MỤC LỤC

[LỜI NÓI ĐẦU 2](#_Toc363439363)

[I.YÊU CẦU BÀI TOÁN 3](#_Toc363439364)

[II.PHÂN TÍCH GIẢI QUYẾT BÀI TOÁN 3](#_Toc363439365)

[1. Phân tích yêu cầu 3](#_Toc363439366)

[2. Phương pháp giải quyết 3](#_Toc363439367)

[2.1 Tìm kiếm nước đi 3](#_Toc363439368)

[2.2 Kỹ thuật lượng giá 8](#_Toc363439369)

[3. Xây dựng các lớp 9](#_Toc363439370)

[4. Giao diện và cách chơi 25](#_Toc363439371)

[KẾT LUẬN 27](#_Toc363439372)

# LỜI NÓI ĐẦU

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 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 vậychúng em đã chọn đề tài Làm game cờ 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 cũng như kỹ năng lập trình ngôn ngữ Java.

Chúng em cũng xin cám ơn sự hướng dẫn tận tình của thầy Phạm Văn Hải, cả về chuyên môn cũng như định hướng. Vì kiến thức còn hạn hẹp nên trong quá trình thực hiện đề tài không thể tránh khỏi thiếu sót. Vì vậy rất mong nhận được nhận sự góp ý của thầy để đề tài có thể hoàn thiện hơn nữa.

1. **YÊU CẦU BÀI TOÁN**

Xây dựng một bàn cờ có kẻ các ô vuông với kích thước 25x25. Có 2 quân cờ là X và O.

Người chơi có thể đánh với máy. Người thắng là người đi được 5 quân cờ cùng kiểu trên hang dọc, hàng ngang hoặc đường chéo. Hai người hoà nhau khi bàn cờ hết chỗ đánh mà vẫn chưa phân được thắng bại

1. **PHÂN TÍCH GIẢI QUYẾT BÀI TOÁN**
2. **Phân tích yêu cầu**

*Mô phỏng bàn cờ*

Bàn cờ (Board) bao gồm các ô cờ ( Pos) được đặt trong một mảng 2 chiều (kích thước a x b)

Trong mỗi Pos có thể xác định được:

Vị trí pos ( Row, collumme)

Trạng thái pos (Status) Bao gồm đang trống (0) nước đi của đối thủ (2) hoặc nước đi của máy (1)

Độ nguy hiểm của ô cờ tuỳ theo trạng thái pos và có thể thay đổi được.

*Đánh giá giá trị các pos*

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 nhưng cụ thể hơn bằng các con số.

1. **Phương pháp giải quyết**

### 2.1 Tìm kiếm nước đi

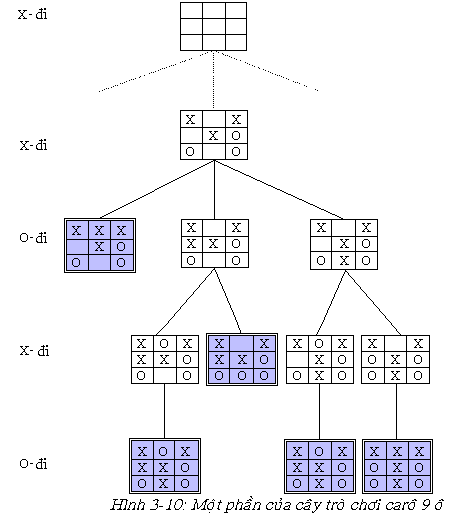
*Giới thiệu về không gian tìm kiếm*

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 đuợc 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 đuợc 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.

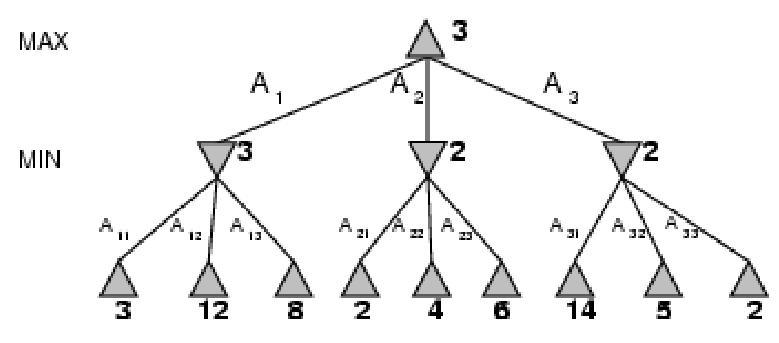
*Chiến lược minimax để tìm kiếm nước đi*

Chiến lược này được xác định bằng cách xét giá trị MINIMAX đối với mỗi nút trong cây biểu diễn trò chơi.

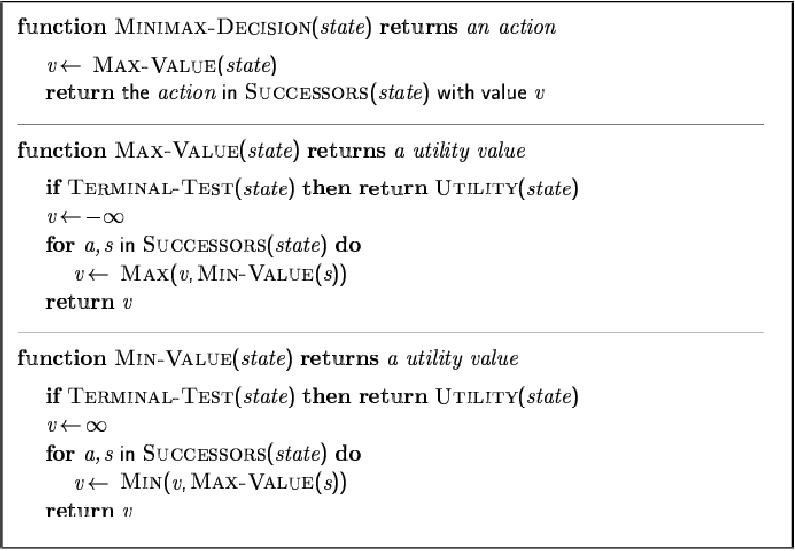
MAX chọn nước đi ứng với giá trị MINIMAX cực đại (để đạt đượcgiátrị cực đạicủa hàm mục tiêu) đạt được giá trị cực đại của hàm mục tiêu)

Ngược lại, MIN chọn nước đi ứng với giá trị MINIMAX cực tiểu.

