HỌC VIỆN HÀNG KHÔNG VIỆT NAM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO TIỂU LUẬN
TRÍ TUỆ NHÂN TẠO
"Ứng dụng thuật toán minimax với cắt tỉa alpha-beta trong trò chơi Caro"

HỌC KỲ 1 – NĂM HỌC: 2023 - 2024

MÃ LỚP HỌC PHẦN: 010100085801

Giảng viên hướng dẫn: TS. Bùi Mạnh Quân

Nhóm sinh viên thực hiện: Lê Cao Tấn Lộc - 2154810086

Nguyễn Thị Thúy Hà - 2154810061

Phan Tường Bảo Trâm - 2154810077

Trịnh Vinh Qui - 2154810110

TP. HCM, tháng 11 năm 2023

MỤC LỤC

NHẬN XÉT CỦA GIẢNG VIÊN

ĐÁNH GIÁ:

ĐIỂM SỐ:----------

ĐIỂM CHỮ: ----------

Chữ ký giảng viên

LỜI CẢM ƠN

Xin chân thành cảm ơn Học viện Hàng không Việt Nam và khoa Công nghệ thông tin đã đưa môn Trí tuệ nhân tạo cũng như tạo điều kiện và hỗ trợ suốt quá trình học tập của chúng em. Đồng thời nhóm em cũng không quên gửi lời cảm ơn đến thầy Bùi Mạnh Quân đã đồng hành và tận tình dẫn dắt, truyền dạy những kiến thức bổ ích, những kinh nghiệm quý báu cho chúng em trong quá trình nghiên cứu đề tài này; đây chắc chắn là những kiến thức quan trọng, là hành trang để nhóm em đạt được nhiều thành quả hơn nữa. Nhờ những kiến thức bổ ích đó cùng với sự dẫn dắt và chỉ bảo tận tình của thầy đã giúp đề tài của nhóm hoàn thành thuận lợi. Đồng thời, cảm ơn những thành viên trong nhóm đã nhiệt tình trao đổi, đóng góp ý kiến giúp cho bài báo cáo hoàn thành đúng thời gian quy định. Tuy đã có nhiều cố gắn nhưng do kiến thức khá sâu rộng và thời gian nghiên cứu có hạn nên đề tài của nhóm em sẽ không tránh khỏi những sai sót. Chúng em rất mong nhận được sự góp ý của các thầy để chủ đề này có điều kiện hoàn thiện hơn. Sau cùng, nhóm chúng em kính chúc quý thầy luôn dồi dào sức khỏe, niềm tin để tiếp tục thực hiện sứ mệnh cao đẹp của mình là truyền đạt kiến thức cho thế hệ sau.

LỜI NÓI ĐẦU

Ngày nay, công nghệ thông tin không chỉ là một lĩnh vực mà còn là một lực lượng mạnh mẽ thúc đẩy sự phát triển toàn cầu. Việc áp dụng công nghệ vào các lĩnh vực và ngành nghề khác nhau không chỉ mang lại hiệu suất cao mà còn tạo ra những cơ hội mới và giải pháp sáng tạo. Trong thời kỳ bùng nổ của công nghệ thông tin này, việc áp dụng các thuật toán thông minh vào trò chơi không chỉ mang lại trải nghiệm chơi game hiện đại mà còn mở ra nhiều khía cạnh mới của nghiên cứu và phát triển, đặc biệt là các thuật toán tìm đường đi tối ưu cho các trò chơi. Thuật toán minimax alpha-beta-pruning là một thuật toán không thể thiếu trong việc tìm nước đi của các trò chơi đối kháng, đặc biệt là cờ Caro. Tuy nhiên, để đáp ứng nhu cầu giải trí đa dạng và linh hoạt của người chơi hiện đại, nhóm em quyết định khám phá khả năng ứng dụng thuật toán này, một trong những thuật toán trí tuệ nhân tạo mạnh mẽ. Bài nghiên cứu này sẽ giúp ta hiểu hơn về thuật toán minimax và cắt tỉa alpha-beta và việc áp dụng thuật toán vào trò chơi cờ caro như thế nào, cũng như cách vận dụng những thuật toán thông minh vào thực tế.

Nhóm chúng em xin cam đoan tiểu luận này do các thành viên trong nhóm thực hiện. Chúng em không sao chép, sử dụng bất kỳ tài liệu, mã nguồn… của người khác mà không ghi nguồn. Chúng em xin chịu hoàn toàn trách nhiệm nếu vi phạm đạo văn.

CHƯƠNG I. GIỚI THIỆU

1. Lý do chọn đề tài

Trong thời đại hiện nay, khi công nghệ thông tin đang trên đà phát triển mạnh mẽ, chứng kiến sự xuất hiện của những tựa game phức tạp và đầy ấn tượng. Mặc dù vậy, những trò chơi cổ điển vẫn giữ được sức hấp dẫn riêng. Trong số những trò chơi cổ điển, cờ caro là một trò chơi phổ biến, thường được người chơi trải nghiệm nhiều nhất ở các lớp học hay giữa những buổi giải lao. Trước đây cờ caro được chơi đơn giản với giấy vở, cờ caro không chỉ mang lại giây phút giải trí mà còn tạo ra không khí giao lưu, tương tác trong nhóm và còn là ván cờ tư duy thách thức biết bao thế hệ. Tuy nhiên, nhận thức về nhu cầu giải trí hiện đại, nơi mà sự thuận tiện và linh hoạt được đặt lên hàng đầu, nhóm em quyết định phát triển một ứng dụng cờ caro mới dựa trên kiến thức đã học ở môn này. Người chơi không chỉ có thể thưởng thức cờ caro mọi lúc, mọi nơi, mà còn đối mặt với một đối thủ máy tính thông minh.

