Trường Đại Học Bách Khoa Hà Nội
Viện Công nghệ Thông Tin Và Truyền Thông

BÁO CÁO

Bài Tập Lớn Môn

TRÍ TUỆ NHÂN TẠO

Đề tài: Áp dụng thuật toán MiniMax xây dựng game cờ Ca-rô

Nhóm sinh viên thực hiện:

- Vũ Thế Đạt — MSSV: 20130889
- Nguyễn Minh Đức — MSSV: 20131022
- Đào Quang Duy — MSSV: 20130592
- Bùi Ngọc Khang — MSSV: 20132039

Giáo viên hướng dẫn:

TS. Nguyễn Nhật Quang

Hà Nội, 11 – 2015

MỤC LỤC

LỜI NÓI ĐẦU

Hiện nay việc ứng dụng Trí tuệ nhân tạo (AI) vào việc phát triển game đã trở nên rất phổ biến, đặc biệt là những game mang tính trí tuệ cao. Điển hình là game Caro (hay còn gọi là game Gomoku) là một trò chơi quen thuộc đối với nhiều đối tượng, dễ chơi, giảm căng thẳng,… Cờ Caro là một trong số những trò chơi rất phổ biến, đặc biệt là trong giới học sinh, sinh viên. Đây cũng là một trò chơi chúng em rất thích, chính vì vậy chúng em chọn đề tài làm game Caro cho môn Trí tuệ nhân tạo.

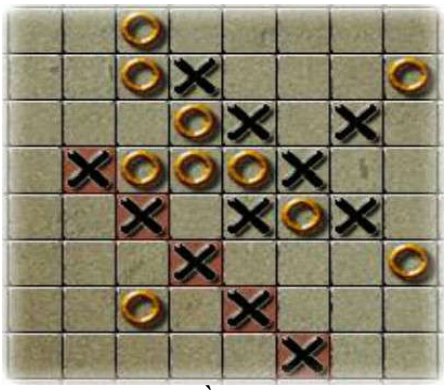
Trong quá trình hoàn thành đề tài này, chúng em đã tìm hiểu được các thuật toán đã được học trong môn Trí tuệ nhân tạo như thuật toán tìm kiếm nước đi MiniMax, giải thuật Alpha-Beta cắt tỉa cũng như kỹ năng lập trình ngôn ngữ Java.

Chúng em cũng xin cảm ơn sự hướng dẫn tận tình của thầy Nguyễn Nhật Quang cả về chuyên môn cũng như định hướng. Nhóm thực hiện đề tài với mục đích xây dựng game Caro có tính nhân tạo cao. Vì kiến thức còn hạn hẹp nên trong quá trình thực hiện đề tài không thể tránh khỏi thiếu xót. Vì vậy rất mong nhận được sự góp ý của thầy để đề tài có thể hoàn thiện hơn nữa.

I. GIỚI THIỆU TRÒ CHƠI CA-RO

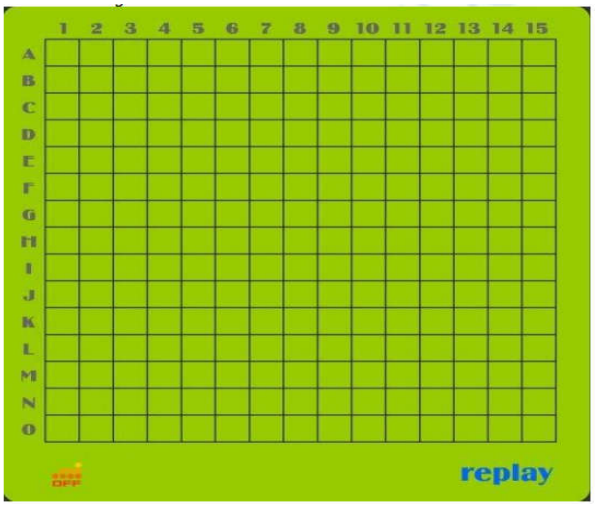
Cờ caro là một loại cờ cổ xưa của người Trung Quốc, một trong những trò chơi logic lâu đời nhất được biết đến trên thế giới.

Cờ caro được chơi trên toàn thế giới, ở mỗi nơi nó lại có tên gọi khác nhau: ở Nhật là Gomoku, ở Nga và các nước Đông Âu gọi là Five in a row, ở Hàn Quốc là Omok, ở Trung Quốc là Wuziqi, Anh là Connect5… và dĩ nhiên ở Việt Nam là Caro.



Game gồm có 2 người chơi, một người cầm quân X, một người cầm quân O. Hai người chơi sẽ lần lượt đánh quân tương ứng của mình trên mặt bàn cờ MxN ô (thường thì M = N)

Luật chơi: Quân X là quân đánh trước. Hai người chơi có thể đánh vào bất kỳ ô nào trên bàn cờ, miễn là ô đó chưa được đánh. Trò chời kết thúc khi người chơi có 5 quân thẳng hàng liên tiếp nhau (ngang, dọc hoặc chéo).



