BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐAI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

ỨNG DỤNG TRÒ CHƠI GOMOKU

SINH VIÊN THỰC HIỆN

Ong Vĩnh Phát – MSSV: 22110394

Nguyễn Hoàng Phúc – MSSV: 22110400

GVHD: ThS. Trần Tiến Đức

Hồ Chí Minh, Tháng 05 Năm 2024

MỤC LỤC

PHẦN MỞ ĐẦU

1. Giới thiệu đề tài

Trong cuộc sống cách mạng công nghiệp 4.0 hiện nay, các ngành giải trí trong lập trình game là một phần phổ biến. Các trò chơi kinh điển như gomoku vẫn rất có giá trị. Gomoku là một trò chơi đối kháng kinh điển và phổ biến, thu hút sự quan tâm của nhiều người chơi với độ phức tạp trò chơi cũng như sự thách thức trong việc tìm kiếm chiến lược chiến thắng. Người chơi khi chơi với nhau sẽ cần phải suy nghĩ tìm đường đi và chiến thuật đúng đắn để có thể tìm cho mình đường đi chiến thắng nhanh nhất và hiệu quả. Đối với thuật toán trên máy, người ta đã áp dụng thuật toán Alpha-Beta Pruning vào Gomoku sẽ giúp cải thiện hiệu suất của các chương trình máy tính chơi game thông minh, tạo ra một trải nghiệm chơi game đầy thú vị và thách thức cho người chơi.

Qua đó bằng cách kết hợp Gomoku với Alpha-Beta Pruning, ta có thể nghiên cứu và phát triển các thuật toán thông minh và hiệu quả cho máy tính trong việc chọn nước đi trong trò chơi này. Việc này không chỉ giúp cải thiện khả năng chơi của máy tính trong Gomoku, trong thực tế, hiểu được sâu thông qua thuật toán, đồng thời mang lại sự phong phú trong việc kết hợp giữa lý thuyết và thực tiễn, rèn luyện được kỹ năng tư duy.

2. Mục tiêu, nhiệm vụ

Nghiên cứu về trò chơi Gomoku: Tìm hiểu sâu về quy tắc, chiến thuật và các vấn đề đặc biệt của trò chơi Gomoku để có cái nhìn toàn diện về bối cảnh và yếu tố cần thiết cho dự án.

Tìm hiểu về thuật toán Alpha-Beta Pruning: Nắm vững cơ bản, cách hoạt động và các phương pháp cải tiến của thuật toán Alpha-Beta Pruning để áp dụng vào việc tối ưu hóa quá trình tìm kiếm nước đi trong Gomoku.

Thiết kế và phát triển hệ thống: Xây dựng một hệ thống máy tính có khả năng chơi Gomoku thông minh bằng cách tích hợp thuật toán Alpha-Beta Pruning và các công nghệ liên quan. Đảm bảo tính linh hoạt, hiệu quả và dễ mở rộng của hệ thống.

Kiểm thử và đánh giá hiệu suất: Tiến hành các bài kiểm thử để đảm bảo tính chính xác và hiệu quả của hệ thống trong việc chọn nước đi. Đồng thời, đánh giá hiệu suất của hệ thống so với các đối thủ máy tính khác và cả với con người.

PHẦN NỘI DUNG

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Thuật toán Minimax

1.1.1 Giới thiệu và mô tả

Thuật toán Minimax là một kỹ thuật tìm kiếm và ra quyết định thường được sử dụng trong các trò chơi hai người chơi đối kháng như cờ vua, cờ caro, và các trò chơi chiến lược khác. Thuật toán này nhằm tìm ra nước đi tối ưu bằng cách giả định rằng đối thủ cũng sẽ chơi một cách tối ưu.

Là một thuật toán duyệt theo chiều sâu. Cơ bản, Minimax hoạt động dựa trên nguyên tắc đánh giá tất cả các khả năng của nước đi, sau đó chọn nước đi có giá trị tốt nhất cho người chơi hiện tại, đồng thời giả định rằng đối thủ sẽ chọn nước đi có giá trị tốt nhất cho họ trong lượt chơi tiếp theo. Để thực hiện điều này, thuật toán Minimax sử dụng một cây trò chơi, trong đó mỗi nút đại diện cho một trạng thái của trò chơi. Các nút con đại diện cho các trạng thái tiếp theo có thể xảy ra từ trạng thái hiện tại.

Quá trình tính toán của Minimax bắt đầu từ lá cây (các trạng thái cuối cùng) và tiến ngược lên gốc cây (trạng thái ban đầu). Ở mỗi nút, giá trị được tính dựa trên giá trị của các nút con: nếu là lượt của người chơi, nút sẽ chọn giá trị lớn nhất từ các nút con (maximizing), ngược lại nếu là lượt của đối thủ, nút sẽ chọn giá trị nhỏ nhất (minimizing). Kết quả là nút gốc sẽ có giá trị đại diện cho giá trị tối ưu của nước đi đầu tiên cho người chơi.

