

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**



**BÁO CÁO ĐỒ ÁN MÔN TRÍ TUỆ NHÂN TẠO  
TRÒ CHƠI Ô ĂN QUAN**

<u>Lớp:</u>	<b>CS106.G22</b>
<u>GVLТ:</u>	<b>Huỳnh Thị Thanh Thương</b>
<u>GVTH:</u>	<b>Phạm Nguyễn Trường An</b>
<u>Nhóm sinh viên thực hiện:</u>	<b>1. Hoàng Thị Thượg – 14520929</b> <b>2. Đồng Minh Trường – 14521027</b> <b>3. Nguyễn Xuân Vũ – 14521103</b>

TP. Hồ Chí Minh, tháng 07 năm 2016.

## Mục lục

<b>CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN.....</b>	<b>5</b>
1.1 Nguồn gốc trò chơi.....	5
1.2 Mô tả trò chơi.....	6
<b>CHƯƠNG 2: CƠ SỞ LÝ THUYẾT .....</b>	<b>10</b>
2.1 TÌM KIẾM CÓ ĐỐI THỦ .....	10
2.2 GIẢI THUẬT MINIMAX.....	12
2.3 GIẢI THUẬT CẮT TỈA ALPHA – BETA.....	12
2.4 Kỹ thuật lượng giá.....	13
<b>CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ .....</b>	<b>14</b>
3.1 Phân tích bài toán .....	14
3.2 Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán .....	19
3.2.1 Mô hình hóa bài toán:.....	19
3.2.2 Biểu diễn trạng thái:.....	19
3.2.2 Không gian trạng thái:.....	19
3.2.3 Cấu trúc dữ liệu: .....	20
3.3 Các vấn đề và thuật giải .....	21
3.3.1 Xây dựng hàm Minimax có cắt tỉa.....	21
3.3.2 Minimax có độ sâu cố định bằng 2 .....	25
3.3.3 Hàm AI theo nguyên lý tham lam.....	27
3.3.4 Các hàm xử lý cơ bản trong game .....	27
3.4 Ví dụ minh họa thuật toán .....	29
<b>CHƯƠNG 4: ỨNG DỤNG .....</b>	<b>33</b>
4.1 Giới thiệu chương trình ứng dụng .....	33
4.2 Cài đặt.....	35
4.2.1 Ngôn ngữ và công cụ sử dụng.....	35
4.2.2 Các hàm các đoạn code chính .....	35
4.3 Kết quả chạy chương trình .....	39
4.2.1 Kết quả thử nghiệm.....	39
<b>CHƯƠNG 5: KẾT LUẬN .....</b>	<b>41</b>
5.1 Kết quả đạt được.....	41
5.2 Hạn chế .....	41

5.3 Hướng phát triển.....	41
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>42</b>

### Bảng phân công công việc

MSSV	Họ tên	Công việc thực hiện	Mức độ hoàn thành	Nhận xét
14520929	Hoàng Thị Thượng	<ul style="list-style-type: none"> <li>- Lên ý tưởng phân công công việc.</li> <li>- Thiết kế và code giao diện</li> <li>- Code các chức năng của giao diện như trở về Menu, tiến lùi, chạy nhạc</li> <li>- Viết báo cáo</li> <li>- Code phụ và đóng góp phần AI</li> <li>- Test game</li> </ul>	100	- Hoàn thành tốt tham gia họp đầy đủ
14521103	Nguyễn Xuân Vũ	<ul style="list-style-type: none"> <li>- Thảo luận đưa ra ý tưởng</li> <li>- Đóng góp ý kiến về thiết kế giao diện</li> <li>- Code chính</li> <li>- Code xử lý nước đi</li> <li>- Hỗ trợ các bạn về code</li> <li>- Code chính AI Minimax độ sâu cố định</li> </ul>	100	Hoàn thành tốt tham gia họp đầy đủ

14521027	Đổng Minh Trường	<ul style="list-style-type: none"> <li>- Thảo luận đóng góp ý tưởng về giao diện.</li> <li>- Code chính về AI Minimax có độ sâu d và cắt tỉa.</li> <li>- Test game</li> <li>- Đóng góp ý kiến báo cáo</li> </ul>	100	<ul style="list-style-type: none"> <li>- Hoàn thành tốt tham gia họp đầy đủ</li> </ul>
----------	---------------------	--	-----	--

## CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN

### 1.1 NGUỒN GỐC TRÒ CHƠI



Ô ăn quan, hay còn gọi tắt là ăn quan hoặc ô quan là một trò chơi dân gian của trẻ em Việt Nam mà chủ yếu là các bé gái. Đây là trò chơi có tính chất chiến thuật thường dành cho hai hoặc ba người chơi và có thể sử dụng các vật liệu đa dạng, dễ kiếm để chuẩn bị cho trò chơi.

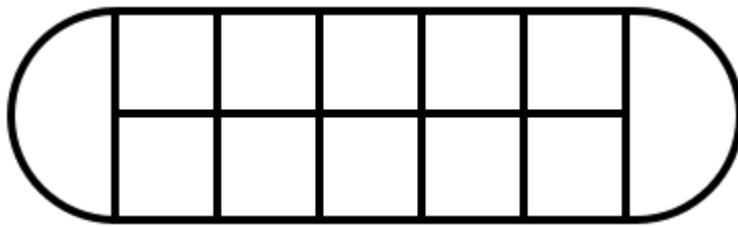
Hiện chưa rõ nguồn gốc cũng như thời điểm bắt đầu nhưng chắc chắn rằng Ô ăn quan đã có ở Việt Nam từ rất lâu đời, có thể nó được lấy cảm hứng từ những cánh đồng lúa nước ở nơi đây. Những câu truyện lưu truyền về Mạc Hiến Tích (chưa rõ năm sinh, năm mất), đỗ Trạng nguyên năm 1086 nói rằng ông đã có một tác phẩm bàn về các phép tính trong trò chơi Ô ăn quan và đề cập đến số ăn (số âm) của ô trống xuất hiện trong khi chơi. Ô ăn quan đã từng phổ biến ở khắp ba miền Bắc, Trung, Nam của Việt Nam nhưng những năm gần đây chỉ còn được rất ít trẻ em chơi. Bảo tàng Dân tộc học Việt Nam có trưng bày, giới thiệu và hướng dẫn trò chơi này.

Theo các nhà nghiên cứu, ô ăn quan thuộc họ trò chơi mancala, tiếng Ả Rập là manqala hoặc minqala (khi phát âm, trọng âm rơi vào âm tiết đầu ở Syria và âm tiết thứ hai ở (Ai Cập) có nguồn gốc từ động từ naqala có nghĩa là di chuyển. Bàn

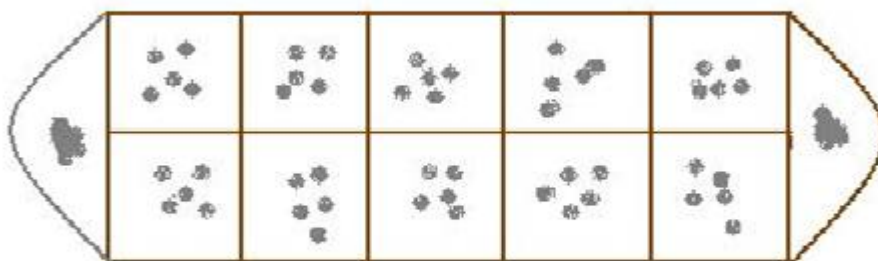
chơi mancala đã hiện diện ở Ai Cập từ thời kỳ Đế chế (khoảng 1580 - 1150 TCN). Tuy nhiên còn một khoảng trống giữa lần xuất hiện này với sự tồn tại của mancala ở Ceylon(Srilanka) những năm đầu Công nguyên và ở Ả Rập trước thời Muhammad.(2)

## 1.2 MÔ TẢ TRÒ CHƠI

### Chuẩn bị



Bàn chơi: bàn chơi Ô ăn quan kẻ trên một mặt bằng tương đối phẳng có kích thước linh hoạt miễn là có thể chia ra đủ số ô cần thiết để chứa quân đồng thời không quá lớn để thuận tiện cho việc di chuyển quân, vì thế có thể được tạo ra trên nền đất, vỉa hè, trên miếng gỗ phẳng.... 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 chơi: gồm hai loại quan và dân, được làm hoặc thu thập từ nhiều chất liệu có hình thể ổn định, kích thước vừa phải để người chơi có thể cầm, nắm nhiều quân bằng một bàn tay khi chơi. Quan có kích thước lớn hơn dân đáng kể cho dễ phân biệt với nhau. Quân chơi có thể là những viên sỏi, gạch, đá, hạt của một số loại quả... hoặc được sản xuất công nghiệp từ vật liệu cứng mà phổ biến là nhựa. Số lượng quan luôn là 2 còn dân có số lượng tùy theo luật chơi nhưng phổ biến nhất là 50.