Nhằm thúc đẩy khả năng ứng dụng thực tế các thuật toán, vận dụng sáng tạo các kiến thức đã học từ môn Trí tuệ nhân tạo để giải quyết những bài toán về tìm đường đi tối ưu cho trò chơi. Sau khi trải qua thời gian tìm hiểu về trò chơi caro, nhóm em đã nhận ra đa số các tác giả của trò chơi đều cố định kích thước của bàn cờ và che dấu độ sâu tìm kiếm của thuật toán. Tuy nhiên, nhóm chúng em có mong muốn việc kích thước bàn cờ hay độ sâu đều thay đổi được để có nhiều trải nghiệm, sáng tạo hơn và giúp người chơi có thể không nhàm chán.

Sau khi nhận được tính cấp thiết trên, nhóm chúng em đã lựa chọn đề tài ứng dụng thuật toán minimax cắt tỉa alpha-beta xây dựng trò chơi cờ Caro có khả năng cài đặt độ khó của trò chơi bằng kích thước bàn cờ và độ sâu tìm kiếm.

2. Đối tượng, phương pháp và phạm vi nghiên cứu của đề tài

- **Đối tượng:** Để làm rõ đề tài trên nhóm chúng em tập trung vào một số đối tượng như giải thuật minimax, phương pháp cắt tỉa alpha-beta, ngôn ngữ Python và các thư viện như: Thư viện turtle, Thư viện tkinter, Thư viện messagebox.

- **Phương pháp nghiên cứu:**

- **Nghiên cứu tài liệu:** Bắt đầu bằng việc thu thập và phân tích thông tin liên quan đến giải thuật minimax, cắt tỉa alpha-beta và ứng dụng của chúng trong trò chơi

Caro. Tài liệu, nghiên cứu trước đây và các nguồn thông tin khác sẽ được tận dụng để hình thành ý tưởng tổng quan về ứng dụng.

- **Xác định yêu cầu phân tích thiết kế:** Xác định các yêu cầu cụ thể để phân tích và thiết kế hệ thống, đảm bảo rằng quy trình ứng dụng giải thuật minimax với cắt tỉa alpha-beta vào trò chơi Caro sẽ được thực hiện một cách chặt chẽ và hiệu quả.
- **Phân tích thiết kế thuật toán:** Sau khi có một kế hoạch chi tiết, tiến hành việc lập trình game theo những yêu cầu đã đặt ra. Xây dựng mô hình, tích hợp giải thuật minimax và cắt tỉa alpha-beta vào hệ thống game, đồng thời đảm bảo tính linh hoạt và khả năng mở rộng của ứng dụng.
- **Phân tích thiết kế giao diện:** Khi các thuật toán đã được tiến hành phân tích hoàn chỉnh, nhóm chúng em sẽ tiến hành thiết kế giao diện phù hợp với game cờ Caro để thể hiện các nước đi của thuật toán một cách chính xác
- **Hoàn thiện và đánh giá:** Cuối cùng, thực hiện quá trình hoàn thiện game và đánh giá kết quả đạt được. Kiểm thử hệ thống để đảm bảo tính ổn định và hiệu suất của giải thuật trong môi trường thực tế của trò chơi Caro. Từ đó, thu được các kết luận và đề xuất các cải tiến nếu cần thiết.

- **Phạm vi nghiên cứu:**

- **Phạm vi không gian:** Đề tài nghiên cứu được nhóm chúng em thực hiện tại Học Viện Hàng Không Việt Nam.
- **Phạm vi nội dung:** Nội dung của đề tài sẽ xoay quanh việc nghiên cứu và phân tích giải thuật minimax với cắt tỉa alpha-beta, đồng thời tập trung vào ứng dụng cụ thể của giải thuật này trong trò chơi Caro. Chúng em sẽ khám phá cơ sở lý thuyết, xây dựng mô hình, thực hiện các thử nghiệm và đánh giá hiệu suất để có cái nhìn toàn diện về khả năng áp dụng của giải thuật trong ngữ cảnh cụ thể của trò chơi Caro.

CHƯƠNG II. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ ỨNG DỤNG

1. CÔNG NGHỆ ỨNG DỤNG

1.1. Môi trường, thư viện

- Thư viện **turtle:** Thư viện Turtle trong Python là một công cụ đồ họa cơ bản được sử dụng để vẽ các hình học và hoạt động về hình ảnh trên màn hình. Thư viện Turtle là một phần của thư viện tiêu chuẩn của Python và cung cấp một cách dễ dàng để tạo ra các hình vẽ đơn giản bằng cách sử dụng các lệnh cơ bản.
- Thư viện **tkinter:** Tkinter là một bộ công cụ GUI (Graphical User Interface) mặc định đi kèm với Python. Tên gọi "Tkinter" là viết tắt của "Tk Interface," trong đó Tk là một toolkit đồ họa.

  Các tính năng:

  + Tạo và quản lý cửa sổ đồ họa.
  + Tạo các thành phần giao diện người dùng như button, label, entry, listbox, và nhiều thành phần khác.
  + Xử lý sự kiện như nút được nhấn, chuột được click, và các sự kiện khác.

- Thư viện **messagebox:** Thư viện messagebox trong Python là một phần của thư viện Tkinter và được sử dụng để hiển thị các hộp thoại thông báo đơn giản cho người dùng. Trong phạm vi ứng dụng này, chúng em sử dụng thư viện để thông báo người chơi thắng cuộc và kết thúc trò chơi.