Ví dụ:



Giải thuật minimax



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

*Phương pháp cắt tỉa α-β (Alpha-beta prunning)*

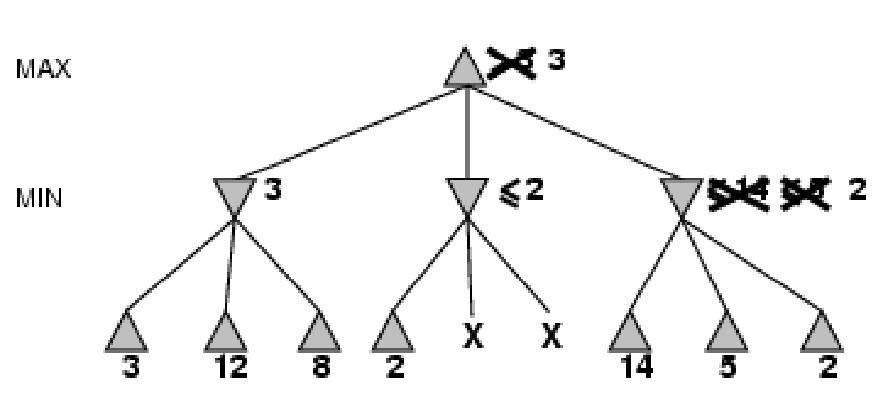
Ý tưởng: Nếu một nhánh tìm kiếm nào đó không thể cải thiện đối với giá trị

(hàm tiện ích) mà chúng ta đã có, thì không cần xét đến nhánh tìm kiếm đónữa!

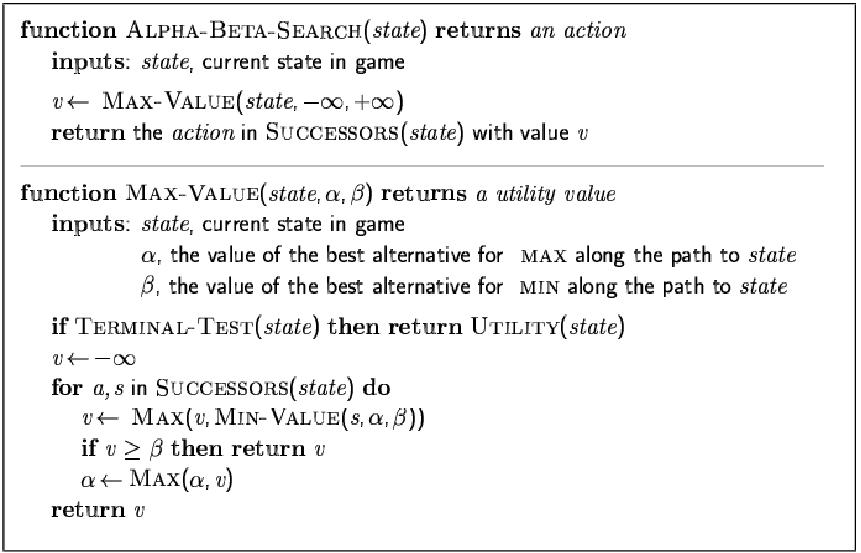
Việc cắt tỉa các nhánh tìm kiếm (“tồi”) không ảnh hưởng đến kết quả cuối cùng α là giá trị của nước đi tốt nhất đối với MAX (giá trị tối đa) tính đến hiện tại đối với nhánh tìm kiếm. Nếu v là giá trị tồi hơn α, MAX sẽ bỏ qua nước đi ứng với

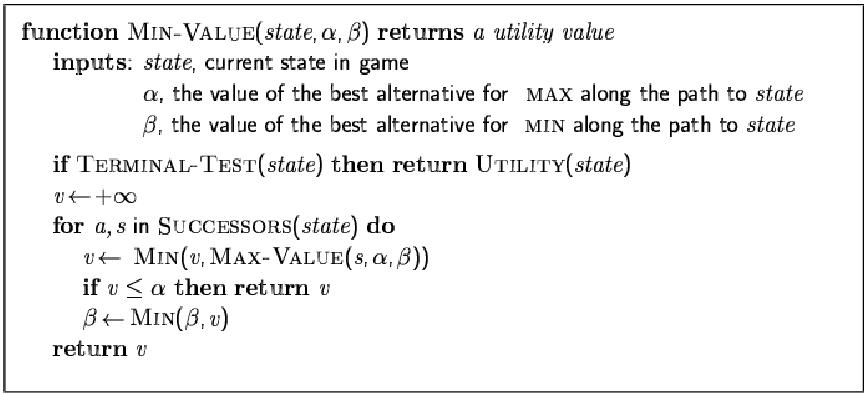
v -> Cắt tỉa nhánh ứng với v β được định nghĩa tương tự đối với MIN..

Ví dụ :

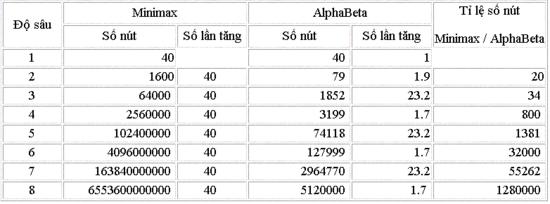


Giải thuật alpha – beta





So sánh số nút phải xét giữa 2 thuật toán Minimax và *α-β* :



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. 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)

{0, 1, 1, 1, 1}, {1, 0, 1, 1, 1}, {1, 1, 0, 1, 1},

\* TH2: Trường hợp thuận lợi (+585 điểm)

{0, 0, 1, 1, 1, 0}, {0, 1, 0, 1, 1, 0}, {1, 0, 1, 0, 1, 0, 1}

\*TH3: ( +73 )

{0, 1, 1, 1, 0}, {0, 0, 1, 1, 1}, {0, 1, 0, 1, 1},

{0, 1, 1, 0, 1}, {0, 0, 1, 1, 1}, {1, 0, 1, 1, 0},

{1, 0, 0, 1, 1}, {1, 0, 1, 0, 1}, {1, 0, 0, 1, 1},

{0, 1, 0, 1, 1}

\*TH3: Trường hợp bình thường (+9 điểm)

{0, 0, 1, 1, 0, 0}, {0, 1, 0, 1, 0, 0},

{0, 1, 1, 0, 0, 0}, {0, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 0, 0},

{0, 1, 0, 0, 1, 0}

1. **Xây dựng các lớp**

* ***Lớp CaroBoard***

Biểu diễn các trạng thái của bàn cờ Caro, danh sách các nước đánh. Cập nhật

tìm điểm đánh cho người chơi "class Player". Có thể coi như thành phần lưu trữ thông tin của class Player