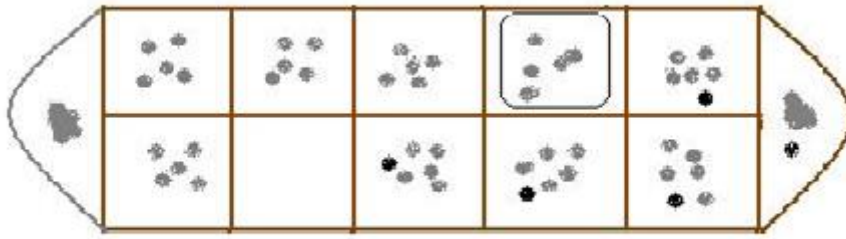
Bố trí quân chơi: quan được đặt trong hai ô hình bán nguyệt hoặc cánh cung, mỗi ô một quân, dân được bố trí vào các ô vuông với số quân đều nhau, mỗi ô 5 dân. Trường hợp không muốn hoặc không thể tìm kiếm được quan phù hợp thì có thể thay quan bằng cách đặt số lượng dân quy đổi vào ô quan.

Người chơi: thường gồm hai người chơi, mỗi người ở phía ngoài cạnh dài hơn của hình chữ nhật và những ô vuông bên nào thuộc quyền kiểm soát của người chơi bên đó.

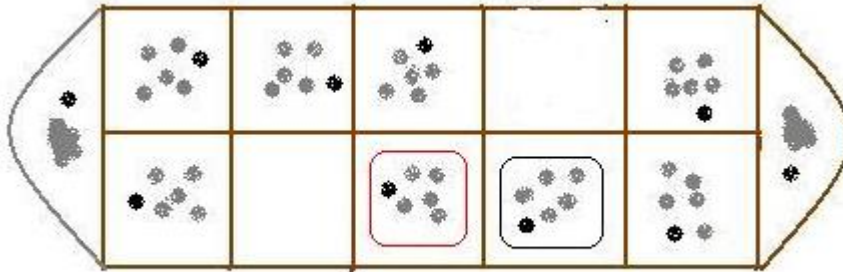
## **Luật chơi**

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. Tùy theo luật chơi từng địa phương hoặc thỏa thuận giữa hai người chơi nhưng phổ biến là 1 quan được quy đổi bằng 10 dân hoặc 5 dân.

Di chuyển quân: từng người chơi khi đến lượt của mình sẽ di chuyển dân theo phương án để có thể ăn được càng nhiều dân và quan hơn đối phương càng tốt. Người thực hiện lượt đi đầu tiên thường được xác định bằng cách oẳn tù tì hay thỏa thuận. 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. Một ô có nhiều dân thường được trẻ em gọi là ô nhà giàu, rất nhiều dân thì gọi là giàu sụ. Người chơi có thể bằng kinh nghiệm hoặc tính toán phương án nhằm nuôi ô nhà giàu rồi mới ăn để được nhiều điểm.

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 5 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; tình huống này được gọi là hết quan, tàn dân, thu quân, kéo về hay hết quan, tàn dân, thu quân, bán ruộng. 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.

### **Biến thể**

Số dân ở mỗi ô vuông là 10 và/hoặc ở ô quan ngoài quan còn có thêm 20 hay 30 dân. Khi quân cuối cùng đã được rải xuống, nếu ô liền sau đó là ô quan thì người chơi cũng mất lượt ngay dù ô đó có chứa quân hay không.

Khi đến lượt đi người chơi có thể tính toán phương án đi của mình trong một khoảng thời gian hợp lý hoặc phải đi ngay mà không được phép tính toán.

Ô ăn quan cũng có thể được chơi với 3 hoặc 4 người chơi trong đó cách di chuyển quân, thể thức tính điểm cũng giống như khi chơi hai người nhưng bàn chơi được thiết kế khác đi cho phù hợp.

## CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

### 2.1 TÌM KIẾM CÓ ĐỐI THỦ

Tìm kiếm luôn là thao tác nền móng cho rất nhiều tác vụ tính toán. Các bài toán tìm kiếm bao gồm việc tìm cách tốt nhất để thu được thông tin cần cho một quyết định. Mỗi bài toán bất kỳ đều chứa trong đó một bài toán con tìm kiếm theo một chiều hướng nào đó, các tình huống tồn tại ở đó việc tìm kiếm cần phải xử lý là: kiểm tra các tài khoản, thanh tra và điều khiển chất lượng...

Một cách tổng quát, tìm kiếm có thể hiểu là tìm một hoặc một số đối tượng thỏa mãn những đòi hỏi nào đó trong tập hợp rộng lớn các đối tượng.

Chúng ta có thể kể ra rất nhiều vấn đề mà việc giải quyết nó được quy về vấn đề tìm kiếm. Trong cờ Ô ăn quan cũng vậy, vấn đề tìm kiếm được thể hiện ở chỗ, trong số rất nhiều nước đi có thể thực hiện, người chơi phải tìm ra nước đi có ưu thế thắng.

Khi muốn giải quyết một vấn đề nào đó bằng tìm kiếm, trước hết ta phải xác định không gian tìm kiếm. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm để tìm ra trong đó đối tượng yêu cầu. Một cách chung nhất, nhiều bài toán phức tạp đều có dạng “*tìm đường đi trong đồ thị*” hay nói một cách hình thức hơn là “*xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến đỉnh nào đó*”. Một phát biểu khác thường gặp của dạng bài toán này là:

Cho trước hai trạng thái  $T_0$  và  $T_G$  hãy xây dựng chuỗi trạng thái  $T_0, T_1, T_2, \dots, T_{n-1}, T_n = T_G$  sao cho :

$$\sum_{i=1}^n Pcost(T_{i-1}, T_i) \text{ thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).}$$

Trong cờ Ô ăn quan, mỗi cách bố trí quân cờ trên bàn cờ là một trạng thái. Trạng thái ban đầu là sự sắp xếp các quân cờ lúc đầu cuộc chơi. Mỗi bước đi hợp lệ là một phép chuyển trạng thái, nó biến đổi một trạng thái trên bàn cờ thành một trạng thái khác.

Như vậy, ta xác định được các yếu tố:

- Trạng thái bắt đầu (TTBĐ): Trạng thái sơ khởi của bàn cờ.
- Trạng thái kết thúc (TTKT): Trạng thái khi ván cờ kết thúc (gồm nhiều trạng thái thỏa điều kiện)
- Tập hợp các toán tử chuyển trạng thái: Tập những nước đi hợp lệ tại một trạng thái.

Tập hợp tất cả các trạng thái có thể đạt tới từ trạng thái ban đầu bằng cách áp dụng một dãy phép chuyển trạng thái, lập thành không gian trạng thái của bài toán. Việc tìm kiếm của bài toán được quy về việc tìm đường đi từ trạng thái đầu đến trạng thái đích. Có nhiều kỹ thuật tìm kiếm khác nhau để giải quyết các bài toán tìm kiếm. Tuy nhiên với mỗi bài toán tùy theo đặc điểm, cách tổ chức dữ liệu mà ta có thể lựa chọn và áp dụng kỹ thuật phù hợp và hiệu quả.

Như chúng ta đã biết, trò chơi Ô ăn quan cũng là một trò chơi đối kháng. Cụ thể trò chơi này thuộc dạng trò chơi có tổng-bằng-không với hai người chơi (Two players, Zero-sum-game). Tức là, không thể có trường hợp cả hai bên đều thắng hoặc đều thua. Nếu một bên thắng thì bên kia nhất định thua và ngược lại. Ta có thể thấy những ví dụ điển hình của trò chơi có tổng-bằng-không trong thể thao, hay kinh doanh. Ví dụ, trong một giải bóng đá, tổng số trận thua luôn bằng tổng số trận thắng; trong đầu tư kinh doanh, số tiền thua lỗ của nhà đầu tư này sẽ là tiền lãi của nhà đầu tư khác.

Ở các trò chơi thể loại này (cờ vua, cờ vây), có một cây trò chơi bao gồm tất cả các nước đi có thể có của cả hai đấu thủ và các cấu hình bàn cờ là kết quả của nước đi đó. Ta có thể tìm kiếm trên cây này để có chiến lược chơi hiệu quả. Trong trò chơi này phải tính đến mọi nước đi đối thủ có thể sử dụng.

Đặc điểm của các trò chơi thể loại này:

- Có hai đấu thủ, mỗi người chỉ được đi một nước khi tới lượt.
- Các đấu thủ đều biết mọi thông tin về tình trạng trận đấu.
- Trận đấu không kéo dài vô tận, phải diễn ra hòa, hoặc một bên thắng và bên kia thua.

Vì thế ta áp dụng tìm kiếm đối kháng (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).

Mặt khác nếu viết chương trình để người chơi với người thì chỉ cần xây dựng các tập luật chơi để người chơi không phạm quy và kết thúc khi có người thắng. Tuy nhiên như vậy lại không phù hợp với đề tài này trong báo cáo này. Vì vậy nhóm sẽ viết chương trình cho Người chơi với Máy, trong trường hợp này ta phải tính toán như thế nào để khả năng máy tính thắng cao hơn người.

