TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN MÔN HỌC
TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: CHƯƠNG TRÌNH TRÒ CHƠI CỜ CA-RÔ

GIẢNG VIÊN HƯỚNG DẪN: TS. Nguyễn Nhật Quang

NHÓM SINH VIÊN	MSSV
1.Ngô Đức Nhật	20132860
2.Phạm Văn Hùng	20131909
3.Trần Đức Anh	20130220

Hà Nội, 11/2015

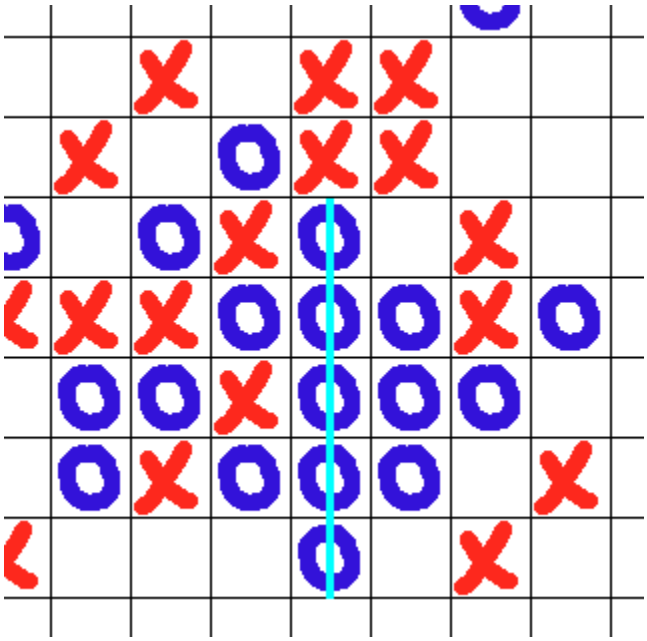
MỤC LỤC

CHƯƠNG I. GIỚI THIỆU VÀ MÔ TẢ BÀI TOÁN

I. GIỚI THIỆU

+ Trò chơi đối kháng Caro(gomoku) : Gồm 2 người chơi(hoặc người với máy), đối thủ này sẽ tìm cách dành chiến thắng trước đối thủ kia trong một số hữu hạn nước đi, mỗi nước đi được tạo ra dựa từ 1 trạng thái bất kỳ của trận đấu. Nếu sau 1 số giới hạn nước đi, nếu chưa ai dành chiến thắng thì xem như hòa

+ Caro tự do (gomoku free style) là 1 solved game, nghĩa là người đi trước sẽ thắng nếu đánh đúng(surewin), ván cờ sẽ kết thúc rất nhanh khoảng 13-15 nước, trình độ người đi sau không quan trọng. Chính vì vậy gomoku free style đã không còn được chơi nhiều mà thay thế bằng gomoku Pro, Renju, Pente cùng những biến thể khác … nhằm công bằng chia đều cho người đi trước và người đi sau



II. MÔ TẢ BÀI TOÁN

1. Mục đích

+ Viết một chương trình biết chơi cờ Caro

+ Chương trình có khả năng đánh thắng những người mới chơi, người chơi trung bình

2. Yêu cầu

+ Đưa ra thuật toán giúp giải quyết bài toán đặt ra: sao cho khả năng để máy tính thắng là lớn nhất

+Hoàn thiện chương trình, tối ưu thuật toán sao cho chương trình làm việc hiệu quả nhất

+Khắc phục lỗi phát sinh trong quá trình giải quyết bài toán

3. Các thức chơi

+ Bàn cờ có kích thước NxN (ví dụ 16x16)