Bàn cờ 15 x 15

Khi các trình độ cờ thủ được nâng cao thì họ nhận ra rằng, nếu chỉ chơi đơn giản trong Gomoku thì đó là lợi thế quá lớn cho bên X (Quy định X đánh trước) vì thế một số cải tiến được đưa ra nhằm cân bằng ưu thế trên như:

- Gomoku. Hiên nay được chơi chính thức với bàn cờ 13x13. Không có hoà. Nếu hết đất thì O thắng. Chưa tìm được chúng minh nào cho thấy X chắc chắn thắng. Tuy nhiên X vẫn có ưu thế rất lớn.
- ProGomoku. Chơi trên bàn 15x15. Nước đầu của X đặt sẵn ô trung tâm. Nước thứ ba (nước thứ hai của X) phải đặt ngoài hình vuông cấm. Hình vuông cấm là hình vuông trung tâm kích thuóc 5x5. Không có hạn chế cho O. Đã có chứng minh X chắc chắn thắng trong biến thể này.
- Pente. Biến thể này không còn giống Gomoku. Luật bổ sung là có thể ăn quân đối phương. Nước ăn quân được thực hiện bằng cách chặn hai đầu một nước hai quân đối phương và ăn hai quân đó. AI tạo được nước năm hoặc ăn được 5 cặp quân trước thì thắng. Rất phổ biến ở Mỹ. Chơi trên bàn 19x19.

II. PHÂN TÍCH GIẢI QUYẾT BÀI TOÁN

1. Phân tích yêu cầu

❖ *Mô phỏng bàn cờ*

Bàn cờ bao gồm các ô cờ được đặt trong một mảng hai chiều (kích thước a x b). Trong mỗi ô cờ có thể xác định được:

- Vị trí ô cờ theo hàng, theo cột
- Trạng thái ô cờ: Đang trống (0), nước đi của người chơi (1), nước đi của máy (2)
- Độ nguy hiểm của ô cờ tùy theo trạng thái của ô cờ có thể thay đổi được

❖ *Đánh giá giá trị các ô cờ*

Giống như trong thực tế người chơi thường đánh giá một số nước cờ là nguy hiểm, bình thường hoặc ít nguy hiểm, máy tính cũng đánh giá như thế nhưng cụ thể hơn bằng các con số.

2. Phương pháp giải quyết

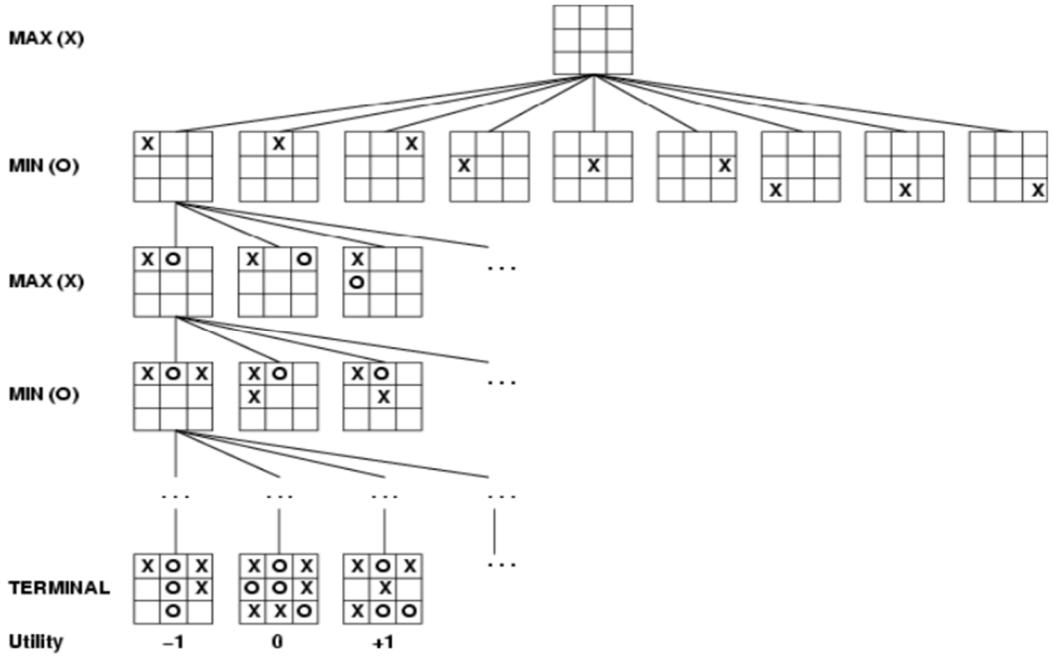
2.1. Tìm kiếm nước đi

❖ *Giới thiệu về không gian tìm kiếm*

Trong trò chơi Caro, cứ sau mỗi nước cờ, mỗi đối thủ sẽ chọn ra từ những ô trống để đi, do đó, sau 1 mỗi nước đi thì số ô trống còn lại sẽ giảm. Như vậy, việc tìm nước đi tiếp theo cho trạng thái có sẵn chỉ là việc tìm kiếm những ô trống còn lại, đồng thời, không gian tìm kiếm sẽ thu hẹp theo số nước đi đã tạo.

Không gian chọn nước đi từ mỗi trạng thái ban đầu là hữu hạn, nhưng không gian tìm kiếm 1 nước đi dẫn đến chiến thắng là rất lớn. Do đó ta không thể vét sạch không gian tìm kiếm nước đi này mà ta phải giới hạn không gian tìm kiếm.

Một không gian tìm kiếm có thể hiện theo 1 cây đa phân và được gọi là cây tìm kiếm hay cây trò chơi. Ví dụ:



Cây trò chơi

Dựa vào cái cây trò chơi đã định nghĩa ở trên, việc tìm kiếm nước đi là chọn 1 nút trên cây (ở mức 1) sao cho nước đó là tốt. Theo thông thường khi chơi, một nước đi tốt hay không là phụ thuộc vào khả năng dành chiến thắng là cao hay thấp sau khi nước đi này được đi. Do đó, muốn chọn 1 nước đi tốt thì nếu chỉ dựa vào thế cờ hiện tại là chưa đủ, mà phải biết thông tin của những thế cờ sau khi chọn nước này để đi.