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 chiến lược Minimax và thuật toán cải tiến Alpha-beta để cài đặt cho máy tính chơi.

## 2.2 GIẢI THUẬT MINIMAX

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).

- Hai đối thủ trong trò chơi có tên là MAX và MIN.
  - + Max: biểu diễn cho mục đích của đối thủ này là làm lớn tối đa lợi thế của mình.
  - + Min: biểu diễn cho mục đích của đối thủ này là làm nhỏ tối đa lợi thế của đối phương.

Trên cây tìm kiếm sẽ phân lớp thành các lớp Max và Min.

Minimax có thể chọn được nước đi tốt nhất theo hàm mục tiêu nếu hàm lượng giá là tốt nhất. Tuy nhiên, Minimax gặp phải một vấn đề đó là không gian tìm kiếm bùng nổ với hàm mũ khi độ sâu tăng dần bởi với Minimax ta cần thực hiện chiến lược tìm kiếm vét cạn. Để giải quyết vấn đề này cần tìm kiếm phương pháp cắt bớt các nhánh tìm kiếm không cải thiện được kết quả.

## 2.3 GIẢI THUẬT CẮT TỈA ALPHA – BETA.

Nếu một nhánh tìm kiếm (nhánh chứa nước đi) không thể cải thiện được hiệu quả của hàm tiện ích mà chúng ta đã có thì không cần xét đến nữa. Việc cắt tỉa nhánh tìm kiếm vô ích không làm ảnh hưởng đến kết quả cuối cùng của hàm tiện ích.

Giải thuật cắt tỉa Alpha-Beta giảm được nhiều nhánh tìm kiếm vô ích, giúp tăng tốc độ tìm kiếm, giảm chi phí thực thi và tăng độ sâu tìm kiếm, từ đó giá trị của hàm tiện ích.

## **2.4 KỸ THUẬT LƯỢNG GIÁ.**

Hàm lượng giá là quan trọng nhất đối với giải thuật Minimax nói chung và giải thuật cắt tỉa Alpha – Beta nói riêng. Lượng giá tốt thì giá trị hàm tiện ích trả về càng chuẩn, việc cắt tỉa được thực hiện sớm giúp giảm được nhiều chi phí tìm kiếm.

## CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ

### 3.1 PHÂN TÍCH BÀI TOÁN

#### 3.1.1 Ý tưởng giải quyết

- Sử dụng chiến lược Minimax, xây dựng cây trạng thái từ trạng thái hiện hành.
- Sử dụng hàm heuristic để đánh giá trạng thái hiện tại của bài toán (chế độ 1 người chơi)
- Đồng thời sử dụng cắt tỉa  $\alpha, \beta$  để tối giản cây trạng thái.
- Xây dựng hàm heuristic để đánh giá giá trị thắng thua tại các nút lá: dựa vào hiệu số giữa điểm người và điểm máy. Một bên sẽ tối đa hiệu số đó, và bên kia sẽ tối thiểu hiệu số đó.
- Bên cạnh đó sử dụng hàm tìm kiếm theo nguyên lý tham lam là duyệt hết tất cả các trạng thái. Trạng thái nào ăn được nhiều quân nhất thì chọn bước đi đó. Nguyên lý này để làm AI của cấp độ 1.
- Xây dựng các thành hàm phần: xét nước đi hợp lệ hay không, xét đến lượt của ai đi, xét thắng thua, hàm di chuyển, xét trạng thái kết thúc game, thay đổi các hiển thị quân trong Ô cờ, undo, redo, ...

#### 3.1.2 Áp dụng cơ sở lý thuyết.

Như đã trình bày ở chương 2, ta sẽ xây dựng cây tìm kiếm, với mỗi nút là một trạng thái có thể có của bàn cờ. Nút gốc biểu diễn cho trạng thái bắt đầu của cuộc chơi. Mỗi nút 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 nút  $n$  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  $x$ .

Áp dụng chiến lược Minimax, Hai đối thủ trong một trò chơi được gọi là MIN và MAX. MAX đại diện cho đối thủ quyết giành thắng lợi hay cố gắng tối đa hóa ưu thế của mình. Ngược lại MIN là đối thủ cố gắng tối thiểu hóa điểm số của MAX. Ở đây, dựa trên ý tưởng là tối đa và tối thiểu hiệu số giữa số quân ăn được của MAX và số quân ăn được của MIN.

Chiến lược Minimax thể hiện qua quy tắc định giá trị cho các nút trên cây trò chơi như sau:

- Nếu nút là nút lá gán cho nút đó một giá trị để phản ánh trạng thái thắng thua hay hòa của các đấu thủ.
- Sử dụng giá trị của các nút lá để xác định giá trị của các nút ở các mức trên trong cây trò chơi theo quy tắc:
  - + Nút thuộc lớp MAX thì gán cho nó giá trị lớn nhất của các nút con của nút đó.
  - + Nút thuộc lớp MIN thì gán cho nó giá trị nhỏ nhất của các nút con của nút 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. 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.

Như chúng ta đã biết, không phải lúc nào cũng đến được các nút lá mang trạng thái kết thúc, vì độ sâu của cây trò chơi là rất lớn. Vì vậy, chúng ta chỉ thực hiện tìm kiếm đến độ sâu cố định nào đó. Độ sâu càng lớn, hàm tìm kiếm càng gần giá trị tối ưu, cũng có nghĩa là “trình độ suy nghĩ” của máy càng cao!. Tuy nhiên sẽ xảy ra tình trạng tốn quá nhiều bộ nhớ cũng như là thời gian tìm kiếm. Vì vậy, chúng ta chỉ thực hiện tìm kiếm đến độ sâu cố định nào đó. Ở đây, theo sự đánh giá của nhóm, độ sâu bằng 2 tương đối phù hợp.

Ta xây dựng giải thuật Minimax như sau:

```

Int Minimax([8,8]A, depth, x)
{
If(depth = 0) or ([8,8]A là trạng thái kết thúc) then return heuristic([8,8]A,x) (hàm heuristic đánh giá trạng thái A khi đến lượt đi của x)
Else
{
    Int best = -1000;

```

```

Test([8,8]A,x); (sinh ra các nước x có thể đi tại trạng thái A)
While còn lấy nước đi X
Do
{
  [8,8] B (trạng thái sau khi đi nước đi X)
  Int value = -Minimax([8,8]B, depth – 1, -x) (gọi đệ quy và đổi dấu)
  If (value > best) then best = value;
}
}
Return best;
}

```

Ngoài ra, hàm *Heuristic* để đánh giá độ tốt của một trạng thái trên bàn cờ, của một người chơi nào đó (MIN hay MAX, 1 hay -1). Đây là hàm quyết định, chi phối hầu hết thời gian và chỉ làm việc trên các nút lá, vì vậy việc thăm các nút không phải các nút lá có thể bỏ qua.

Thuật toán Minimax thăm toàn bộ cây trò chơi bằng việc dùng chiến lược tìm kiếm theo chiều sâu. Nên độ phức tạp của thuật toán tương ứng trực tiếp với kích thước không gian tìm kiếm. Ta có thể tiết kiệm được nhiều thời gian bằng việc dùng thuật toán cải tiến Alpha – beta, thuật toán này không thăm tất cả các nút lá mà vẫn cho kết quả đúng với thuật toán Minimax phía trên.

### **Giải thuật alpha, beta:**

Giải thuật alpha, beta là một cải tiến của giải thuật Minimax nhằm tĩa nhánh của cây trò chơi, làm giảm số lượng nút phải sinh và lượng giá, do đó có thể tăng độ sâu của cây tìm kiếm.

Ý tưởng của tìm kiếm Alpha-beta rất đơn giản: Có hai giá trị, gọi là alpha và beta được tạo ra trong quá trình tìm kiếm:

- Giá trị alpha liên quan với các nút MAX và có khuynh hướng không bao giờ giảm.
- Ngược lại giá trị beta liên quan đến các nút MIN và có khuynh hướng không bao giờ tăng.

Để bắt đầu thuật toán tìm kiếm alpha, beta, ta 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 thuyết tất cả đều là nút MIN. Giá trị tối đa của các nút MIN này sẽ được truyền ngược lên cho nút cha mẹ (là một nút MAX). Sau đó giá trị này được gán cho ông bà của các nút MIN như là một giá trị beta



kết thúc tốt nhất. Tiếp theo thuật toán này sẽ đi xuống các nút cháu khác và kết thúc việc tìm kiếm đối với nút cha mẹ 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 nút cháu của một nút MAX.

