHÚ Ý:***")
```

```
    print("Nhập số lượng sỏi cần loại bỏ, tiếp theo mới nhập vị trí của
đống để loại bỏ: ")
    print("Người chơi loại bỏ viên sỏi cuối cùng sẽ thua cuộc --> LOSE!")
    print("Example: Loại bỏ 3 viên sỏi từ đống thứ 2 --> 3 2, ENTER")

    player_win = True
    while True:
        print("Lượng sỏi hiện tại: %s" %(game.board))
        # player's turn
        user = str(input("Lượt người chơi: "))
        player_remove = [int(i) for i in user.strip().split(' ')]
        while not isValid(player_remove, game.board):
            print("ERROR! Giá trị đầu vào không chính xác!\nNhập lại dữ
liệu!")
            user = str(input("Lượt người chơi: "))
            player_remove = [int(i) for i in user.split(' ')]
        print(f"Bạn đã thực hiện loại bỏ {player_remove[0]} viên sỏi tại
đống thứ {player_remove[1]}!")
        game.update(int(player_remove[1] - 1), player_remove[0])
        if sum(game.board) == 0:
            player_win = False
            break
        elif sum(game.board) == 1:
            break

        print("Lượng sỏi hiện tại: %s" %(game.board))
        print("Lượt người máy AI!...")
        pre_state = game.board[:]
        game.computerUpdate()

        for index, value in enumerate(pre_state):
            if (value != game.board[index]):
                aiplayer = [int(value-game.board[index]), index+1]
        print(f"Người máy đã thực hiện loại bỏ {aiplayer[0]} viên sỏi tại
đống thứ {aiplayer[1]}!")
        if sum(game.board) == 0:
            break
        elif sum(game.board) == 1:
            player_win = False
            break
    if player_win:
        print(game.board)
        print("Bạn đã thắng --> VICTORY!!!")
    else:
        print(game.board)
        print("Bạn thua cuộc --> LOSE!!!")
```

5.1.2. Mã nguồn file alphabeta_solution.py:

```
# ===== Giải thuật Minimax sử dụng phương pháp cắt tỉa Alpha-Beta =====
def alphabetaFunction(state, alpha, beta, maxTurn):
    """
    Phương thức alphabetaFunction thực hiện xác định lượng giá cho một
trạng thái "state" hiện tại.
    """
    # Nếu đây là trạng thái của node lá hoặc là node gốc và các trường hợp
tổng số lượng token xác định bằng 0 hoặc 1 thì đưa ra kết quả
    if (sum(state) == 1 and maxTurn) or (sum(state) == 0 and not maxTurn):
return (-1, [state])
    if (sum(state) == 1 and not maxTurn) or (sum(state) == 0 and maxTurn):
return (1, [state])
    # Trường hợp với mức MAX
    if maxTurn:
        res_max = -float('inf')
        res = None
        for i in findNodeChild(state):
            val, temp = alphabetaFunction(i, alpha, beta, not maxTurn)
            if val > res_max:
                res_max = val
                res = temp
            alpha = max(alpha, val)
            # Cắt tỉa
            if alpha >= beta:
                break
        return res_max, [state] + res
    # Thực hiện với mức MIN
    else:
        res_min = float('inf')
        res = None
        for i in findNodeChild(state):
            val, temp = alphabetaFunction(i, alpha, beta, not maxTurn)
            if val < res_min:
                res_min = val
                res = temp
            beta = min(beta, val)
            if alpha >= beta:
                break
        return res_min, [state] + res
def findNodeChild(state):
    """
    Phương thức findNodeChild tìm tất cả các trạng thái con (trạng thái
tiếp theo) của trạng thái hiện tại.
    """
    visited = set()
    res = []
```

```
    for i in range(len(state)):
        for m in range(1, state[i] + 1):
            temp = list(state[:])
            temp[i] -= m
            rearranged = tuple(sorted(temp))
            if rearranged not in visited:
                res.append(temp)
                visited.add(rearranged)
    return res
```

5.1.3. Đánh giá chương trình

a. Ưu điểm

Một điểm tối ưu của chương trình là thực hiện không gian trạng thái cho máy tính sau mỗi bước đi của người chơi, nghĩa là, node gốc sẽ là trạng thái sau khi người dùng đã thực hiện bước đi. Điều này là giảm đi quá trình bùng nổ nhánh, làm phức tạp quá trình tính toán bước đi tối ưu hơn khi ta thực hiện tính toán không gian trạng thái ban đầu.

Về bản chất, người dùng khó có thể suy nghĩ và đưa ra bước đi tốt nhất đối với không gian trạng thái đó, do đó, sử dụng trạng thái sau mỗi bước đi người dùng đều tối ưu hơn. Điểm tối này đã được sử dụng trong phương thức ***computerUpdate()*** trong class ***Board*** thuộc file *main.py*.