1.2. Ngôn ngữ lập trình

Python - một trong những ngôn ngữ lập trình phổ biến và được sử dụng rộng rãi nhất hiện nay. Với nhiều ứng dụng trong công việc và tính đơn giản của nó, Python đang dần trở thành kỹ năng cần nắm bắt và học tập với nhiều người. Là ngôn ngữ lập trình hướng đối tượng đa năng có cấu trúc dữ liệu cấp cao mạnh mẽ và hệ thống thư viện lớn, Python hoàn toàn tạo kiểu động và sử dụng cơ chế cấp phát bộ nhớ tự động, cú pháp rõ ràng, đơn giản. Python dễ hiểu, dễ tiếp cận, ngay cả đối với người chưa từng học lập trình.

2. CƠ SỞ LÝ THUYẾT.

2.1 Thuật toán Minimax

Thuật toán Minimax là một thuật toán được sử dụng trong trò chơi hai người, như cờ vua hoặc tic-tac-toe, để đưa ra quyết định cho mỗi nước đi dựa trên việc tối ưu hóa lợi ích (điểm số) của người chơi và đồng thời giảm thiểu lợi ích của đối thủ.

Thuật toán Minimax hoạt động bằng cách lựa chọn nước đi tối ưu cho người chơi (MAX) và đồng thời giả định đối thủ sẽ chọn nước đi tối ưu nhất cho mình (MIN). Nó thực hiện đệ quy để duyệt qua cây game, mô phỏng tất cả các nước đi có thể có và tính toán giá trị đánh giá cho mỗi trạng thái.

Quá trình chơi cờ là quá trình mà Trắng và Đen thay phiên nhau đưa ra các nước đi hợp lệ cho đến khi dẫn đến trạng thái kết thúc cuộc chơi. Quá trình này biểu diễn bởi đường đi từ nút gốc tới nút lá trên cây trò chơi. Giả sử tại một đỉnh u nào
đó trên đường đi, nếu u là đỉnh Trắng (Đen) thì cần chọn một nước đi nào đó đến một trong các đỉnh con Đen (Trắng) v của u. Tại đỉnh Đen (Trắng) v sẽ chọn đi tiếp đến một đỉnh con Trắng (Đen) w của v. Quá trình này tiếp tục cho đến khi đạt đến một đỉnh lá của cây. Chiến lược tìm nước đi của Trắng hay Đen là luôn tìm những nước đi dẫn tới trạng thái tốt nhất cho mình và tồi nhất cho đối thủ. Giả sử Trắng cần tìm nước đi tại đỉnh u, nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con v sao cho v là tốt nhất trong số các đỉnh con của u. Đến lượt Đen chọn nước đi từ v, Đen cũng chọn nước đi tốt nhất cho mình. Để chọn nước đi tối ưu cho Trắng tại đỉnh u, cần xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá ứng với giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen.

Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất (các đỉnh lá) lên gốc u. Giả sử cần xác định giá trị của đỉnh v mà các đỉnh con của nó đã xác định. Khi đó, nếu v là đỉnh Trắng thì giá trị của nó là giá trị lớn nhất trong các đỉnh con, nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các đỉnh con.

- **MINIMAX-SEARCH**

Đây là hàm chính của thuật toán Minimax. Hàm này nhận hai tham số: game, đại diện cho trò chơi đang được chơi và state: đại diện cho trạng thái hiện tại của trò
chơi. P layer được thiết lập là người chơi hiện tại, được xác định bởi game.TOMOVE(state). Sau đó, gọi hàm MAX-VALUE(game, state) để tìm nước đi tốt nhất. Cuối cùng, nó trả về nước đi tốt nhất được tìm thấy.

- **MAX-VALUE:**

Hàm này sẽ tìm giá trị cực đại mà người chơi hiện tại có thể đạt được trong trạng thái đã cho. Nếu trạng thái là trạng thái kết thúc (cuối trò chơi), nó trả về giá trị tiện ích của trạng thái đó cho người chơi hiện tại và một nước đi null (không cần nước đi). Biến v được khởi tạo là âm vô cùng (đại diện cho giá trị tồi nhất có thể đối với người chơi đang cố gắng tối đa hóa). Lặp qua tất cả các hành động có thể thực hiện (a) từ trạng thái hiện tại. Đối với mỗi hành động, nó đệ quy gọi MINVALUE để tìm giá trị tiện ích tối thiểu mà đối thủ có thể đạt được trong trạng thái kết quả (v2). Nếu v2 lớn hơn v hiện tại, nó cập nhật v và move với các giá trị mới. Cuối cùng, trả về giá trị tiện ích cực đại (v) và nước đi tương ứng (move) dẫn đến nó

- **MIN-VALUE**

Hàm này tương tự như MAX-VALUE, nhưng nó được sử dụng cho đối thủ (người chơi cố gắng tối thiểu hóa). Nếu trạng thái là trạng thái kết thúc, nó trả về giá trị tiện ích cho người chơi hiện tại và nước đi null. Khởi tạo v là dương vô cùng (đại diện cho giá trị tốt nhất có thể đối với người chơi đang cố gắng tối thiểu hóa). Tương tự như MAX-VALUE, nó lặp qua tất cả các hành động có thể thực hiện, nhưng lần này nó cố gắng tìm giá trị tiện ích cực đại (v2) cho người chơi đang cố gắng tối đa hóa. Nếu v2 nhỏ hơn v hiện tại, nó cập nhật v và move với các giá trị mới. Cuối cùng, trả về giá trị tiện ích tối thiểu (v) và nước đi tương ứng (move) dẫn đến nó.