Từ đó ta có luật cắt tỉa dựa trên các giá trị alpha và beta:

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

Từ ý tưởng và luật trên ta sẽ xây dựng hàm cải tiến từ hàm Minimax phía trên có cắt tỉa alpha, beta.

```

Int AB([8,8]A, depth, x, alpha, beta)
{
    If(depth = 0) or ([8,8]A là trạng thái kết thúc) then return heuristic([8,8]A,x)
    (hàm heuristic đánh giá trạng thái A khi đến lượt đi của x)

    Else
        {
            Int best = -1000;

            Test([8,8]A,x); (sinh ra các nước x có thể đi tại trạng thái A)

            While (còn lấy nước đi X) and (best < beta) (nếu best >= beta thì không thực hiện)

            Do
                {
                    If(best > alpha) then alpha = best;

                    [8,8] B (trạng thái sau khi đi nước đi X)

                    Int value = -AB([8,8]B, depth - 1, -x, -beta, -alpha) (gọi đệ quy và đổi dấu)
                }
        }
    }

```

```

        If (value > best) then best = value;
    }
}
Return best;
}

```

Ví dụ: Khi ta gọi hàm này với độ sâu  $d$ :  $AB(-1000,1000,d)$ ;

Ta thấy cứ mỗi khi  $best \geq \beta$  thì thuật toán không thực hiện tiếp vòng lặp, có nghĩa là nó không thực hiện việc mở rộng những nhánh còn lại. Các nhánh đó đã bị cắt tía và do đó ta sẽ tiết kiệm được thời gian. Việc cắt tía này hoàn toàn an toàn với những lý do trên. Ta thấy ở phần ý tưởng,  $\alpha$  chỉ liên quan đến MAX và  $\beta$  chỉ liên quan đến MIN, nên khi hạ xuống một độ sâu, 2 giá trị  $\beta$  và  $\alpha$  đã đổi chỗ cho nhau (và đổi cả dấu). 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  $\beta$  ngày càng thu hẹp và dẫn đến các nhánh cây có giá trị nằm ngoài khoảng này nhanh chóng bị cắt bỏ.

Thuật giải Alpha-beta nói chung giúp ta tiết kiệm nhiều thời gian so với Minimax mà vẫn đảm bảo kết quả tìm kiếm chính xác. Tuy nhiên lượng tiết kiệm này không ổn định – phụ thuộc vào số nút mà nó cắt bỏ. Trong trường hợp xấu nhất, giải thuật không cắt được một nhánh nào và phải xét số nút lá bằng với giải thuật Minimax. Vì thế ta cần đẩy mạnh việc cắt bỏ nhờ đẩy nhanh sự thu hẹp của hai giá trị  $\alpha$  và  $\beta$ . Khoảng cách này được thu hẹp một bước khi gặp một giá trị mới tốt hơn giá trị cũ. Như vậy, nếu gặp giá trị tốt nhất càng sớm, thì số nhánh được cắt bỏ càng nhiều.

### 3.1.3 Phát biểu bài toán một cách hình thức:

- Không gian trạng thái: toàn bộ trạng thái có thể có của bàn cờ.
- Trạng thái đầu: bàn cờ có 10 ô quân và 2 ô quan, mỗi ô quân có 5 quân cờ, ô quan có 1 quan và không có quân (dân) nào rải vào.
- Trạng thái cuối: Có 2 trường hợp là:

+ Khi 2 quan đồng thời bị ăn hết không còn dân nào trong ô quan.

+ Khi các ô thuộc một bên nắm giữ hết sỏi mà trong kho của mình không đủ 5 sỏi để rải đều cho 5 ô của mình.

- Toán tử: cách bốc ô quân và chiều rải quân trong bàn cờ một cách hợp lý
- Vấn đề tìm kiếm trong KGTT: Tìm kiếm trạng thái sao cho có lợi nhất và ngắn nhất.

### 3.2 CẤU TRÚC DỮ LIỆU VÀ CÁCH BIỂU DIỄN CÁC TRẠNG THÁI CỦA BÀI TOÁN

#### 3.2.1 Mô hình hóa bài toán:

Cho cây tìm kiếm, với mỗi nút là một trạng thái của bàn cờ. Nút con mang trạng thái có thể sinh ra từ trạng thái tại nút cha. Từ một nút, tìm nút con thích hợp để để tìm được nút đích có trạng thái mong muốn.

#### 3.2.2 Biểu diễn trạng thái:

- Mỗi cách bố trí quân cờ trên bàn cờ là một trạng thái. Sử dụng mảng BanCo[] kiểu nguyên 12 phần tử. Trong đó các ô quân của máy là vị trí từ 1 đến 5 và của người là 7 đến 11, quan là vị trí 0 và 6.
- Mỗi bước đi hợp lệ là 1 toán tử chuyển trạng thái.

#### 3.2.2 Không gian trạng thái:

Trang thái đầu: Gồm 1 bàn cờ. Trong bàn cờ gồm có 2 quan và mỗi ô dân thì có 5 quân.

10	5	5	5	5	5	10
	5	5	5	5	5	

Trang thái kết thúc: Khi 2 quan đồng thời bị ăn hết không còn dân nào trong ô quan

0	0	0	0	0	0	0
	0	0	0	0	0	

*Nói cách khác, không còn đối thủ nào có thể thực hiện được nước đi hợp lệ.*

Các toán tử: chọn ô để rải theo chiều trái hay phải (-1 hay 1).

### 3.2.3 Cấu trúc dữ liệu:

Các biến và hàm sử dụng trong cài đặt:

#### Danh sách biến cần dùng:

- BanCo[12]: mảng kiểu nguyên lưu số quân của các ô trên bàn cờ.
- QuayLai[14], TienToi[14]: mảng kiểu nguyên lưu số quân trên bàn cờ và điểm của người và máy để sử dụng trong hàm Undo và Redo.
- \_quaylai: kiểu bool dùng để kiểm tra xem đã undo chưa trước khi redo.
- LbDan[12], \_LbDan[12]: mảng kiểu Label dùng để lưu các Label để thay đổi hiển thị.
- hientai: là biến kiểu Node dùng để lưu giá trị Min, Max tại một vị trí ô.
- hiemmay, diemnguo: biến kiểu nguyên dùng để lưu điểm máy, điểm người.
- capdo: là biến kiểu nguyên dùng để chọn độ khó của máy.
- oMayChon: kiểu số nguyên dùng để đánh dấu vị trí của máy đã đi.
- endgame: kiểu bool là biến dùng để kiểm tra trò chơi đã đạt đến trạng thái kết thúc hay chưa.
- luatnguo: kiểu bool dùng để kiểm tra xem đây là lượt của người đi hay máy đi nếu là true thì là của người.
- mauthu: kiểu bool dùng để kiểm tra xem đây là nước đi chính thức hay nước đi giả định.
- odedi: kiểu nguyên dùng để lưu lại vị trí ô sẽ rải
- chieuDi: kiểu nguyên dùng để lưu lại chiều đi của ô nếu là -1 thì đi bên trái, 1 thì đi bên phải.

#### Danh sách hàm:

- DiChuyen(int chieu, int[] ss): Input là chiều chọn để rải và trạng thái bàn cờ hiện tại. Output là trạng thái sau khi di chuyển để cung cấp giá trị cho hàm định giá.
- DiChuyen1(int Chieu, int[] ss): Input là chiều chọn để rải và trạng thái bàn cờ hiện tại. Output là trạng thái thay đổi thực sự sau khi di chuyển hay đây là trạng thái mà ta thấy được trên màn hình game.
- Kiemtra(): Dùng để kiểm tra có phải là trạng thái kết thúc hay không để thông báo kết quả.

- Thaydoi(): Hàm thay đổi dùng để thay đổi Text, Image của Label hiển thị trạng thái của bàn cờ và điểm của người chơi.
- Newgame(): Dùng để thiết lập lại trạng thái ban đầu của trò chơi.
- MayChoi(): Là dùng để kiểm bạn đang chơi ở cấp độ nào để máy đánh cấp độ đó và thực hiện các bước đi của nó.
- AI(): là hàm tìm kiếm áp dụng thuật giải nguyên lý tham lam.
- AIes(): là hàm áp dụng nguyên lý Minimax có độ sâu cố định là 2:
- SuaViTri(): Để kiểm tra và đảm bảo vị trí luôn nằm trong mảng bàn cờ.
- ThieuQuan(): Dùng để kiểm tra số lượng quân trên bàn cờ của mỗi người còn quân để rải hay không.

### 3.3 CÁC VẤN ĐỀ VÀ THUẬT GIẢI

#### 3.3.1 Xây dựng hàm Minimax có cắt tỉa

Xây dựng class Node gồm các thành viên dữ liệu:

- player: cho biết node hiện tại là của người đi hay máy đi
- điểm người
- điểm máy
- cách đi từ node hiện tại đến node này, nếu node đang là node hiện tại thì cách đi trống
- mảng s[ ] gồm 12 phần tử chứa trạng thái hiện tại của các quân trên bàn cờ, với s[0] và s[6] là 2 ô quan, s[1] đến s[5] là các ô dân của máy, s[7] đến s[11] là các ô dân của người.

Xây dựng Class Minimax gồm:

Một thuộc tính danh sách Danh Sach[ ];

Các phương thức của class Minimax:

#### **Hàm đánh giá**

Hàm DanhGia(node)

Input: Một node.

Output: Trả về một giá trị lượng giá.

