**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

KHOA KHOA HỌC MÁY TÍNH

****



**BÁO CÁO ĐỒ ÁN MÔN TRÍ TUỆ NHÂN TẠO**

**GAME CỜ CARO SỬ DỤNG THUẬT TOÁN MINIMAX CÓ CẮT TỈA ALPHA BETA**

LỚP: CS106.K22.KHCL

GVLT: Nguyễn Đình Hiển

SVTH: Hồ Hoàng Tùng - 17521232

Ngô Anh Vũ - 17521272

TP.Hồ Chí Minh, ngày 02 tháng 07 năm 2020

**Mục lục:**

**CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN**

1.1. Nguồn gốc trò chơi

1.2. Mô tả trò chơi

1.3. Ứng dụng của trò chơi

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ**

3.1. Phân tích bài toán

3.2. Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán

3.3. Các vấn đề và thuật giải

3.4. Ví dụ minh họa thuật toán/ thuật giải (2 ví dụ)

**CHƯƠNG 4: ỨNG DỤNG**

4.1. Giới thiệu chương trình ứng dụng

4.2. Cài đặt

4.3. Kết quả chạy chương trình

**CHƯƠNG 5: KẾT LUẬN**

5.1. Kết quả đạt được

5.2. Hạn chế

5.3. Hướng phát triển

**TÀI LIỆU THAM KHẢO**

**CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN**

* 1. **Nguồn gốc trò chơi:**

Cờ caro là một trò chơi dân gian phổ biến, quốc tế gọi là trò *gomoku,* Ở Việt Nam, cờ này thường chơi trên giấy tập học sinh (đã có sẵn các ô ca-rô), dùng bút đánh dấu hình tròn (O) và chữ X để đại diện cho 2 quân cờ.

* 1. **Mô tả trò chơi:**

Được chơi bằng 2 loại quân, thường được kí hiệu là X hoặc O, cũng có thể là quân cờ hình tròn có 2 màu khác nhau, được đánh dấu lên bàn cờ, thường có kích thước 19x19 với các ô cờ dạng hình ô vuông. Người thắng là người đầu tiên có được một chuỗi liên tục gồm 4 quân hàng ngang, hoặc dọc, hoặc chéo không bị chặn đầu nào. Nếu bị chặn một đầu thì người đó cần có chuỗi 5 quân liên tục mới thắng. Nếu chuỗi 5 quân liên tục bị chặn cả hai đầu thì trò chơi tiếp tục. Một khi đã đặt xuống, các quân cờ không thể di chuyển hoặc bỏ ra khỏi bàn, nhưng trong game thì có thể sử dụng chức năng của nút Undo để hoàn lại.

* 1. **Ứng dụng của trò chơi:**

Vì là 1 trò chơi phổ biến, đơn giản, dễ chơi, đặc biệt mang “hơi hướng” của một “trò chơi chiến thuật”, cờ caro có thể giúp con người giải tỏa stress, căng thẳng sau những giờ học tập và làm việc, hoặc giúp những cá nhân có sở thích hoặc hứng thú muốn nâng cao trình độ chơi cờ caro của bản thân có thể luyện tập thêm khả năng đánh cờ và “tư duy chiến thuật” bằng chế độ chơi với máy (PvsCom)

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

Bài toán cờ Caro là một game đối kháng trực tiếp giữa 2 người chơi với nhau. Trong đó, mỗi người sẽ bằng mọi cách, tìm được nước đi sao cho nước đi đó có thể tạo ra một thế cờ là “tối ưu” nhất có thể, mang lại nhiều lợi thế cho mình nhất và hạn chế tối đa lợi thế của đối thủ. Dựa vào cở sở đó, ta sẽ áp dụng chiến lược *Minimax* để xử lý vấn đề này.

Chiến lược *Minimax* được thể hiện bằng giải thuật minimax, dựa trên 2 giả thuyết sau:

* Cả 2 đối thủ có cùng kiến thức như nhau về không gian trạng thái của trò chơi.
* Cả 2 đối thủ đều có cùng mức cố gắng thắng như nhau.

Về chiến lược *Minimax* được trình bày như sau:

* Max: biểu diễn cho mục đích của người chơi này là làm tối đa hóa lợi thế của mình.
* Min: biểu diễn cho mục đích của người chơi này là làm tối thiểu hóa lợi thế của đối thủ.

Trên cây tìm kiếm (Cây trò chơi) sẽ phân lớp thành các lớp Max và Min.

Về giải thuật Minimax được minh họa như sau:

Với một State (node) u bất kỳ,

* Node u là node lá có giá trị xác định => Trả về giá trị hàm kết cuộc.(vd: 1 nếu Max thắng, -1 nếu Min thắng)
* Node u thuộc lớp Max => gán giá trị lớn nhất cuả các node con
* Node u thuộc lớp Min => gán giá trị nhỏ nhất của các node con

(Đỉnh có giá trị càng lớn càng tốt cho Max, càng nhỏ càng tốt cho Min)

Ngoài ra, trên thực tế, không gian trạng thái thường sẽ rất lớn, nên ta không thể nào có thể duyệt đầy đủ hết tất cả các trạng thái đề tìm được trạng thái “tối ưu” nhất, nhằm giành được chiến thắng tuyệt đối trước đối thủ. Vì thế, ta sẽ áp dụng biến thể của chiến lược Minimax, bằng cách thêm vào một mức độ giới hạn nhìn thấy trước các trạng thái, gọi là “chiều sâu tối đa” của thuật giải. Dựa vào thông số này, thuật giải có thể chạy nhanh hơn rất nhiều và cho ra được kết quả khá tốt, có thể chấp nhận được.

Ngoài ra ta còn có thể sử dụng thêm phương pháp *cắt tỉa alpha – beta,* nhằm loại bỏ bớt các nhánh dư thừa tồn tại trong cây trạng thái, nhờ đó có thể giảm được thêm một khoảng thời gian dể duyệt qua các trạng thái mà đã biết chắc chắn sẽ không làm thay đổi giá trị của các trạng thái ở trên.

Về phương pháp cắt tỉa alpha – beta, được trình bày như sau:

Với mỗi State u, sẽ tồn tại song song 2 giá trị alpha và beta:

* Nếu u đang thuộc lớp Max, giá trị của u sẽ là giá trị của alpha, được tính bằng max(alpha, min của các nhánh con), xét thấy nếu alpha > beta => cắt tỉa.
* Nếu u đang thuộc lớp Min, giá trị của u sẽ là giá trị của beta, được tính bằng min(beta, max của các nhánh con), xét thấy nếu alpha >= beta => cắt tỉa

**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ**

**3.1. Phân tích bài toán:**

**-** Ý tưởng giải quyết bài toán:

Trong bài toán cờ caro, mỗi người chơi nếu muốn dành chiến thắng thì phải cố gắng sao cho mỗi khi đánh một quân cờ mới lên bàn cờ, sẽ tạo ra được thế cờ tốt nhất, nghĩa là phải hạn chế tối thiểu lợi thế của đối thủ và làm tăng tối đa lợi thế của bản thân. Cụ thể, người chơi sẽ phải dựa vào các thông tin có sẵn như trạng thái của bàn cờ hiện tại, số lượng các quân cờ của mỗi bên, vị trí của các quân cờ hiện tại trên bàn cờ để có thể đánh giá độ tốt sơ bộ cho các nước đi kế tiếp, qua đó có thể so sánh giữa các nước đi với nhau và chọn được nước đi mà được đánh giá là tốt nhất, làm cho lợi thế của thế cờ tiếp theo nghiêng về mình nhiều hơn, tiến sát hơn tới việc chiến thắng ván cờ.