+ Máy sẽ được đánh nước đầu tiên ngay giữa bàn cờ, sau đó thay phiên nhau đánh

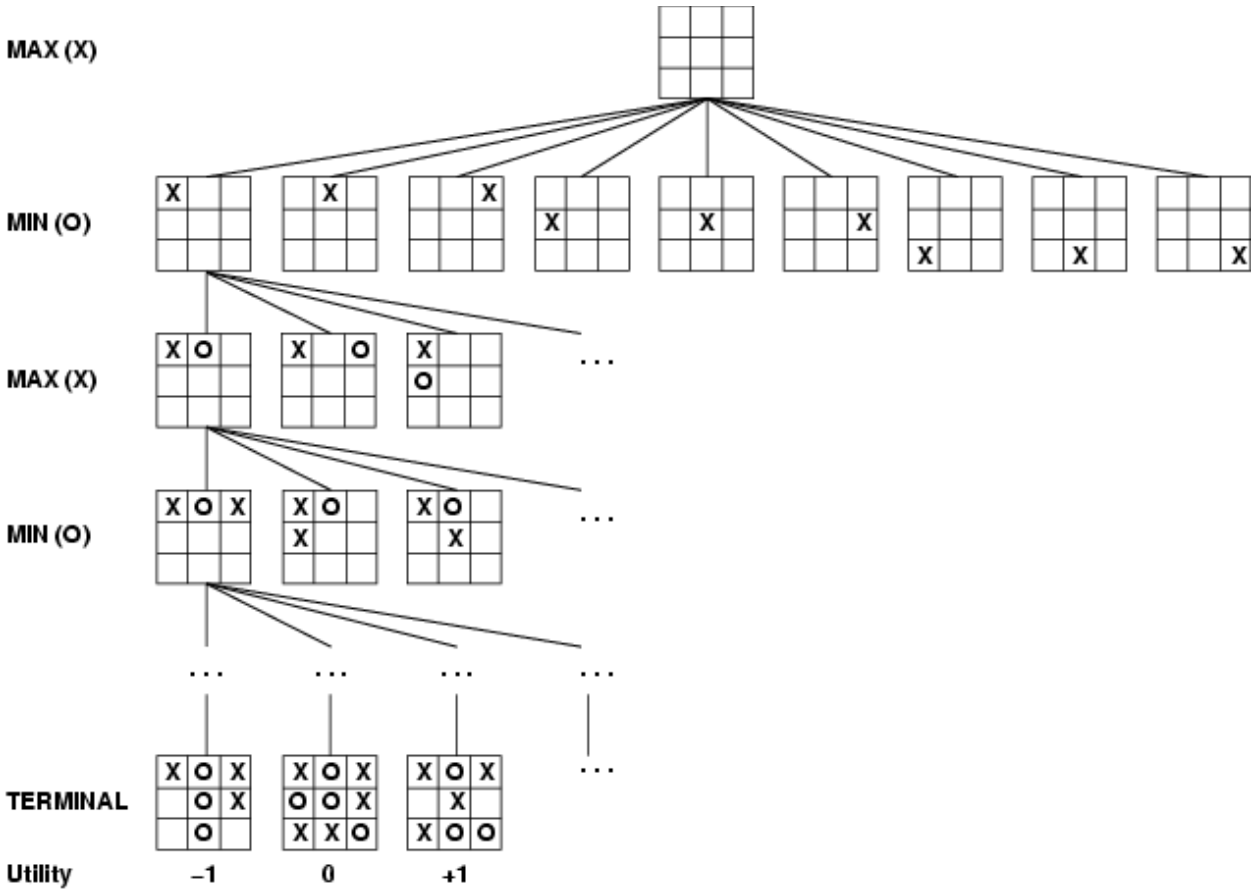
+ Nếu tồn tại đúng 5 con liên tiếp trên 1 đường, không bị chặn 2 đầu là thắng(chéo, ngang, dọc)

CHƯƠNG II. PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN

I. Tìm kiếm nước đi

1.Không gian trạng thái của trò chơi

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.

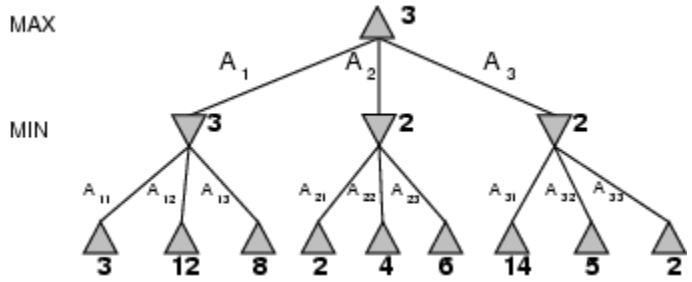


Dựa vào cái cây trò chơi được định nghĩa ở trên, việc tìm kiếm nước đi là chọn 1 nút trên cây (ở mức 1) sao cho nước đó là tốt. Theo thông thường khi chơi, một nước đi tốt hay không là phụ thuộc vào khả năng dành chiến thắng là cao hay thấp sau khi nước đi này được đi. Do đó, muốn chọn 1 nước đi tốt 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.

2. Thuật toán MINIMAX

+Chiến lược này được xác định bằng cách xét giá trị MINIMAX đối với mỗi nút trong cây biểu diễn tr. chơi.
+MAX chọn nước đi ứng với giá trị MINIMAX cực đại (để đạt được giá trị cực đại của hàm mục tiêu) đạt được giá trị cực đại của hàm mục tiêu)
+Ngược lại, MIN chọn nước đi ứng với giá trị MINIMAX cực tiểu.