-*public void updateStateOfBoard(Point point):* cập nhật trạng thái bàn cờ. Bỏ điểm đã đánh ra khỏi các đỉnh chưa đánh, cập nhật các điểm còn lại:

public void updateStateOfBoard(Point point) {

// Bo diem da danh ra khoi danh sach cac dinh chua danh

MarkNonTickedOfOnePlayer[point.x][point.y] = 0;

MarkNonTickedOfTwoPlayer[point.x][point.y] = 0;

// Cap nhat lai danh sach cac o chua danh

// Phan cap nhat nay can sua sau cho nhanh hon

numberNonTicked = 0;

for (int i = 0; i < number\_step; i++) {

for (int j = 0; j < 8; j++) {

if (board[listAreaTicked[i].x + dx[j]][listAreaTicked[i].y + dy[j]] == 0) {

Point p = new Point(listAreaTicked[i].x + dx[j], listAreaTicked[i].y + dy[j]);

int k;

for (k = 0; k < numberNonTicked; k++) {

Point p2 = listAreaNonTicked[k];

if (p.x == p2.x && p.y == p2.y) {

break;

}

}

if (k >= numberNonTicked) {

listAreaNonTicked[numberNonTicked++] = p;

}

}

}

}

// Cap nhat lai diem cua 32 o xung quanh o vua danh

// Tim cac o duong doc

for (int j = 0; j < 11; j++) {

if ((point.x + stepValue[j] < rows) && (point.x + stepValue[j] >= 0) && board[point.x + stepValue[j]][point.y] == 0) {

updateMarkOfPoint(new Point(point.x + stepValue[j], point.y));

}

}

// Tim cac o duong ngang

for (int j = 0; j < 11; j++) {

if ((point.y + stepValue[j] < cols) && (point.y + stepValue[j] >= 0) && board[point.x][point.y + stepValue[j]] == 0) {

updateMarkOfPoint(new Point(point.x, point.y + stepValue[j]));

}

}

// Tim cac o duong cheo chinh

for (int j = 0; j < 11; j++) {

if (point.x + stepValue[j] < rows && point.x + stepValue[j] >= 0

&& point.y + stepValue[j] < cols && point.y + stepValue[j] >= 0

&& board[point.x + stepValue[j]][point.y + stepValue[j]] == 0) {

updateMarkOfPoint(new Point(point.x + stepValue[j], point.y + stepValue[j]));

}

}

// Tim cac o duong cheo phu

for (int j = 0; j < 11; j++) {

if (point.x + stepValue[10 - j] < rows && point.x + stepValue[10 - j] >= 0

&& point.y + stepValue[j] < cols && point.y + stepValue[j] >= 0

&& board[point.x + stepValue[10 - j]][point.y + stepValue[j]] == 0) {

updateMarkOfPoint(new Point(point.x + stepValue[10 - j], point.y + stepValue[j]));

}

}

}

-*public int getMarkOfPointByPlayer(Point p, int player):* tínhđiểm cho một pos ( mộtô trên bàn cờ ):

public int getMarkOfPointByPlayer(Point p, int player) {

int mark = 0;

// Tim cac o duong doc

for (int j = 0; j < 11; j++) {

StepChess[j] = ((p.x + stepValue[j] < rows) && (p.x + stepValue[j] >= 0) ? board[p.x + stepValue[j]][p.y] : -1);

}

mark += ChessMark.MarkOfChessArea(StepChess, player);

// Tim cac o duong ngang

for (int j = 0; j < 11; j++) {

StepChess[j] = ((p.y + stepValue[j] < cols) && (p.y + stepValue[j] >= 0) ? board[p.x][p.y + stepValue[j]] : -1);

}

mark += ChessMark.MarkOfChessArea(StepChess, player);

// Tim cac o duong cheo chinh

for (int j = 0; j < 11; j++) {

StepChess[j] = ((p.x + stepValue[j] < rows && p.x + stepValue[j] >= 0) && (p.y + stepValue[j] < cols && p.y + stepValue[j] >= 0) ? board[p.x + stepValue[j]][p.y + stepValue[j]] : -1);

}

mark += ChessMark.MarkOfChessArea(StepChess, player);

// Tim cac o duong cheo phu

for (int j = 0; j < 11; j++) {

StepChess[j] = ((p.x + stepValue[10 - j] < rows && p.x + stepValue[10 - j] >= 0) && (p.y + stepValue[j] < cols && p.y + stepValue[j] >= 0) ? board[p.x + stepValue[10 - j]][p.y + stepValue[j]] : -1);

}

mark += ChessMark.MarkOfChessArea(StepChess, player);

// Tinh toan nuoc doi

if (mark >= 2 \* ChessMark.MARK\_2 && mark < ChessMark.MARK\_3) {

mark = ChessMark.MARK\_3 + 4 \* ChessMark.MARK\_1;

} else if (mark > (ChessMark.MARK\_3 + ChessMark.MARK\_2) && mark < ChessMark.WIN\_MARK) {

mark = ChessMark.WIN\_MARK;

}

return mark;

}

-*public Point getPointMaxMark(int diff, int defendOrAttack):* Chọn ô có điểm lớn nhất

public Point getPointMaxMark(int diff, int defendOrAttack) {

if (number\_step == 0) {

return new Point(cols / 2, rows / 2);

}

int pos = selectPoint(diff, defendOrAttack);

return listAreaNonTicked[pos];

//return null;

}

-*public float getAttackDefend(Point point, int player):* Tính tỉ lệ điểm của máy và tỉ lệ điểm của người chơi nếuđánh thử vào ô point một quân player.

public float getAttackDefend(Point point, int player) {

int decreaseMark = 0;

int increaseMark = 0;

board[point.x][point.y] = player;

// Tim cac o duong doc

for (int j = 0; j < 11; j++) {

if ((point.x + stepValue[j] < rows) && (point.x + stepValue[j] >= 0) && board[point.x + stepValue[j]][point.y] == 0) {

Point p = new Point(point.x + stepValue[j], point.y);

increaseMark = increaseMark + getMarkOfPointByPlayer(p, 1) - MarkNonTickedOfOnePlayer[p.x][p.y];

decreaseMark = decreaseMark + MarkNonTickedOfTwoPlayer[p.x][p.y] - getMarkOfPointByPlayer(p, 2);

}

}

// Tim cac o duong ngang

for (int j = 0; j < 11; j++) {

if ((point.y + stepValue[j] < cols) && (point.y + stepValue[j] >= 0) && board[point.x][point.y + stepValue[j]] == 0) {