```
class Board(object):
    # Phương thức khởi tạo
    def __init__(self, board):
        self.board = board
    # Cập nhật lại trạng thái sau mỗi lượt chơi
    def update(self, piles, num):
        self.board[piles] -= num
    # Phương thức tính toán của người chơi là máy tính
    def computerUpdate(self):
        self.board = alphabetaFunction(self.board, -float('inf'), float('inf'), True)[1][1]
```

Hình 5.1: Sử dụng lại trạng thái hiện tại của dữ liệu để tính toán

Điểm thứ hai, không mất tính tổng quát, các đống sỏi có thể hoán đổi vị trí cho nhau, do đó, để giảm sự phân nhánh, ta có thể thực hiện sắp xếp lại số lượng sỏi theo thứ tự từ bé đến lớn. Điều này đã được thực hiện trong phương thức ***findNodeChild()*** trong file *alphabeta_solution.py*

```python
def findNodeChild(state):
    visited = set()
    res = []
    for i in range(len(state)):
        for m in range(1, state[i] + 1):
            temp = list(state[:])
            temp[i] -= m
            # Nếu trạng thái đó đã tồn tại thì bỏ qua, ngược lại thêm vào res
            rearranged = tuple(sorted(temp))
            if rearranged not in visited:
                res.append(temp)
                visited.add(rearranged)
    return res
```

Hình 5.2: Giảm sự phân nhánh cây trò chơi bằng cách sắp xếp các đống sỏi

b. Nhược điểm

Tuy nhiên, chương trình vẫn tồn tại một số nhược điểm có thể kể đến như sau:

+ Chưa có giao diện người dùng: Chương trình hiện tại chỉ thực hiện chạy trên một file nhất định, chưa có giao diện để người dùng có thể thực hiện một cách trực quan, điều này là do hạn chế về kiến thức viết giao diện và cách xử lý.

+ Tồn đọng một số lỗi chưa khắc phục hết: Chương trình vẫn chưa khắc phục các lỗi dữ liệu đầu vào một cách tốt nhất, điều này một phần cũng xuất phát từ ngôn ngữ lập trình.

5.2. Demo chương trình

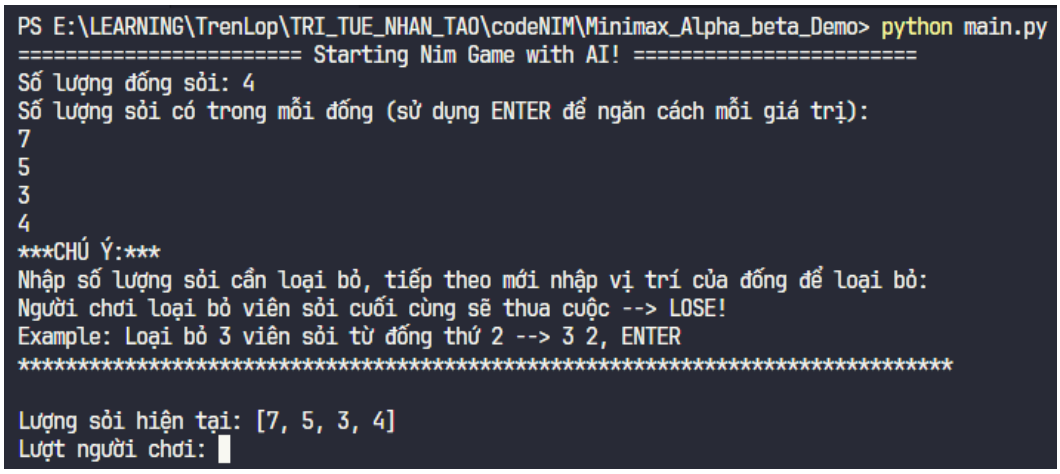
5.2.1. Sử dụng chương trình

Đầu tiên, người dùng sẽ thực hiện tải mã nguồn chương trình bằng câu lệnh: ***git clone https://github.com/Huynhngockhanh52/Minimax_Alpha_beta_Demo.git***. Sau đó, thực hiện mở thư mục đã tải và chạy lệnh: ***python main.py***. Chương trình bắt đầu khởi chạy, giao diện ban đầu sẽ như mô tả tại Hình:

```
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\Minimax_Alpha_beta_Demo> python main.py