Giải thuật MINIMAX

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

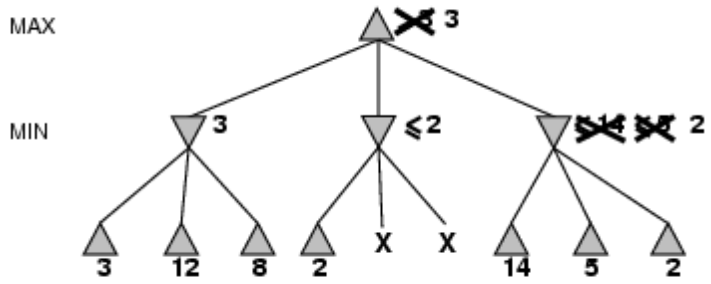
+ Giải thuật tìm kiếm MINIMAX vấp phải vấn đề bùng nổ (mức hàm mũ) các khả năng nước đi cần phải xét không phù hợp với nhiều bài toán trò chơi thực tế.

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

3. Thuật toán cắt tỉa α-β (Alpha-beta prunning)

+ 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.



Giải thuật ALPHA-BETA

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

```java
public Point search(CaroBoard bb) {
    CaroBoard b = new CaroBoard(bb.getSize());
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            b.getSquare()[i][j] = bb.getSquare()[i][j];
        }
    }
    evaluateEachSquare(b, 2);
    ArrayList<State> list = new ArrayList();
    for (int i = 0; i < maxSquare; i++) {
        list.add(getMaxSquare());
    }
    int maxp = Integer.MIN_VALUE;
    ArrayList<State> ListChoose = new ArrayList();
    for (int i = 0; i < list.size(); i++) {
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 2;
        int t = MinVal(b, list.get(i), Integer.MIN_VALUE, Integer.MAX_VALUE, 0);
        if (maxp < t) {
            maxp = t;
            ListChoose.clear();
            ListChoose.add(list.get(i));
        } else if (maxp == t) {
            ListChoose.add(list.get(i));
        }
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 0;
    }
    int x = rand.nextInt(ListChoose.size());
    return ListChoose.get(x).getP();
}
```

```java
private int MaxVal(CaroBoard b, State s, int alpha, int beta, int depth) {
    int val = evaluationBoard(b);
    if (depth >= maxDepth || Math.abs(val) > 3000) {
        return val;
    }
    evaluateEachSquare(b, 2);
    ArrayList<State> list = new ArrayList();
    for (int i = 0; i < maxSquare; i++) {
        list.add(getMaxSquare());
    }
    for (int i = 0; i < list.size(); i++) {
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 2;
        alpha = Math.max(alpha, MinVal(b, list.get(i), alpha, beta, depth + 1));
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 0;
        if (alpha > beta) {
            break;
        }
    }
    return alpha;
}
```

```
private int MinVal(CaroBoard b, State s, int alpha, int beta, int depth) {
    int val = evaluationBoard(b);
    if (depth >= maxDepth || Math.abs(val) > 3000) {
        return val;
    }
    evaluateEachSquare(b, 1);
    ArrayList<State> list = new ArrayList();
    for (int i = 0; i < maxSquare; i++) {
        list.add(getMaxSquare());
    }
    for (int i = 0; i < list.size(); i++) {
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 1;
        beta = Math.min(beta, MaxVal(b, list.get(i), alpha, beta, depth + 1));
        b.getSquare()[list.get(i).getP().x][list.get(i).getP().y] = 0;
        if (alpha >= beta) {
            break;
        }
    }
    return beta;
}
```

So sánh 2 thuật toán Minimax và alpha-beta

Độ sâu	Minimax		AlphaBeta		Tỉ lệ số nút
	Số nút	Số lần tăng	Số nút	Số lần tăng	Minimax / AlphaBeta
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

+ 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*

II Kỹ thuật lượng giá

1. Đánh giá điểm của các ô trống (chưa đánh)

Quét tất cả các block 5 ô của bàn cờ(ngang, dọc, chéo), đếm số quân của người và máy, sau đó cộng điểm cho các ô trống dựa trên số quân đếm được:

int[] defenseScore = {0, 1, 9, 85, 769};

int[] attackScore = {0, 4, 28, 256, 2308};

Ví dụ: 0 là ô trống, 1 là của người, 2 là của máy

0 1 1 2 0

thì điểm của các ô trống được tính như sau: (đánh giá với người (1))

evaluateSquare[][] += attackScore[2] + defenseScore[1];

2. Đánh giá điểm của cả bàn cờ

Trường hợp 1 : 4 điểm

+“02221”, “12220”,

Trường hợp 2 : 6 điểm

+ “0220”