1.1.2 Ví dụ về thuật toán

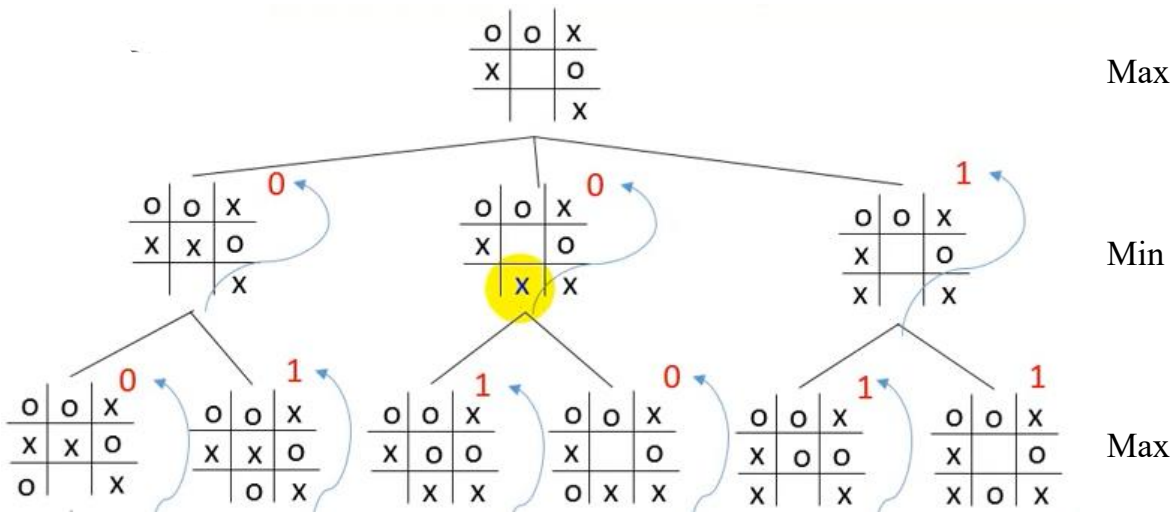
Giả sử rằng bạn đang chơi với máy tính và đến lượt máy tính đi. Thuật toán Minimax sẽ xây dựng một cây trò chơi từ trạng thái hiện tại của bàn cờ. Ở mỗi nút của cây, trạng thái của bàn cờ được thể hiện và các nút con của nó là các trạng thái tiếp theo có thể xảy ra từ nước đi của người chơi hiện tại.

Đầu tiên, thuật toán sẽ đánh giá tất cả các nước đi có thể từ trạng thái hiện tại của bàn cờ. Giả sử có năm nước đi khả dĩ cho máy tính, Minimax sẽ xem xét từng nước đi một. Đối với mỗi nước đi, máy tính sẽ giả định rằng người chơi sẽ phản ứng bằng cách đi nước đi tốt nhất của họ. Từ đó, máy tính lại đánh giá tiếp các phản ứng tiếp theo của mình và người chơi, tạo ra một cây trò chơi.

Trong quá trình này, Minimax sẽ gán giá trị cho từng trạng thái cuối cùng (lá cây) dựa trên kết quả thắng, thua hoặc hòa. Các giá trị này sau đó sẽ được truyền ngược lại từ lá cây lên gốc cây. Nếu máy tính đang ở lượt đi, nó sẽ chọn giá trị lớn nhất từ các nút con (maximizing), còn nếu người chơi ở lượt đi, nó sẽ chọn giá trị nhỏ nhất (minimizing).

Ví dụ, nếu một nước đi của máy tính dẫn đến một loạt các nước đi mà kết quả cuối cùng là máy tính thắng, thua hoặc hòa, thì nước đi dẫn đến kết quả thắng sẽ được ưu tiên. Ngược lại, nếu một nước đi của máy tính dẫn đến việc người chơi có thể thắng ngay trong lượt tiếp theo, thì nước đi đó sẽ bị loại trừ.

Như trong ví dụ dưới hình sẽ xen kẽ nhau min, max sinh ra các trạng thái, xét từ dưới lên trên. Ở tầng lấy Min, node 1 ta có min(0, 1) = 0, node 2 ta có min(1, 0) = 0, node 3 ta có min(1, 1) = 0. Ở tầng nút gốc ta lấy Max sẽ được max(0, 0, 1) = 1



Hình 1: Ví dụ về thuật toán

1.1.3 Chương trình cơ bản

```
def minimax(node, depth, maximizingPlayer):
    if node == "End Node" or depth == 0:
        return value(node)

    if maximizingPlayer:
        Mx = float('-inf')
        for child in node:
            Mx = max(Mx, minimax(child, depth - 1, False))
        return Mx
    else:
        Mn = float('inf')
        for child in node:
```

```
            Mn = min(Mn, minimax(child, depth - 1, True))
        return Mn
```

Trong đó: Nhận đầu vào là node - nút hiện tại trong cây trò chơi, depth - độ sâu còn lại để khám phá, và maximizingPlayer - biến đánh dấu xem người chơi đang ở vai trò MAX hoặc MIN.

Nếu đang ở bước maximizing (tối đa hóa), thuật toán sẽ chọn nước đi tốt nhất cho người chơi tối đa bằng cách duyệt qua tất cả các nước đi con và chọn giá trị lớn nhất (Tìm đường đi thắng cao nhất). Ngược lại, nếu đang ở bước minimizing, thuật toán sẽ chọn nước đi tốt nhất cho người chơi tối thiểu bằng cách chọn giá trị nhỏ nhất (Tìm đường đi để đối thủ có khả năng thắng thấp nhất)

1.2 Thuật toán Alpha-Beta Pruning

1.2.1 Mô tả và cải tiến từ Minimax

Thuật toán Alpha-Beta Pruning là một cải tiến của thuật toán Minimax, được sử dụng để tìm kiếm nước đi tối ưu trong các trò chơi đối kháng như cờ vua, cờ caro và Gomoku. Alpha-Beta Pruning giúp giảm số lượng nút cần duyệt trong cây trò chơi mà không ảnh hưởng đến kết quả cuối cùng, từ đó cải thiện hiệu suất của thuật toán.