❖ *Thuật toán MiniMax*

- **Nguyên lý:**

  + Có 2 người chơi là MIN và MAX.

  + Một chiến lược tối ưu là một chuỗi các nước đi giúp đưa đến trạng thái đích mong muốn.

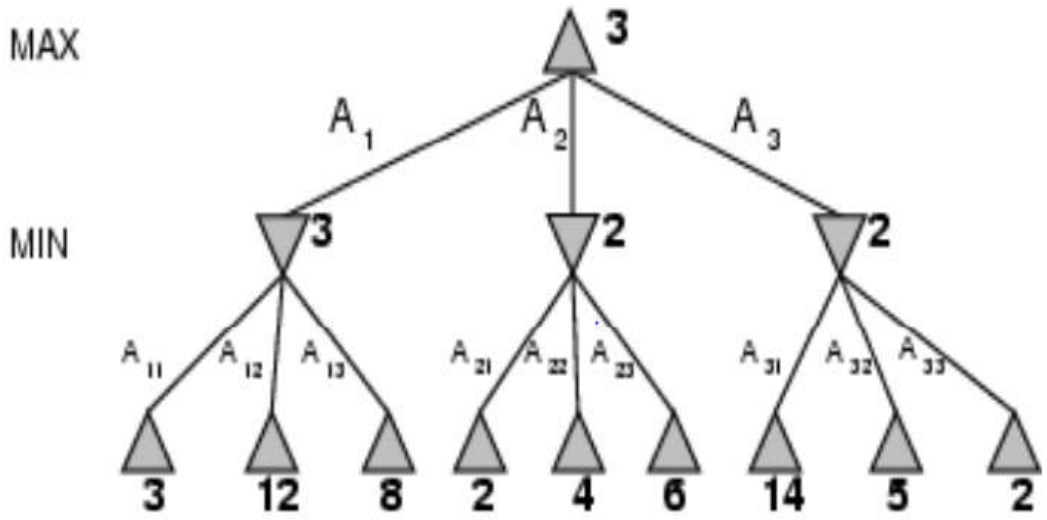
  + Chiến lược của MAX bị ảnh hưởng (phụ thuộc) vào các nước đi của MIN và ngược lại

  + MAX cần chọn một chiến lược giúp cực đại hóa giá trị của hàm mục tiêu với giả sử là MIN đi các nước tối ưu.

  + Chiến lược này được xác định bằng việc xét các giá trị MINMAX đối với mỗi nút trong cây biểu diễn trò chơi.

  + MAX chọn các nước đi tương ứng với giá trị MINMAX cực đại (MIN chọn các nước đi ứng với giá trị MINMAX cực tiểu).

Ví dụ:



- **Giải thuật:**

```
int max_value(state s, int dept) {
        if(terminal_test() || dept >= 4) return eval(s);
                v = -∞
        for( duyệt tất cả các trạng thái con s' của s){
                v = max(v,min_value(s',dept +1));
        }
        return v;
}
```

```
int min_value(state s, int dept) {
        if(terminal_test() || dept >= 4) return eval(s);
                v = +∞
        for( duyệt tất cả các trạng thái con s' của s){
                v = min(v,min_value(s',dept +1));
        }
        return v;
}
```

Giải thuật tìm kiếm MINIMAX vấp phải vấn đề bùng nổ (mức hàm mũ) các khả năng nước đi cần phải xét → không phù hợp với nhiều bài toán trò chơi thực tế.

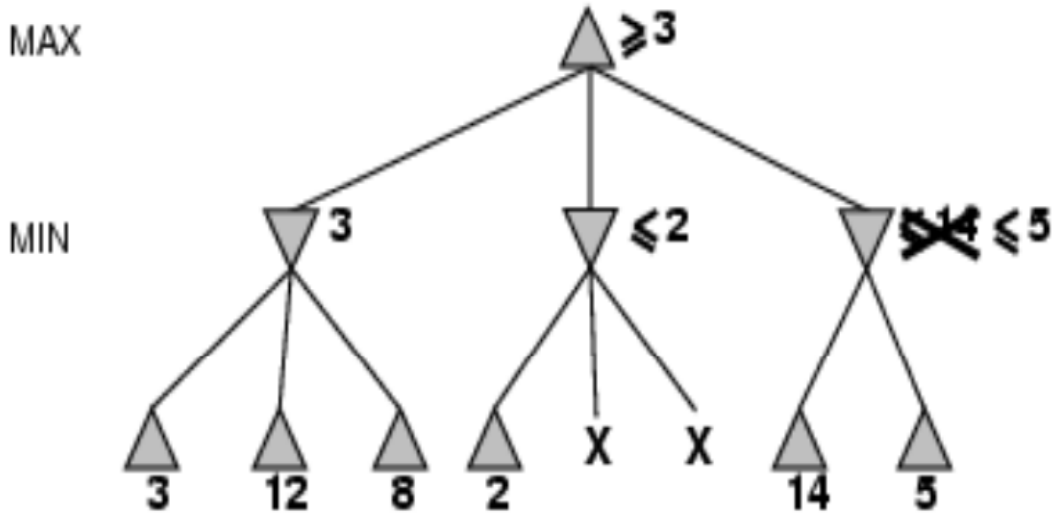
Chúng ta có thể cắt tỉa (bỏ đi – không xét đến) một số nhánh tìm kiếm trong cây biểu diễn trò chơi.