Trường hợp 3 : 12 điểm

+ “020220” ,“022020” , “02220”

Trường hợp 4 : 1000 điểm

+ "0022221", "1222200","1222020","0202221","0222021", “1202220”,

"0202220", "0222020", "0222201", “1022220”, "220220", “022022” ,

Trường hợp 5: 3000 điểm

+ "022220", “2022202"

Trường hợp 6 : 10000 điểm

+ “22222”

Tiến hành quét cả bàn cờ theo chiều ngang, dọc, chéo (chéo phải, chéo trái) và lưu giá trị của từng ô vào 1 String

Phương thức count(String s, String find) để đếm số lần xuất hiện của xâu find trong xâu s, dùng Pattern và Matcher

```
public int count(String s, String find) {
        Pattern  pattern = Pattern.compile(find);
        Matcher  matcher = pattern.matcher(s);
        int i = 0;
        while (matcher.find()) {
            i++;
        }
        return i;
    }
```

CHƯƠNG III. CHỨC NĂNG VÀ CÁCH SỬ DỤNG HỆ THỐNG

Cài đặt và chạy chương trình

+ Yêu cầu : Máy tính đã cài đặt J2SE

+ Chạy trực tiếp file Caro.jar

Giao diện game

+ Bàn cờ kích thước 20x20, các ô cờ màu DARK_GRAY

+ Máy quân O màu xanh, người quân X màu đỏ

+Mặc định khi chạy chương trình máy sẽ đánh trước ở giữa bàn cờ

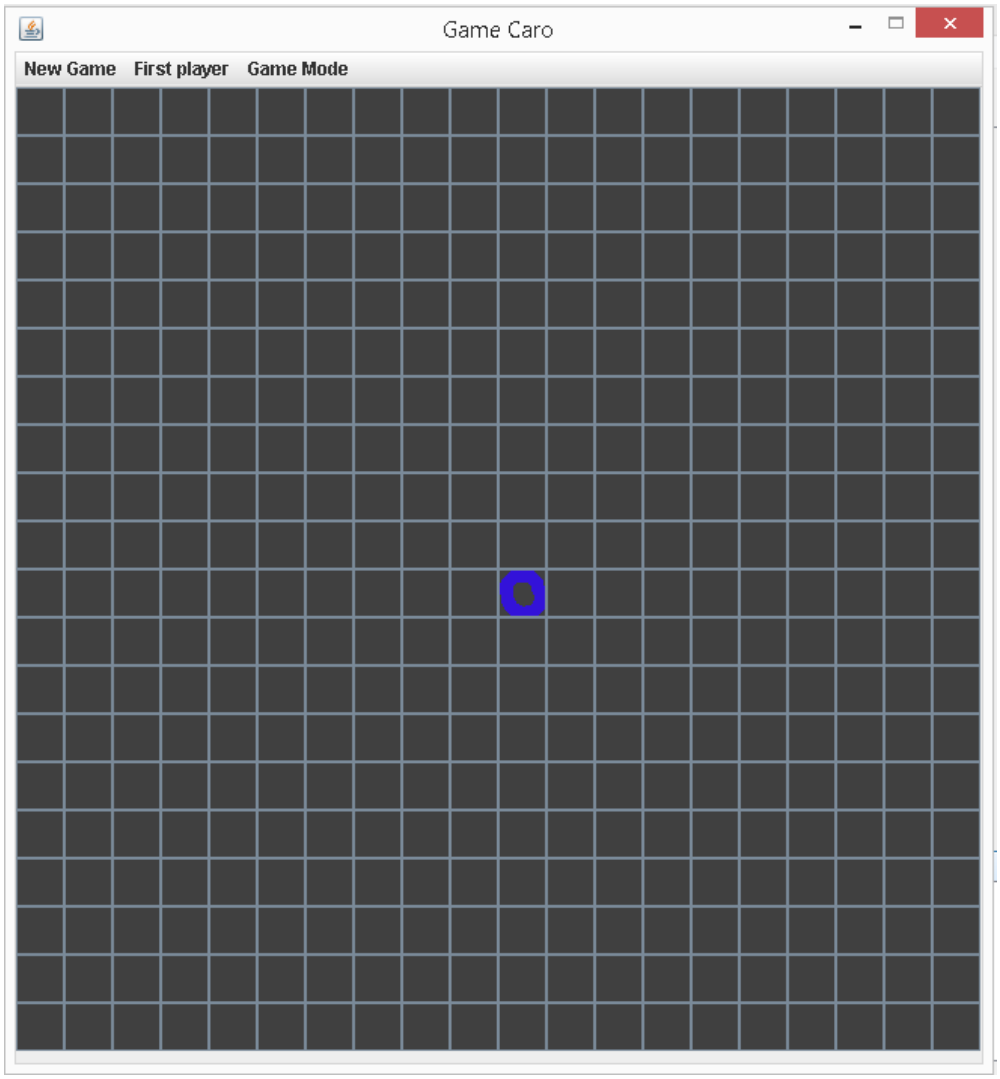
+ Có 3 menu : New Game, First player, Game Mode