Thuật toán này hoạt động bằng cách duyệt cây trò chơi theo cùng cấu trúc như Minimax, nhưng với việc cắt tỉa các nhánh không cần thiết. Khi duyệt qua cây, Alpha-Beta Pruning sử dụng hai giá trị, là alpha và beta, để xác định các giới hạn tối ưu cho giá trị của nước đi.

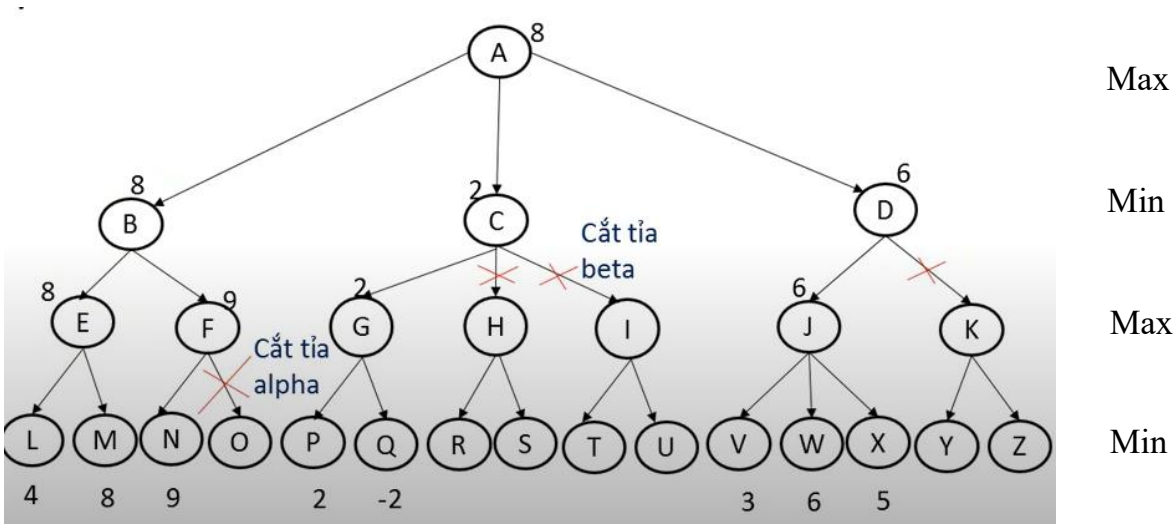
Alpha đại diện cho giá trị tốt nhất mà người chơi hiện tại đã tìm thấy cho đến nay trong các nút con đã duyệt qua. Beta đại diện cho giá trị tốt nhất mà đối thủ có thể đạt được thông qua các nước đi của họ. Khi một nút con được duyệt qua, giá trị của nó sẽ được kiểm tra so với alpha và beta.

Nếu giá trị của nút con lớn hơn hoặc bằng beta, điều này có nghĩa là nước đi tương ứng sẽ không bao giờ được chọn bởi người chơi hiện tại, vì đối thủ có thể chọn một nước đi khác để giảm giá trị của nó. Do đó, nhánh này có thể được cắt tỉa (pruned), không cần phải tiếp tục duyệt. Giống như ví dụ bởi hình ảnh, cắt tỉa nhánh H và I vì giá trị ở C lấy Min mà Min C luôn nhỏ hơn min B nên nhánh có thể được cắt tỉa

Ngược lại, nếu giá trị của nút con nhỏ hơn hoặc bằng alpha, điều này có nghĩa là nước đi tương ứng sẽ không được chọn bởi đối thủ, vì người chơi hiện tại đã tìm thấy

một nước đi tốt hơn. Nhánh này cũng có thể được cắt tỉa. Giống như ví dụ bởi hình ảnh, cắt tỉa nhánh O vì giá trị ở F lấy Max mà cùng nhắn đó có giá trị E tốt hơn để lấy Min tiếp theo nên ta có thể cắt tỉa nhánh O

Qua quá trình này, Alpha-Beta Pruning loại bỏ các nhánh không cần thiết trong cây trò chơi, giảm đáng kể số lượng nút cần duyệt, tăng tốc độ tính toán và hiệu suất của thuật toán. Điều này làm cho thuật toán trở nên hiệu quả hơn.



Hình 2: Ví dụ về thuật toán cải tiến

1.2.2 Chương trình cơ bản

```
def alphabeta(node, depth, a, b, maximizingPlayer):
    if node == "End Node" or depth == 0:
        return value(node)

    if maximizingPlayer:
        for child in node:
            a = max(a, alphabeta(child, depth - 1, a, b, False))
            if a >= b:
                break
        return a
    else:
        for child in node:
            b = min(b, alphabeta(child, depth - 1, a, b, True))
            if a >= b:
                break
        return b
```

Trong đó: nhận đầu vào là node - nút hiện tại trong cây trò chơi, depth - độ sâu còn lại để khám phá, a và b là hai giá trị alpha và beta, và maximizingPlayer - biến đánh dấu xem người chơi đang ở vai trò MAX hoặc MIN.

Nếu đang ở bước maximizing, thuật toán sẽ cố gắng tối đa hóa giá trị của a bằng cách duyệt qua tất cả các nút con của nút hiện tại. Nếu giá trị của a lớn hơn hoặc bằng giá trị của b, thì thuật toán sẽ dừng và cắt tỉa cây. Tương tự, nếu đang ở bước minimizing, thuật toán sẽ cố gắng tối thiểu hóa giá trị của b.

1.3 Thuật toán Heuristic Alpha-Beta Tree

Thuật toán Heuristic Alpha-Beta Tree Search là một biến thể nâng cao của thuật toán Alpha-Beta Pruning, được thiết kế để cải thiện hiệu suất tìm kiếm nước đi tối ưu trong các trò chơi đối kháng như cờ vua, cờ caro và đặc biệt là Gomoku. Thuật toán này kết hợp các kỹ thuật cắt tỉa nhánh không cần thiết của Alpha-Beta Pruning với các hàm heuristic để đánh giá giá trị của các trạng thái trung gian, giúp giảm đáng kể số lượng nút cần duyệt trong cây tìm kiếm.