❖ *Giải thuật cắt tỉa α-β (Alpha-beta Pruning)*

- **Ý tưởng:**

Nếu một nhánh tìm kiếm nào đó không thể cải thiện đối với giá trị (hàm tiện ích) mà chúng ta đã có, thì không cần xét đến nhánh tìm kiếm đó nữa!

Việc cắt tỉa các nhánh tìm kiếm ("tồi") không ảnh hưởng đến kết quả cuối cùng α là giá trị của nước đi tốt nhất đối với MAX (giá trị tối đa) tính đến hiện tại đối với nhánh tìm kiếm. Nếu v là giá trị tồi hơn α, MAX sẽ bỏ qua nước đi ứng với v -> Cắt tỉa nhánh ứng với v, β được định nghĩa tương tự đối với MIN. Ví dụ :



- **Giải thuật:**

```
int max_value(state s, int dept, int alpha, int beta) {
        if( terminal_test() || dept >=4)
              return eval(s);
        v = -∞;
        for(duyệt tất cả các trạng thái con s' của s){
              v = max(v,min_value(s',dept +1,alpha,beta));
        }
        if(v > beta)
             return v;
        alpha = max(alpha, v);
       return v;
}
```

```
int min_value(state s, int dept, int alpha, int beta) {
        if( terminal_test() || dept >=4)
              return eval(s);
        v = +∞;
        for(duyệt tất cả các trạng thái con s' của s){
              v = min(v,mã_value(s',dept +1,alpha,beta));
        }
        if(v < alpha)
              return v;
        beta = min(beta, v);
       return v;
}
```

Đối với các trò chơi có không gian trạng thái lớn, thì phương pháp cắt tỉa α-β vẫn không phù hợp. Không gian tìm kiếm (kết hợp cắt tỉa) vẫn lớn.

Có thể hạn chế không gian tìm kiếm bằng cách sử dụng các tri thức cụ thể của bài toán.

- Tri thức để cho phép đánh giá mỗi trạng thái của trò chơi.
- Tri thức bổ sung (heuristic) này đóng vai trò tương tự như là hàm ước lượng h(n) trong giải thuật tìm kiếm A*.

2.2. Kỹ thuật lượng giá

Kỹ thuật lượng giá là một kỹ thuật quan trọng trong việc xây dựng trò chơi cở caro. Kĩ thuật này giúp cho điểm trạng thái của bàn cờ để từ đó xây dựng cây trò chơi. Việc xây dựng hàm lượng giá hợp lý, chính xác sẽ giúp cho hệ thống có đánh giá chính xác về trạng thái bàn cờ để đưa ra nước đi thông minh hơn.

Đối với bài toán cờ caro, ta có thể dùng 1 hàm lượng giá để đánh giá tính "tốt, xấu" tại 1 thời điểm. Những ô nào ở gần các quân đã đánh trước sẽ được điểm cao hơn. Những ô càng xa thì được càng ít điểm. Cụ thể:

Khởi tạo hai mảng điểm Tấn công (AScore) và Phòng thủ (Dscore):

DScore = { 0, 1, 9, 81, 729 }

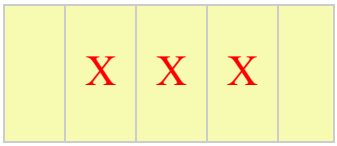
AScore = { 0, 2, 18, 162, 1458 }

Tiến hành quét một block 5 ô theo 4 hướng là dọc, ngang, chéo lên, chéo xuống, đếm số quân cờ mỗi bên và tiến hành cộng điểm cho các ô trống khi trong block 5 ô đó chỉ có toàn quân ta, hoặc toàn quân địch.

- Nếu có n quân ta thì các ô trống được cộng AScore[n] điểm.
- Nếu có n quân địch thì các ô trống được cộng DScore[n] điểm.
- Điểm của ô[i][j] trống:

Score[i][j] = Score[i][j] + Ascore[n] + Dscore[n]

VD:



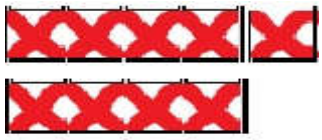
Hai ô trống sẽ có AScore[3] = 81, Dscore[0] = 0 → Score[i][j] = 81

Như thế điểm các ô trống sẽ được cộng dồn (do một ô được quét 5 lần mỗi chiều). Điều này có ý nghĩa trong việc xác định các nước đi ăn đôi, ăn ba.

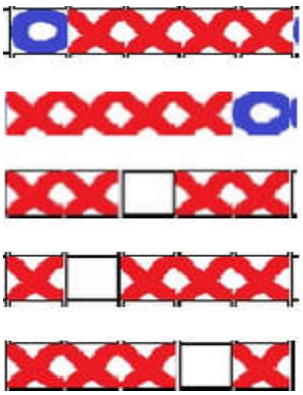
Các ô có điểm bằng nhau một cách tương đối thì việc đánh vào 2 ô đó như nhau. Nếu chọn, ta chỉ chọn một trong các nước đó để tránh lãng phí thời gian khi duyệt. Chọn 3 ô có điểm cao nhất. Sau đó sử dụng giải thuật MiniMax và Alpha-Beta cắt tỉa để tìm nước đi.