======================== Starting Nim Game with AI! ========================
Số lượng đống sỏi:
```

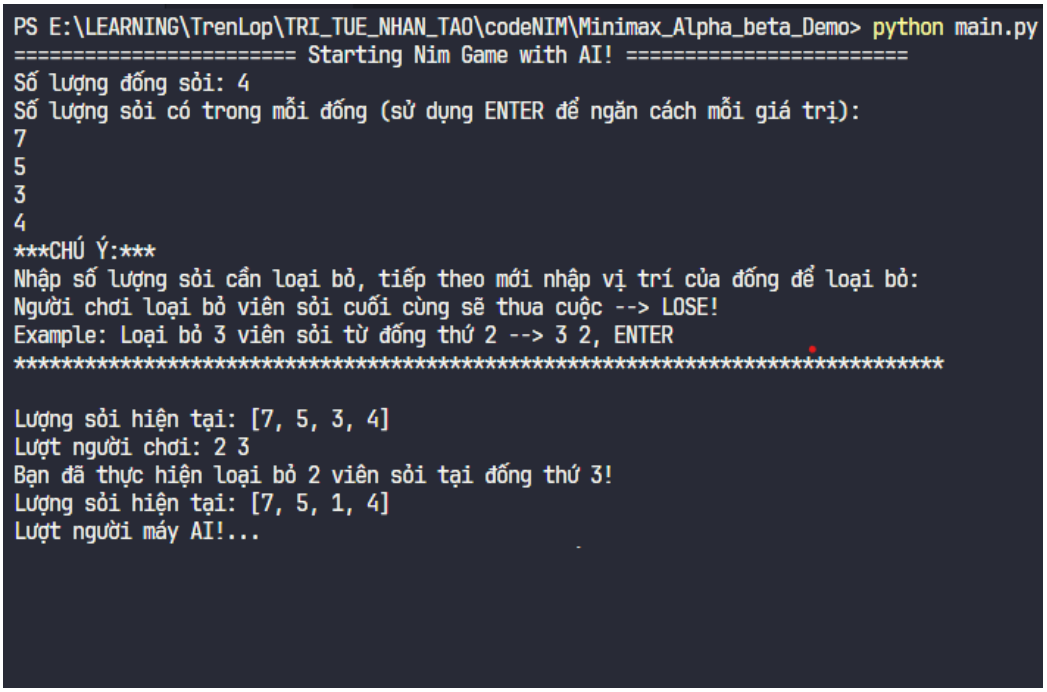
Hình 5.3: Giao diện bắt đầu khởi chạy chương trình

Tiếp theo, người dùng sẽ cài đặt các đầu vào của đề bài như số lượng đống sỏi, số lượng sỏi có trong mỗi đống. Chú ý, người dùng sẽ nhập các giá trị được phân tách bởi phím ENTER, nếu không chương trình sẽ dừng và thông báo lỗi không phù hợp dữ liệu đầu vào. Hình mô tả nhập dữ liệu đầu vào phù hợp.



Hình 5.4: Ví dụ về dữ liệu đầu vào hợp lệ

Sau khi nhập thông tin dữ liệu đầu vào phù hợp, chương trình sẽ thông báo một số chú ý về cách đưa dữ liệu lượt chơi sao cho phù hợp với chương trình. Trò chơi mặc định người chơi đầu tiên là con người, người dùng sẽ nhập số lượng sỏi cần loại bỏ và tiếp theo sẽ nhập vị trí đống cần thực hiện loại bỏ. Dữ liệu lượt chơi phù hợp thì chương trình thực hiện tính toán cho lượt chơi của người máy dựa trên giải thuật Minimax sử dụng phương pháp cắt tỉa Alpha-Beta.



Hình 5.5: Chương trình tính toán và đưa ra lượt chơi phù hợp nhất cho AI

Sau khi lựa chọn được nước đi tối ưu nhất, chương trình sẽ tiếp tục hỏi về lượt chơi của người dùng. Cứ tiếp tục như vậy đến khi tổng số lượng sỏi còn lại là 0 hoặc 1 thì chương trình sẽ kết thúc và thông báo người thắng cuộc!

```
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\Minimax_Alpha_beta_Demo> python main.py
====================== Starting Nim Game with AI! ======================
Số lượng đống sỏi: 4
Số lượng sỏi có trong mỗi đống (sử dụng ENTER để ngăn cách mỗi giá trị):
7
5
3
4
***CHÚ Ý:***
Nhập số lượng sỏi cần loại bỏ, tiếp theo mới nhập vị trí của đống để loại bỏ:
Người chơi loại bỏ viên sỏi cuối cùng sẽ thua cuộc --> LOSE!
Example: Loại bỏ 3 viên sỏi từ đống thứ 2 --> 3 2, ENTER
**************************************************************************

Lượng sỏi hiện tại: [7, 5, 3, 4]
Lượt người chơi: 2 3
Bạn đã thực hiện loại bỏ 2 viên sỏi tại đống thứ 3!
Lượng sỏi hiện tại: [7, 5, 1, 4]
Lượt người máy AI!...
Người máy đã thực hiện loại bỏ 7 viên sỏi tại đống thứ 1!
Lượng sỏi hiện tại: [0, 5, 1, 4]
Lượt người chơi: 5 2
Bạn đã thực hiện loại bỏ 5 viên sỏi tại đống thứ 2!
Lượng sỏi hiện tại: [0, 0, 1, 4]
Lượt người máy AI!...
Người máy đã thực hiện loại bỏ 4 viên sỏi tại đống thứ 4!
[0, 0, 1, 0]
Bạn thua cuộc --> LOSE!!!