New Game : Bấm chuột trái vào để chơi ván mới

First player: Chọn nước đánh đầu tiên của 1 ván(Ai : máy đánh trước, Human : người đánh trước)

Game Mode: Có 3 chế độ chơi (Easy : độ sâu 1, Medium: độ sâu 4, Hard : độ sâu 6)

Sau khi tùy chỉnh ở First player, Game Mode xong thì bấm vào New Game để chơi ván mới ứng với tùy chọn đó

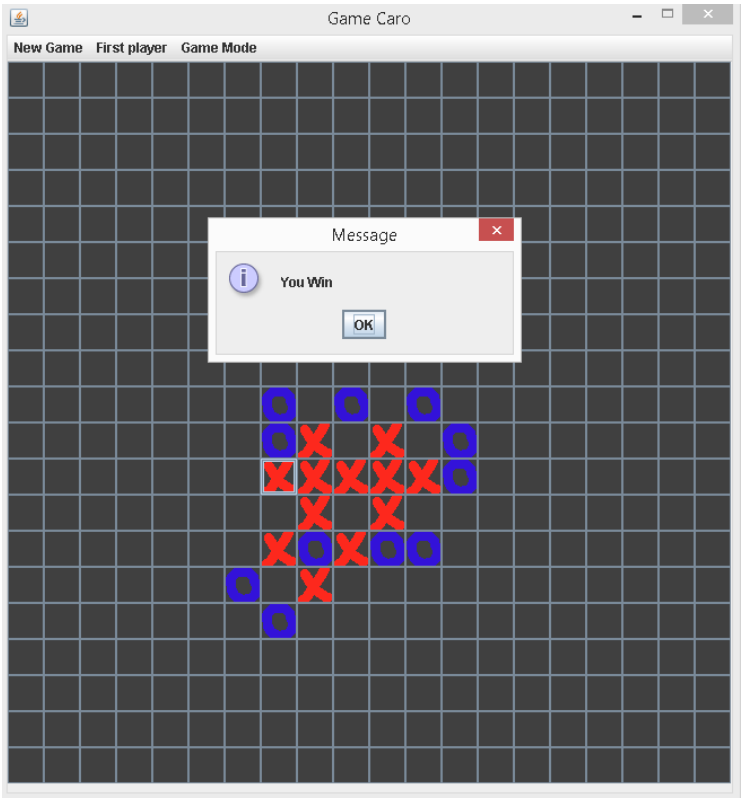


Cách chơi

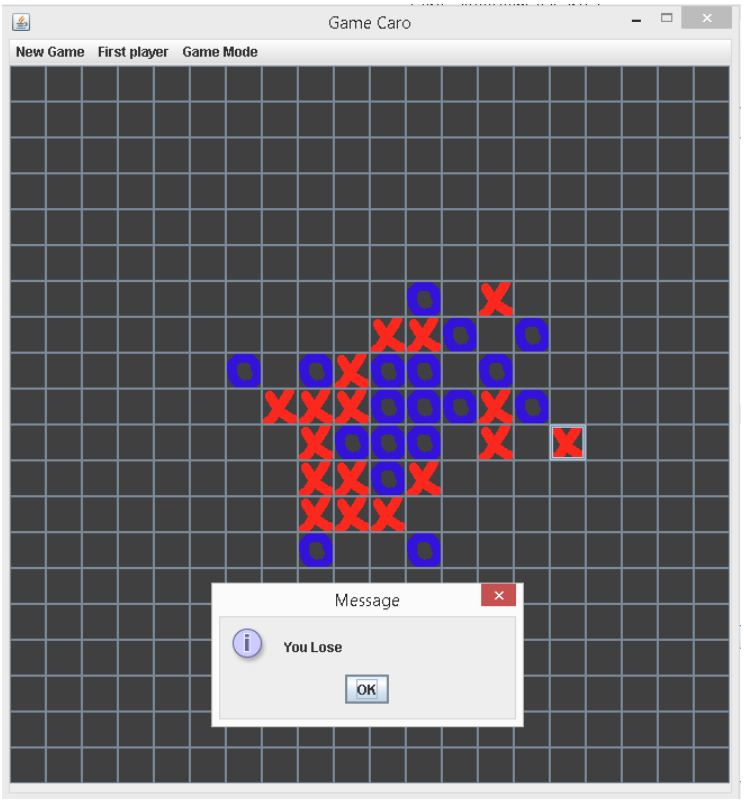
+ Để đi nước cờ nào ta ấn chuột trái vào vị trí cần đi, chỉ đi được vào những ô trống, những ô đã đánh không đi được