Cốt lõi của thuật toán này là việc sử dụng các hàm heuristic để ước lượng giá trị của các trạng thái chưa được hoàn thành trong trò chơi. Thay vì phải duyệt hết tất cả các trạng thái đến cuối cùng, hàm heuristic sẽ đưa ra một đánh giá tạm thời dựa trên các yếu tố như số lượng quân cờ của mỗi bên, các chuỗi quân cờ liên tiếp và các vị trí chiến lược trên bàn cờ. Những đánh giá này giúp thuật toán có thể cắt tỉa các nhánh không có khả năng dẫn đến kết quả tốt hơn, từ đó tập trung vào các nhánh tiềm năng hơn.

Khi duyệt qua cây tìm kiếm, thuật toán Alpha-Beta sử dụng hai giá trị alpha và beta để giới hạn khoảng giá trị mà nước đi hiện tại có thể đạt được. Giá trị alpha là giá trị tối đa mà người chơi có thể đảm bảo, còn giá trị beta là giá trị tối thiểu mà đối thủ có thể đảm bảo. Nếu một nhánh có giá trị thấp hơn alpha hoặc cao hơn beta, nhánh đó sẽ được cắt tỉa vì nó không thể cải thiện kết quả của người chơi hoặc đối thủ.

Bằng cách kết hợp các đánh giá heuristic vào quá trình tìm kiếm Alpha-Beta, thuật toán Heuristic Alpha-Beta Tree Search không chỉ tăng tốc quá trình tìm kiếm mà còn nâng cao chất lượng của các nước đi được chọn. Điều này đặc biệt quan trọng trong các trò chơi như Gomoku, nơi không gian trạng thái rất lớn và việc duyệt toàn bộ cây tìm kiếm là không khả thi trong thời gian thực.

CHƯƠNG 2: GIỚI THIỆU BÀI TOÁN GOMOKU

2.1 Bài toán Gomoku

2.1.1 Mô tả bài toán

Gomoku, một trò chơi cổ điển với tên gọi phổ biến như "Caro" hoặc "Five in a Row", đã thu hút người chơi trí tuệ trên khắp thế giới, nó là một trò chơi trên bảng ô vuông, nơi mà hai người chơi luân phiên đặt biểu tượng của họ (thường là "X" và "O") trên bảng, với mục tiêu tạo ra một dãy liên tiếp gồm năm biểu tượng giống nhau theo hàng ngang, hàng dọc hoặc đường chéo.

2.1.2 Luật chơi và cách thức chiến thắng

Luật chơi của Gomoku rất đơn giản, khi một người chơi đến lượt chơi của họ, họ sẽ ghi một kí tự X hoặc O lên bàn cờ, bàn cơ này thường có kích thước khoảng 10x10, hoặc 20x20. Người chơi chỉ có thể đi nước tại những vị trí còn trống, không được xóa nước đi của mình và đối thủ và không được đi lại. Người chiến thắng là người tạo được một dãy liên tiếp 5 kí tự.

Tại một số nơi khác cũng có thể quy định các cách chiến thắng khác nhau, ví dụ như 5 nước liên tiếp phải không bị chặn 2 đầu nhằm giảm lợi thế của người đi trước. Nhưng trong chương trình này, chúng em chỉ đơn giản quyết định người chiến thắng là người tạo được một dãy 5 kí tự giống nhau liên tiếp trước là người chiến thắng.

Ván đấu sẽ hòa khi không ai có thể chiến thắng trong khi đã hết nước đi.

2.2 Thuật toán trong bài toán

2.2.1 Chiến lược cơ bản

Khi đến một người chơi đến lượt, họ sẽ có các lựa chọn là các ô trống để đi. Để đạt được 5 kí tự giống nhau liên tiếp, họ cũng phải tính toán làm sao để đạt được những mục tiêu nhỏ hơn như 4 nước liên tiếp, 3 nước liên tiếp, hai chuỗi 3 nước liên tiếp,… Đồng thời cũng phải xem xét việc chặn đường thắng của đối thủ nếu như đối thủ có khả năng tạo được 5 nước liên tiếp trước mình. Việc khởi đầu khá đơn giản nhưng khi số lượng nước đi tăng lên, việc tính toán nước đi nào là tốt sẽ trở nên phức tạp dần với người chơi. Các chuỗi ngang dọc chéo chồng chéo nhau tạo nên vô số các tổ hợp các nước đi.

Có rất đa dạng các chiến thuật được đưa ra từ những người chơi có kinh nghiệm, nhưng về cơ bản sẽ có những bước không thể thiếu như sau:

Xem xét các nước đi của đối thủ

Xem xét các nước đi của mình

Suy luận để tìm ra duy nhất 1 nước đi hợp lý và tối ưu

2.2.2 Nước đi đe doạ

Nhìn chung thì việc xem xét các nước đi của đối thủ hay của mình đều là như nhau, chỉ khác về nước đi nên ở đây chúng em xin lấy người chơi X là người tấn công, và người chơi O là người phòng thủ.

Khi nhìn vào một bàn cờ, sẽ có những chuỗi 2, chuỗi 3, chuỗi 4,… Các chuỗi này sẽ tạo ra 2 đầu của nó những vị trí có mức điểm đe dọa khác nhau. Với những vị trí nước đi đe dọa của X thì X sẽ cố gắng đi vào để tối đa hóa mức đe dọa của mình để tìm đến chiến thắng, ngược lại, người chơi O nếu không có nước đi có mức đe dọa cao hơn sẽ phải tìm cách ngăn chặn X đi vào những nước đi đe dọa này.