```

Hình 5.6: Ví dụ một chương trình trò chơi, khi đó người dùng đã thua cuộc

5.2.2. Một số vấn đề chú ý:

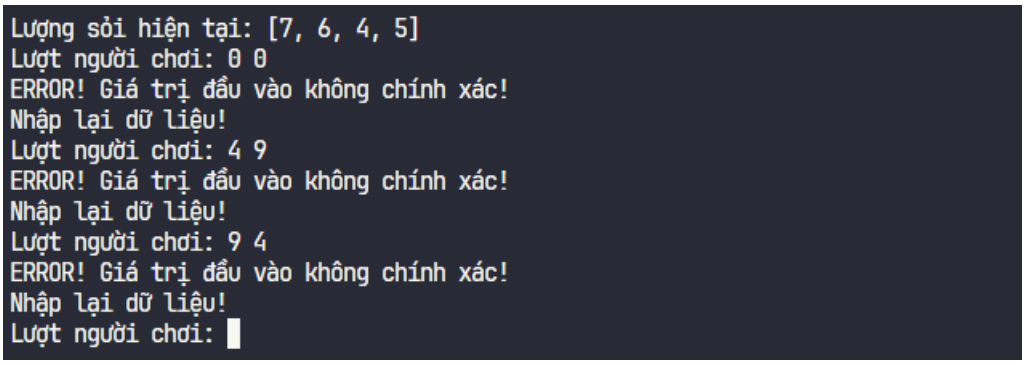
1. Các trường hợp dữ liệu không được chấp nhận:

Đầu tiên, như đã mô tả ở mục 2.1.1, nếu dữ liệu đầu vào không phù hợp (định dạng nhập vào không chính xác) chương trình sẽ dừng lại và đưa ra thông báo lỗi như ví dụ sau đây.

```
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\Minimax_Alpha_beta_Demo> python main.py
====================== Starting Nim Game with AI! ======================
Số lượng đống sỏi: 4
Số lượng sỏi có trong mỗi đống (sử dụng ENTER để ngăn cách mỗi giá trị):
4 5
Traceback (most recent call last):
  File "E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\Minimax_Alpha_beta_Demo\main.py", line
52, in <module>
    lis.append(int(input()))
               ^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: '4 5'
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\Minimax_Alpha_beta_Demo>
```

Hình 5.7: Thông báo lỗi nếu định dạng dữ liệu nhập vào không phù hợp

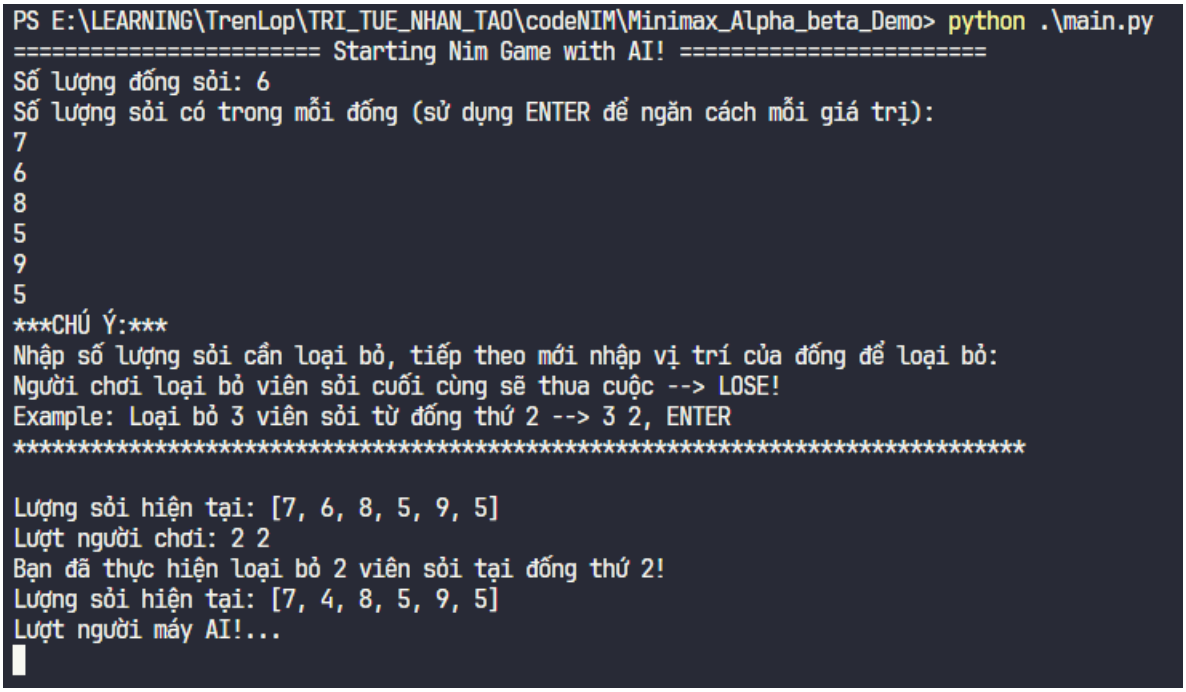
Tương tự, lỗi dữ liệu nhập vào cho từng lượt chơi người dùng sẽ được thông báo, tuy nhiên chương trình vẫn tiếp tục và chỉ yêu cầu người dùng nhập lại.



Hình 5.8: Chương trình vẫn tiếp tục nếu người dùng nhập sai dữ liệu

2. Thời gian tính toán:

Về bản chất, thuật toán minimax có độ phức tạp theo thời gian theo cấp số nhân với độ sâu của cây trò chơi khi thực hiện tìm kiếm sâu. Để có thể thực hiện mô phỏng trò chơi một cách tốt nhất, chương trình nên nhận dữ liệu đầu vào với không gian trạng thái nhỏ. Không gian trạng thái tối ưu nhất là hãy tạo một dữ liệu đầu vào với nhiều nhất là 05 đống và tổng số lượng sỏi trong tất cả đống không quá 25 (Thuật toán sẽ thực hiện hơn 60s nếu sử dụng tối đa dữ liệu).