Bước 1. Gán  $f = \text{điểm máy} - \text{điểm người}$ .

Bước 2. Gán `DanhSach[ ]` để lưu độ tốt của các node lá đã đánh giá và cách đi đến node lá đó, thêm vào mảng này cặp giá trị  $[f, \text{cách đi}]$ .

Bước 3. Trả về giá trị  $f$  và kết thúc.

### **Hàm SuaViTri()**

Input : một giá trị số nguyên

Output : trả về một giá trị số nguyên

Bước 1. Gán số nguyên truyền vào là  $n$

Bước 2. Nếu  $n < 0$  thì gán  $n = 11$ .

Bước 3. Nếu  $n > 11$  thì gán  $n = 0$ .

Bước 4. Trả về  $n$  và kết thúc

### **Hàm Move()**

Input : một node cần tính toán bước đi tiếp theo, một mảng bước đi chỉ định

Output : trả về giá trị node kề của node trong input

Bước 1. Gọi mảng bước đi truyền vào là `buocdi[ ]`, gán biến `vitri = buocdi[0]`, biến `chieu = buocdi[1]`, tạo một mảng mới `_s`, sao chép giá trị các phần tử trong mảng bàn cờ `s` (mảng `s` trong node) vào mảng `_s`, gán biến `diem = 0`; biến `soluong = _s[vitri]` lưu số quân cờ mình bóc lên để rải; gán một cờ mất lượt `MATLUOT = false`;

Bước 2. Nếu `MATLUOT = false` thì đi tiếp, ngược lại thì nhảy qua bước 3.

Bước 2.1. Nếu `soluong > 0` thì gán `vitri = chieu + vitri`, gọi hàm `vitri = SuaViTri(vitri)`, `soluong = soluong - 1`, `_s[vitri] = _s[vitri] + 1`.

Bước 2.2. Nếu `soluong = 0` thì gán biến `vitri_ke = vitri + chieu`, `vitri_keke = vitri_ke + 1`, gọi hàm `vitri_ke = SuaViTri(vitri_ke)`, `vitri_keke = SuaViTri(vitri_keke)`.

Bước 2.2.1. Trường hợp mất lượt khi rải xong mà gặp ô quan hoặc 2 ô trống liên tiếp. Nếu `_s[vitri_ke] = 0` và `vitri_keke = 0` hoặc `vitri_ke = 0` hoặc `vitri_ke = 6` thì cờ `MATLUOT = true`, trở về bước 2.

Bước 2.2.2. Trường hợp ăn điểm. Nếu `_s[vitri_ke] = 0` và `_s[vitri_keke] > 0` thì gán `diem = diem + _s[vitri_keke]`, `_s[vitri_keke] = 0`, cập nhật lại `vitri = vitri_keke`, gán biến `t = vitri + chieu`. Xét xem nếu `_s[t] > 0`

thì không thể ăn tiếp được, cò  $MATLUOT = true$ , trở về bước 2. Ngược lại nếu  $_s[t] \leq 0$  thì không bắt cò và trở về bước 2 xét tiếp xem có ăn được thêm điểm không.

Bước 2.2.3. Trường hợp rải quân xong ô kế tiếp là quân dân. Nếu  $_s[vitri\_ke] > 0$  thì cập nhật  $soluong = _s[vitri\_ke]$ ,  $_s[vitri\_ke] = 0$ ,  $vitri = vitri\_ke$  sau đó trở về bước 2.1 để tiếp tục rải quân.

Bước 3. Nếu cách đi của node rỗng thì gán mảng  $cachdi[] = buocdi[]$  ngược lại thì sao chép mảng cách đi của node vào mảng  $cachdi[]$ .

Bước 4. Tạo một node mới

Bước 4.1 Nếu node ở input là máy đi thì cập nhật lại node mới này với các giá trị mảng  $s[] = _s[]$ , cách đi =  $cachdi[]$ , cò lượt sẽ là lượt của người, điểm máy = điểm máy trong node input + diem, điểm người = điểm của người trong node input.

Bước 4.2 Nếu node ở input là người đi thì cập nhật lại node mới này với các giá trị mảng  $s[] = _s[]$ , cách đi =  $cachdi[]$ , cò lượt sẽ là lượt của máy, điểm máy = điểm máy trong node input, điểm người = điểm của người trong node input + diem.

Bước 5. Trả về giá trị node và kết thúc.

### **Hàm BuocDi()**

Input: một node

Output: trả về một danh sách chứa các mảng bước đi có thể đi được của node

Bước 1. Tạo danh sách các mảng, mỗi mảng gồm 2 phần tử. Gọi danh sách này là  $buocdi$ .

Bước 2. Nếu node là lượt chơi của máy, gán  $i = 1$

Bước 2.1. Nếu  $i \leq 5$  thì đi tiếp, ngược lại trả về A và kết thúc

Bước 2.2. Nếu  $node.s[i] > 0$  thì thêm vào danh sách A 2 mảng  $\{i, -1\}$  và  $\{i, 1\}$  (với  $i$  là vị trí bước đi, 1 là chiều đi bên phải, -1 là chiều đi bên trái), tăng biến  $i$  ( $i = i + 1$ ), trở về bước 2.1.

Bước 3. Nếu node là lượt chơi của người, gán  $i = 7$

Bước 3.1. Nếu  $i \leq 11$  thì đi tiếp, ngược lại trả về A và kết thúc

Bước 3.2. Nếu  $node.s[i] > 0$  thì thêm vào danh sách A 2 mảng  $\{i, -1\}$  và  $\{i, 1\}$ , tăng biến  $i$  ( $i = i + 1$ ), trở về bước 3.1.

### Hàm NodeKe()

Input: một node

Output: danh sách các node kề của node hiện tại

Bước 1. Tạo danh sách chứa các mảng bước đi có thể đi được của node bằng cách gọi hàm BuocDi(node), gán là A. Tạo một danh sách chứa các node kề của node input truyền vào, gọi là B.

Bước 2. Với mỗi bước mảng buocdi[ ] trong a, ta gọi hàm move(node, buocdi[ ]), hàm này trả về một node, thêm node này vào B.

Bước 3. Trả về danh sách node kề B.

### Hàm DeQuy()

Input: trạng thái hiện tại của bàn cờ, độ sâu cho trước.

Output: trả về độ tốt của một node lá tương ứng với độ sâu

Bước 1. Gán  $a = -500$ ,  $b = 500$ , lưu trạng thái hiện tại của bàn cờ vào biến node, lưu độ sâu vào biến d, xét xem nếu node là node kết thúc hoặc nếu độ sâu = 0 thì gọi hàm DanhGia(node), gán giá trị đánh giá của node = f, trả về f và kết thúc.

Bước 2. Nếu node là lượt chơi của máy, dùng hàm NodeKe(node) tìm tất cả các node con của node, lưu vào mảng nodeke[ ]. Với mỗi node thuộc nodecon[ ], Gán  $d = d - 1$ , node = 1 node con, thực hiện đệ quy quay lại bước 1 cho đến khi trả về giá trị f. So sánh tất cả các f trả về của các node con, chọn cái lớn nhất. Trả về giá trị f và kết thúc.

Bước 3. Nếu node là lượt chơi của người, dùng hàm NodeKe(node) tìm tất cả các node con của node, lưu vào mảng nodeke[ ]. Với mỗi node thuộc nodecon[ ], Gán  $d = d - 1$ , node = 1 node con, thực hiện đệ quy quay lại bước 1 cho đến khi trả về giá trị f. So sánh tất cả các f trả về của các node con, chọn cái nhỏ nhất. Trả về giá trị f và kết thúc.

### Hàm MinimaxSearch()

Input: node hiện tại của bàn cờ, độ sâu cho trước

Output: mảng nguyên chứa bước đi tốt nhất cho node hiện tại theo cách đánh giá của Minimax()



Bước 1. Gán  $node = node$  ở input,  $d = độ\ sâu\ ở\ input$ . Gọi hàm  $DeQuy(node, d)$ , hàm trả về giá trị nguyên, gọi là  $f$ .

Bước 2. Tìm trong  $DanhSach[]$  có cặp giá trị  $[f, bước\ đi\ nào\ đó]$ . Gán mảng  $cachdi[] = bước\ đi\ nào\ đó$ .

Bước 3. Trả về mảng  $cachdi[]$  và kết thúc.

### 3.3.2 Minimax có độ sâu cố định bằng 2

Minimax có độ sâu cố định là 2

Input: Trạng thái của bàn cờ ( $Banco[]$ )

Output: Nước đi ( $vitrimax$ ) và chiều phù hợp ( $chieumax$ )

Bước 1: Gán biến  $diemmax = -100$ ;  $diemmin = 100$ ,  $vitrimax = 0$ ;  $chieumax = 0$ .