+ Sau khi kết thúc 1 ván chơi thì ấn vào New Game để chơi ván mới

+Ở chế độ Hard máy tính cần thời gian chạy > 1s, nên sau nước đi của người chơi cần 1 thời gian (1s < <2s) mới cập nhật nước đi của người và máy lên bàn cờ



Giao diện lúc kết thúc ván đấu



CHƯƠNG IV. CÁC PHƯƠNG PHÁP CÓ SẴN ĐƯỢC KHAI THÁC TRONG CÔNG VIỆC CỦA ĐỒ ÁN

Đánh giá các ô trống(chưa đánh) tham khảo từ

http://icetea09.com/blog/2014/01/27/co-caro-co-ai/

Như đã đề cập ở phần kỹ thuật lượng giá, việc đánh giá các ô trống được tiến hành như sau :

Quét tất cả các bàn cờ(ngang, dọc, chéo), đếm số quân của người và máy, sau đó cộng điểm cho các ô trống dựa trên số quân đếm được:

int[] defenseScore = {0, 1, 9, 85, 769};

int[] attackScore = {0, 4, 28, 256, 2308};

Ví dụ: 0 là ô trống, 1 là của người, 2 là của máy

0 1 1 2 0

thì điểm của các ô trống được tính như sau: (đánh giá với người (1))

evaluateSquare[][] += attackScore[2] + defenseScore[1];

Việc đánh giá các ô trống giúp chọn ra những ô trống có khả năng đánh vào đó cao, để giảm bớt không gian tìm kiếm khi duyệt alpha-beta

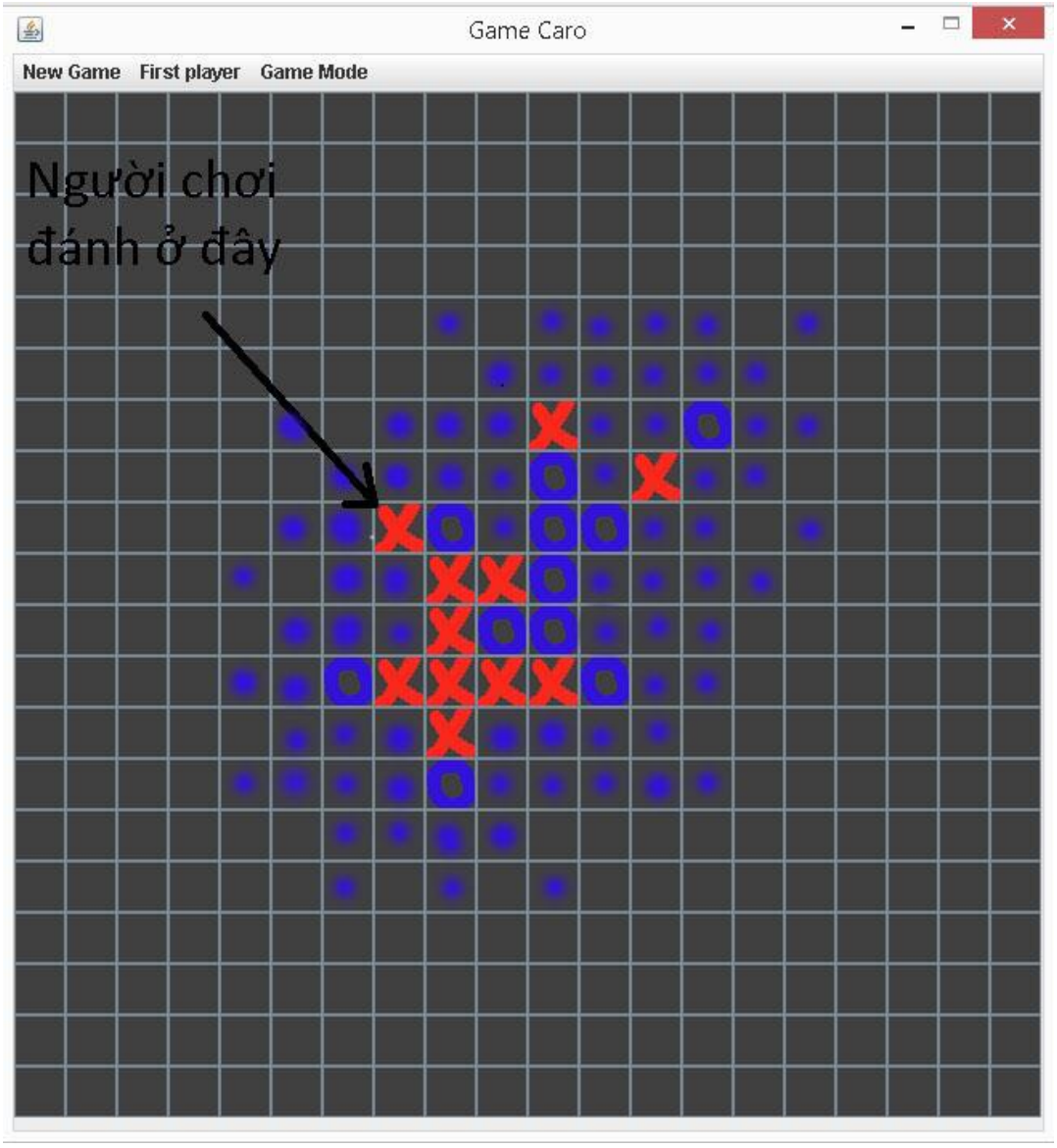
CHƯƠNG V. CÁC VẤN ĐỀ GẶP PHẢI VÀ HƯỚNG GIẢI QUYẾT