Hình 5.9: Chương trình tính toán nước đi tối ưu cho người máy AI bị tốn nhiều thời gian do không gian trạng thái lớn

5.2.3. Mô hình trò chơi NIM giữa 2 người với nhau

Một chương trình khác là chương trình đối kháng giữa hai người với nhau. Chương trình sẽ thực hiện tìm kiếm nước đi tối ưu nhất cho mỗi người chơi, sau đó gọi ý cho người chơi về nước đi tốt nhất đó, dữ liệu ban đầu được sinh tự động và người dùng chỉ cần nhập tên người chơi trước khi bắt đầu. Tuy nhiên, chương trình không có giao diện và việc sử dụng đều thực hiện trên các câu lệnh. Có thể nói, đây cũng là một chương trình minh họa về sử dụng chiến thuật Minimax với phương pháp cắt tỉa Alpha-Beta để lựa chọn nước đi tốt nhất cho mỗi người trong trò chơi NIM. Chương trình này nằm trong thư mục ***doikhang2man***, ta thực hiện khởi chạy file ***CODE.py*** để bắt đầu chương trình.

```
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\NIM-GAME> python .\CODE.py
Enter player 1 name: Ngoc Khanh
Enter player 2 name: Van Anh
Let's look at the board now.
------------------------
Pile 1: OO
Pile 2: OOOO
------------------------
Hint: nim sum is 6.
Pick 2 stones from pile 2
Ngoc Khanh, how many stones to remove? 2
Pick a pile to remove from: 2
Let's look at the board now.
------------------------
Pile 1: OO
Pile 2: OO
------------------------
Hint: nim sum is 0.
Pick all stones from pile 1.
Van Anh, how many stones to remove? 2
Pick a pile to remove from: 1
Let's look at the board now.
------------------------
Pile 1:
Pile 2: OO
------------------------
Hint: nim sum is 2.
Pick 2 stones from pile 2
Ngoc Khanh, how many stones to remove? 2
Pick a pile to remove from: 2
Let's look at the board now.
------------------------
Pile 1:
Pile 2:
------------------------
Ngoc Khanh is the winner of this round!
Do you want to play again? Enter y for yes, anything for no: n
PS E:\LEARNING\TrenLop\TRI_TUE_NHAN_TAO\codeNIM\NIM-GAME>
```

Hình 5.10: Ví dụ về chương trình đối kháng giữa 2 người

5.3. Phương pháp cải tiến

5.3.1. Sử dụng cây tìm kiếm Monte-Carlo [16]

Các chương trình truyền thống được xây dựng trên cây tìm kiếm Minimax, Alpha-Beta với hàm đánh giá heuristic được xây dựng dựa trên tri thức của người chơi cờ. Việc thiết kế một hàm lượng giá trạng thái tốt thường rất khó, hơn nữa các cây tìm kiếm truyền thống chỉ phù hợp với những trò chơi có hệ số phân nhánh thấp. Cây tìm kiếm Monte Carlo là một hướng tiếp cận hiện đại và hiệu quả trên nhiều trò chơi có hệ số phân nhánh cao như cờ vua, cờ vây, hơn nữa, giải thuật trên có thể giải quyết được các nhược điểm mà giải thuật cắt tỉa Alpha-Beta gặp phải khi xây dựng.

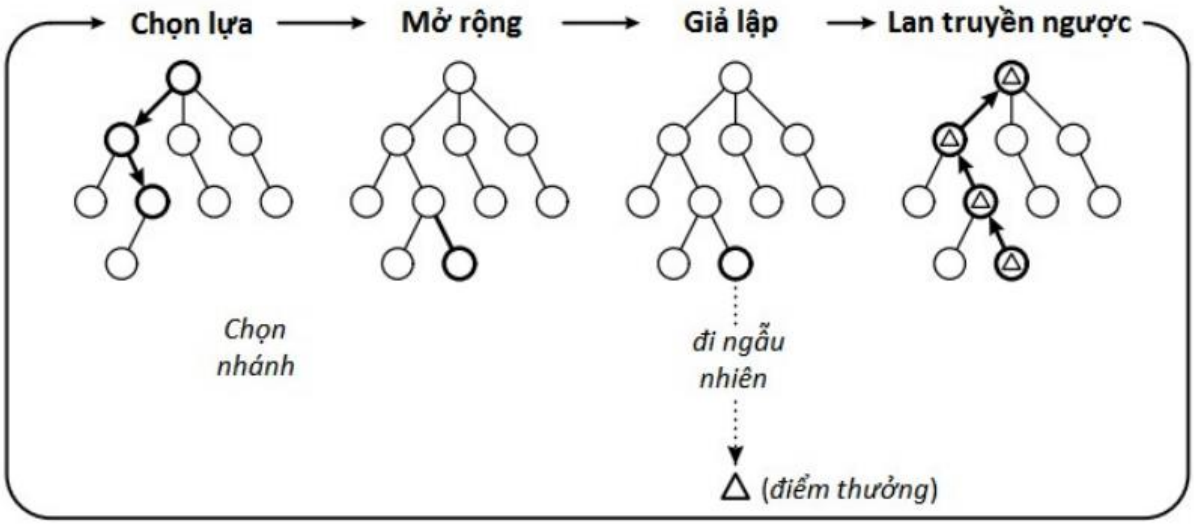
Mô hình cây tìm kiếm Monte Carlo được kết hợp từ Cây tìm kiếm, Học tăng cường và giả lập Monte Carlo. Với cách tiếp cận này, ta có thể cải tiến hiệu suất của cây tìm kiếm Monte Carlo bằng cách tìm hiểu phương pháp cải tiến Học tăng cường và cải tiến giả lập Monte Carlo. Qua đó, có thể sử dụng nhằm nâng cao hiệu suất tính toán hơn là sử dụng giải thuật Alpha-beta.