Dưới đây là một số chuỗi đe dọa thường thấy trong một ván cờ

O X X X X N

Hình 3: 4 nước liên tiếp, bị chặn 1 đầu và trống 1 đầu

N X X X X N

Hình 4: 4 nước liên tiếp và trống 2 đầu

N X X X N

Hình 5: 3 nước liên tiếp và trống 2 đầu

O N X X X N N

Hình 6: 3 nước liên tiếp, bị chặn cách một ô trống và đầu còn lại là 2 ô trống

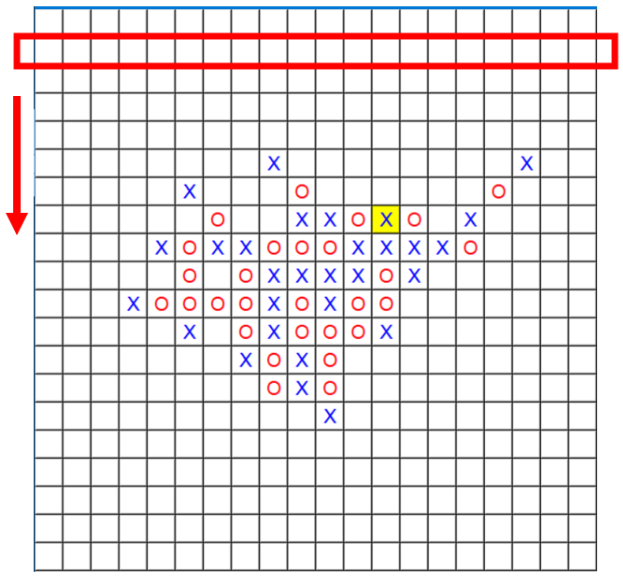
N X N X X N

Hình 7: 3 nước đứt đoạn, bị trống 2 đầu

2.2.3 Hàm heuristic trong giải thuật

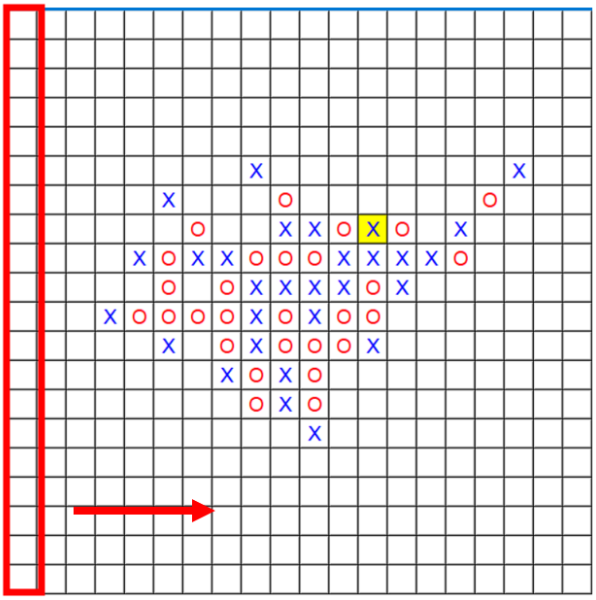
Hàm heuristic được dùng để tìm ra những vị trí được xem là có tính đe dọa. Thứ tự duyệt bảng để tìm các vị trí đe dọa của một mẫu đe dọa như sau:

Duyệt theo dòng:



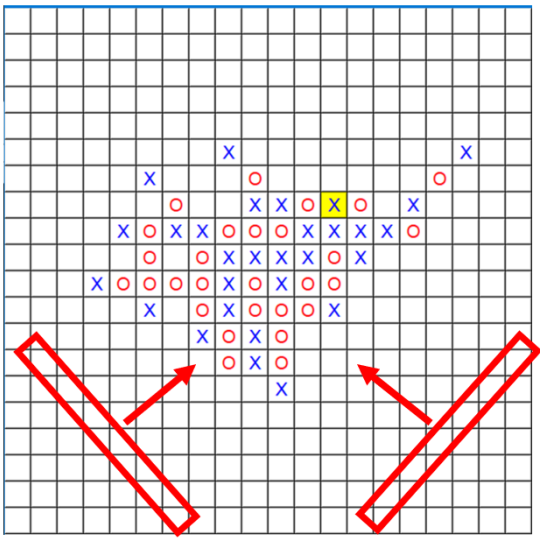
Hình 8: Cách duyệt theo dòng

Duyệt theo cột:



Hình 9: Cách duyệt theo cột

Duyệt theo đường chéo chính và đường chéo phụ



Hình 10: Cách duyệt theo đường chéo chính / phụ

Tại mỗi dòng, cột hay đường chéo được duyệt, ta sẽ duyệt xem có mẫu nguy hiểm trong dòng, cột hay đường chéo đó hay không và lưu lại những vị trí đe dọa cùng với điểm đe dọa của nó.

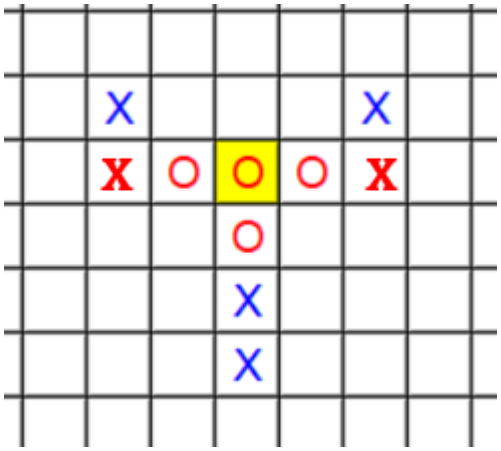
Trường hợp nếu như có nhiều vị trí đe dọa xuất hiện nhiều lần tại cùng một vị trí thì vị trí đó sẽ được cộng thêm một tí điểm đe dọa nữa để tăng thêm độ ưu tiên có vị trí đe dọa cộng dồn này.