2.2 Phương pháp cắt tỉa Alpha – Beta

Cắt tỉa Alpha-Beta (Alpha-Beta Pruning) là một kỹ thuật được sử dụng trong thuật toán Minimax để giảm số lượng nút cần được đánh giá trên cây tìm kiếm, cải thiện hiệu suất tính toán. Kỹ thuật này làm giảm đáng kể thời gian tính toán và làm cho thuật toán có khả năng xử lý các cây tìm kiếm lớn hơn.

Ý tưởng chính của cắt tỉa Alpha-Beta là loại bỏ những nút trên cây tìm kiếm mà chúng ta đã biết sẽ không ảnh hưởng đến kết quả cuối cùng của thuật toán Minimax. Kỹ thuật này sử dụng hai giá trị, được gọi là “alpha” và “beta” để theo dõi giới hạn tốt nhất hiện tại đối với các giá trị tối đa và tối thiểu tương ứng.

“Alpha” là giới hạn tốt nhất hiện tại cho các giá trị tối đa mà người chơi Max có thể đạt được. Ban đầu, alpha được gán là âm vô cùng (đại diện cho giá trị tối thiểu). “Beta” là giới hạn tốt nhất hiện tại cho các giá trị tối thiểu mà đối thủ Min có thể đạt được. Ban đầu, beta được gán là dương vô cùng (đại diện cho giá trị tối đa).

Thuật toán Alpha-beta pruning hoạt động với tìm kiếm theo chiều sâu đến một giới hạn độ sâu đã được đặt trước. Như vậy, cây tìm kiếm được duyệt từ trái sang phải. Khi cây tìm kiếm được duyệt qua, Alpha-Beta Pruning kiểm tra giữa alpha và beta để quyết định xem nếu chúng ta có thể cắt tỉa (prune) các nhánh con hoặc không. Cụ thể:
Khi duyệt một nút tối đa (Max node), nếu giá trị của nút này (đại diện cho giá trị tối đa của tất cả các nút con) lớn hơn hoặc bằng beta, chúng ta có thể cắt tỉa (prune) các nút con khác của nút này vì chúng sẽ không ảnh hưởng đến kết quả (đối thủ Min đã tìm được giá trị tối thiểu nhỏ hơn). Khi duyệt một nút tối thiểu (Min node), nếu giá trị của nút này (đại diện cho giá trị tối thiểu của tất cả các nút con) nhỏ hơn hoặc bằng alpha, chúng ta có thể cắt tỉa các nút con khác của nút này vì chúng sẽ không ảnh hưởng đến kết quả (người chơi Max đã tìm được giá trị tối đa lớn hơn). Bằng cách áp dụng cắt tỉa Alpha-Beta, chúng ta có thể loại bỏ một số nút trong cây tìm kiếm, giúp giảm đáng kể thời gian tính toán và tăng hiệu suất của thuật toán Minimax.

3. Ưu điểm

Chiến lược chiến thắng: Thuật toán minimax cùng cắt tỉa alpha-beta mang lại chiến lược chơi game thông minh, giúp đối thủ máy tính đưa ra những quyết định có khả năng chiến thắng cao trong trò chơi Caro.

Tối ưu hóa tìm kiếm: Cắt tỉa alpha-beta giúp giảm độ phức tạp tính toán bằng cách loại bỏ các nhánh tìm kiếm không cần thiết, tối ưu hóa hiệu suất và giảm thời gian đáng kể.

Điều chỉnh khả năng chơi: Trong ứng dụng thực tế của chúng em, người chơi có thể cài đặt được kích thước của bàn cờ Caro và độ sâu của thuật toán. Đây là một trong những điểm linh hoạt hơn rất nhiều so với các trò chơi Caro khác với cài đặt kích thước bàn cờ cố định và che dấu độ sâu của thuật toán. Ngoài ra, đa số các trò chơi thường ứng dụng giải thuật tìm kiếm tham lam để đưa ra nước đi tốt nhất. Còn trong ứng dụng của chúng em, thuật toán minimax cắt tỉa alpha beta(Alpha-beta-pruning) được ứng dụng để tìm đường đi tối ưu cho trò chơi.

4. Nhược điểm

Tuy được cài đặt với thuật toán tìm đường đi tối ưu, nhưng trò chơi của nhóm chúng em vẫn có một số lỗi cần cải thiện để có thể hoạt động tốt hơn:

- Về tốc độ chơi game, thời gian để thuật toán đưa ra đường đi tối ưu thật sự chưa nhanh và cần nhiều xử lý phức tạp trước khi có thể đưa ra lời giải tối ưu nhất.
- Về logic trò chơi, khi tăng độ sâu của trò chơi lớn hơn 5, thuật toán thực sự cần rất nhiều thời gian để xử lý và có thể đưa ra nước đi chưa thật sự tối ưu.
- Về giao diện trò chơi, khi người chơi chọn những điểm cờ sát với viền bàn cờ, thuật toán chưa thật sự bắt được hết tất cả các trường hợp để xử lý.