Point p = new Point(point.x, point.y + stepValue[j]);

increaseMark = increaseMark + getMarkOfPointByPlayer(p, 1) - MarkNonTickedOfOnePlayer[p.x][p.y];

decreaseMark = decreaseMark + MarkNonTickedOfTwoPlayer[p.x][p.y] - getMarkOfPointByPlayer(p, 2);

}

}

// Tim cac o duong cheo chinh

for (int j = 0; j < 11; j++) {

if (point.x + stepValue[j] < rows && point.x + stepValue[j] >= 0

&& point.y + stepValue[j] < cols && point.y + stepValue[j] >= 0

&& board[point.x + stepValue[j]][point.y + stepValue[j]] == 0) {

Point p = new Point(point.x + stepValue[j], point.y + stepValue[j]);

increaseMark = increaseMark + getMarkOfPointByPlayer(p, 1) - MarkNonTickedOfOnePlayer[p.x][p.y];

decreaseMark = decreaseMark + MarkNonTickedOfTwoPlayer[p.x][p.y] - getMarkOfPointByPlayer(p, 2);

}

}

// Tim cac o duong cheo phu

for (int j = 0; j < 11; j++) {

if (point.x + stepValue[10 - j] < rows && point.x + stepValue[10 - j] >= 0

&& point.y + stepValue[j] < cols && point.y + stepValue[j] >= 0

&& board[point.x + stepValue[10 - j]][point.y + stepValue[j]] == 0) {

Point p = new Point(point.x + stepValue[10 - j], point.y + stepValue[j]);

increaseMark = increaseMark + getMarkOfPointByPlayer(p, 1) - MarkNonTickedOfOnePlayer[p.x][p.y];

decreaseMark = decreaseMark + MarkNonTickedOfTwoPlayer[p.x][p.y] - getMarkOfPointByPlayer(p, 2);

}

}

board[point.x][point.y] = 0;

return (increaseMark + 1) / (decreaseMark + 1);

}

-*public int selectPoint(int diff, int defendOrAttack):* Chọn mộtô (pos) trong số cácô trống đểđánh, ô này thỏa mãn giải thuật Minimax.

public int selectPoint(int diff, int defendOrAttack) {

int max1 = -1;

int max2 = -1;

for (int i = 0; i < numberNonTicked; i++) {

Point p = listAreaNonTicked[i];

if (max1 == -1 || MarkNonTickedOfOnePlayer[p.x][p.y] > MarkNonTickedOfOnePlayer[listAreaNonTicked[max1].x][listAreaNonTicked[max1].y]) {

max1 = i;

}

if (max2 == -1 || MarkNonTickedOfTwoPlayer[p.x][p.y] > MarkNonTickedOfTwoPlayer[listAreaNonTicked[max2].x][listAreaNonTicked[max2].y]) {

max2 = i;

}

}

Point p1 = listAreaNonTicked[max1];

Point p2 = listAreaNonTicked[max2];

if (MarkNonTickedOfOnePlayer[p1.x][p1.y] >= MarkNonTickedOfTwoPlayer[p2.x][p2.y]

&& MarkNonTickedOfOnePlayer[p1.x][p1.y] >= ChessMark.WIN\_MARK) {

return max1;

}

if (MarkNonTickedOfOnePlayer[p1.x][p1.y] < MarkNonTickedOfTwoPlayer[p2.x][p2.y]

&& MarkNonTickedOfTwoPlayer[p2.x][p2.y] >= ChessMark.WIN\_MARK) {

// Tim nuoc ma co kha nang chan duoc nuoc tan cong cua doi phuong

int max = MarkNonTickedOfTwoPlayer[p2.x][p2.y];

return getPosition(max);

}

return getNormalPos(diff, defendOrAttack);

}

-*public int getNormalPos(int diff, int DefendOrAttack):* Đánh nướcđi nếuđiểm các pos bình thường

public int getNormalPos(int diff, int DefendOrAttack) {

int[] candidates = new int[numberNonTicked];

int i, j;

int temp;

Point p1, p2;

boolean isAttack = true;

for (i = 0; i < numberNonTicked; i++) {

candidates[i] = i;

}

if (DefendOrAttack > 2) {

// Lay phong thu

isAttack = false;

for (i = 0; i < numberNonTicked - 1; i++) {

p1 = listAreaNonTicked[i];

for (j = i + 1; j < numberNonTicked; j++) {

p2 = listAreaNonTicked[j];

if (MarkNonTickedOfTwoPlayer[p1.x][p1.y] < MarkNonTickedOfTwoPlayer[p2.x][p2.y]) {

temp = candidates[i];

candidates[i] = candidates[j];

candidates[j] = temp;

}

}

}

} else {

// Lay tan cong

isAttack = true;

for (i = 0; i < numberNonTicked - 1; i++) {

p1 = listAreaNonTicked[i];

for (j = i + 1; j < numberNonTicked; j++) {

p2 = listAreaNonTicked[j];

if (MarkNonTickedOfOnePlayer[p1.x][p1.y] < MarkNonTickedOfOnePlayer[p2.x][p2.y]) {

temp = candidates[i];

candidates[i] = candidates[j];

candidates[j] = temp;

}

}

}

}

// Lay do kho theo khoang

int level;

switch (diff) {

case 0:

case 1:

case 2:

level = 0;

break;

case 3:

case 4:

level = 1;

break;

case 5:

case 6:

level = 2;

break;

default:

level = 2;

break;

}

// Tim muc diem tuong ung voi level

int max = -1;

for (i = 0; i < numberNonTicked; i++) {

p1 = listAreaNonTicked[candidates[i]];

if (isAttack && max < MarkNonTickedOfOnePlayer[p1.x][p1.y]) {

max = MarkNonTickedOfOnePlayer[p1.x][p1.y];

level--;

}

if (!isAttack && max < MarkNonTickedOfTwoPlayer[p1.x][p1.y]) {

max = MarkNonTickedOfTwoPlayer[p1.x][p1.y];

level--;

}

if (level < 0) {

break;

}

}

if (isAttack) {

return getPostOnePlayer(max, diff, DefendOrAttack);

} else {

return getPostOfTwoPlayer(max, diff, DefendOrAttack);