Bảng 5.1: Bảng so sánh phương pháp tìm kiếm minimax kết hợp cắt tỉa Alpha-beta với phương pháp tìm kiếm Monte Carlo

	Cây tìm kiếm truyền thống	***Cây tìm kiếm Monte Carlo***
Mô hình	Minimax + Tỉa Alpha-Beta + tri thức Heuristic.	Cây tìm kiếm + Học tăng cường + giả lập Monte Carlo
Tích cực	Phù hợp cây có hệ số phân nhánh nhỏ, và có sẵn hàm lượng giá trạng thái tốt.	Không phụ thuộc tri thức Heuristic, phù hợp trên cây có hệ số phân nhánh rất lớn.
Hạn chế	Chi phí xây dựng hàm đành giá trạng thái thường rất cao	Phải cân bằng việc khai thác và khám phá, phải có chiến thuật giả lập hiệu quả.

MCTS là một quá trình lặp đi lặp lại bốn bước trong một khoảng thời gian hữu hạn. Chọn lựa, từ một node gốc (trạng thái bàn cờ hiện hành) cho đến node

lá, vì vậy sẽ có nhiều hướng đi được mang ra đánh giá. Mở rộng, thêm một node con vào node lá của hướng được chọn trong bước chọn lựa, việc mở rộng không thực hiện trừ khi kết thúc ván cờ tại node lá. Giả lập, một ván cờ giả lập được chơi từ node mở rộng, sau đó kết quả thắng thua của ván cờ giả lập sẽ được xác định. Lan truyền ngược, kết quả thắng thua sẽ được cập nhật cho tất cả các node của hướng được chọn theo cách lan truyền ngược [17].



Hình 5.11: Sơ đồ cây tìm kiếm Monte Carlo

Điểm mấu chốt của MCTS so với các phương pháp tìm kiếm truyền thống như AlphaBeta và A* là nó không phụ thuộc vào tri thức đặc trưng của trò chơi, nói cách khác là không phụ thuộc vào hàm lượng giá trạng thái. Khi đó, MCTS có thể áp dụng vào nhiều trò chơi dạng không may rủi và thông tin trạng thái trò chơi rõ ràng sau mỗi lượt đi. Đối với các trò chơi mà khó xây đựng hàm lượng giá trạng thái tốt như cờ Vây thì việc áp dụng MCTS rất hiệu quả [18].

Phương pháp Monte-Carlo hiện nay đang rất hiệu quả cho các chương trình trò chơi hiện nay, đặc biệt là các trò chơi như cờ vua, cờ vây, … Hiện nay, thuật toán UCT và các biến thể của nó là các thuật toán chủ đạo cho các chương trình có sử dụng MCTS. MCTS là phương pháp tìm kiếm dựa theo lấy mẫu dùng giả lập ngẫu nhiên để ước lượng tỷ lệ thắng thua của một trạng thái bàn cờ nhằm tìm ra nước đi tốt nhất và để cân bằng giữa việc khám phá và khai thác của tất cả các nước đi. Như vậy, có thể thấy, ta có thể áp dụng kỹ thuật cây tìm kiếm Monte Carlo để thực hiện tìm kiếm, tính toán phương pháp tối ưu nhất cho mỗi người

chơi trong trò chơi NIM để cải thiện hiệu suất, giảm thời gian tính toán bước đi.

5.3.2. Sử dụng phương pháp học tăng cường Q-learning

Một dự án trò chơi NIM sử dụng máy tính lượng tử cơ bản để giải quyết phương pháp tìm kiếm tối ưu cho trò chơi NIM được thực hiện bởi James Weaver. Dự án sử dụng phương pháp học tăng cường với mô hình Q-learning [19].

Chiến lược thực hiện ở đây sử dụng thực hiện phép XOR các giá trị của các đống, sau đó dựa trên các nguyên tắc, lý thuyết toán học dành cho bài toán NIM [14] đã được chứng minh để tìm ra người thắng cuộc [20]. Chiến lược này sẽ được triển khai khi tự động tạo ra một mạch dựa trên các đống đối tượng còn lại mỗi khi đến lượt máy tính chơi.