CHƯƠNG III. SẢN PHẨM ĐỒ ÁN

1. Xây dựng các hàm tính toán

1.1. Thuật toán cắt tỉa alpha-beta

Theo cơ sở lý thuyết, ta sẽ xây dựng giải thuật cắt tỉa alpha-beta (Alpha-beta-pruning). Hàm này sẽ nhận vào mảng biểu diễn bàn cờ và trạng thái đi của quân cờ hiện tại để tính toán nước đi tối ưu (điểm cao nhất) cho máy và tính toán nước đi tối thiểu (điểm thấp nhất) của người chơi.

```
def alpha_beta_pruning_search(board, col):
    if col == "w":
        anticol = "b"
    else:
        anticol = "w"

    def max_value(board, col, anticol, depth, alpha, beta):
        if depth == 0:
            return evaluate(board, col, anticol, 0, 0)  # Đánh giá trạng thái hiện tại của bảng

        max_eval = -float("inf")
        moves = sorted(possible_moves(board), key=lambda move: evaluate(board, col, anticol, move[0], move[1]), reverse=True)[:10]

        for move in moves:
            y, x = move
            board[y][x] = col
            eval = min_value(board, col, anticol, depth - 1, alpha, beta)
            board[y][x] = " "  # Hoàn nguyên trạng thái

            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)

            if beta <= alpha:
                break

        return max_eval

    def min_value(board, col, anticol, depth, alpha, beta):
```

```
    if depth == 0:
        return evaluate(board, col, anticol, 0, 0)  # Đánh giá trạng thái hiện tại của bảng

    min_eval = float("inf")
    moves = sorted(possible_moves(board), key=lambda move: evaluate(board, col, anticol, move[0], move[1]), reverse=True)[:10]

    for move in moves:
        y, x = move
        board[y][x] = anticol
        eval = max_value(board, col, anticol, depth - 1, alpha, beta)
        board[y][x] = " "  # Hoàn nguyên trạng thái

        min_eval = min(min_eval, eval)
        beta = min(beta, eval)

        if beta <= alpha:
            break

    return min_eval

def alpha_beta(board, col, anticol, depth, alpha, beta):
    if col == "w":
        best_eval = -float("inf")
    else:
        best_eval = float("inf")
    best_move = None

    moves = sorted(possible_moves(board), key = lambda move: evaluate(board, col, anticol, move[0], move[1]), reverse=True)[:10]

    for move in moves:
        y, x = move
        board[y][x] = col
        eval = min_value(board, col, anticol, depth - 1, alpha, beta)
        board[y][x] = " "  # Hoàn nguyên trạng thái

        if col == "w":
```

```
            if eval > best_eval:
                best_eval = eval
                best_move = move
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        else:
            if eval < best_eval:
                best_eval = eval
                best_move = move
            beta = min(beta, eval)
            if beta <= alpha:
                break

    return best_move

    max_eval = -float("inf")
    best_move = None
    depth = DEPTH  # Điều chỉnh độ sâu tìm kiếm
    alpha = -float("inf")
    beta = float("inf")

    move = alpha_beta(board, col, anticol, depth, alpha, beta)
    if move:
        return move
    else:
        # Nếu không có nước đi nào, trả về giá trị mặc định
        return (0, 0)
```

Trong đó, các đối số hàm sẽ nhận về là board - chứa trạng thái bàn cờ đang chơi và col - chứa trạng thái hiện tại của trò chơi (người chơi nào đang thực hiện nước đi)

Đối số col sẽ giúp cho thuật toán biết được sẽ sử dụng hàm giá trị nào để tính toán nước đi. Nếu nước đi tiếp theo của máy, thuật toán sẽ gọi hàm max_value để tính toán nước đi tốt nhất cho máy, và ngược lại, nếu nước đi của người chơi, thuật toán sẽ trả về giá trị bất lợi.

Ngoài ra, trong thuật toán cắt tỉa alpha-beta, chúng em còn xây dựng thêm chức năng giúp thuật toán có thể được triển khai nhanh hơn và hiệu quả hơn:

1.2. Cắt tỉa hiệu quả hơn

Alpha-beta pruning cắt tỉa một số nhánh không cần thiết của cây tìm kiếm, giảm đáng kể số lượng trạng thái cần đánh giá. Điều này làm cho thuật toán nhanh hơn so với việc không sử dụng cắt tỉa.

```
if beta <= alpha:
    break
```

1.3. Chọn các nút để đánh giá

Bằng cách chọn cẩn thận các nút để đánh giá (sử dụng giá trị alpha và beta), thuật toán giúp tối ưu hóa thời gian thực hiện và không đánh giá một số nút không cần thiết.

1.4. Sắp xếp nước đi theo ưu tiên

Trong mỗi bước, nước đi được sắp xếp theo một tiêu chí để đánh giá trước những nước đi có khả năng tối ưu hóa lớn hơn. Điều này giúp tối ưu hóa việc cắt tỉa alpha-beta.

```
moves = sorted(possible_moves(board), key=lambda move: evaluate(board, col, anticol, move[0], move[1]), reverse=True)[:10]
```

1.5. Giảm độ sâu tìm kiếm không cần thiết

Thuật toán sử dụng độ sâu tìm kiếm để đảm bảo rằng thuật toán sẽ kết thúc trong thời gian hợp lý mà vẫn đưa ra nước đi tốt nhất.