I Thuật toán AI

1.Khó khăn

Ban đầu do xét không gian tìm kiếm xung quanh những ô đã đánh bán kính 2 ô, nên không gian tìm kiếm khá lớn -> thời gian duyệt alpha-beta lâu

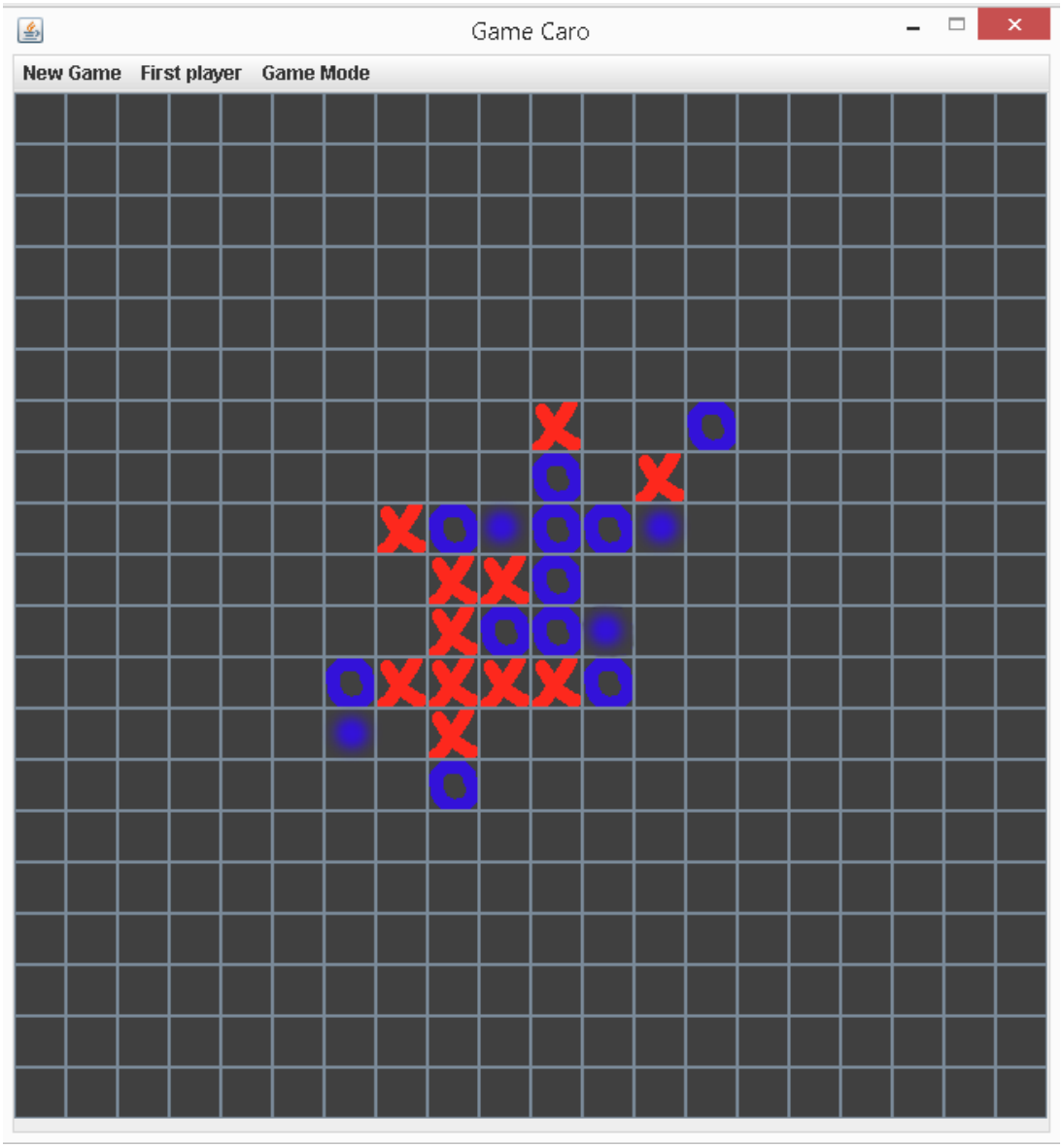
(những chấm màu xanh là những ô sẽ xét tới)