Bước 2: `for(int i = 1; i < 6; i++)` //các ô bên phía máy

Bước 2.1: Gán  $nodeAI[] = BanCo[]$ , gán  $odedi = i$

Bước 2.2: Nếu  $nodeAI[odedi]$  khác 0 thì gán  $diem0 = diem1 = diem2 = 0$  (dùng để lưu tạm điểm), gán  $diemmin$  lại bằng 100; di chuyển  $nodeAI[]$  tại ô  $i$  theo chiều 1 (chiều kim đồng hồ) cho trả về  $diem0$ , gán  $diem1 = diem0$ . Nếu bằng  $nodeA[odedi] = 0$  thì trở lại bước 2

Bước 2.2.1: `for(int j = 7; j < 12; j++)` //các ô bên phía người chơi

Bước 2.2.1.1: Gán  $diem0$  lại = 0; gán  $node2 = nodeAI$  sao khi đã di chuyển; gán  $node2[] = nodeAI[]$ ;  $odedi = j$

Bước 2.2.1.2: Nếu  $node2[]$  khác 0 thì di chuyển  $node2[odedi]$  tại ô  $j$  theo chiều 1 cho trả về  $diem0$ ; gán  $diem2 = diem1 - diem0$ . Nếu  $diem2 \leq diemmin$  thì gán  $diemmin = diem$   $vitrimax = i$ ;  $chieumax = 1$ . Nếu  $node2[] = 0$  thì trở lại bước 2.2.1

Bước 2.2.2: `for(int j = 7; j < 12; j++)`

Bước 2.2.2.1: Gán  $diem0$  lại = 0; gán  $node2 = nodeAI$  sao khi đã di chuyển; gán  $node2[] = nodeAI[]$ ;  $odedi = j$

Bước 2.2.2.2: Nếu  $node2[]$  khác 0 thì di chuyển  $node2[odedi]$  tại ô  $j$  theo chiều -1 cho trả về  $diem0$ ; gán  $diem2 = diem1 - diem0$ . Nếu  $diem2 \leq diemmin$  thì gán  $diemmin = diem$   $vitrimax = i$ ;  $chieumax = 1$ . Nếu  $node2[] = 0$  thì trở lại bước 2.2.2

Bước 2.2.3: Nếu  $diemmin \geq diemmax$  thì gán  $diemmax = diemmin$ ;  $vitrimax = i$ ,  $chieumax = 1$ ;

Bước 2.2.4: gán  $diem0 = diem1 = diem2 = 0$ , gán  $diemmin$  lại bằng 100, gán  $nodeAI[] = BanCo[]$ ,  $odedi = i$ ; di chuyển  $nodeAI[]$  tại ô  $i$  theo chiều -1 (ngược chiều kim đồng hồ) cho trả về  $diem0$ , gán  $diem1 = diem0$

Bước 2.2.5:  $for(int j=7; j<12; j++)$  //các ô bên phía người chơi

Bước 2.2.5.1: Gán  $diem0$  lại = 0; gán  $node2 = nodeAI$  sao khi đã di chuyển; gán  $node2[] = nodeAI[]$ ;  $odedi = j$

Bước 2.2.5.2: Nếu  $node2[]$  khác 0 thì di chuyển  $node2[odedi]$  tại ô  $j$  theo chiều 1 cho trả về  $diem0$ ; gán  $diem2 = diem1 - diem0$ . Nếu  $diem2 \leq diemmin$  thì gán  $diemmin = diem$   $vitrimax = i$ ;  $chieumax = 1$ . Nếu  $node2[] = 0$  thì trở lại bước 2.2.4

Bước 2.2.6:  $for(int j=7; j<12; j++)$

Bước 2.2.6.1: Gán  $diem0$  lại = 0; gán  $node2 = nodeAI$  sao khi đã di chuyển; gán  $node2[] = nodeAI[]$ ;  $odedi = j$

Bước 2.2.6.2: Nếu  $node2[]$  khác 0 thì di chuyển  $node2[odedi]$  tại ô  $j$  theo chiều -1 cho trả về  $diem0$ ; gán  $diem2 = diem1 - diem0$ . Nếu  $diem2 \leq diemmin$  thì gán  $diemmin = diem$   $vitrimax = i$ ;  $chieumax = 1$ . Nếu  $node2[] = 0$  thì trở lại bước 2.2.5

Bước 2.2.7: Nếu  $diemmin \geq diemmax$  thì gán  $diemmax = diemmin$ ;  $vitrimax = i$ ,  $chieumax = -1$ ;

Bước 3: Gán  $odedi = vitrimax$ ; di chuyển  $BanCo[]$  tại ô  $odedi$  và chiều  $chieumax$ .

Bước 4: Kết thúc.

### 3.3.3 Hàm AI theo nguyên lý tham lam

Input: Trạng thái của bàn cờ (BanCo[])

Output: Nước đi (vitrimax) và chiều phù hợp (chieumax)

Bước 1: Gán diemmax=-100; vitrimax=0; chieumax=0

Bước 2: for (i=1; i<6; i++).

Bước 2.1: Gán diem0=0; diem1=0; nodeAI[]=BanCo[], odedi=i;

Bước 2.2: Nếu odedi !=0 thì DiChuyen(1, nodeAI[]); diem1=diem0. Nếu diem1>diemmax thì diemmax gán bằng diem1; vitrimax=I; chieumax=1. Ngược lại trở lại bước 2.

Bước 3: for (i=1; i<6; i++).

Bước 3.1: Gán diem0=0; diem1=0; nodeAI[]=BanCo[], odedi=i;

Bước 3.2: Nếu odedi !=0 thì DiChuyen(-1, nodeAI[]); diem1=diem0. Nếu diem1>diemmax thì diemmax gán bằng diem1; vitrimax=I; chieumax=-1. Ngược lại trở lại bước 3.

Bước 4: : Gán odedi = vitrimax; di chuyển BanCo[] tại ô odedi và chiều chieumax

### 3.3.4 Các hàm xử lý cơ bản trong game

#### Hàm xử lý nước đi

Input: Trạng thái bàn cờ ss[], vị trí ô rải odedi và chiều đi chieu.

Output: Trạng thái bàn cờ sau khi di chuyển ss[].

Bước 1: Kiểm tra số lượng quân của ô được chọn có lớn hơn 0 không. Nếu không thì chuyển qua Bước 6.

Bước 2: Kiểm tra xem có phải lượt người đi hay không. Nếu là lượt người thì lưu trạng thái qua mảng phụ ( QuayLai[]) để làm dữ liệu cho hàm Undo và Redo.

Bước 3: Gán biến corai = số lượng quân của ô được chọn (ss[odedi]), cho biến kiểm tra còn lượt = true. Gán số lượng quân của ô được chọn = 0. Gán diem=0.

Bước 4: Trong khi còn lượt bằng true. Thì kiểm tra corai có lớn hơn 0 hay không.

Bước 4.1: Trong khi corai lớn hơn 0 thì tăng odedi theo chiều đã chọn.

Chỉnh sửa odedi cho tương ứng với vị trí trong mảng bàn cờ. Tăng số lượng quân của ô được rải (ss[odedi]++). Giảm số lượng corai xuống 1 quân.

Bước 4.2: Nếu corai=0. Gán contro1 = odedi +chieu;  
Gán contro2=contro1+chieu. Chỉnh sửa contro1 và contro2 tương ứng với vị trí trong mảng bàn cờ.

Bước 4.2.1: Nếu contro1=0 hoặc contro1=6 hoặc ss[contro1]=0 và ss[contro2]=0 thì chuyển sang Bước 5.

Bước 4.2.2: Nếu ss[contro1]=0 và ss[contro2]>0 thì diem=diem+ ss[contro2]. Gán ss[contro2]=0, odedi=contro2; contro1=odedi+chieu. Chỉnh sửa contro1 tương ứng với vị trí trong mảng bàn cờ. Nếu ss[contro1]>0 thì gán còn lượt = false và chuyển tới bước 5 .

Bước 4.3 Nếu ss[contro1] >0 thì corai=ss[contro1];

Ss[contro1]=0; odedi=contro1;

Bước 5: Nếu diem !=0;

Bước 5.1: Nếu lượt người = true thì diemnguoi=diemnguoi+diem.

Bước 5.2: Ngược lại diemmay=diemmay+diem;

Bước 6: Kết thúc.

### **Hàm kiểm tra kết thúc:**

Input: Trạng thái bàn cờ ss[], vị trí ô rải odedi và chiều đi chieu.

Output: Trạng thái bàn cờ.

Bước 1: Nếu thuộc 1 trong 3 điều kiện trạng thái kết thúc.

+ Khi 2 quan đồng thời bị ăn hết không còn dân nào trong ô quan.

+ Khi các ô thuộc một bên nắm giữ hết sỏi mà trong kho của mình không đủ 5 sỏi để rải đều cho 5 ô của mình.

Thì trạng thái kết thúc của trò chơi endgame=true; luotnguoi=false.  
Tăng điểm của người chơi tương ứng với số quân trên bàn và máy tương trên bàn.

Bước 1.1 diemmay>diemnguoi thì thông báo máy thắng.

Bước 1.2 diemmay=diemnguoi thì thông báo hòa.

Bước 1.3 Ngược lại thông báo bạn thắng.