1.6. Áp dụng hàm đánh giá trạng thái thông minh

Hàm đánh giá trạng thái (trong trường hợp này là evaluate) được thiết kế để càng nhanh càng đưa ra ước lượng tốt về chất lượng của trạng thái hiện tại. Điều này giúp thuật toán đưa ra quyết định tốt hơn về cắt tỉa

2. Hàm đánh giá – evaluate

2.1 Hàm đánh giá

```
def evaluate(board, col, anticol, y, x):
    global colors
    M = 1000
    res, adv, dis = 0, 0, 0
```

```
# tấn công
board[y][x] = col
# draw_stone(x,y,colors[col])
sumcol = score_of_col_one(board, col, y, x)
a = winning_situation(sumcol)
adv += a * M
sum_sumcol_values(sumcol)
# {0: 0, 1: 15, 2: 0, 3: 0, 4: 0, 5: 0, -1: 0}
adv += sumcol[-1] + sumcol[1] + 4 * sumcol[2] + 8 * sumcol[3] + 16 * sumcol[4]

# phòng thủ
board[y][x] = anticol
sumanticol = score_of_col_one(board, anticol, y, x)
d = winning_situation(sumanticol)
dis += d * (M - 100)
sum_sumcol_values(sumanticol)
dis += (
    sumanticol[-1]
    + sumanticol[1]
    + 4 * sumanticol[2]
    + 8 * sumanticol[3]
    + 16 * sumanticol[4]
)

res = adv + dis

board[y][x] = " "
return res
```

Đây là một hàm đánh giá cho trò chơi cờ Caro dựa trên bảng cờ được đưa vào (board). Hàm nhận các đối số như là màu cờ hiện tại (col), màu cờ đối phương (anticol), tọa độ y và x của nước đi cần đánh giá.

Hàm này có hai phần chính: một phần tấn công (adv) và một phần phòng thủ (dis). Mỗi phần tính điểm dựa trên một hàm score_of_col_one, sau đó kiểm tra xem một trong hai màu cờ có đạt được tình huống chiến thắng không thông qua hàm winning_situation. Điểm số được tính dựa trên giá trị trả về của các hàm này.

Cuối cùng, hàm trả về tổng điểm (res) bằng cách cộng điểm tấn công và điểm phòng thủ lại với nhau. Trong quá trình tính điểm, giá trị M được sử dụng để làm cho điểm tấn công có trọng số cao hơn điểm phòng thủ.

2.2 Hàm trả về các tình huống chiến thắng

```
def winning situation(sumcol):

  if 1 in sumcol[5].values():
    return 5
  elif len(sumcol[4]) >= 2 or (len(sumcol[4]) >= 1 and max(sumcol[4].values()) >=
2):
    return 4
  elif TF34score(sumcol[3], sumcol[4]):
    return 4
  else:
    score3 = sorted(sumcol[3].values(), reverse=True)
    if len(score3) >= 2 and score3[0] >= score3[1] >= 2:
      return 3
  return 0
```

Hàm winning_situation nhận một tham số là sumcol, một từ điển chứa thông tin về điểm số của mỗi cột trên bảng cờ. Hàm này cố gắng xác định tình huống chiến thắng của một màu cờ dựa trên các điểm số đã tính toán.

Sử dụng hàm này để kiểm tra các tình huống chiến thắng có thể xuất hiện trên bảng cờ và trả về một giá trị thể hiện mức độ nguy hiểm của tình huống đó.

2.3 Các hàm khác

Ngoài ra còn có một số hàm khác để tính toán điểm số và trạng thái của bản cờ theo cột và theo hàng, đánh giá theo các hướng khác nhau để trả về trạng thái của bàn cờ, nhằm đưa ra các trạng thái của bàn cờ để tính toán điểm số, tấn công hoặc cảnh báo

cho máy rằng người chơi đang có những nước đi có thể chiến thắng để đưa ra nước đi tối ưu nhất.

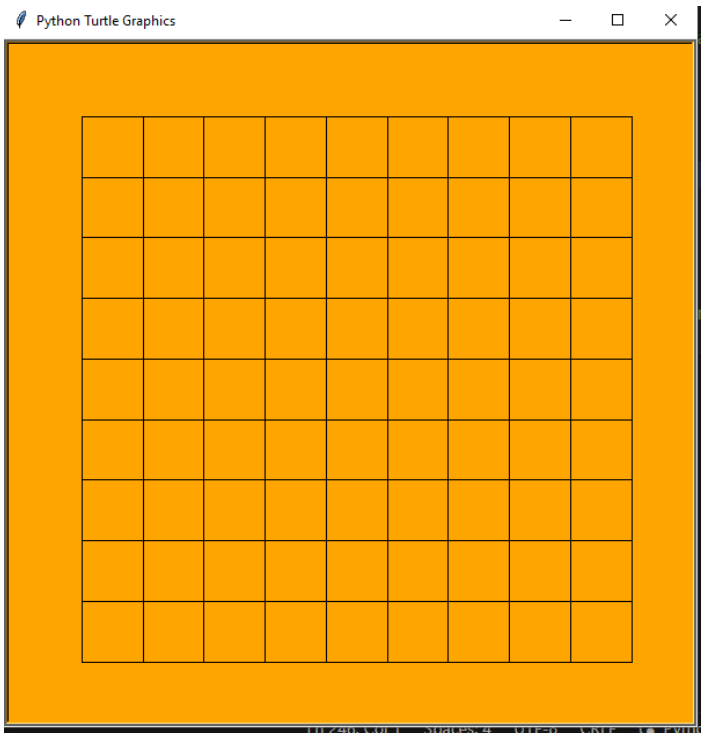
3. Xây dựng các giao diện đồ họa

3.1 Hàm khởi tạo

Hàm khởi tạo sẽ là hàm nhận vào giá trị kích thước của bàn cờ, sau đó khởi tạo các biến giá trị lưu trạng thái của trò chơi. Sau đó khởi tạo màn hình và thiết lập kích thước cho màn hình đồ họa.

```
global win, board, screen, colors, move_history
    # lưu lịch sử nước đi, trạng thái trò chơi, vẽ bảng trống cho trò chơi
    move_history = []
    win = False
    board = make_empty_board(size)
    # tạo 1 màn hình
    screen = turtle.Screen()
    # set onclick cho màn hình
    screen.onclick(click)
    # thiết lập kích thước màn hình đồ họa
    screen.setup(screen.screensize()[1] * 2, screen.screensize()[1] * 2)
    # thiết lập hệ tọa độ của bảng
    screen.setworldcoordinates(-1, size, size, -1)
    # màu nền của game
    screen.bgcolor("orange")
```

Sử dụng thư viện turtle để vẽ các lưới của bảng trò chơi và các quân cờ và thiết lập trạng thái để duy trì màn hình luôn được hiển thị và thay đổi khi người chơi thao tác trên màn hình.