Động thái của máy tính chỉ dựa trên việc đo mạch lượng tử được tạo ra. Được phép chỉ định nhiều lần chụp trong phép đo và lấy kết quả xảy ra thường xuyên nhất. Bất cứ khi nào lý thuyết toán học cho Nim chỉ ra rằng có nhiều hơn một nước đi chiến thắng, mạch lượng tử phải làm cho mỗi nước đi đó có khả năng xảy ra như nhau là kết quả của việc đo mạch. Nếu kết quả của phép đo là một nước đi vô nghĩa đối với máy tính thì máy tính sẽ tự động thua trò chơi.

Hình 5.12: Giao diện ứng dụng NIM sử dụng phương pháp Q-learning

Hình 5.13: Giao diện người chơi sử dụng

Dự án này có giao diện người dùng và cho biết người chơi nào đã thắng trò chơi. Trò chơi bắt đầu chọn số lượng bóng đèn ta muốn ở mỗi hàng trong số bốn hàng trên bảng đèn và ai sẽ phát đầu tiên trên trang cấu hình và nhấn bắt đầu. Người tắt bóng đèn cuối cùng sẽ là người thua cuộc.

Dự án này sử dụng công nghệ Flask framework để tạo giao diện và viết bằng ngôn ngữ lập trình Python. Mã nguồn của chương trình được lưu trữ trên GitHub với đường dẫn: https://github.com/ritu-thombre99/QNim-App.

KẾT LUẬN

Với yêu cầu đặt ra của bài tiểu luận là tìm hiểu, nghiên cứu chiến lược Minimax và phương pháp cắt tỉa Alpha-Beta, áp dụng chiến lược vào trò chơi NIM. Dựa trên các kiến thức đã có và kiến thức từ các bài báo quốc tế, bài tiểu luận đã đạt được những vấn đề, có thể tóm tắt như sau:

- Đã tìm hiểu tổng quan được về lý thuyết trò chơi như không gian trạng thái, xác định một số khái niệm cần chú ý, sơ bộ về các bài toán có tổng bằng 0, lý thuyết về cây trò chơi cũng như các phương pháp giải thuật được sử dụng trong việc tìm kiếm tối ưu.

- Tìm hiểu chiến lược Minimax cơ bản đầy đủ, bao gồm các nội dung như: tổng quan về giải thuật; ý tưởng, xây dựng giải thuật; các ưu, nhược điểm còn tồn đọng; đánh giá hướng phát triển của giải thuật.

- Tương tự, cũng đã tìm hiểu tổng quan về phương pháp cắt tỉa Alpha-Beta, tổng quan về ý tưởng, cách thức xây dựng giải thuật; đánh giá sử dụng giải thuật và các ứng dụng của giải thuật trong tình hình hiện nay.

- Tìm hiểu tổng quan, một cách chi tiết về bài toán NIM, các nguyên tắc được sử dụng khi chơi truyền thống; các yêu cầu đặt ra và phương hướng xây dựng chương trình áp dụng giải thuật Minimax và phương pháp cắt tỉa Alpha-beta để tìm ra lời giải tốt nhất cho bài toán trò chơi NIM giữa con người và máy tính.

- Thực hiện xây dựng được một chương trình mô phỏng đơn giản về trò chơi NIM giữa con người và máy tính. Đồng thời, cũng nêu ra được những hạn chế của chương trình cần phải khắc phục cũng như đưa ra các biện pháp, phương hướng cải tiến để giải quyết bài toán. Ngoài ra, cũng tìm hiểu về các chương trình trò chơi được sử dụng với các phương pháp khác nhau cho bài toán trên.

Mặc dù có nhiều cố gắng nhưng chắc chắn rằng các kết quả đã cài đặt được không tránh khỏi những thiếu sót và hạn chế, hy vọng rằng trong tương lai vấn đề này sẽ được nghiên cứu sâu hơn và phát triển với thuật toán được cải tiến tốt hơn, chẳng hạn chúng ta có thể đi vào nghiên cứu vấn đề song song hóa thuật toán trên.

Trên cơ sở các kết quả đã đạt được, chúng ta có thể phát triển những nghiên cứu tiếp về thuật toán Alpha- Beta song song. Hơn nữa, chúng ta có thể nghĩ đến việc triển khai những nghiên cứu về ứng dụng của những kết quả này trong các lĩnh vực khác của xã hội, đặc biệt là kinh tế, chính trị nói chung và sự phát triển của trí tuệ nhân tạo trong tương lai nói riêng.