Hàm heuristic sau khi chạy xong ta sẽ được một danh sách các vị trí đe dọa cùng với điểm đe dọa của nó

2.2.4 Ứng dụng Alpha-Beta Pruning

Alpha-Beta Pruning là một thuật toán phù hợp để thực hiện được các ý tưởng mà ta đã đề cập. Cụ thể, các con của một nút cha của trong cây tìm kiếm là một loạt các phương án mà người chơi đó có thể đi, tốc độ tìm ra nước đi tốt nhất phụ thuộc rất nhiều vào số nút mà thuật toán phải duyệt tại một mức và do đó hàm heuristic là một nhân tố cực kỳ quan trọng trong việc cải thiện tốc độ của thuật toán bằng cách tìm các nước đi có mức đe dọa cao hơn trước để duyệt trước, và hoặc chỉ tìm các nước đi có mức đe dọa cao nhất để giảm số lượng nút cần phải duyệt.

Ví dụ trong một thế cờ sau, đây là thế chờ sau khi O và X đã di chuyển được 4 nước và đang đến lượt X di chuyển. Khi đến lượt X di chuyển, thuật toán Alpha-Beta Pruning sẽ được chạy để tìm nước đi tốt nhất



Hình 11: Ví dụ một thế cờ

Bằng hàm heuristic, ta biết được có 2 vị trí đe dọa được đánh dấu X đỏ trên bàn cờ, thuật toán cần phải ưu tiên duyệt 2 nước này trước, và sau đó sẽ gọi đệ quy để duyệt tiếp và cứ như vậy, thuật toán sẽ cho ta biết nên đi nước nào trong 2 nước có mức đe dọa cao này. Ta dễ thấy rằng, thay vì mất thời gian duyệt các nước ít nguy hiểm hơn, gây lãng phí tài nguyên và nhất là thời gian, ta đã dùng hàm heuristic để duyệt các nước nguy hiểm hơn trước để khắc phục những nhược điểm lớn này.

Sau khi X di chuyển xong, đến lượt của O thì việc tìm nước đi cũng xảy ra tương tự, O cũng sẽ tìm các vị trí đe dọa nhiều hơn và duyệt trước để tìm ra nước đi tốt cho mình.

Cũng cần lưu ý rằng, không nhất thiết khi đến lượt X đi thì chỉ duyệt các nước đi gây đe dọa của O, X cũng có thể duyệt các nước đi đe dọa của mình và cũng sẽ cần phải

quyết định xem nên đi nước nào, vì bởi một lẽ đơn giản là ngoài phòng thủ là đi chặn các nước đi của đối thủ thì rõ ràng người chơi cũng có thể tấn công nếu như họ có nước đi nhiều đe dọa hơn.

2.2.5 Một số hàm quan trọng trong việc tìm nước đi

Hàm minimax_alpha_beta

```
def minimax_alpha_beta(self, state, depth, alpha, beta):
        winner = state.win()
        if winner != 'N':
            final_score = self.score_alpha_beta(winner, depth)
            return final_score
        elif depth > MAX_DEPTH:
            return 0

        best_moves, threatening_point = state.get_best_moves()

        if state.active_turn == self.name:
            if depth == 0 and threatening_point >= 4:
                self.choice = best_moves[0]
                print("GET 1 QUICK SOLUTION!!!")
                return

            max_score = -MINIMAX_INFINITY
            for move in best_moves:
                possible_game = state.get_new_state(move)
                score = self.minimax_alpha_beta(possible_game, depth + 1,
alpha, beta)
                # get the best move
                if depth == 0 and score > max_score:
                    self.choice = move
                    print("GET 1 SOLUTION!!!")
                # -----
                max_score = max(max_score, score)
                alpha = max(alpha, max_score)
                if alpha >= beta:
                    break
            return max_score
        else:
            min_score = MINIMAX_INFINITY
            for move in best_moves:
                possible_game = state.get_new_state(move)
                score = self.minimax_alpha_beta(possible_game, depth + 1,
alpha, beta)
                min_score = min(min_score, score)
                beta = min(beta, min_score)
                if alpha >= beta:
                    break
```

```
            return min_score
```