}

}

-*public int getPostOnePlayer(int max, int diff, int defendOrAttack):* chọn nướcđi ở thế cơ tấn công

public int getPostOnePlayer(int max, int diff, int defendOrAttack) {

float min = 1000000;

int pos = 0;

int[] postions = new int[numberNonTicked];

int counter = 0;

int playercounter;

int spaces = -1;

for (int i = 0; i < numberNonTicked; i++) {

Point p = listAreaNonTicked[i];

if (MarkNonTickedOfOnePlayer[p.x][p.y] == max) {

float temp = getAttackDefend(p, 1);

if (temp < min) {

min = temp;

pos = i;

counter = 0;

postions[counter++] = i;

} else if (temp == min) {

postions[counter++] = i;

}

}

}

// Xet ve khong gian 0 + 2 - 1

// Lay max ve khong gian

if (diff % 2 == 0) {

return postions[0];

}

for (int i = 0; i < counter; i++) {

Point p = listAreaNonTicked[postions[i]];

// Xet khong gian

playercounter = 0;

for (int j = 0; j < 8; j++) {

if (board[p.x + dx[j]][p.y + dy[j]] == 2) {

playercounter++;

}

}

if (spaces < (8 - playercounter)) {

spaces = 8 - playercounter;

pos = postions[i];

}

}

return pos;

}

-*public int getPostOfTwoPlayer(int max, int diff, int defendOrAttack):*Chọn nướcđi ở thế cờ phòng thủ

public int getPostOfTwoPlayer(int max, int diff, int defendOrAttack) {

float min = 1000000;

int pos = 0;

int[] postions = new int[numberNonTicked];

int counter = 0;

int playercounter;

int spaces = -1;

for (int i = 0; i < numberNonTicked; i++) {

Point p = listAreaNonTicked[i];

if (MarkNonTickedOfTwoPlayer[p.x][p.y] == max) {

float temp = getAttackDefend(p, 1);

if (temp < min) {

min = temp;

pos = i;

counter = 0;

postions[counter++] = i;

} else if (temp == min) {

postions[counter++] = i;

}

}

}

// Xet ve khong gian 0 + 2 - 1

// Lay max ve khong gian

if (diff % 2 == 0) {

return postions[0];

}

for (int i = 0; i < counter; i++) {

Point p = listAreaNonTicked[postions[i]];

// Xet khong gian

playercounter = 0;

for (int j = 0; j < 8; j++) {

if (board[p.x + dx[j]][p.y + dy[j]] == 1) {

playercounter++;

}

}

if (spaces < (8 - playercounter)) {

spaces = 8 - playercounter;

pos = postions[i];

}

}

return pos;

}

-*public int getPosition(int max):* chọn nướcđi để chặn người chơi không thắng được

public int getPosition(int max) {

float min = 1000000;

int pos = 0;

int[] postions = new int[numberNonTicked];

int counter = 0;

int playercounter;

int spaces = -1;

for (int i = 0; i < numberNonTicked; i++) {

Point p = listAreaNonTicked[i];

if (MarkNonTickedOfTwoPlayer[p.x][p.y] == max) {

float temp = getAttackDefend(p, 1);

if (temp < min) {

min = temp;

pos = i;

counter = 0;

postions[counter++] = i;

} else if (temp == min) {

postions[counter++] = i;

}

}

}

// Xet ve khong gian 0 + 2 - 1

// Lay max ve khong gian

for (int i = 0; i < counter; i++) {

Point p = listAreaNonTicked[postions[i]];

// Xet khong gian

playercounter = 0;

for (int j = 0; j < 8; j++) {

if (board[p.x + dx[j]][p.y + dy[j]] == 1) {

playercounter++;

}

}

if (spaces < (8 - playercounter)) {

spaces = 8 - playercounter;

pos = postions[i];

}

}

return pos;

}

-*public int checkHasWiner(Point p):* kiểm tra xem ai đã giành phần thắng

public int checkHasWiner(Point p) {

if (number\_step < 9) {

return 0;

}

int player = board[p.x][p.y];

// Check hang ngang

if (board[p.x + 1][p.y] == player && board[p.x + 2][p.y] == player && board[p.x + 3][p.y] == player

&& board[p.x + 4][p.y] == player) {

return player;

}

if (board[p.x + 1][p.y] == player && board[p.x + 2][p.y] == player && board[p.x + 3][p.y] == player

&& board[p.x - 1][p.y] == player) {

return player;

}

if (board[p.x + 1][p.y] == player && board[p.x + 2][p.y] == player && board[p.x - 1][p.y] == player

&& board[p.x - 2][p.y] == player) {

return player;

}

if (board[p.x + 1][p.y] == player && board[p.x - 1][p.y] == player && board[p.x - 2][p.y] == player

&& board[p.x - 3][p.y] == player) {

return player;

}

if (board[p.x - 1][p.y] == player && board[p.x - 2][p.y] == player && board[p.x - 3][p.y] == player

&& board[p.x - 4][p.y] == player) {

return player;

}

// Check hang doc

if (board[p.x][p.y + 1] == player && board[p.x][p.y + 2] == player && board[p.x][p.y + 3] == player

&& board[p.x][p.y + 4] == player) {

return player;

}

if (board[p.x][p.y + 1] == player && board[p.x][p.y + 2] == player && board[p.x][p.y + 3] == player

&& board[p.x][p.y - 1] == player) {

return player;

}

if (board[p.x][p.y + 1] == player && board[p.x][p.y + 2] == player && board[p.x][p.y - 1] == player

&& board[p.x][p.y - 2] == player) {

return player;

}

if (board[p.x][p.y + 1] == player && board[p.x][p.y - 1] == player && board[p.x][p.y - 2] == player

&& board[p.x][p.y - 3] == player) {

return player;

}

if (board[p.x][p.y - 1] == player && board[p.x][p.y - 2] == player && board[p.x][p.y - 3] == player

&& board[p.x][p.y - 4] == player) {

return player;

}

// Check duong cheo

if (board[p.x + 1][p.y + 1] == player && board[p.x + 2][p.y + 2] == player && board[p.x + 3][p.y + 3] == player

&& board[p.x + 4][p.y + 4] == player) {

return player;

}

if (board[p.x + 1][p.y + 1] == player && board[p.x + 2][p.y + 2] == player && board[p.x + 3][p.y + 3] == player

&& board[p.x - 1][p.y - 1] == player) {

return player;

}

if (board[p.x + 1][p.y + 1] == player && board[p.x + 2][p.y + 2] == player && board[p.x - 1][p.y - 1] == player

&& board[p.x - 2][p.y - 2] == player) {

return player;

}

if (board[p.x + 1][p.y + 1] == player && board[p.x - 1][p.y - 1] == player && board[p.x - 2][p.y - 2] == player