- Áp dụng lý thuyết:

Ta sẽ áp dụng phần lý thuyết về *thuật toán Minimax* và các phương pháp làm gia tăng khả năng xử lý và hiệu suất như *phương pháp cắt tỉa alpha – beta, leo đồi*, để biến thuật toán thành thuật giải có thể đem lại hiệu quả.

- Phát biểu bài toán:

Bài toán lập trình chương trình đánh cờ caro được quy về bài toán đi tìm một bước đi, tức là tìm một vị trí để đặt quân cờ sao cho tiến gần tới trạng thái chiến thắng nhanh nhất hoặc có thể ngăn chặn được đối thủ đạt được trạng thái chiến thắng.

Mỗi trạng thái của thuật giải là một trạng thái của bàn cờ, trạng thái của bàn cờ là mảng trạng thái của các ô cờ trên bàn cờ. Vậy, *không gian trạng thái* của bài toán được coi như là tập hợp của tất cả các trạng thái của bàn cờ có thể có được, cụ thể hơn là thứ tự sắp xếp của các quân cờ trên bàn cờ, mỗi sự thay đổi dù là nhỏ nhất về vị trí của các quân cờ với nhau cũng sẽ biến trạng thái hiện tại trở thành một trạng thái khác.

Ta có thể thấy được nếu muốn thay đổi trạng thái hiện tại của bàn cờ sang một trạng thái mới, ta sẽ thay đổi vị trí của các quân cờ hoặc 1 quân cờ, mà trong trò chơi Caro, các quân cờ sẽ mặc định là không thể di chuyển, vậy *toán tử chuyển trạng thái* của bài toán sẽ là tạo mới một quân cờ trên bàn cờ, với điều kiện ô cờ được tạo mới chưa tồn tại bất kì quân cờ của bên nào.

Như vậy, trạng thái bắt đầu của bài toán sẽ là một bàn cờ mà tất cả ô cờ đều trống, hai bên sẽ tuân theo thứ tự, luân phiên đặt những quân cờ của mình, sẽ lần lượt thay đổi trạng thái của trò chơi cho tới khi một trong hai đạt tới được trạng thái kết thúc, tức là dành chiến thắng trước bên kia. Trạng thái kết thúc của trò chơi là một trong hai bên phải sở hữu 5 quân cờ liên tiếp nhau, xếp thành một đường thẳng ở bất kì phương nào và thỏa mãn điều kiện không bị chặn 2 đầu, tức là trong cùng một phương, chuỗi không được nằm trong “phạm vi giới hạn” của 2 quân cờ của đối thủ.

Tóm lại, ta có:

Không gian trạng thái: Sự sắp xếp, bố trí của các quân cờ trên bàn cờ.

Trong đó, trạng thái bắt đầu là bàn cờ trống, trạng thái kết thúc là bàn cờ thỏa điều kiện có một dãy 5 quân cờ của một bên được xếp thẳng hàng và không bị chặn ở hai đầu.

Toán tử chuyển trạng thái: Thêm một quân cờ mới vào bàn cờ hiện tại.

Mục tiêu của trò chơi: Một trong hai bên đạt được đến trạng thái kết thúc bằng cách thêm các quân cờ, làm thay đổi trạng thái của trò chơi.

**3.2. Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán**

- Cấu trúc dữ liệu:

Quá trình chơi cờ trên máy tính thực chất là quá trình biến đổi của các trạng thái bàn cờ. Vậy, đầu tiên ta cần phải có một lớp đối tượng là *bàn cờ* để lưu trữ các thông số cơ bản là số cột và số dòng. Lớp *bàn cờ* được định nghĩa bằng 2 thuộc tính cơ bản là số dòng, số cột và các phương thức khởi tạo với giá trị cho trước của 2 thuộc tính trên. Ngoài ra, vì chương trình được lập trình bằng *Winform C Shape*, cần thêm một đối tượng *Graphics* đi kèm để có thể vẽ được bàn cờ và các quân cờ với màu sắc tương ứng.

Tiếp theo, chúng ta sẽ có một lớp đối tượng là *ô cờ,* đây là đối tượng rất quan trọng trong cấu trúc dữ liệu của trò chơi, mỗi đối tượng *ô cờ* được định nghĩa với các thuộc tính là: 2 hằng số chiều dài, chiều rộng cố định, là kích thước thực tế để biểu diễn ô cờ trên bàn cờ; 2 thuộc tính *dòng, cột* là các thông số của ô cờ cho biết vị trí chính xác của ô cờ trên bàn cờ; thuộc tính *vị trí* là một biến kiểu Point lưu trữ 2 giá trị là X và Y tương ứng với tọa độ thực tế của ô cờ được biểu diễn, hỗ trợ cho việc vẽ ô cờ một cách trực quan; thuộc tính *sở hữu* cho biết thông tin ô cờ này thuộc sỡ hữu của bên 1 hoặc bên 2, còn nếu ô cờ chưa thuộc sở hữu của bên nào thì sẽ mang giá trị là 0. Lớp đối tượng *ô cờ* có phương thức khởi tạo với các thuộc tính trên đã được cho biết trước hoặc với giá trị mặc định là sở hữu bằng 0.

Ngoài ra, ta cần một lớp đối tượng có thể bao quát và có các phương thức mang chức năng điều khiển trò chơi là lớp *CaroChess.* Lớp đối tượng *CaroChess* được xem như là “engine” của trò chơi, quy định cách thức hoạt động và đảm bảo được tính logic của trò chơi. Lớp *CaroChess* được định nghĩa như sau:

- Các thuộc tính:

+ BanCo: là một biến thuộc class *Bàn Cờ* dùng để khởi tạo và lưu trữ kích thước cho bàn cờ hiện tại của trò chơi.

+ MangOCo: Một mảng 2 chiều có kích thước lần lượt bằng số dòng và số cột của bàn cờ hiện tại, là 1 ma trận mà mỗi phần tử là một đối tượng *ô cờ,* mảng nàydùng để lưu trữ trạng thái của bàn cờ hiện tại bằng cách lưu trữ tất cả các trạng thái của mỗi ô cờ theo cấu trúc ma trận. Vì dùng ma trận nên có thể biểu diễn được vị trí của các ô cờ một cách trực quan, thông số dòng và cột của mỗi ô cờ sẽ tương ứng với chỉ số của ô cờ trong ma trận.

+ stkCacNuocDaDi: mang cấu trúc stack là một tập hợp dùng để lưu trữ các thao tác tạo ra quân cờ mới của một bên nào đó theo thứ tự thực tế, mà ở đây mỗi thao tác là một đối tượng *ô cờ* tương ứng với vị trí, tọa độ, sở hữu của nước cờ vừa đi, hỗ trợ cho chức năng Undo hoàn lại nước đi) của trò chơi.

+ stkCacNuocUndo: là một stack tương tự như stkCacNuocDaDi, nhưng stack này dùng để lưu lại các nước cờ đã được undo bằng thao tác Undo, dùng để hỗ trợ cho thao chức năng Redo (phục hồi lại các nước cờ đã Undo trước đó).

+ LuotDi: là một biến kiểu int, đại diện và kiểm soát lượt đi của mỗi bên.

+ SanSang: biến bool dùng để điều khiển trạng thái bật hoặc tắt trò chơi.