Tuy nhiên đây chỉ là Heuristic nên ta phải bổ sung thêm các Heuristic khác nữa, ví dụ vùng có 2, 3, 4 quân liên tiếp thì sẽ được cộng thêm 1 số điểm thưởng nào đó cho vùng đó dựa vào trọng số quân (tức là nhiều quân liên tiếp thì được cộng nhiều điểm thưởng hơn). Sau mỗi nước đi, hệ thống sẽ kiểm tra bàn cờ tìm các thế cờ đó rồi tùy vào độ lợi thế đã định trước để tính ra điểm. Cụ thể là:

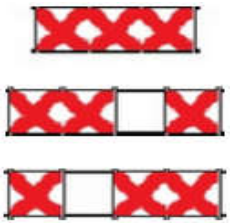
TH1: Trường hợp chắc thắng (+5000 điểm)



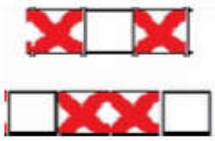
TH2: Trường hợp rất thuận lợi (+600 điểm)



TH3: Trường hợp thuận lợi (+500 điểm)



TH4: Trường hợp bình thường (+5 điểm)



2.3. Đánh giá chi phí

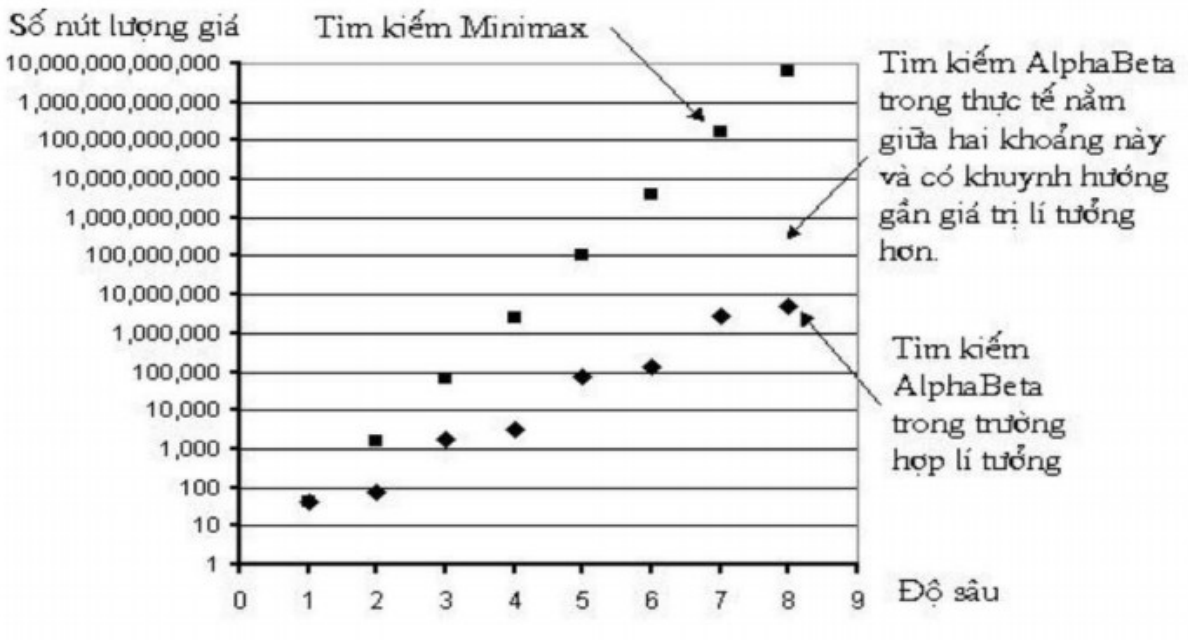
- Độ phức tạp của thuật toán MiniMax:
  + Không gian tìm kiếm: b^d (b là hệ số phân nhánh của cây hay nước đi hợp pháp tại mỗi điểm, d là độ sâu tối đa của cây).
  + Số lượng các nút được thăm là: $b(b^d-1)(b-1)$
  + Độ phức tạp thời gian: $O(b^d)$
  + Độ phức tạp không gian bộ nhớ: O(bd)
- Độ phức tạp của thuật toán Alpha-Beta:
  + Tổng số nút được thực hiện: $O(b^{3d/4})$ (b là độ rộng của cây – hệ số phân nhánh trung bình của các con, d là độ sâu của cây). Trường hợp lý tưởng số nút cần xét:
    - $2b^{d/2-1}$ với d chẵn
    - $b^{(d+1)/2} + b^{d/2} - 1$ với d lẻ

➔ Số nút thực hiện: $O(b^{d/2})$
  + Hệ số phân nhánh: $\sqrt{b}$

- So sánh giải thuật MiniMax và Alpha-Beta

Độ sâu	Minimax		AlphaBeta		Tỉ lệ số nút Minimax/Alpha-beta
	Số nút	Số lần tăng	Số nút	Số lần tăng	
1	40		40	1	
2	1600	40	79	1.9	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

Số nút phải xét giữa hai giải thuật



Khảo sát sự bùng nổ tổ hợp

- Bảng đánh giá chi phí: Hệ số phân nhánh là 3

Độ sâu	**Số nút**	**Thời gian trung bình (ms)**
1	3	136,3
3	27	328,2
5	243	354,5
11	177147	2468

III. XÂY DỰNG CÁC LỚP

1. Lớp ChessBoard

Lớp này đóng vai trò như một bàn cờ trong thực tế. Lớp này chứa các thông tin về bàn cờ:

- ***Thuộc tính:***
  - Row : Số dòng của bàn cờ
  - Col : Số cột của bàn cờ
- ***Phương thức:***
  - ChessBoard(){}: Phương thức khởi tạo mặc định
  - ChessBoard(int, int){}: Phương thức khởi tạo có tham số
  - drawChessBoard(Graphics){}: Phương thức vẽ bàn cờ
  - drawChessMan(Graphics, int, int, Image){}: Phương thức vẽ quân cờ

2. Lớp EvalBoard

Lớp này đóng vai trò lưu trữ điểm của các ô trên bàn cờ sau khi đã lượng giá

- ***Thuộc tính:***
  - Height : Chiều cao bàn cờ ứng với số dòng
  - Width: Chiều rộng bàn cờ ứng với số cột
  - Eboard: Mảng lưu trữ giá trị - điểm các ô cờ
- ***Phương thức:***
  - EvalBoard(ChessBoard){}: Phương thức khởi tạo với tham số truyền vào là bàn cờ

3. Lớp CaroChess

Lớp đóng vai trò chính của chương trình , xử lý các vấn đề của game như:

- Xử lý AI cho game: Tìm kiếm nước đi cho máy, lượng giá bàn cờ
- Lưu trữ các trạng thái của bàn cờ, thế cờ
- Xử lý việc đánh cờ, kiểm tra thắng thua – kết thúc của ván cờ

- ***Thuộc tính:***
  - ChessBoard: Đối tượng lớp bàn cờ
  - EvalBoard: Đối tượng lớp lưu trữ điểm ô cơ
  - arrBoard[][]: Mảng lưu trữ trạng thái bàn cờ
  - listUndo: Mảng ArrayList lưu trữ các nước đã đánh rồi của 2 bên
  - maxDepth: Độ sâu tối đa khi tìm kiếm nước đi
  - depth: Độ sâu hiện tại
  - playerFlag: Kiểm tra lượt đi
  - x, y : Vị trí nước đi của máy theo chiều ngang và dọc
  - SIZE: Kích thước 1 ô cờ
  - Ascore[], DScore[]: Mảng điểm tấn công và phòng thủ
  - Point[]: Mảng điểm các trường hợp nguy hiểm
- ***Phương thức:***
  - CaroChess(){}: Phương thức khởi tạo
  - drawChessBoard(Graphics) {}: Phương thức gọi đến phương thức vẽ bàn vờ của lớp ChessBoard
  - playChess(int, int, Graphics) {}: Phương thức đánh cờ

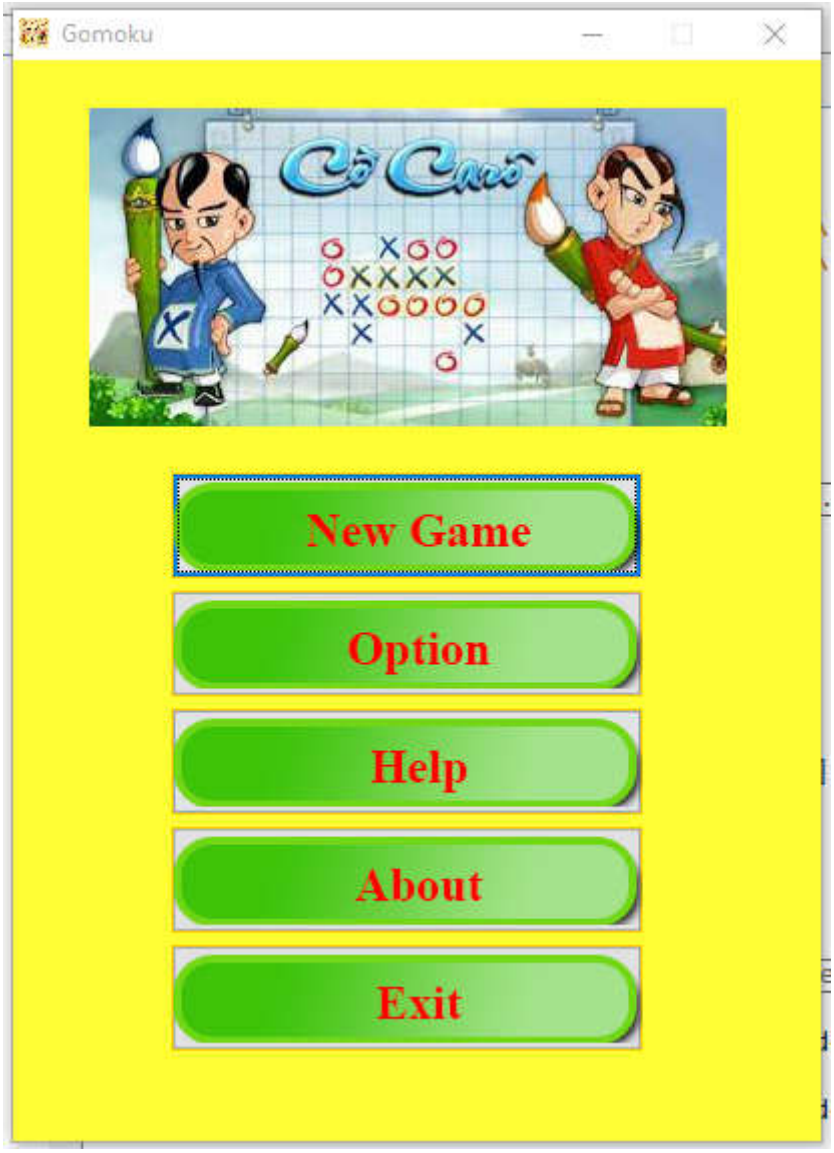
- repaintChessMan(Graphics){}: Phương thức vẽ lại quân cờ
- playerVsCom(Graphics){}: Phương thức khởi tạo để đánh cờ với máy
- resetBoard(EvalBoard){}: Phương thức gán các giá trị điểm của ô cờ về trạng thái ban đầu (tức là các ô có điểm là 0)
- evalChessBoard(int, EvalBoard, int[][]){}: Phương thức lượng giá tĩnh thế cờ bằng cách tính điểm cho các ô cờ
- evalDangerous(int[][]){}: Phương thức lượng giá các trường hợp nguy hiểm, trả về điểm của thế cờ khi thỏa mãn các trường hợp nguy hiểm
- getMaxPoint(){}: Phương thức trả về vị trí của ô có điểm lớn nhất
- findMoveOfCom (){}: Trả về vị trí ô cờ sau khi lượng giá và tìm nước đi cho máy. Đây chính là nước đi tiếp theo của máy.
- maxVal(int[][], int, int, int){}, minVal(int[][], int, int, int){}: Phương thức thể hiển giải thuật Alpha-Beta Pruning
- checkWin(int, int){}: Phương thức kiểm tra kết thúc ván cờ - thắng thua
- newGame(Graphics){}: Phương thức khởi tạo ván cờ mới
- undoGame(Graphics){}: Phương thức xử lý việc đánh lại nước đi