&& board[p.x - 3][p.y - 3] == player) {

return player;

}

if (board[p.x - 1][p.y - 1] == player && board[p.x - 2][p.y - 2] == player && board[p.x - 3][p.y - 3] == player

&& board[p.x - 4][p.y - 4] == player) {

return player;

}

// Check duong cheo phu

if (board[p.x + 1][p.y - 1] == player && board[p.x + 2][p.y - 2] == player && board[p.x + 3][p.y - 3] == player

&& board[p.x + 4][p.y - 4] == player) {

return player;

}

if (board[p.x + 1][p.y - 1] == player && board[p.x + 2][p.y - 2] == player && board[p.x + 3][p.y - 3] == player

&& board[p.x - 1][p.y + 1] == player) {

return player;

}

if (board[p.x + 1][p.y - 1] == player && board[p.x + 2][p.y - 2] == player && board[p.x - 1][p.y + 1] == player

&& board[p.x - 2][p.y + 2] == player) {

return player;

}

if (board[p.x + 1][p.y - 1] == player && board[p.x - 1][p.y + 1] == player && board[p.x - 2][p.y + 2] == player

&& board[p.x - 3][p.y + 3] == player) {

return player;

}

if (board[p.x - 1][p.y + 1] == player && board[p.x - 2][p.y + 2] == player && board[p.x - 3][p.y + 3] == player

&& board[p.x - 4][p.y + 4] == player) {

return player;

}

return 0;

}

}

* ***Lớp Chessmark***

Danh sách cách thế cờ dùng để tính điểm được goi trong một

số thủ tục của class CaroBoard

-*public static int MarkOfChessArea(int[] arr, int player):* tínhđiểm củaô, tương ứng với thế cờ

public static int MarkOfChessArea(int[] arr, int player) {

int j;

int i;

for (i = 0; i < thecos.length; i++) {

int[] theco = thecos[i];

int pos = 5 - circle[i];

int lengOfTheco = theco.length;

for (j = 0; j < lengOfTheco; j++) {

if (arr[j + pos] != player \* theco[j]) {

break;

}

}

if (j >= lengOfTheco) {

return mark[i];

}

//

pos = 5 + circle[i];

for (j = 0; j < lengOfTheco; j++) {

if (arr[pos - j] != player \* theco[j]) {

break;

}

}

if (j >= lengOfTheco) {

return mark[i];

}

}

return 0;

}

}

* ***Lớp player***

Class Player : tượng trưng cho một máy tính với các thông số

nhằm thực hiện nước đi, thao tác với người sử dụng.

-*public Player(int pDiff, int pAttackOrDepend, int width, int heigth)*

public Player(int pDiff, int pAttackOrDepend, int width, int heigth)

{

this.difficult = pDiff;

this.attackOrDepend = pAttackOrDepend;

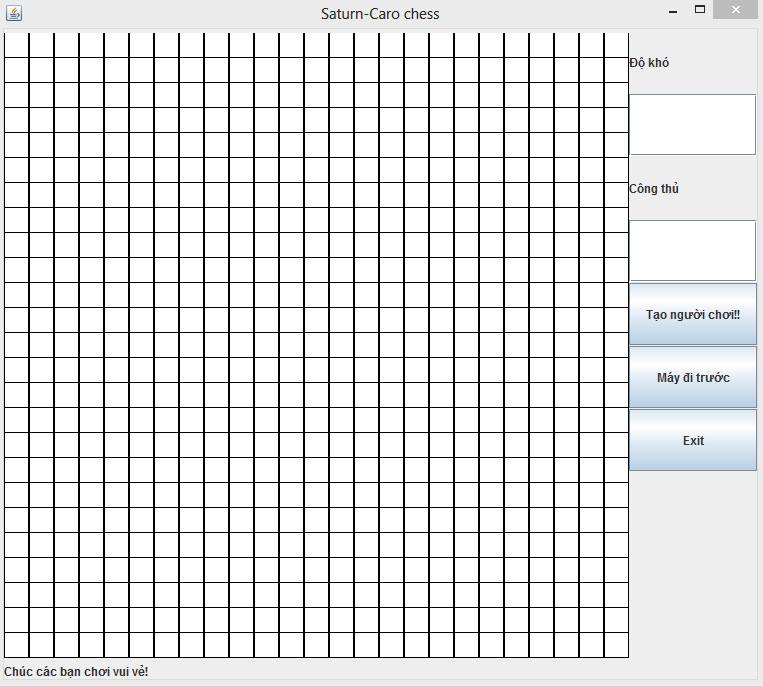
caroTable = new CaroBoard(width, heigth);

}

* ***Lớp point***

Biểu diễn 1 điểm đánh trong bàn cờ caro

1. **Giao diện và cách chơi**

****

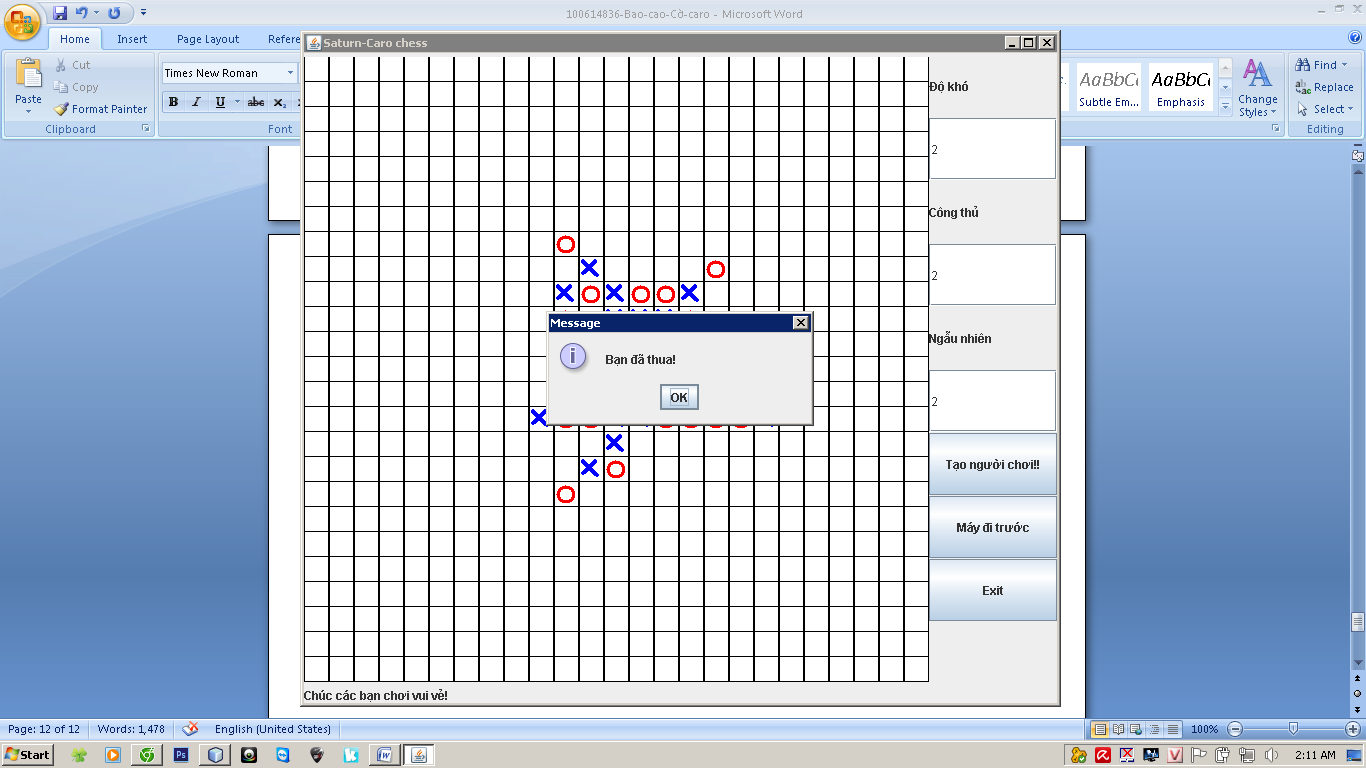
Trước khi bắt đầu chơi ta thiết lập các thông số về:

* Độ khó: Chọn độ khó của máy
* Công thủ: Bạn muốn máy đánh ở thế công hay thế thủ? Nhập 0 để máy đánh ở thế 0 và 1 để máy đánh ở thế thủ

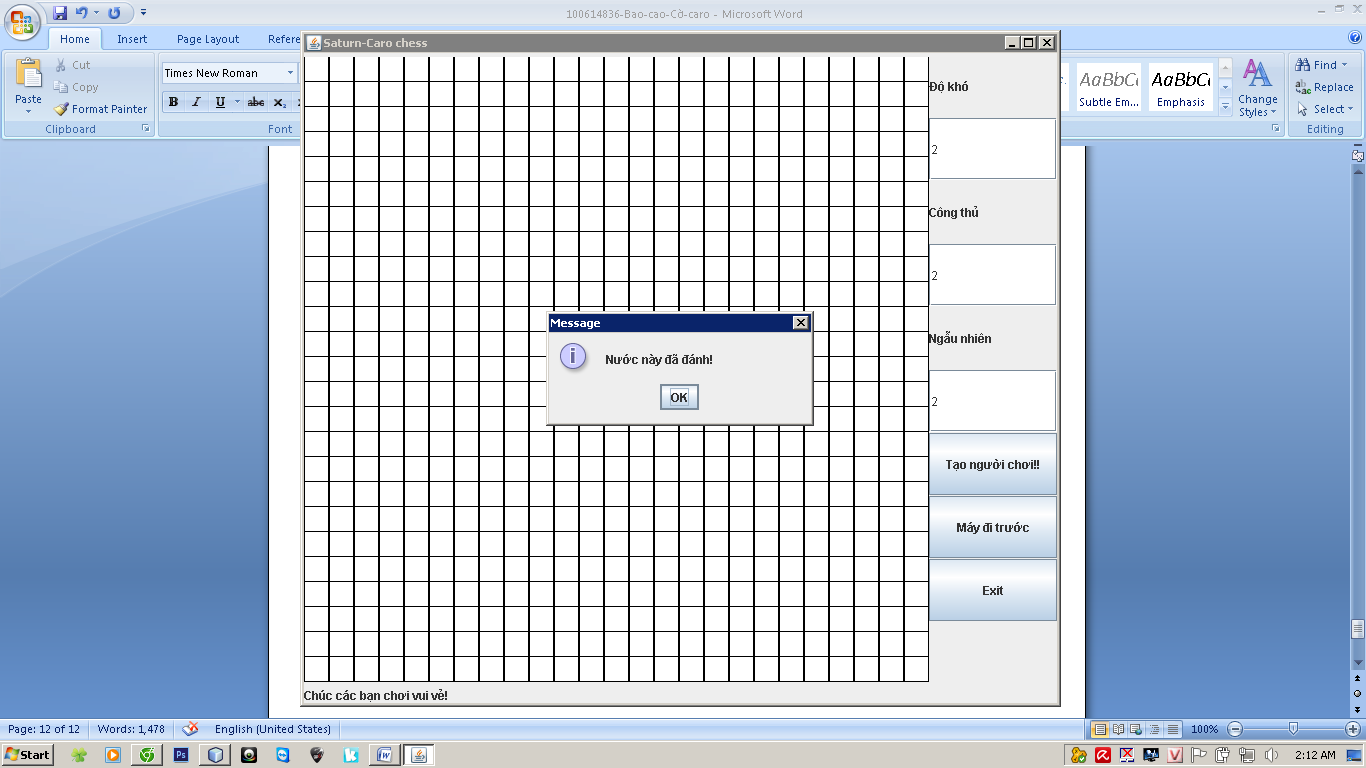
Sau đó nếu bạn muốn đánh trước thì bạn chọn Tạo người chơi và bắt đầu chơi, nếu muốn máy đánh trước thì sau khi chọn Tạo người chơi bạn chọn Máy đi trước.