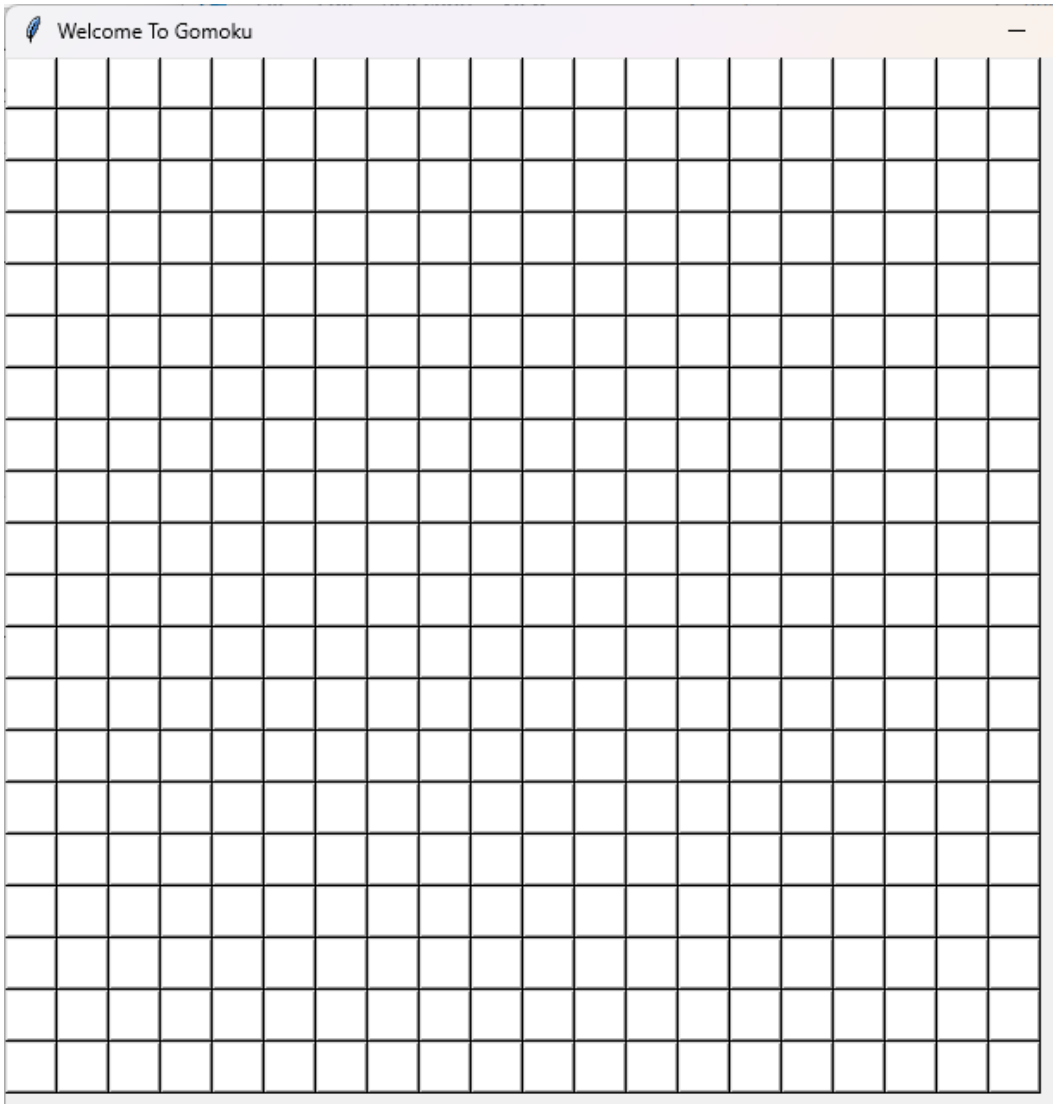
Hàm take_turn_alpha_beta

```
def take_turn_alpha_beta(self):
        """
        Changes the self.choice and then return the best move it can move
from its current board game\n
        This function does not change the given board game
        """
        self.minimax_alpha_beta(self.gmk_game, 0, -MINIMAX_INFINITY,
MINIMAX_INFINITY)
        print(f"solution:({self.choice.x}, {self.choice.y})")
        return self.choice
```