4. Các lớp giao diện của chương trình

- ***InterfaceGame:*** Lớp chứa giao diện đồ họa chương trình
- ***MenuGame:*** Lớp chứa các chức năng của chương trình
- ***About:*** Lớp chứa thông tinh chương trình
- ***Level:*** Lớp chọn cấp độ chơi
- ***Help:*** Lớp hướng dẫn chơi – luật chơi
- ***Main:*** Lớp chứa luồng main để thực thi chương trình

IV. GIAO DIỆN HỆ THỐNG

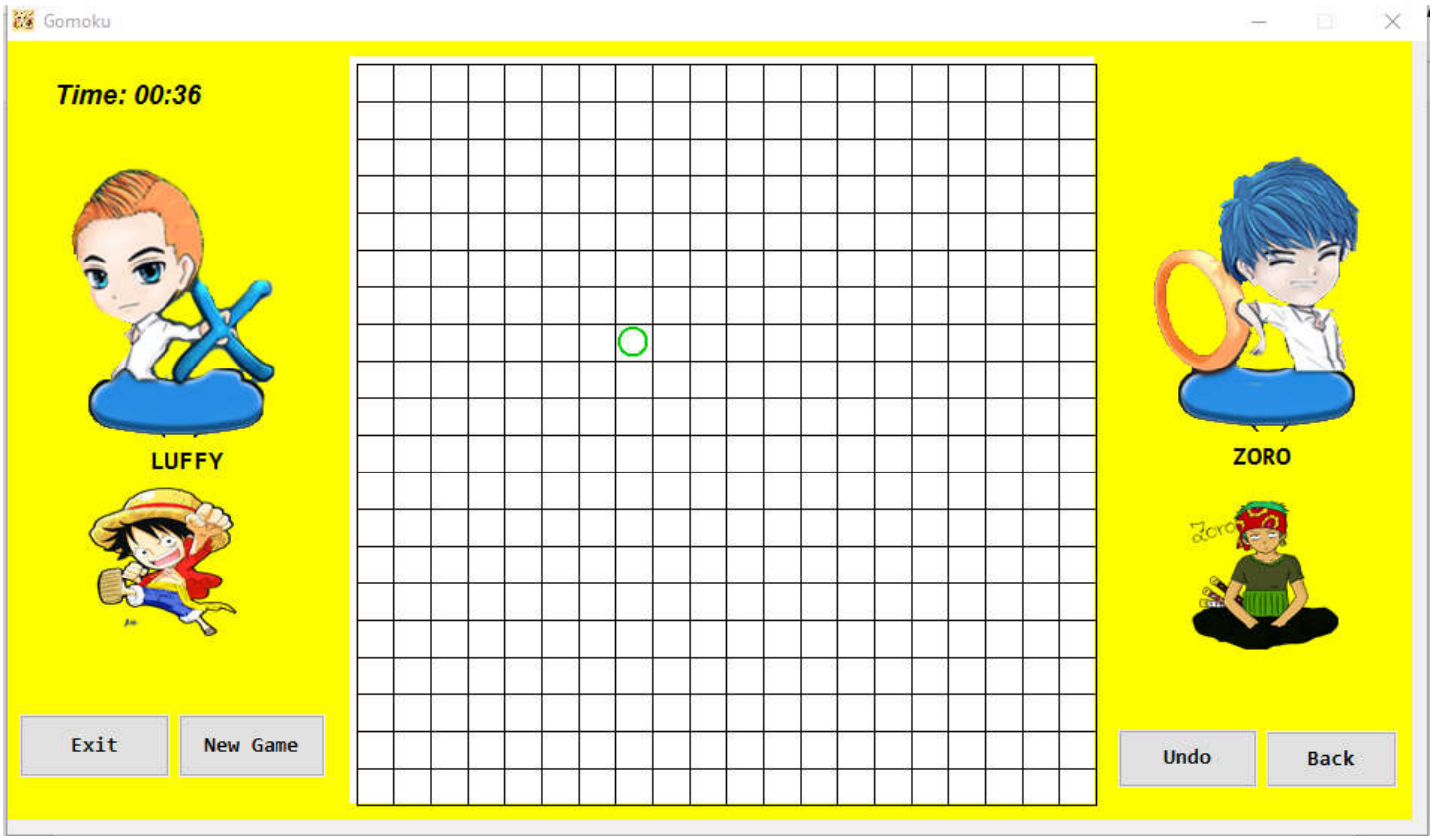
1. Giao diện Menu



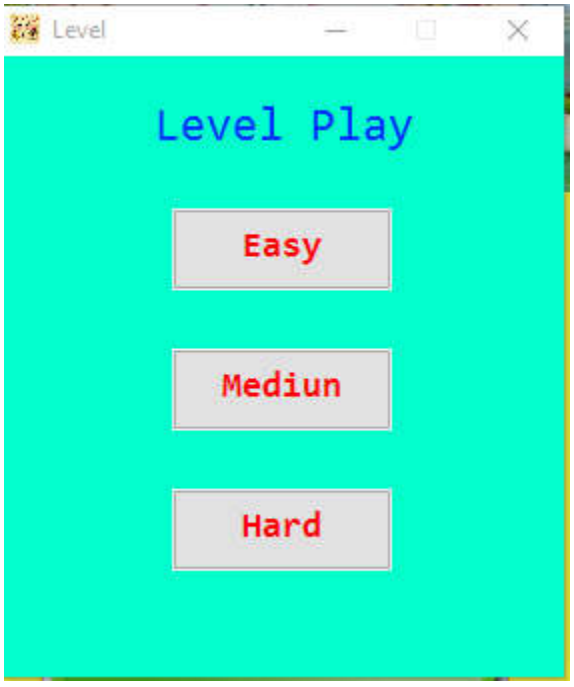
- NewGame: Chơi game
- Option: Chọn cấp độ chơi
- Help: Hướng dẫn chơi
- About: Thông tin sản phẩm
- Exit: Thoát chương trình

2. Giao diện bàn cờ

- NewGame: Chơi ván mới
- Exit: Thoát chương trình
- Undo: Đánh lại nước đi trước
- Back: Quay lại giao diện Menu



3. *Giao diện Option*



- Easy: Mức dễ
- Medium: Mức trung bình
- Hard: Mức khó

4. *Giao diện Help*

5. *Giao diện About*

TỔNG KẾT

Khó khăn và hướng giải quyết

- Trong quá trình đánh chương trình sẽ thông báo khi nhìn thấy đường có thể thắng đối thủ khoảng 90% (không thể 100% vì ta phải giới hạn chiều rộng phân nhánh để tránh các nước đi vô ích).

➔ Ta có thể dùng các biến toàn cục để kiểm tra đường thắng đó có chắc hay không, nhưng làm vậy tốn nhiều thời gian, và độ chính xác còn phụ thuộc vào cách đánh giá.

- Đối với vài trường hợp đặc biệt (chỉ xảy ra khi ta chọn đối thủ trình độ thấp). Không chặn các đường thua trước mắt… Những vị trí mà máy đánh sẽ là đường đánh thắng của đối thủ trong nước kế tiếp – Tức là máy trong tình trạng thua chắc !

➔ Ta sẽ khởi tạo mảng lưu trữ các trường hợp đặc biệt đó. Và kiểm tra khi tìm kiếm nước đi cho máy.