+ MangDiemTanCong: là một bộ số kiểu int, mỗi phần tử là một hằng số lưu trữ giá trị ước chừng của người lập trình về độ tốt của điểm tấn công của một ô cờ còn trống ( chưa thuộc sở hữu của bên nào) tương ứng với số lượng quân cờ của bên đang xét, được sắp xếp liền nhau trên cùng một hàng. ( vi dụ: bên đang xét là bên 1; có 0 quân 1 => 0 điểm; có 1 quân 1 => 1 điểm; có 2 quân 1 => 9 điể, …)

+ MangDiemPhongNgu: là một bộ số kiểu int tương tự với MangDiemTanCong, nhưng khác ở chỗ mảng số này được dùng để tính điểm phòng ngự cho từng ô cờ chưa thuộc sở hữu của bên nào trong bàn cờ hiện tại (ví dụ: bên đang xêt là bên 1; có 0 quân cờ 2 => 0 điểm, có 1 quân cờ 2 => 4 điểm, có 2 quân cờ 2 => 28 điểm, …)

+ Truonghop1, Truonghop2: là mảng string, mỗi phần tử là một trường hợp có thể xảy ra trên một trạng thái của bàn cờ, hỗ trợ cho việc đánh giá độ tốt của trạng thái đó.

+ point: là một bộ số kiểu int, lưu trữ giá trị tương ứng với từng trường hợp trong mảng Truonghop1, Truonghop2.

+ maxdepth: là 1 biến int cho biết độ sâu tối đa của thuật giải, dùng để giới hạn số bước đi có thể “nhìn trước” được của thuật giải.

+ branch: là 1 biến int cho biết số lượng tối đa các nhánh được mở rộng.

+ Pen: là 1 biến kiểu pen, dùng để “vẽ” bàn cờ và quân cờ, xử lý các thao tác đồ họa.

+ sbWhite, sbBlack, sbBG: là các biến để lưu màu sắc cho quân cờ và màu nền của bàn cờ.

+ Ketthuc: là biến enum lưu các giá trị khi kết thúc trò chơi, hỗ trợ cho việc xác định trạng thái khi kết thúc trò chơi. Các trạng thái là: hòa cờ, người chơi 1 thắng, người chơi 2 thắng, computer thắng.

- Các trạng thái khởi tạo, kết thúc và toán tử chuyển trạng thái của trò chơi:

+ Trạng thái khởi tạo của trò chơi là tạo một đối tượng bàn cờ có kích thước 20 x 20, khởi tạo gía trị bằng 0 cho tất cả phần tử của mảng MangTrangThaiHienTai, khởi tạo mảng MangOCo với giá trị các ô cờ mặc định.

+ Trạng thái kết thúc của trò chơi là một bàn cờ có một chuối các ô cờ thỏa điều kiện chiến thắng, hoặc là một bàn cờ không còn tồn tại ô cờ nào có sở hữu bằng 0. Được biểu diễn và xác định thông qua mảng MangOCo.

+ Toán tử chuyển trạng thái của trò chơi là một thao tác làm thay đổi bàn cờ, cụ thể là thao tác gán lại giá trị cho thuộc tính *sở hữu* của một ô cờ còn trống (sở hữu bằng 0) trong mảng MangOCo, sau đó cập nhật lại giá trị của mảng MangTrangThaiHienTai theo mảng MangOCo.

**3.3. Các vấn đề và thuật giải**

- Các vấn đề quan trọng cần phải giải quyết:

+ Xử lý thao tác đánh một nước cờ mới:

Đây là một thao tác cơ bản nhất của trò chơi, giúp cho người chơi hiện thực hóa các nước cờ của mình.

Hàm void DanhCo(int X,int Y)

Mã giả:

Intput: X, Y

B1: Kiểm tra xem có ô cờ nào tồn tại tại tọa độ (X, Y) hay không. Nếu có thì return false, nếu không thì chuyển sang B2

B2: Kiểm tra xem thuộc tính sở hữu của ô cờ đang xét bằng bao nhiêu, nếu bằng 0 tức là chưa thuộc sở hữu của bên nào thì tiếp tục chuyển sang B3, nếu khác 0 thì return false.

B3: Xét xem biến LuotDi là lượt đi hiện tại thuộc về bên nào. Nếu LuotDi bằng 1 thì gán *sở hữu* tại ô cờ đang xét bằng 1, ngược lại thì gán *sở hữu* tại ô cờ đang xét bằng 2.

B4: Thêm ô cờ vừa xét vào stack stkCacNuocDaDi, khởi tạo stack stkCacNuocUndo cho trường hợp người chơi muốn thực hiện thao tác Redo.

B5: Gán lại giá trị cho biến LuotDi ngược lại với giá trị cũ để đảm bảo luân phiên giữa hai người chơi.

+ Xử lý thao tác tự động đánh cờ của máy:

Đánh cờ vào ô cờ được đánh giá là tốt nhất. Vậy, ta sẽ có hàm tìm ô cờ đi tốt nhất cho trạng thái hiện tại. Hàm sẽ nhận vào một trạng thái là một ma trận 2 chiều tương ứng với kích thước bàn cờ, mỗi phần tử là giá trị *sở hữu* của ô cờ tương ứng và sẽ trả về output là một ô cờ tốt được xem là tốt nhất, với hi vọng sẽ tạo được thế cờ tốt nhất. Đây cũng là hàm quan trọng nhất của thuật giải, bao quát hết tất cả các vấn đề và hàm xử lý khác của thuật giải tìm bước đi tiếp theo cho máy.

Hàm OCo ChonNuocDiTotNhat()

Mã giả:

Intput: Biến toàn cục là mảng các ô cờ MangOCo

Output: Oco Ketqua, là một ô cờ được chọn với giá trị *sở hữu* được gán *=* 1. (mặc định máy = 1, player = 2)

B1: Khởi tạo một mảng 2 chiều kiểu int[,] là mangTrangThaiHienTai có cấu trúc mô phỏng lại mảng MangOCo nhưng mỗi phần tử chứa giá trị *sở hữu* tương ứng với các ô cờ trong MangOCo để tiện thao tạc.

B2:

2.1. Khởi tạo danh sách các điểm kiểu list<Point> DanhSachCacTrangThaiCanXemXet chứa các tọa độ của các ô cờ cần xem xét.

2.2. For (i = 0, i < branch (số lượng nhánh mở rộng tối đa), i++ ):

DanhSachCacTrangThaiCanXemXet.Add( Nước đi có điểm tấn công và phòng ngự cao nhất của trạng thái (mangTrangThaiHienTai) khởi tạo ở B1). (Hàm tìm nước đi có điểm tấn công và phòng ngự cao nhất được định nghĩa ở dưới)

2.3. Tạo một mảng kiểu int[DanhSachCacTrangThaiCanXemXet.size] MangGiaTriCuaTungTrangThai để lưu giá trị được đánh giá tương ứng của mỗi phần tử trong mảng DanhSachCacTrangThaiCanXemXet, làm cơ sở để so sánh và giúp lựa chọn trạng thái tốt nhất để đi tiếp.

2.4. Với mỗi phần tử i thuộc mảng DanhSachCacTrangThaiCanXemXet, ta gán giá trị của hàm MinValue(Trạng thái hiện tại, i, alpha, beta, độ sâu) cho phần tử tương ứng ở mảng MangGiaTriCuaTungTrangThai. (Hàm MinValue được định nghĩa ở phía dưới)

B3: Dựa vào mảng giá trị của từng trạng thái (MangGiaTriCuaTungTrangThai) được khởi tạo và gán giá trị ở B2, ta sẽ chọn ra được một hoặc nhiều vị trí có giá trị cao nhất và thêm vào danh sách mới kiểu list<Point>. Sau đó chọn ngẫu nhiên 1 trạng thái để đi tiếp trong list vừa tạo để đảm bảo tính tự nhiên của thuật giải.