DANH MỤC THAM KHẢO

[1] T. M. Phương, "Giáo trình Nhập môn trí tuệ nhân tạo." 2016. Accessed: Dec. 05, 2023. [Online]. Available: http://thuvienso.thanglong.edu.vn/handle/TLU/6385

[2] H. CHÍNH and V. THÔNG, "NHẬP MÔN TRÍ TUỆ NHÂN TẠO." 2016. Accessed: Dec. 05, 2023. [Online]. Available: https://www.academia.edu/download/34349251/TTNT-ptit.pdf

[3] N. V. Nam, "Trí tuệ nhân tạo," 2023, Accessed: Dec. 05, 2023. [Online]. Available: http://tailieuso.tlu.edu.vn/handle/DHTL/12932

[4] C. \DJẮNG and L. THÔNG, "\DJỀ CƯƠNG CHI TIẾT HỌC PHẦN Trí tuệ nhân tạo (Artificial Intelligence)", Accessed: Dec. 05, 2023. [Online]. Available: https://fita.vnua.edu.vn/wp-content/uploads/2013/05/TTNT.pdf

[5] \DJình Hiển Nguyễn, T. N. D. Nguyễn, and V. L. Hồ, "Tài liệu hướng dẫn thực hành Trí tuệ nhân tạo," 2015, Accessed: Dec. 05, 2023. [Online]. Available: http://ir.vnulib.edu.vn/handle/VNUHCM/8253

[6] Trình D. Đ., "Thuật toán Mini-Max trong Trí tuệ nhân tạo," w3seo. Accessed: Nov. 28, 2023. [Online]. Available: https://websitehcm.com/thuat-toan-mini-max-trong-tri-tue-nhan-tao/

[7] "[123doc] - giai-thuat-tim-kiem-minimax-va-ung-dung-trong-cac-tro-choi-co-tong-bang-khong - Website: - Studocu." Accessed: Nov. 28, 2023. [Online]. Available: https://www.studocu.com/vn/document/truong-dai-hoc-noi-vu-ha-noi/giao-trinh-kinh-te-vi-mo/123doc-giai-thuat-tim-kiem-minimax-va-ung-dung-trong-cac-tro-choi-co-tong-bang-khong/63323869

[8] "Tài liệu Chiến lược minimax và phương pháp cắt tỉa alpha beta | XEMTAILIEU," https://xemtailieu.net/. Accessed: Nov. 28, 2023. [Online]. Available: https://xemtailieu.net//tai-lieu/chien-luoc-minimax-va-phuong-phap-cat-tia-alpha-beta-2436710.html

[9] "NgTienHungg/TicTacToe: Basic Tic-Tac-Toe using Minimax algorithm." Accessed: Nov. 28, 2023. [Online]. Available: https://github.com/NgTienHungg/TicTacToe

[10] "Minimax | Brilliant Math & Science Wiki." Accessed: Nov. 28, 2023. [Online]. Available: https://brilliant.org/wiki/minimax/

[11] "1702.03401.pdf." Accessed: Dec. 05, 2023. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1702/1702.03401.pdf

[12] "đồ áN cấu trúc dữ liệu và thuật toán trò chơi nim." Accessed: Dec. 03, 2023. [Online]. Available: https://www.slideshare.net/nataliej4/n-cu-trc-d-liu-v-thut-ton-tr-chi-nim

[13] "Nim," *Wikipedia*. Oct. 31, 2023. Accessed: Dec. 03, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Nim&oldid=1182831666#Proof_of_the_winning_formula

[14] "Một số bài toán trò chơi có nội dung toán học." Accessed: Dec. 03, 2023. [Online]. Available:

https://repository.vnu.edu.vn/flowpaper/simple_document.php?subfolder=36/64/91/&doc=366491899089424322710071132949233754 35&bitsid=c15fd121-8e00-4608-83ee-fe6ab1023f4a&uid=

[15] "Cách thắng trò chơi Nim - Toán học lý thú," Diễn đàn Toán học. Accessed: Dec. 03, 2023. [Online]. Available: https://diendantoanhoc.org/topic/153451-c%C3%A1ch-th%E1%BA%AFng-tr%C3%B2-ch%C6%A1i-nim/

[16] Huy N. Q. and Chiến N. K., "PHƯƠNG PHÁP CẢI TIẾN HIỆU QUẢ CÂY TÌM KIẾM MONTE CARLO," *Tạp Chí Khoa Học Đại Học Cần Thơ*, pp. 17–24, Dec. 2015.

[17] kamil, "Monte Carlo Tree Search - beginners guide," int8.io. Accessed: Dec. 04, 2023. [Online]. Available: https://int8.io/monte-carlo-tree-search-beginners-guide/

[18] "Khảo sát và đánh giá về các hướng tiếp cận lựa chọn đặc trưng trong bài to," 123doc Cộng đồng chia sẻ, upload, upload sách, upload tài liệu. Accessed: Dec. 04, 2023. [Online]. Available: https://123docz.net/document/6919742-khao-sat-va-danh-gia-ve-cac-huong-tiep-can-lua-chon-dac-trung-trong-bai-toan-danh-co-co-do-phan-nhanh-cao.htm

[19] R. Thombre, "Ứng dụng QNim." Sep. 28, 2023. Accessed: Dec. 04, 2023. [Online]. Available: https://github.com/ritu-thombre99/QNim-App

[20] "QNim/QAMP Checkopint 1.ipynb at main · ritu-thombre99/QNim," GitHub. Accessed: Dec. 04, 2023. [Online]. Available: https://github.com/ritu-thombre99/QNim/blob/main/QAMP%20Checkopint%201.ipynb