Bước 2: Kết thúc.

### 3.4 VÍ DỤ MINH HỌA THUẬT TOÁN

#### 3.4.1 MinimaxSearch()

Ví dụ đánh giá bằng hàm MinimaxSearch() 1 bước đi hiện tại (độ sâu = 2):

Giả sử Node hiện tại (máy chọn):

$s = \{ 1, 0, 0, 11, 1, 0, 4, 9, 0, 0, 0, 0 \}$

$d(\text{điểm người} = 0, \text{điểm máy} = 0)$

Từ node hiện tại ta tính được 2 nước tiếp theo có thể đi là:

- Node 1:

$s = \{ 0, 0, 0, 1, 1, 1, 2, 1, 1, 12, 2, 0 \}$

$d(0, 5)$

- Node 2:

$s = \{ 5, 0, 0, 0, 1, 1, 2, 1, 1, 12, 2, 1 \}$

$d(0, 0)$

Từ node 1 ta lại tính được 8 nước tiếp theo có thể đi

Node 1.1 :  $s = \{ 2, 1, 1, 12, 0, 1, 1, 0, 0, 1, 1, 1 \}$  ,  $d(0, 5)$ ,

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Node 1.2 :  $s = \{ 3, 0, 2, 13, 0, 0, 0, 0, 1, 0, 2, 0 \}$  ,  $d(0, 5)$

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Node 1.3 :  $s = \{ 4, 0, 3, 2, 0, 0, 2, 2, 0, 3, 0, 3 \}$  ,  $d(2, 5)$

Hết độ sâu, đánh giá  $f = 5 - 2 = 3$

Node 1.4 :  $s = \{ 4, 3, 0, 1, 3, 1, 1, 2, 0, 3, 3, 0 \}$  ,  $d(0, 5)$

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Node 1.5 :  $s = \{ 2, 1, 0, 13, 0, 1, 1, 0, 0, 1, 1, 1 \}$  ,  $d(0, 5)$

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Node 1.6 :  $s = \{ 2, 2, 0, 12, 2, 0, 0, 0, 0, 1, 1, 1 \}$  ,  $d(0, 5)$

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Node 1.7 :  $s = \{ 4, 0, 4, 0, 1, 0, 3, 3, 0, 0, 0, 0 \}$  ,  $d(6, 5)$

Hết độ sâu, đánh giá  $f = 5 - 6 = -1$

Node 1.8 :  $s = \{ 3, 0, 1, 12, 2, 0, 0, 0, 1, 0, 2, 0 \}$  ,  $d(0, 5)$

Hết độ sâu, đánh giá  $f = 5 - 0 = 5$

Từ node 2 ta lại tính được 10 nước tiếp theo có thể đi:

Node 2.1 =  $\{ 2, 1, 1, 12, 2, 0, 6, 0, 0, 0, 1, 1 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 0 = -0$

Node 2.2 :  $s = \{ 4, 3, 0, 1, 4, 1, 6, 2, 0, 2, 3, 0 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 0 = 0$

Node 2.3 :  $s = \{ 2, 1, 1, 12, 0, 2, 6, 0, 0, 0, 1, 1 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 0 = 0$

Node 2.4 :  $s = \{ 3, 0, 2, 13, 0, 1, 5, 0, 0, 0, 2, 0 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 0 = 0$

Node 2.5 :  $s = \{ 4, 0, 3, 2, 0, 0, 7, 2, 0, 2, 0, 3 \}$  ,  $d(3, 0)$

Hết độ sâu, đánh giá  $f = 0 - 3 = -3$

Node 2.6 :  $s = \{ 4, 3, 0, 1, 3, 2, 6, 2, 0, 2, 3, 0 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 5 = -5$

Node 2.7 :  $s = \{ 2, 1, 0, 13, 0, 2, 6, 0, 0, 0, 1, 1 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 5 = -5$

Node 2.8 :  $s = \{ 2, 2, 0, 12, 2, 1, 5, 0, 0, 0, 1, 1 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 5 = -5$

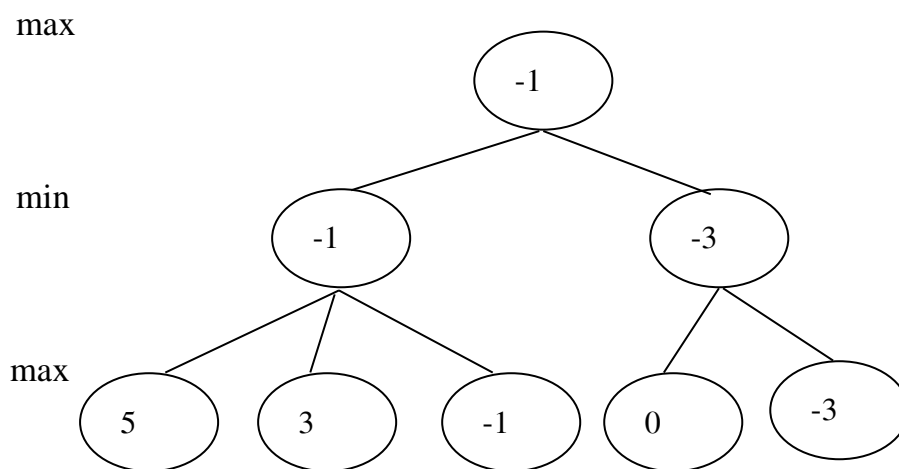
Node 2.9 :  $s = \{ 4, 0, 4, 0, 1, 0, 8, 3, 1, 0, 1, 4 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - 5 = -5$

Node 2.10 :  $s = \{ 3, 0, 1, 12, 2, 1, 5, 0, 0, 0, 2, 0 \}$  ,  $d(0, 0)$

Hết độ sâu, đánh giá  $f = 0 - = 0$

Cây Minimax:



Ý tưởng chính của thuật giải này là chọn trong 2 node 1 và 2, đánh giá node nào có thể có trường hợp tệ nhất cho máy, sau đó chọn trường hợp lớn hơn trong 2 trường hợp tệ nhất đó. Ví dụ ở đây node 1 tệ nhất là máy kém người 1 điểm, node 2 tệ nhất là máy kém người 3. Vậy ta sẽ chọn node 1 để tệ nhất thì máy chỉ bị kém người 1 điểm thôi.

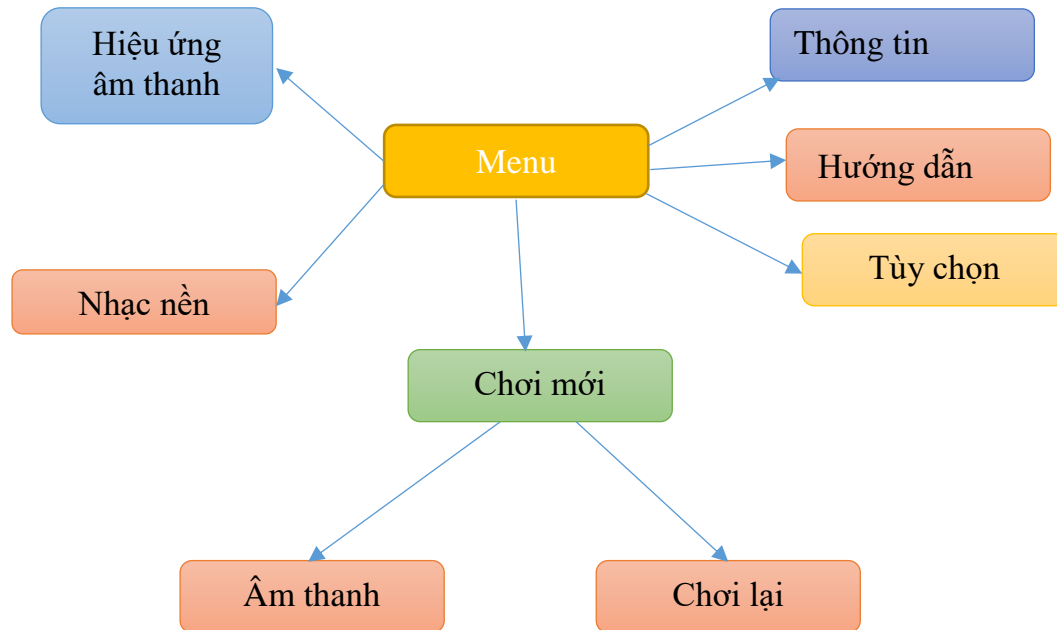


## CHƯƠNG 4: ỨNG DỤNG

### 4.1 GIỚI THIỆU CHƯƠNG TRÌNH ỨNG DỤNG

Trò chơi Ô ăn quan.

#### 4.1.1 Hệ thống Menu trong game



Menu chính chứa các đối tượng lựa chọn:

- Chơi mới: Chọn để bắt đầu chơi game. Sau khi chọn sẽ hiện lên một cửa sổ mới để chơi game.
- Thông tin: Sẽ hiện lên thông tin của người viết game.
- Tùy chọn: Hiện lên cấp độ chơi
- Hiệu ứng âm thanh: Âm thanh khi di chuyển và ăn quân
- Nhạc nền: Bật tắt nhạc nền.
- Hướng dẫn: Xem hướng dẫn để chơi.