- Các thao tác cần phải xử lý để hỗ trợ cho việc đánh giá độ tốt của trạng thái hiện tại của bàn cờ, điểm tấn công và phòng thủ của mỗi ô cờ ở trạng thái hiện tại:

+ Đánh giá độ quan trọng bằng thông số điểm tấn công và điểm phòng thủ của mỗi ô cờ trên bàn cờ, giúp chọn các bước đi ban đầu để bắt đầu quá trình “nhìn trước” và đánh giá:

Hàm Point GetMaxNode(int [,] mangTrangThaiHienTai)

Input: Nhận đầu vào là 1 mảng 2 kiểu int 2 chiều có cấu trúc của bàn cờ, cho biết trạng thái của bàn cờ hiện tại là giá trị *sở hữu* của từng ô cờ trên bàn cờ.

Output: Mội đối tượng Point chưa giá trị tọa độ của ô cờ mang điểm tấn công và phòng ngự cao nhất bàn cờ.

Mã giả:

B1: Khởi tạo 2 biến đếm là cComputer và cPlayer, là biến lưu trữ số lượng quân cờ của máy và người chơi. Khởi tạo một mảng 2 chiều kiểu int[,] MangGiaTri, là mảng có cấu trúc của bàn cờ, dùng để lưu trữ giá trị sẽ được tính toán của mỗi ô cờ, dùng để so sánh và đánh giá. Ngoài ra khởi tạo thêm các biến đếm kiểu int, hỗ trợ duyệt dữ liệu thuận tiện hơn.

B2:

2.1. Duyệt ô cờ, nếu *sỡ hữu = 0* , đếm số lượng quân ta, tương ứng với hệ số điểm tấn công và số lượng quân địch, tương ứng với hệ số điểm phòng ngự theo các phương ngang, dọc, chéo lên, chéo xuống, chuyển đến B2.2. Nếu *sỡ hữu* != 0 thì gán phần tử ở vị trí tương ứng trong mảng giá trị (MangGiaTri) khởi tạo ở B1 = -1, lặp lại B2.1.

2.2. Thực hiện nhân hệ số vừa đếm được lần lượt với mảng lưu giá trị tấn công ước lượng (MangDiemTanCong) và mảng giá trị phòng thủ ước lượng (MangDiemPhongNgu).

2.3. Gán tổng 2 giá trị tấn công và phòng ngự vừa tính được cho phần tử có thứ tự tương ứng trong mảng giá trị (MangGiaTri) khởi tạo ở B1. Lặp lại B2.1. cho đến khi tất cả các phàn tử của MangGiaTri đều có giá trị.

B3: Từ mảng giá trị (MangGiaTri), chọn ra các phần tử có giá trị cao nhất, tương ứng với thứ tự các điểm trên bàn cờ. Thêm các điểm được chọn vào một danh sách, sau đó tiến hành lấy random một điểm bất kì trong danh sách để làm output cho hàm, đảm bảo tính tự nhiên của thuật giải.

+ Đánh giá độ tốt của một trạng thái bàn cờ dựa vào sự sắp xếp của các quân cờ đang xét so với trạng thái chiến thắng của mỗi bên đang xét.

Hàm int DinhGiaTrangThaiKetThuc (int [,] MangTrangThaiHienTai, int LuotDi HienTai)

Input: Nhận đầu vào là 1 mảng 2 kiểu int 2 chiều có cấu trúc của bàn cờ, cho biết trạng thái của bàn cờ hiện tại là giá trị *sở hữu* của từng ô cờ trên bàn cờ và một biến int cho biết lượt đi đang được xét đến thuộc về bên nào.

Output: Một biến int mang giá trị đã được đánh giá cho trạng thái đầu vào.

Mã giả:

B1: Khởi tạo biến Diem = 0 là đầu ra của hàm, chuối s kiểu string, lần lượt duyệt theo các phương dọc, ngang, chéo lên, chéo xuống, để thêm tất cả các giá trị *sỡ hữu* của từng ô cờ vào chuỗi s.

B2: Xét xem lượt đi hiện tại thuộc về bên nào dựa vào biến đầu vào LuotDiHienTai. Sau đó tiến hành cộng điểm cho biến Diem dựa vào giá trị của mảng điểm ước lượng point tương ứng với các trường hợp có xuất hiện trong chuỗi s.

B3: Trả về biến Diem chứa giá trị của trạng thái đầu vào.

- Thuật giải Minimax có cắt tỉa alpha, beta dùng để “nhìn trước” các trạng thái chưa được phát triển và lượng giá cho từng trạng thái kế tiếp sắp được chọn:

+ Trong thuật giải Minimax, tồn tại 2 đối tượng là Max đại diện cho bên luôn muốn chiến thắng và Min đại diện cho bên cố gắng để hạn chế tối thiểu số điểm của Max. Vậy, ta sẽ có 2 hàm để thực hiện từng thao tác tương ứng với từng lớp.

+ Hàm MaxValue, MinValue: hàm sẽ nhận vào input gồm một trạng thái hiện tại t có dạng mảng 2 chiều có cấu trúc của bàn cờ, một đối tượng Point p có ý nghĩa là giá trị tọa độ của nước cờ kế tiếp và các giá trị khác là biến alpha, beta hỗ trợ cho việc cắt tỉa các nhánh dư thừa. Ngoài ra còn có chỉ số depth là độ sâu hiện tại của hàm, dùng để giới hạn khoảng nhìn trước, làm tăng tốc độ xử lý của chương trình. Sau khi được gọi với bộ input tương ứng, hàm sẽ thực hiện các bước tính toán, “nhìn trước” các trạng thái của bàn cờ nếu giả định sẽ đánh nước cờ tại vị trí p vào trạng thái t bằng kĩ thuật gọi lại hàm – đệ quy. Vì trò chơi xen kẽ là từng bước đi của 2 bên nên nếu đang ở lớp Max, ta sẽ đệ quy hàm MinValue và ngược lại, nếu đang ở lớp Min, ta sẽ gọi lại hàm MaxValue. Với mỗi lần được gọi, hàm sẽ tính toán giá trị của từng trạng thái giả định tiếp theo qua các lần lặp cho đến khi đạt điều kiện dừng là tới được trạng thái kết thúc, hoặc là độ sâu hiện tại (depth) của hàm đã đạt tới độ sâu tối đa (maxdepth), thì sẽ return giá trị beta nếu đang ở hàm MinValue hoặc alpha nếu đang ở hàm MaxValue. Giá trị alpha và beta cũng tương ứng là giá trị của một trạng thái được giả định với một nước cờ mới. Nếu trạng thái ở lớp Max thì sẽ lấy giá trị alpha làm giá trị của trạng thái t với nước đi tại tọa độ p, giá trị alpha được cập nhật bằng cách lấy max(giá trị của các trạng thái con). Ngược lại, lớp Min sẽ lấy giá trị beta làm giá trị của trạng thái t với nước đi tại tọa độ p, beta được cập nhật bằng cách lấy min(giá trị của các trạng thái con).

Hàm MaxValue, MinValue:

int MinValue(int [,] MangTrangThaiHinTai, Point NuocDi, int alpha, int beta, int depth ) e

Mã giả:

Input: Nhận đầu vào là 1 mảng 2 kiểu int 2 chiều có cấu trúc của bàn cờ, cho biết trạng thái của bàn cờ hiện tại là giá trị *sở hữu* của từng ô cờ trên bàn cờ, một Point p chứa tọa độ của nước đi, các biến int alpha, beta, depth.

Output: Giá trị của trạng thái thuộc lớp Min, bằng với giá trị của beta.

B1: Tạo biến lượt đi giả định, là lượt đi tiếp theo của trạng thái hiện tại, lớp Min tượng trưng cho người chơi nên gán lượt đi = 2. Tiếp theo tạo biến value = Định giá trạng thái kết thúc (int[,] Trạng thái hiện tại,int Lượt đi giả định), là biến lưu giá trị của trạng thái hiện tại được định giá bằng hàm định giá trạng thái kết thúc đã định nghĩa ở trên.