3.2 Hàm click - hiển thị quân cờ trên màn hình tại tọa độ cho trước

Khi click vào màn hình, biến x, y sẽ lưu vị trí của trỏ chuột và chuyển tọa độ đó thành tọa độ trong màn hình trò chơi. Sau đó hàm sẽ tiến hành kiểm tra tọa độ x, y có nằm trong màn hình trò chơi hay không. Nếu kiểm tra xong và điểm đó có nằm trong màn hình sẽ tiến hành vẽ điểm và lưu vào lịch sử di chuyển. Sau khi vẽ xong sẽ tiến hành kiểm tra trạng thái trò chơi và tiến hành.

```
def click(x, y):
    global board, colors, win, move_history


    # chuyển tọa độ click thành tọa độ màn hình trò chơi
    x, y = getindexposition(x, y)


    # hoàn nguyên nước đi trước
    if x == -1 and y == -1 and len(move_history) != 0:
        x, y = move_history[-1]
        del move_history[-1]
        board[y][x] = " "
        x, y = move_history[-1]
        del move_history[-1]
        board[y][x] = " "
        return
```

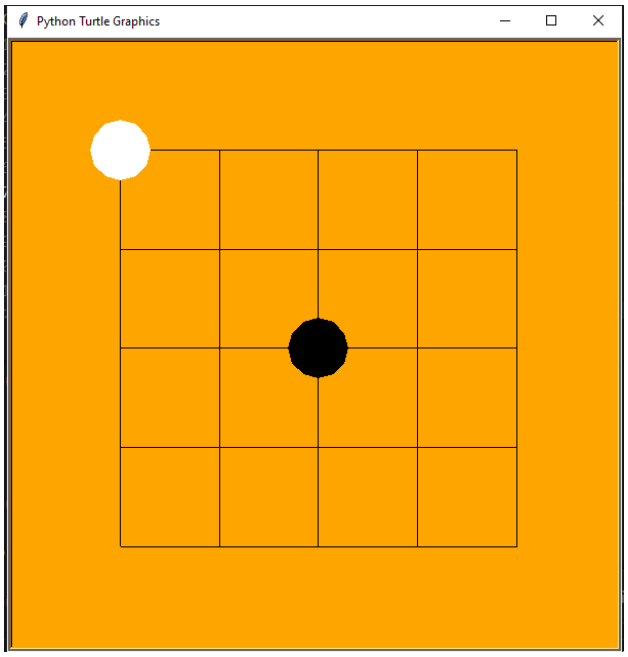
```
# kiểm tra tọa độ x, y không nằm trong bàn cờ
if not is_in(board, y, x):
    return
# có thể vẽ quân cờ, set màu (b) lưu lịch sử
if board[y][x] == " ":
    draw_stone(x, y, colors["b"])
    board[y][x] = "b"
    move_history.append((x, y))

    # kiểm tra trò chơi kết thúc
    game_res = is_win(board)
    if game_res in ["White won", "Black won", "Draw"]:
        print(game_res)
        showdialog(game_res)
        win = True
        return
    # lượt đi của máy
    ay, ax = alpha_beta_pruning_search(board, "w")
    draw_stone(ax, ay, colors["w"])
    board[ay][ax] = "w"
    move_history.append((ax, ay))
    game_res = is_win(board)
    if game_res in ["White won", "Black won", "Draw"]:
        print(game_res)
        showdialog(game_res)
        win = True
        return
```

3.3 Hàm vẽ quân cờ

Hàm vẽ quân cờ sẽ xác định vị trí tọa độ x, y của quân cờ, sau đó di chuyển đến vị trí tọa độ và bắt đầu đặt bút vẽ và tiến hành fill màu theo hình tròn với đường kính theo tung độ là 0.3 và sau khi fill xong sẽ kết thúc vẽ.

Kết thúc hàm ta được 1 chấm tròn, tượng trưng cho quân cờ theo màu truyền vào colturtle