2. Hướng giải quyết

Sau khi áp dụng kỹ thuật đánh giá điểm các ô trống (đã đề cập ở các phần trước) thì giảm không gian tìm kiếm xuống, nên việc duyệt alpha-beta nhanh hơn

Cụ thể ở đây sau khi đánh giá từng ô của bàn cờ, ta chọn ra n ô có điểm cao nhất(ở đây lấy n = 4) để xét(n càng lớn thì tìm được nước đi càng tốt và thời gian duyệt lâu)



II LỖI VÀ HẠN CHẾ CỦA CHƯƠNG TRÌNH

Hiện tại chương trình vẫn chưa phát hiện lỗi

1. Hạn chế của chương trình: Trong lúc đánh nếu trên bàn cờ có nhiều quân cờ, ở 1 số trường người chơi khó theo dõi được ô máy vừa đánh vào

2. Hướng giải quyết : Viết thêm phần hiển thị tọa độ ô máy vừa đánh, hoặc tạo hiệu ứng nổi bật của ô cờ mà máy đánh vào

CHƯƠNG VI : HƯỚNG PHÁT TRIỂN CỦA BÀI TOÁN VÀ KẾT LUẬN

I. HƯỚNG PHÁT TRIỂN CỦA BÀI TOÁN

1. Về thuật toán

+ Cải thiện hàm đánh giá từng ô : thay vì đánh giá toàn bộ ô trống trên bàn cờ, sẽ đánh giá những ô trống nằm trong bán kính 2 đến 3 ô của những ô đã đánh

+ Cải thiện hàm đánh giá cả bàn cờ :

-Bổ sung thêm nhiều thế cờ hơn, tính toán lại việc cho điểm từng thế cờ 1 cách chính xác nhất.

-Thay vì chỉ quét theo chiều dọc, ngang, chéo lần lượt. Ta sẽ quét bàn cờ đồng thời theo dọc và ngang, ngang và chéo, dọc và chéo để tìm đc 1 số thế cờ kết hợp theo 2 chiều(có thể mở rộng trong việc kết hợp 3 chiều)

+ Nghiên cứu thêm 1 số thuật toán mới trong tìm kiếm có đối thủ như thuật toán: **Threat-Space Search,** để ứng dụng viết game caro người máy trên mobile cũng như máy tính

2. Về giao diện và chức năng của chương trình

+Cải thiện lại đồ họa, cũng như thêm âm thanh vào

+Thêm chế độ chơi người-người, máy-máy

+Thêm chức năng lưu lại ván đấu, chức năng đánh lại

II KẾT LUẬN

Qua môn học và trong quá trình tìm hiểu để thực hiện đề tài này, nhóm em đã có cái nhìn toàn diện hơn trong việc ứng dụng trí tuệ nhân tạo vào giải quyết vấn đề trong thực tế. Cờ caro là một trò chơi ứng dụng tốt thuật toán minimax và giải thuật alpha-beta. Tuy nhiên trong quá trình thực thi chương trình không thể tránh khỏi những sai sót và chưa thực sự tối ưu. Chúng em mong được sự góp của thầy để có thể hoàn thiện hơn trong tương lai! Chúng em xin chân thành cảm ơn !

TÀI LIỆU THAM KHẢO

Bài giảng Trí tuệ nhân tạo – thầy Nguyễn Nhật Quang

http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

http://www.mimuw.edu.pl/~awojna/SID/referaty/Go-Moku.pdf