B2: Xét điều kiện dừng: Nếu độ sâu hiện tại (depth) >= độ sâu tối đa (maxdepth) thì return giá trị của trạng thái hiện tại bằng giá trị của value. Nếu không, chuyển tới B3.

B3: Tìm các ô cờ có giá trị tấn công, phòng thủ tốt nhất của trạng thái hiện tại, sau đó thêm tất cả vào một danh sách gọi là Danh sách các node con (DanhSachCacNodeCon).

B4:

4.1. Với mỗi phần tử i của mảng danh sách các node con, tiến hành thao tác “đánh thử” bằng cách gán giá trị của mảng trạng thái hiện tại (MangTrangThaiHienTai) tại vị trí (i.X, i.Y) = 2 (sở hữu của Player).

4.2. Gọi hàm MaxValue( Mảng trạng thái hiện tại đã được “đánh thử”, Point i, alpha, beta, chiều sâu +1) nhằm đệ quy tới hàm tiếp theo với chiều sâu được tăng lên 1, cùng với trạng thái bàn cờ đã được tạm thay đổi để thực hiện việc “nhìn trước”.

4.3. Tiến hành cập nhật giá trị beta = min(beta cũ, các giá trị lớn nhất của các trạng thái con)

4.4. Bỏ đánh thử bằng cách gán lại giá trị của mảng trạng thái hiện tại tại vị trí i = 0(không thuộc sở hữu của bên nào)

B5: Sau khi cập nhật beta mới, nếu alpha >= beta thì break, tức là cắt tỉa tại trạng thái hiện tại, không xét tiếp nữa.

B6: Trả về giá trị beta là giá trị output của hàm MinValue.

int MaxValue(int [,] MangTrangThaiHienTai, Point NuocDi, int alpha, int beta, int depth )

Mã giả:

Input: Nhận đầu vào là 1 mảng 2 kiểu int 2 chiều có cấu trúc của bàn cờ, cho biết trạng thái của bàn cờ hiện tại là giá trị *sở hữu* của từng ô cờ trên bàn cờ, một Point p chứa tọa độ của nước đi, các biến int alpha, beta, depth.

Output: Giá trị của trạng thái thuộc lớp Max, bằng với giá trị của alpha.

B1: Tạo biến lượt đi giả định, là lượt đi tiếp theo của trạng thái hiện tại, lớp Max tượng trưng cho máy nên gán lượt đi = 1. Tiếp theo tạo biến value = Định giá trạng thái kết thúc (int[,] Trạng thái hiện tại,int Lượt đi giả định), là biến lưu giá trị của trạng thái hiện tại được định giá bằng hàm định giá trạng thái kết thúc đã định nghĩa ở trên.

B2: Xét điều kiện dừng: Nếu độ sâu hiện tại (depth) >= độ sâu tối đa (maxdepth) thì return giá trị của trạng thái hiện tại bằng giá trị của value. Nếu không, chuyển tới B3.

B3: Tìm các ô cờ có giá trị tấn công, phòng thủ tốt nhất của trạng thái hiện tại, sau đó thêm tất cả vào một danh sách gọi là Danh sách các node con (DanhSachCacNodeCon).

B4:

4.1. Với mỗi phần tử i của mảng danh sách các node con, tiến hành thao tác “đánh thử” bằng cách gán giá trị của mảng trạng thái hiện tại (MangTrangThaiHienTai) tại vị trí (i.X, i.Y) = 1 (sở hữu của Máy).

4.2. Gọi hàm MinValue( Mảng trạng thái hiện tại đã được “đánh thử”, Point i, alpha, beta, chiều sâu +1) nhằm đệ quy tới hàm tiếp theo với chiều sâu được tăng lên 1, cùng với trạng thái bàn cờ đã được tạm thay đổi để thực hiện việc “nhìn trước”.

4.3. Tiến hành cập nhật giá trị alpha = max(alpha cũ, các giá trị nhỏ nhất của các trạng thái con)

4.4. Bỏ đánh thử bằng cách gán lại giá trị của mảng trạng thái hiện tại tại vị trí i = 0(không thuộc sở hữu của bên nào)

B5: Sau khi cập nhật beta mới, nếu alpha > beta thì break, tức là cắt tỉa tại trạng thái hiện tại, không xét tiếp nữa.

B6: Trả về giá trị alpha là giá trị output của hàm MaxValue.

**3.4. Ví dụ minh họa thuật toán/ thuật giải:**

- Khi cần định giá cho trạng thái “t” là trạng thái hiện tại, hàm GetMaxNode sẽ được gọi với trạng thái t. Bên trong hàm GetMaxNode sẽ tiến hành tính toán giá trị tấn công, phòng thủ ứng với mỗi ô cờ có thể đi tiếp được từ trạng thái t (tức là những ô cờ có sở hữu =0), sau đó sẽ ngẫu nhiên chọn ra một trong các ô cờ có giá trị tấn công, phòng thủ lớn nhất để làm output. Ta gọi output vừa tìm được là t1, là một trong các trang thái con của t. Tương tự, ta gọi lại hàm GetMaxNode với t thêm 2 lần để tìm tiếp t2, t3. Như vậy ta có tập các node tiếp theo của t là {t1,t2,t3}. Sau đó, với t1, gọi hàm MinValue để định giá trị cho t1, trong hàm MinValue, ta gọi hàm Định giá trạng thái kết thúc để định giá cho t1, tiếp theo gọi lại hàm GetMaxNode để lấy 3 node con của t1 là tập {t11,t12,t13}. Với từng phần tử con của t1 là t11, t12, t13, ta lần lượt gọi MaxValue với độ sâu + 1 để định giá cho chúng. Tương tự vậy, đến khi đạt điều diện dừng là độ sâu hiện tại >= độ sâu tối đa (depth >= maxdepth) hoặc giá trị kết thúc quá lớn (nghĩa là trạng thái đó đã rất gần trạng thái kết thúc) thì dừng đệ quy và xuất ngược về điểm trạng thái cho hàm trên.

**CHƯƠNG 4: ỨNG DỤNG**

**4.1. Giới thiệu chương trình ứng dụng**

- Ứng dụng game caro là một ứng dụng có chức năng giải trí. Giao diện trò chơi gồm có một bàn cờ kích thước 20 x 20 ô cờ, một thanh menu và các button dùng để thực hiện các thao tác cơ bản của một trò chơi đánh cờ như bắt đầu chế độ chơi với người, bắt đầu chế độ chơi với máy, hoàn lại nước đi, bỏ hoàn lại nước đi.

- Cách chơi: Người dùng sẽ chọn chế độ chơi mong muốn, nếu chọn chế độ chơi người với người, lượt đi sẽ được thay đổi luân phiên giữa 2 người chơi với nhau. Nếu chọn chế độ chơi với máy, máy tính được mặc định được đánh trước vào ô cờ ở vị trí trung tâm bàn cờ, sau đó lượt chơi sẽ chuyển lại cho người chơi và luân phiên như vậy đến khi kết thúc trò chơi.

**4.2. Cài đặt**

- Chương trình được viết trên nền ngôn ngữ Winform C#, sử dụng IDE Visual Studio phiên bản Community 2019.