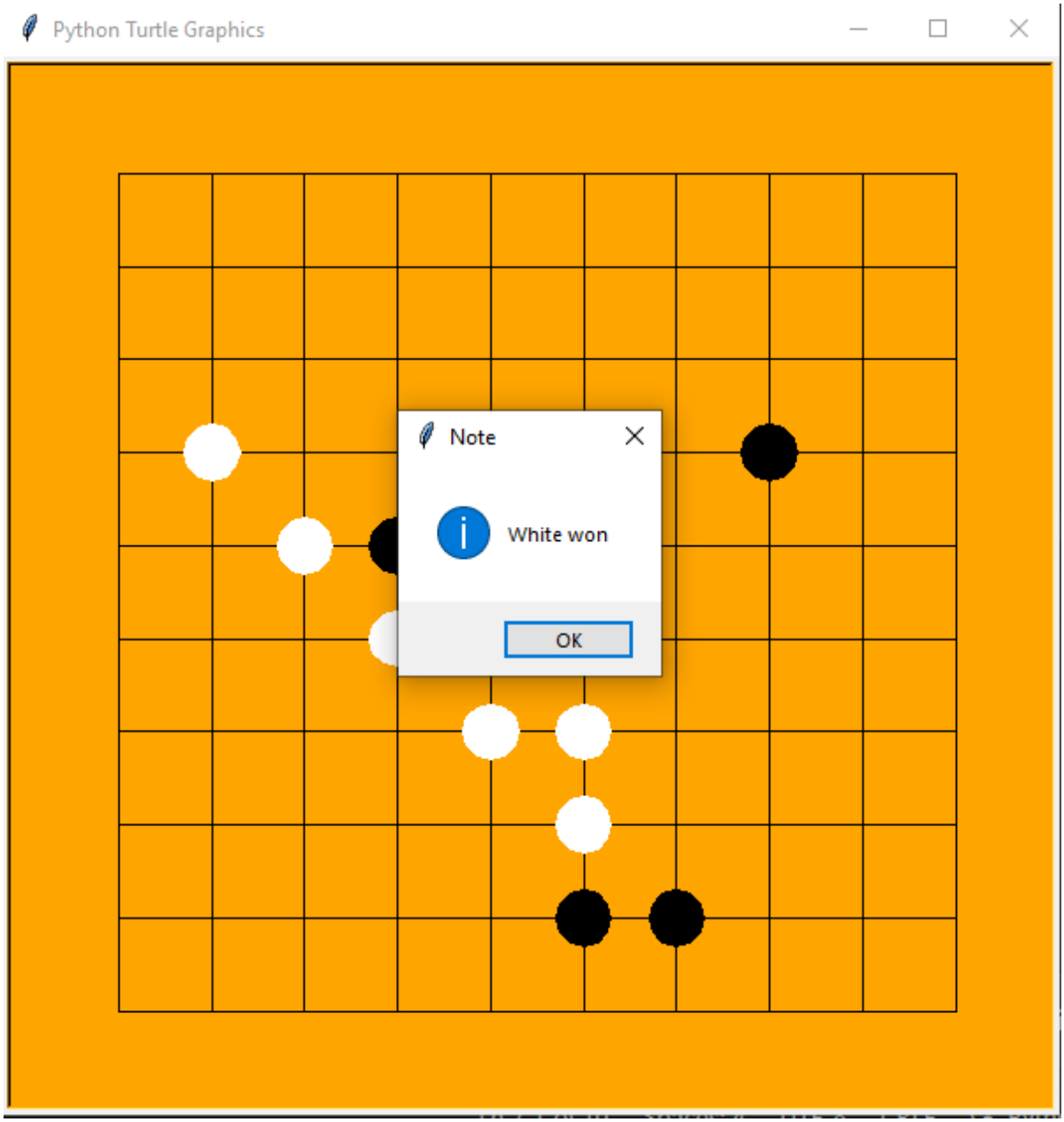
```
def draw_stone(x, y, colturtle):
    colturtle.goto(x, y - 0.3)
    colturtle.pendown()
    colturtle.begin_fill()
    colturtle.circle(0.3)
    colturtle.end_fill()
    colturtle.penup()
```



3.4 Hàm thông báo người chơi thắng

Hàm thông báo sử dụng thư viện messagebox để show thông báo cho người dùng biết trạng thái hiện tại của trò chơi.

```
def showdialog(string):
    root = tk.Tk()
    root.withdraw()
    # Hiển thị một dialog thông báo
    messagebox.showinfo("Note",string)
    # Mainloop để giữ cửa sổ mở và chờ người dùng tương tác
    root.mainloop()
```


Python Turtle Graphics
Note
White won
OK

CHƯƠNG IV. KẾT LUẬN

Qua quá trình nghiên cứu, chúng em cũng đã có những cái nhìn rộng hơn về cách hoạt động, ưu điểm và ứng dụng của thuật toán minimax với cắt tỉa alpha-beta, đặc biệt là khi ứng dụng vào trò chơi Caro. Thuật toán mặc dù đơn giản nhưng lại mang đến chiến lược thông minh trong trò chơi Caro, điều này thậm chí còn được cải thiện hơn nữa thông qua cắt tỉa alpha-beta, giúp tối ưu hóa quá trình tìm kiếm và giảm độ phức tạp tính toán và tạo ra một đối thủ máy tính có khả năng tư duy chiến thuật. Cuối cùng, bài tiểu luận này không chỉ là một tìm hiểu qua lý thuyết và ứng dụng thuật toán, mà còn là một bước nhảy vào thế giới rộng lớn của trí tuệ nhân tạo. Hy vọng rằng nó sẽ góp phần làm phong phú thêm kiến thức vào sự hiểu biết để chúng em có thể tận dụng sức mạnh của máy tính tạo ra trải nghiệm giải trí tuyệt vời hơn.

TÀI LIỆU THAM KHẢO

- Shichiki Lê (18/08/2020) Giải thuật tìm kiếm minimax, Giải Thuật Tìm Kiếm Minimax, (https://www.iostream.co/article/giai-thuat-tim-kiem-minimax-s1EVnH)
- Sebastian Lague (20/04/2018) Algorithms Explained – minimax and alpha-beta pruning, *Youtube*, (https://www.youtube.com/watch?v=l-hh51ncgDI&t=546s)
- Wolfgang Ertel; (2017). Search, Games and Problem Solving, *Introduction to Artificial Intelligence*, 2, Springer International Publishing AG; Switzerland.
- Geeksforgeeks (16/01/2023) Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning), Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning), (https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/)
- Vũ Trọng Quang (18/07/2020) Giải Thuật Cắt Tỉa Alpha-beta, iostream.co, (https://www.iostream.co/article/giai-thuat-cat-tia-alpha-beta-Wu7F1)