- Tính điểm các nước chặn đối thủ các rất nhiều tình huống trong khi việc duyệt mất khá nhiều thời gian.

➔ Sử dụng kỹ thuật cắt tỉa Alpha – Beta để tăng tốc độ duyệt MiniMax để loại bỏ các nước "tồi"

- Khởi tạo mảng điểm tấn công và phòng thủ làm sao cho hợp lí. Trong khi lượng giá bàn cờ, có những trường hợp điểm tương đương nhau nhưng 1 trường hợp lại nguy hiểm hơn hẳn trường hợp kia.

➔ Test các luật nhân điểm sao cho đúng đắn.

Kết Luận

Qua quá trình thực hiện đề tài này chúng em đã hiểu hơn về việc ứng dụng Trí tuệ nhân tạo trong việc giải quyết các vấn đề thực tế. Những điều thu được trong quá trình thực hiện:

- Hiểu thêm về hai giải thuật MiniMax và Alpha-Beta Pruning
- Áp dụng được hai thuật toán trên vào game Caro
- Hiểu và xây dựng được hàm Heuristic trong game

Mặc dù có nhiều cố gắng nhưng chắc chắn không tránh khỏi thiếu xót và hạn chế trong quá trình cài đặt chương trình. Chúng em mong nhận được sự góp ý của thầy để có thể hoàn thiện và phát triển chương trình hơn.

Hướng phát triển

- Cải thiện, nâng cao hàm lượng giá
- Xây dựng thêm một số tính năng
  + Giới hạn thời gian đánh cho người chơi
  + Gợi ý nước đi cho người chơi
  + Lưu, load ván cờ
  + Âm nhạc trong lúc đánh, lúc thắng, lúc hòa
  + Máy đánh với máy

TÀI LIỆU THAM KHẢO

1. *Báo cáo bài tập lớn môn Trí tuệ nhân tạo của nhóm sinh viên: Vũ Văn Phước, Bùi Viết Trung , Nguyễn Đình Nhu Trịnh Công Nam.*
2. *Mã nguồn source code của anh Lê Bá Khánh Trình*
3. *Một số trang web:*
   - *https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning*
   - *https://en.wikipedia.org/wiki/Minimax*