- Các hàm code chính của chương trình:

+Hàm khởi tạo

public void KhoiTaoMangOCo() // khởi tạo một mảng 2 chiều gồm các phần tử là Ô cờ, là cơ sở để vẽ bàn cờ lên giao diện chính

{

for (int i = 0; i < \_BanCo.SoDong; i++)

{

for (int j = 0; j < \_BanCo.SoCot; j++)

{

\_MangOCo[i, j] = new OCo(i, j, new Point(j \* OCo.\_ChieuRong, i \* OCo.\_ChieuCao), 0); //khởi tạo các giá trị mặc định cho mỗi ô cờ,( dòng, cột, tọa độ thực, sở hữu)

}

}

}

+ Hàm đánh cờ

public bool DanhCo(int MouseX, int MouseY, Graphics g) // hàm thực hiện thao tác đánh 1 ô cờ vào 1 vị trí chỉ định bằng thao tác click chuột trái

{

if (MouseX % OCo.\_ChieuRong == 0 || MouseY % OCo.\_ChieuCao == 0) // kiểm tra vị trí chuột có trùng với đường kẻ giữa bàn cờ hay không.

{

return false; // nếu có không thực hiện thao tác vẽ

}

int Cot = MouseX / OCo.\_ChieuRong; // tạo biến lưu giá trị tại cột tương ứng với tọa độ chuột

int Dong = MouseY / OCo.\_ChieuCao; // tạo biến lưu giá trị tại dòng tương ứng với tọa độ chuột

if (\_MangOCo[Dong, Cot].SoHuu != 0) // nếu ô cờ đã thuộc sở hữu của bên nào đó

return false; // nếu có không thực hiện thao tác vẽ

switch (\_LuotDi) // kiểm tra lượt đi hiện tại

{

default:

MessageBox.Show("Có lỗi");

break;

case 1: // nếu = 1

\_MangOCo[Dong, Cot].SoHuu = 1; // gán sở hữu của phần tử Ô cờ tương ứng với tại vị trí dòng cột đã khai báo ở trên = 1

\_BanCo.VeQuanCo(g, \_MangOCo[Dong, Cot].ViTri, sbBlack); /vẽ ô cờ lên giao diện

\_LuotDi = 2; //chuyển lượt đi hiện tại thành 2

break;

case 2: // tương tự như case 1

\_MangOCo[Dong, Cot].SoHuu = 2;

\_BanCo.VeQuanCo(g, \_MangOCo[Dong, Cot].ViTri, sbWhite);

\_LuotDi = 1;

break;

}

+ Hàm bật chế độ chơi với máy, thực hiện thao tác đánh một nước cờ mới cho máy

public void KhoiDongComputer(Graphics g)

{

if (stkCacNuocDaDi.Count == 0)

{

DanhCo(\_BanCo.SoCot / 2 \* OCo.\_ChieuRong + 1, \_BanCo.SoDong / 2 \* OCo.\_ChieuCao + 1, g); //nếu bàn cờ chưa có gì, đánh vào ô ở giữa bàn cờ

}

Else

{

OCo oco = ChonNuocDiTotNhat(); // tạo 1 ô cờ được coi là tốt nhất bằng hàm ChonNuocDiTotNhat()

DanhCo(oco.ViTri.X + 1, oco.ViTri.Y + 1, g); // đánh cờ vào ô cờ vừa tạo

}

}

+ Hàm chọn nước đi tốt nhất dựa vào trạng thái của bàn cờ hiện tại:

private OCo ChonNuocDiTotNhat()

{

int[,] mangTrangthaiHienTai = new int[\_BanCo.SoDong, \_BanCo.SoCot]; // khởi tạo một mảng trạng thái tạm để thực hiện các thao tác giả định, giống hệt với cấu trúc bàn cờ

for (int i = 0; i < \_BanCo.SoDong; i++)

{

for (int k = 0; k < \_BanCo.SoCot; k++)

{

mangTrangthaiHienTai[i, k] = \_MangOCo[i, k].SoHuu; // khởi tạo giá trị cho mảng trạng thái tạm

}

}

OCo KetQua; // tạo một ô cờ để lưu kêt quả, là output cho hàm

List<Point> DanhSachTrangThaiCanXemXet = new List<Point>(); //tạo danh sách lưu các giá trị trang thái kế tiếp

for (int i = 0; i < \_branch; i++) // tiến hành thêm giá trị cho danh sách các trạng thái kế tiếp, số làn lặp tương ứng với số lượng nhánh mong muốn mở rộng

DanhSachTrangThaiCanXemXet.Add(GetMaxNode(mangTrangthaiHienTai)); // lần lượt tìm các trạng thái kế tiếp tốt nhất từ trạng thái hiện tại

int[] MangGiaTriCuaTungTrangThai = new int[\_branch]; // tạo một mảng giá trị để lưu giá trị độ tốt của danh sách các trạng thái kế tiếp ở trên, sẽ được đánh giá bằng hàm MinValue cho từng trạng thái tiếp theo.

int alpha = -10000, beta = 10000, depth = 0; // khởi tạo các giá trị alpha, beta, độ sâu hiện tại cho trạng thái hiện tại

int j = 0;

foreach (Point i in DanhSachTrangThaiCanXemXet) // với mỗi trạng thái trong danh sách các trạng thái kế tiếp

{

int v = MinValue(mangTrangthaiHienTai, i, alpha, beta, depth); // tính toán giá trị tương ứng của từng trạng thái.

MangGiaTriCuaTungTrangThai[j] = v; // Gán cho mảng lưu độ tốt tương ứng. Vì đây mặc định là lượt đi của máy, nên trạng thái hiện tại thuộc lớp Max => giá trị của các trạng thái kế tiếp sẽ được tính bằng hàm MinValue, sau đó sẽ chọn max.

j++;

}

int tmp = 0; // tạo biến tạm, hỗ trợ cho việc tìm các trạng thái có giá trị định giá max

List<Point> list = new List<Point>(); //tạo danh sách chứa tọa độ các trạng thái mang giá trị được định giá tốt nhất, dùng để random chọn ra được tọa độ kết quả

for (int i = 0; i < MangGiaTriCuaTungTrangThai.Length; i++)

{

if (MangGiaTriCuaTungTrangThai[tmp] < MangGiaTriCuaTungTrangThai[i])

{

tmp = i; //tmp là giá trị định giá lớn nhất

}

}

for (int i = 0; i < MangGiaTriCuaTungTrangThai.Length; i++)

{

if (MangGiaTriCuaTungTrangThai[tmp] == MangGiaTriCuaTungTrangThai[i])

{

list.Add(new Point(DanhSachTrangThaiCanXemXet[i].X, DanhSachTrangThaiCanXemXet[i].Y)); //lần lượt thêm tọa độ các trạng thái có giá trị tốt nhất trong mảng giá trị vào danh sách tọa độ các trạng thái kết quả cuối cùng

}

}

int x = rand.Next(0, list.Count); //random 1 index

KetQua = new OCo(list[x].X, list[x].Y, \_MangOCo[list[x].X, list[x].Y].ViTri, 1); // gán giá trị cho ô cờ kết quả với sở hữu = 1 và có vị trí tại tọa độ được random ở trên. Đây chính là ô cờ được đánh giá tốt nhất, là bước đi mà máy sẽ chọn để đi.

return KetQua;

}

+ Hàm tính giá trị cho các node con của lớp Max và thực hiện đệ quy tới lớp Min:

private int MaxValue(int[,] MangTrangThaiHienTai, Point NuocDi, int alpha, int beta, int depth)

{

int LuotDiGiaDinh = 1; //Lượt đi hiện tại, là lượt đi của máy – thuộc về lớp Max

int value = DinhGiaTrangThaiKetThuc(MangTrangThaiHienTai, LuotDiGiaDinh); //Biến lưu giá trị kết thúc được đánh giá bằng hàm DinhGiaTrangThaiKetThuc của trạng thái hiện tại

if (depth >= maxdepth || Math.Abs(value) > 3000) // nếu thỏa điều kiện dừng

return value; // trả về value, coi như node này là node dừng (không có node con)

List<Point> DanhSachCacNodeCon = new List<Point>(); //tạo một danh sách lưu tọa độ của các trạng thái kế

for (int i = 0; i < \_branch; i++)