#### 4.1.2 Giao diện game

Menu:




Play game:



#### 4.1.3 Các nút chức năng cơ bản trong game



:**Chức năng Undo - Redo**: người chơi có thể thu hồi nước cờ bằng cách click

chuột vào biểu tượng  và đi tới các nước sau (nếu có) bằng cách click chuột vào



Trở về Menu



Chơi lại



Chọn chiều di chuyển qua phải.



Chọn chiều di chuyển qua trái



Bật tắt âm thanh nền.



Lựa chọn các cấp khi chơi.

- **Chức năng hiển thị lượt chơi:** khi tới lượt chơi của quân nào thì màn hình sẽ hiện lên lượt đánh của quân đó.

*Chọn nước đi và hướng*

## 4.2 CÀI ĐẶT

### 4.2.1 Ngôn ngữ và công cụ sử dụng.

Để xây dựng game Ô ăn quan , nhóm sử dụng:

- Ngôn ngữ lập trình: C# đây là một ngôn ngữ mới và mạnh mẽ. Tuy nhiên trong đề tài này, nhóm lựa chọn ngôn ngữ C# vì dễ thao tác và phù hợp với công cụ lập trình
- Công cụ: Visual Studio, Photoshop để thiết kế giao diện

### 4.2.2 Các hàm các đoạn code chính

Vì trọng tâm của đề tài là về trí tuệ nhân tạo cho game, nên ở đây nhóm chỉ trình bày những đoạn code chính.

- Hàm Minimax

```
public int[] MinimaxSearch(Node hientai, int d)
{
```

```

//Lượng giá node hientai
int f = DeQuy(hientai, d, -500, 500);
if (DANHSAATCHDI.Count == 0)
{
    int vitri = 0;
    int chieu = 1;
    if (!hientai.luotnguoi)
    {
        for (int i = 1; i <= 5; i++)
            if (hientai.s[i] > 0)
            {
                vitri = i;
                break;
            }
    }
    else
    {
        for (int i = 7; i <= 11; i++)
            if (hientai.s[i] > 0)
            {
                vitri = i;
                break;
            }
    }
    return new int[] { vitri, chieu };
}
}

```

- Hàm Độ quy: dùng để cắt tỉa

```

public int DeQuy(Node node, int depth, int alpha, int beta)
{
    //Khi gặp node kết thúc game hoặc độ sâu = 0 thì trả
    về độ tốt của node qua
    //hàm DanhGia()
    if ((node.s[0] == 0 && node.s[6] == 0) || depth <=
0)
        return DanhGia(node);

    //Sử dụng đệ quy để duyệt cây theo ý chí của Minimax
    if (!node.luotnguoi)
    {
        List<Node> nodeke = NodeKe(node);
        foreach (Node ke in nodeke)
        {

```

```

        alpha = Math.Max(alpha, DeQuy(ke, depth - 1,
alpha, beta));
        if (beta < alpha) break; //cắt tỉa, không
duyệt các node kế tiếp trong nodeke
    }

    return alpha;
}
else
{
    List<Node> nodeke = NodeKe(node);
    foreach (Node ke in nodeke)
    {
        beta = Math.Min(beta, DeQuy(ke, depth - 1,
alpha, beta));
        if (beta < alpha) break; //cắt tỉa, không
duyệt các node kế tiếp trong nodeke
    }

    return beta;
}
}

```

- Hàm đánh giá độ tốt:

```

//Hàm đánh giá độ tốt của một trạng thái
public int DanhGia(Node hientai)
{
    int f;
    f = hientai.max_scored - hientai.min_scored;
    //Lưu độ tốt cùng với cách đi của node vào
DANHSAATCHDI
    //Hàm này sẽ chỉ được gọi khi node hientai là node
lá hoặc là node kết thúc,
    //sẽ nói rõ hơn ở hàm DeQuy()
    if (!DANHSAATCHDI.ContainsKey(f))
        DANHSAATCHDI.Add(f, hientai.CachDi);

    return f;
}

```

- Hàm kiểm tra: Kiểm tra xem đã đến trạng thái cuối cùng hay chưa

```

public void KiemTra()//kiểm tra xem đã đến lúc kết thúc game
chưa.

```

```

{
    if (
        (s[0]==0 && s[6]==0) ||
        (diemmay <5 && s[1] ==0 && s[2] == 0 && s[3] ==0
&& s[4]==0 && s[5] == 0 ) ||
        (diemnguoi < 5 && s[7] == 0 && s[8] == 0 && s[9]
== 0 && s[10] == 0 && s[11] == 0)
    )
    {
        endgame = true;
        luotnguoi = false;
        diemmay = diemmay + s[1] + s[2] + s[3] + s[4] +
s[5];
        diemnguoi = diemnguoi + s[7] + s[8] + s[9] +
s[10] + s[11];
        if(diemmay>diemnguoi)
        {
            labThongbao.Text = "Máy Thắng!!";
            LostGameSoundPlayer.Play();// âm thanh trong
game.

            for(int i=0; i < 12;i++)
            {
                s[i] = 0;
            }
            ThayDoi();
        }
        else if(diemmay == diemnguoi)
        {
            labThongbao.Text= "Hòa Rồi!!";
            for (int i = 0; i < 12; i++)
            {
                s[i] = 0;
            }
            ThayDoi();
        }
        else
        {
            labThongbao.Text = "Bạn Thắng";
            WinGameSoundPlayer.Play();
            for (int i = 0; i < 12; i++)
            {
                s[i] = 0;
            }
            ThayDoi();
        }
    }
}

```

}

- Hàm kiểm tra quân: Kiểm tra xem còn đủ quân trong bàn cờ của người chơi hay không. Nếu thiếu thì rải quân từ những quân đã ăn được.

```
public void ThieuQuan()
{
    if (endgame == false)
    {
        if ((luotnguoi == true) && (diemnguoi >= 5 &&
s[7] == 0 && s[8] == 0 && s[9] == 0 && s[10] == 0 && s[11] ==
0))
        {
            s[7] = 1;
            s[8] = 1;
            s[9] = 1;
            s[10] = 1;
            s[11] = 1;
            diemnguoi = diemnguoi - 5;
            ThayDoi();
        }
        if ((luotnguoi == false) && diemmay >= 5 && s[1]
== 0 && s[2] == 0 && s[3] == 0 && s[4] == 0 && s[5] == 0)
        {
            s[1] = 1;
            s[2] = 1;
            s[3] = 1;
            s[4] = 1;
            s[5] = 1;
            diemmay = diemmay - 5;
            ThayDoi();
        }
    }
}
```

### 4.3 KẾT QUẢ CHẠY CHƯƠNG TRÌNH

#### 4.2.1 Kết quả thử nghiệm

- Các nước đi đã thực hiện đúng luật của trò chơi.
- Các chức năng Undo, Redo hoạt động.
- Các hiệu ứng âm thanh hoạt động tốt.

Kết thúc game:

- Máy thắng



- Bạn thắng





## **CHƯƠNG 5: KẾT LUẬN**

### **5.1 KẾT QUẢ ĐẠT ĐƯỢC**

- Game đã chạy đúng luật của trò chơi.
- Đã hoàn thành các chức năng cơ bản trong game.
- Có hiệu ứng âm thanh.
- Giao diện để dùng có các chức năng cơ bản. Nhóm khá ưng ý với giao diện nhưng cần hoàn thiện thêm.
- Tìm hiểu và hiểu được cách máy tính lựa chọn nước đi, tính toán trước nhiều bước giống như suy nghĩ của con người, thông qua giải thuật Minimax, Alpha-beta.
- Đã ứng dụng hàm đánh giá heuristic dựa trên kinh nghiệm của con người.
- Có các cấp độ cho người chơi lựa chọn.

### **5.2 HẠN CHẾ**

- Chưa có chế độ 2 người chơi.
- Cách chọn cấp độ
- Chưa đưa thời gian vào trong game
- Chưa cài đặt được danh sách lưu điểm người chơi. Và chưa có form thông báo kết quả riêng hiện lên màn hình.

### **5.3 HƯỚNG PHÁT TRIỂN**

- Khắc phục các mặt hạn chế, hoàn thiện các chức năng cơ bản trong game.
- Tối ưu hóa các vấn đề trong game: tăng sự thông minh cho máy mà tránh được việc tràn bộ nhớ.
- Thiết game cho 3 4 người cùng chơi.
- Hoàn thiện giao diện hiện tại.

## **TÀI LIỆU THAM KHẢO**

- (1) Slides Trí Tuệ Nhân Tạo – Ms. Huỳnh Thị Thanh Thương
- (2) [wikipedia.org/wiki/%C3%94\\_%C4%83n\\_quan](https://wikipedia.org/wiki/%C3%94_%C4%83n_quan)
- (3) Giáo trình Trí tuệ nhân tạo