Các hộp thoại hiện lên khi chơi:

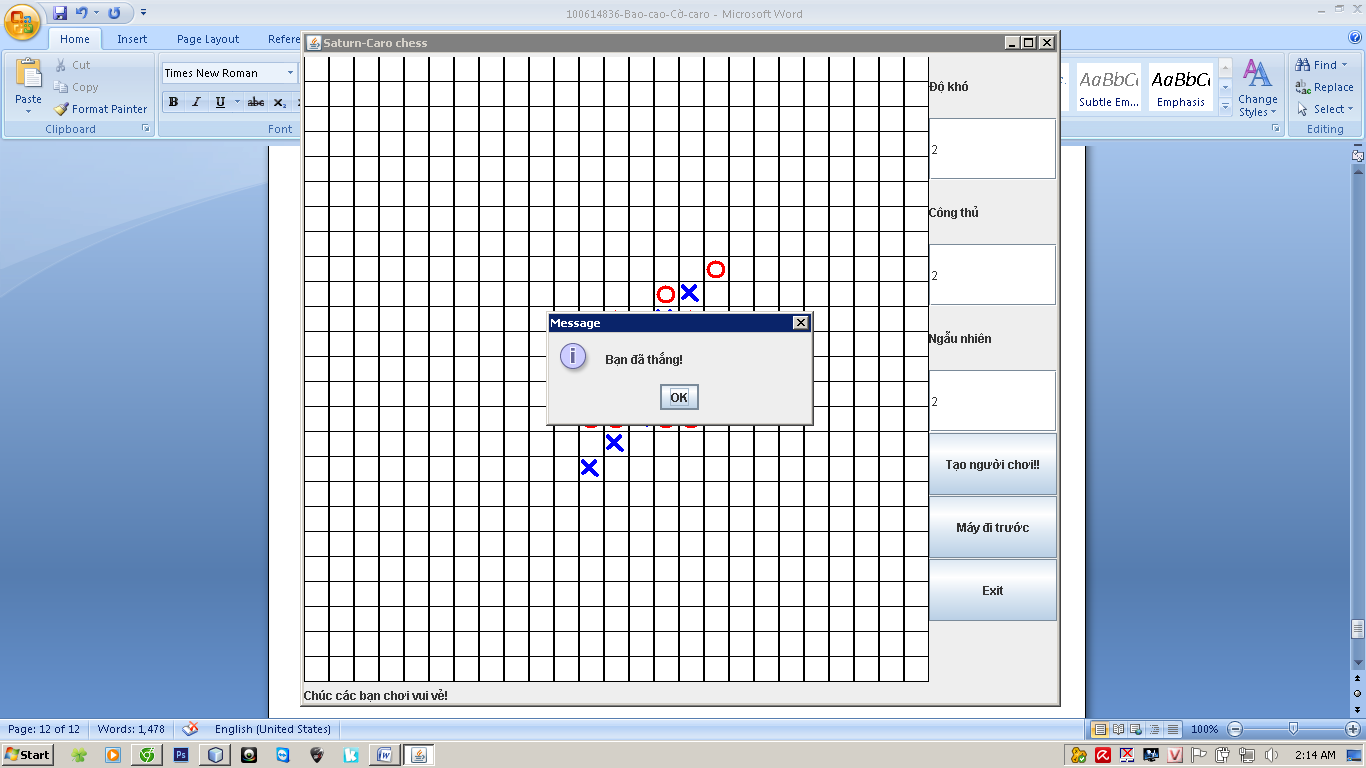
Khi bạn thua



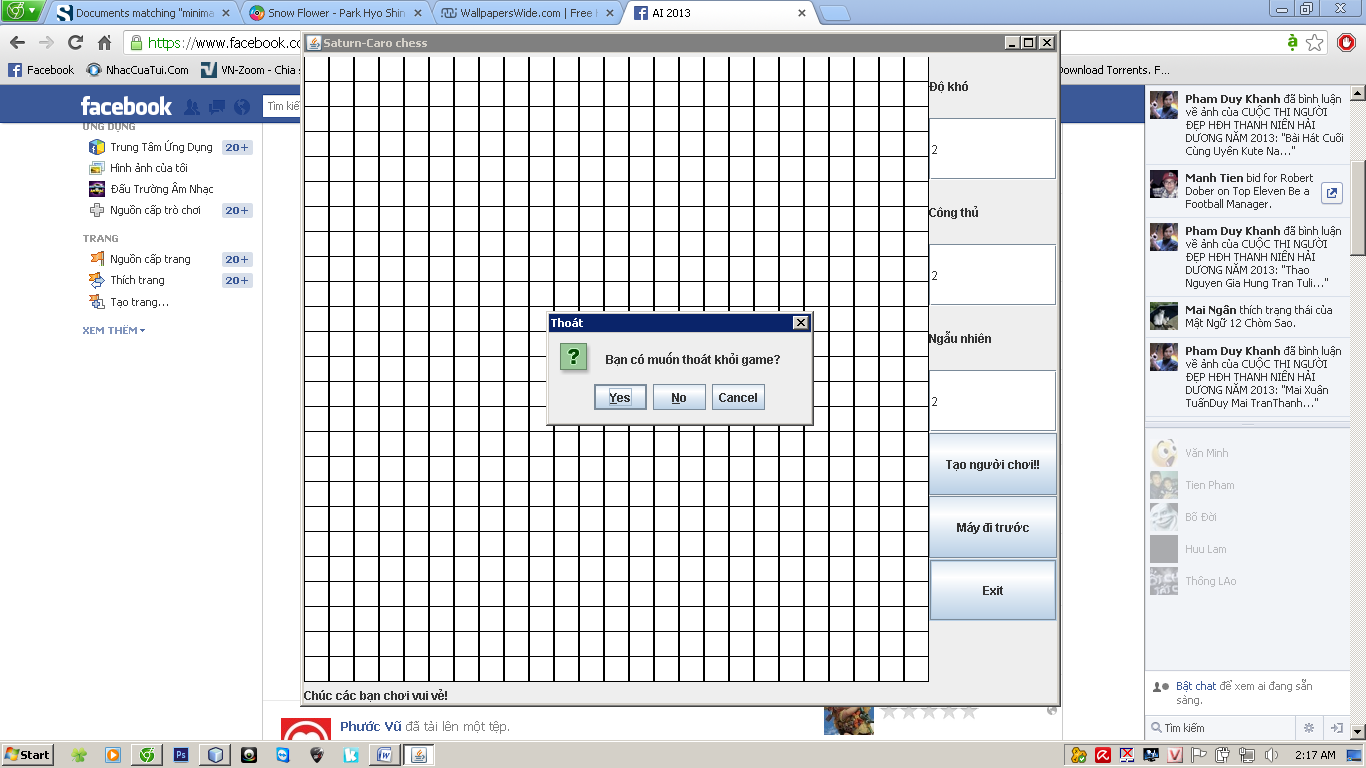
Khi bạn đánh vào nước đã đánh rồi



Khi bạn thắng:



Khi bạn thắng hoặc thua, bạn chọn Tạo người chơi để chơi lại hoặc Exit đê thoát game!



# 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 trinh thực thi chương trình không thể tránh khỏi những sai sót và chưa thực sự tối ưu. Chúng em mong được sự góp ý của thầy để có thể hoàn thiện hơn trong tương lai!

Chúng em xin chân thành cảm ơn!