{

DanhSachCacNodeCon.Add(GetMaxNode(MangTrangThaiHienTai)); // thêm các trạng thái kế

}

for (int i = 0; i < DanhSachCacNodeCon.Count; i++) //với mỗi phần tử của mảng các trạng thái kề

{

MangTrangThaiHienTai[DanhSachCacNodeCon[i].X, DanhSachCacNodeCon[i].Y] = 1; //đánh thử

alpha = Math.Max(beta, MinValue(MangTrangThaiHienTai, DanhSachCacNodeCon[i], alpha, beta, depth + 1)); // giá trị alpha được tính bằng cách lấy max(alpha hiện tại, giá trị của các node con ở lớp Min tiếp theo)

MangTrangThaiHienTai[DanhSachCacNodeCon[i].X, DanhSachCacNodeCon[i].Y] = 0; //bỏ đánh thử

if (alpha >= beta) //nếu alpha >= beta

break; // cắt tỉa

}

return alpha; //trả về giá trị được định giá của trạng thái hiện tại là alpha (đang ở lớp Max nên lấy alpha làm giá trị của node)

}

+ Hàm tính giá trị cho các node con của lớp Min và thực hiện đệ quy tới lớp Max:

private int MinValue(int[,] MangTrangThaiHienTai, Point NuocDi, int alpha, int beta, int depth)

{

int LuotDiGiaDinh = 2; //Lượt đi hiện tại, là lượt đi của máy – thuộc về lớp Min

int value = DinhGiaTrangThaiKetThuc(MangTrangThaiHienTai, LuotDiGiaDinh);

//Biến lưu giá trị kết thúc được đánh giá bằng hàm DinhGiaTrangThaiKetThuc của trạng thái hiện tại

if (depth >= maxdepth || Math.Abs(value) > 3000) // nếu thỏa điều kiện dừng

return value; // trả về value, coi như node này là node dừng (không có node con)

List<Point> DanhSachCacNodeCon = new List<Point>();//tạo một danh sách lưu tọa độ của các trạng thái kế

for (int i = 0; i < \_branch; i++)

{

DanhSachCacNodeCon.Add(GetMaxNode(MangTrangThaiHienTai)); // thêm các trạng thái kế

}

for (int i = 0; i < DanhSachCacNodeCon.Count; i++)//với mỗi phần tử của mảng các trạng thái kề

{

MangTrangThaiHienTai[DanhSachCacNodeCon[i].X, DanhSachCacNodeCon[i].Y] = 2; //đánh thử

beta = Math.Min(beta, MaxValue(MangTrangThaiHienTai, DanhSachCacNodeCon[i], alpha, beta, depth + 1)); // giá trị beta được tính bằng cách lấy min(beta hiện tại, giá trị của các node con ở lớp Min tiếp theo)

MangTrangThaiHienTai[DanhSachCacNodeCon[i].X, DanhSachCacNodeCon[i].Y] = 0; //bỏ đánh thử

if (alpha >= beta) //nếu alpha >= beta

break; // cắt tỉa

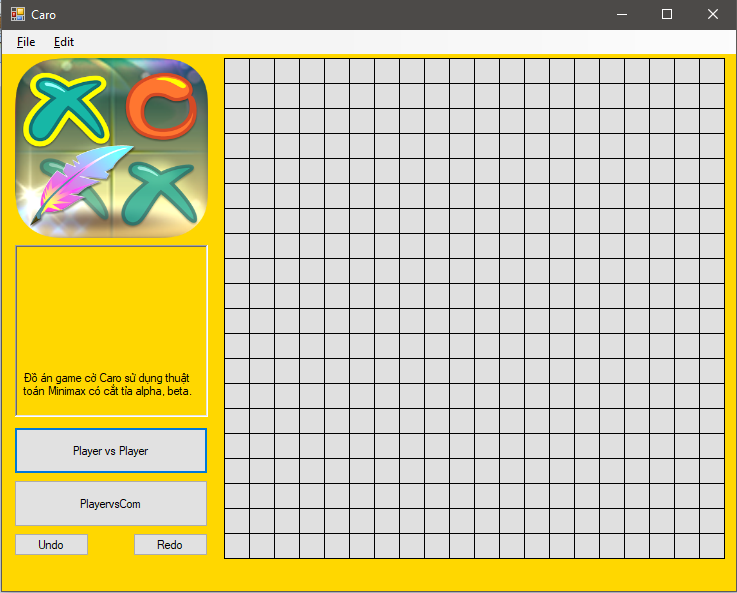
}

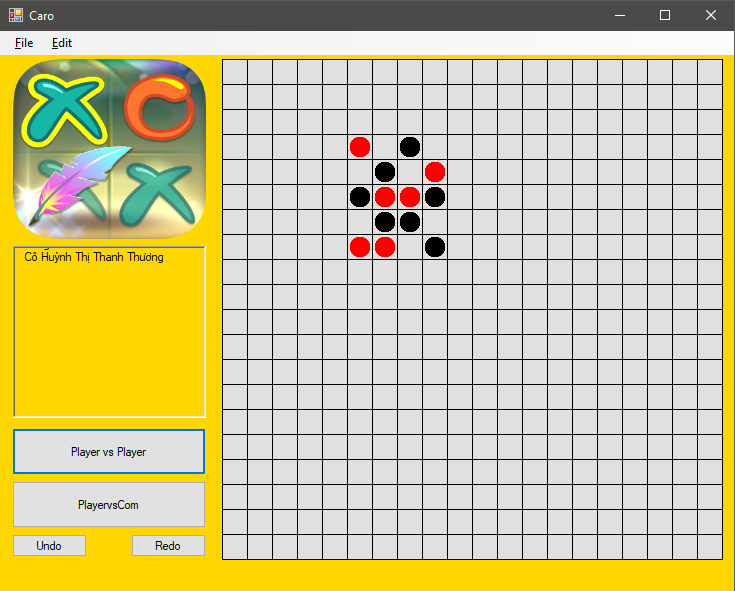
return beta; //trả về giá trị được định giá của trạng thái hiện tại là alpha (đang ở lớp Max nên lấy alpha làm giá trị của node)

}

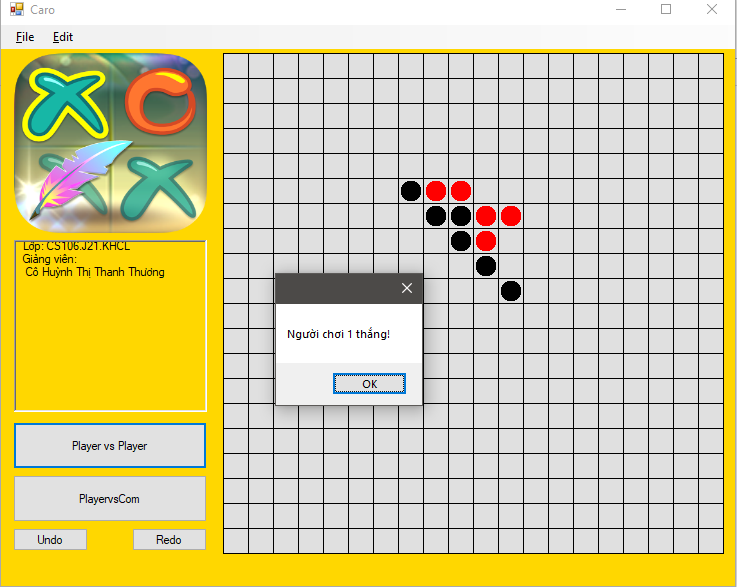
**4.3. Kết quả chạy chương trình:**

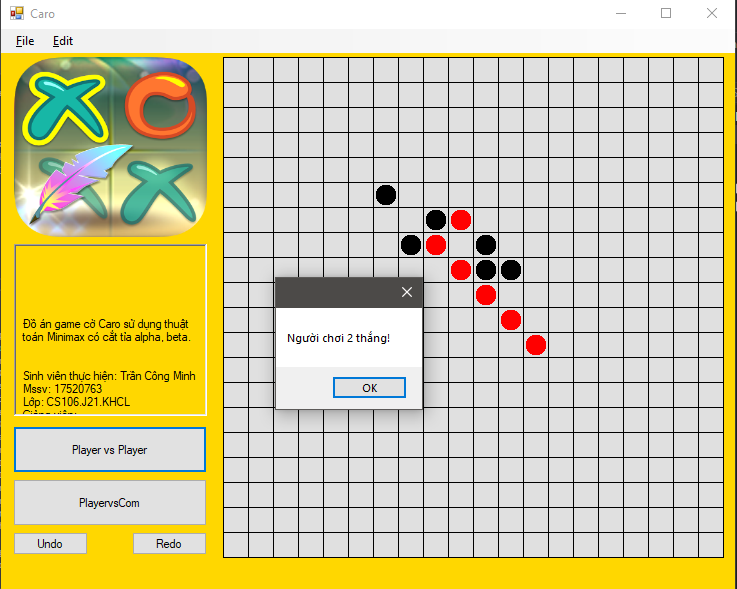
**-** Giao diện khởi tạo của trò chơi:

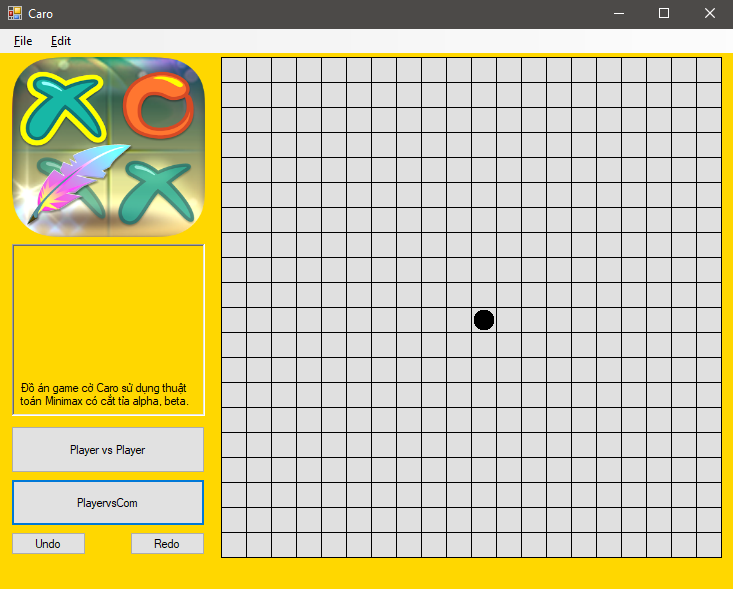
****

**-** Khi ở chế độ người với người:

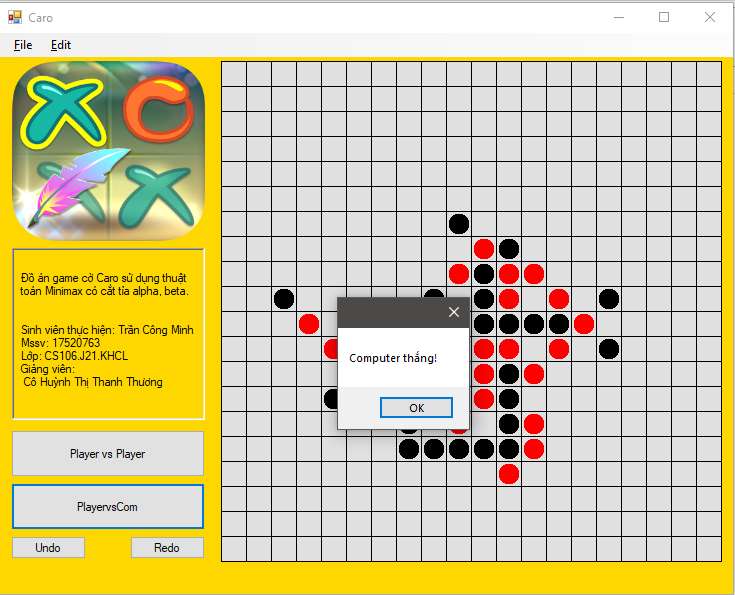
**-** Khi người chơi 1 thắng:



- Khi người chơi 2 thắng:

- Khi bắt đầu chế độ chơi với máy:

- Khi máy thắng:



- Nhận xét chung:

+ Ưu điểm: Chương trình có giao diện đơn giản, thân thiện với người dùng, có các chức năng đáp ứng được các thao tác cơ bản của một ứng dụng chơi cờ. Chương trình đánh cờ với máy cho kết quả chấp nhận được.

+ Hạn chế: Chương trình còn sơ sài, thiếu nhiều chức năng cơ bản như lưu trữ kết quả, lựa chọn trình độ chơi cho máy,... Phần thiết kế chức năng tự động đánh cờ còn kém thông minh vì lựa chọn bộ số để đánh giá thế cờ chưa thực sự tốt, các nước đi đầu tiên của máy còn bị chậm do thời gian tính toán khá lâu.

**CHƯƠNG 5: KẾT LUẬN**

**5.1. Kết quả đạt được**

- Một chương trình đánh cờ, có thể đáp ứng tương đối nhu cầu chơi cờ Caro cho mọi đối tượng người dùng.

**5.2. Hạn chế**

- Chương trình còn nhiều thiếu sót và chưa đầy đủ về các thao tác của một ứng dụng chơi cờ. Khả năng chơi cờ của máy cần được cải tiến thêm.

**5.3. Hướng phát triển**

- Có thể cải tiến thành phiên bản game cờ Caro mới với đầy đủ các chức năng của một game chơi cờ. Cài đặt thêm một số thao tác nâng cao và gia tăng khả năng xử lý của máy thông qua việc xem xét, chọn lựa được các bộ giá trị tốt hơn để định giá nước đi. Có thể thêm chức năng chọn trình độ chơi của máy dựa vào thay đổi giá trị “độ sâu tối đa” của thuật giải tìm nước đi tốt nhất cho máy.

**TÀI LIỆU THAM KHẢO:**

Ý tưởng giải quyết vấn đề: tham khảo từ source code tìm được trên mạng, nguồn: “<http://diendan.congdongcviet.com/threads/t57585::source-code-game-caro-viet-bang-csharp.cpp>”

- Về lập trình: Sinh viên tự viết hầu hết các hàm và giao diện dựa vào tài liệu tham khảo và kế thừa cách tính toán của source code chứ không copy nguyên si source, cụ thể:

+ Tự viết lại hàm MinValue, MaxValue, ChonNuocDiTotNhat, sử dụng cấu trúc dữ liệu khác với cấu trúc dữ liệu của source tham khảo. (Chỉ tham khảo ý tưởng, và cách thức viết hàm, tính toán giá trị cho từng hàm, không copy nguyên si lại cấu trúc dữ liệu của source)

+ Các hàm sinh viên kế thừa gần giống với source code: Hàm GetMaxNode, sinh viên kế thừa cách tính toán các nước đi, cách random và chọn nước đi bất kì của source, chỉ điều chỉnh lại các thao tác cho phù hợp với cấu trúc dữ liệu đang sử dụng.

+ Về phần giao diện và xử lý thắng thua: sinh viên tự code lại theo series video “Hướng dẫn lập trình C# nâng cao bằng game Caro”