CHƯƠNG 3: ỨNG DỤNG TRÒ CHƠI GOMOKU

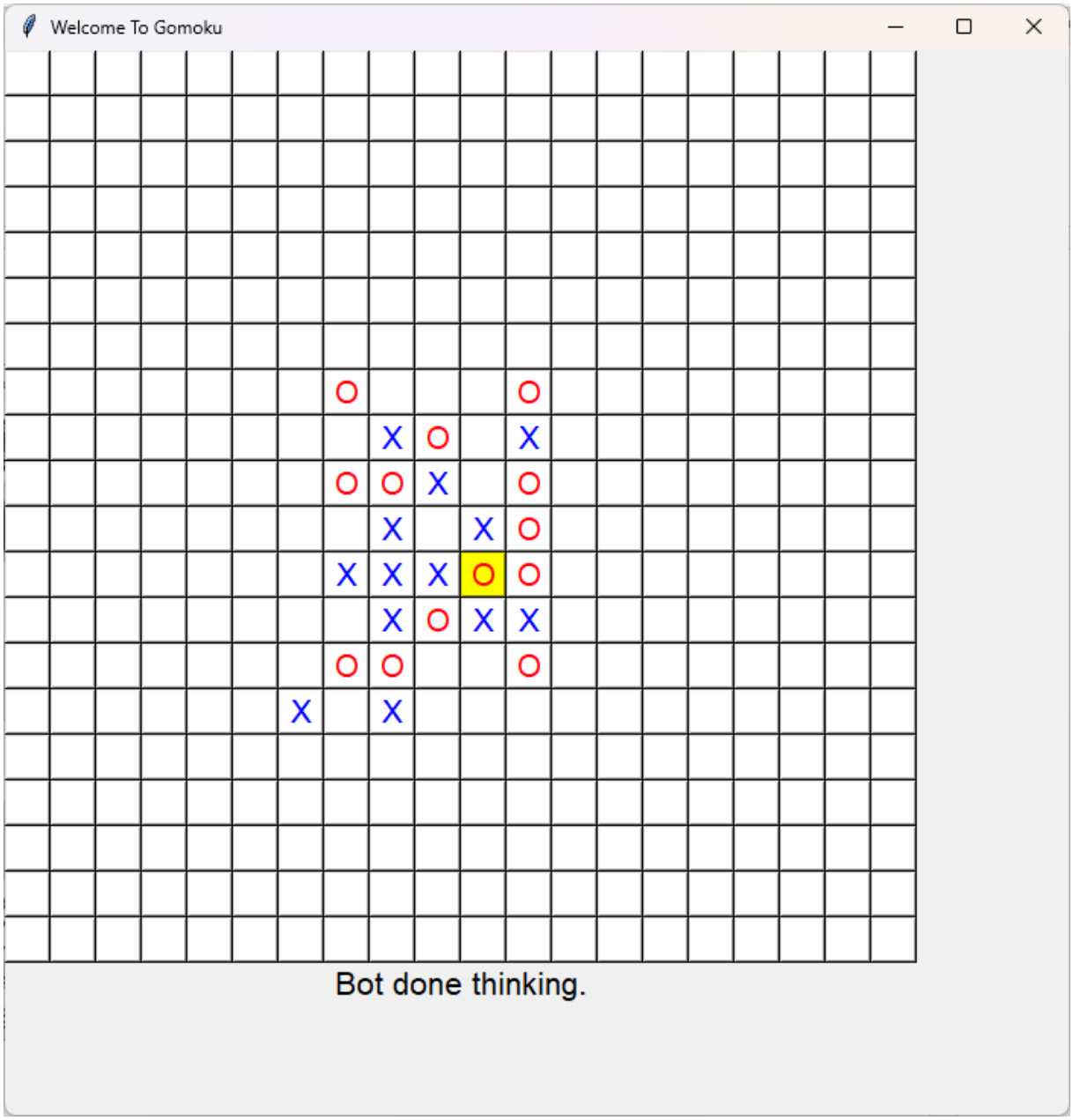
3.1 Giao diện dùng Tkinter

3.1.1 Giao diện khi khởi động



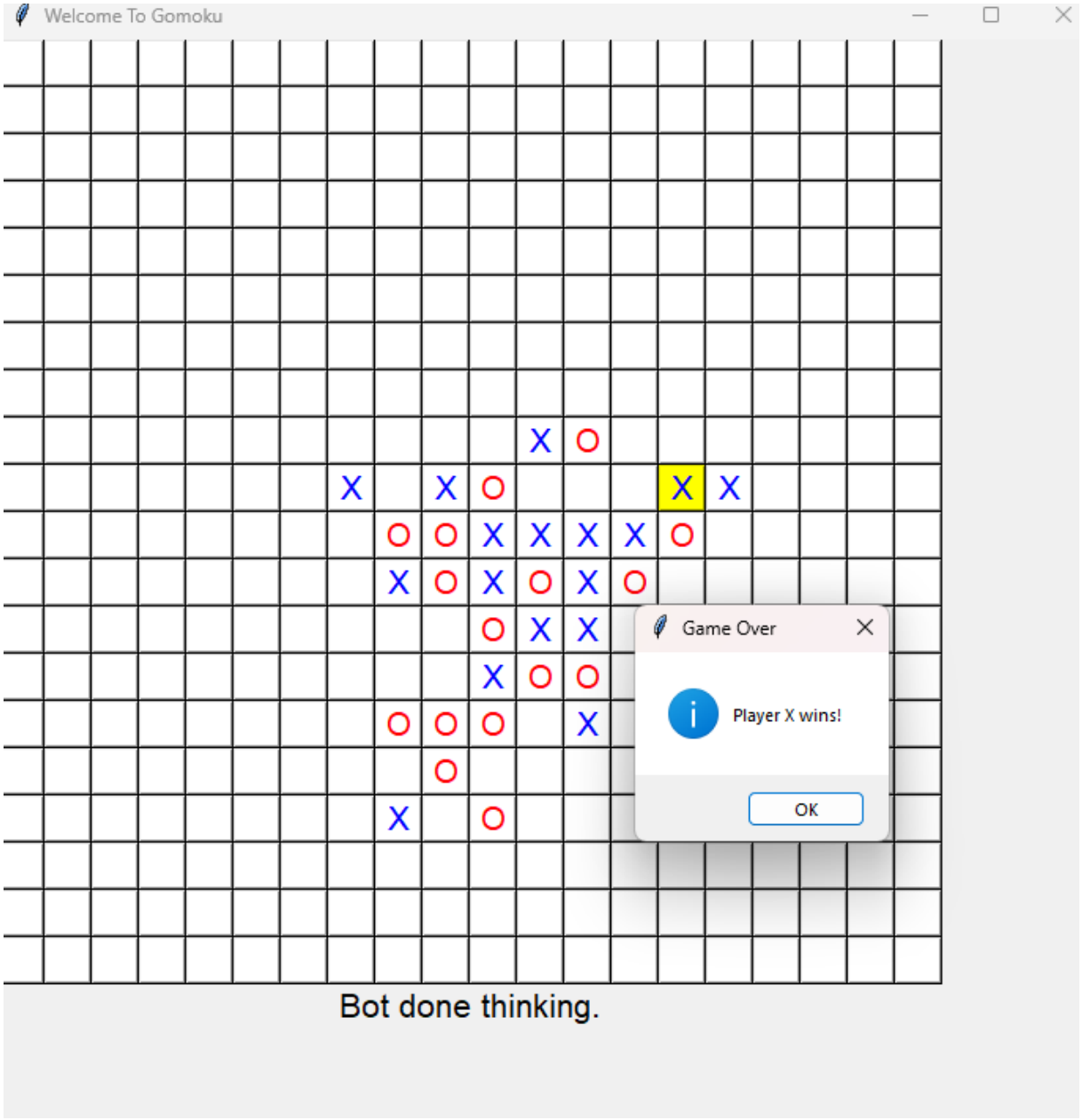
Hình 12: Giao diện khi khởi động

3.1.2 Giao diện trong quá trình chơi



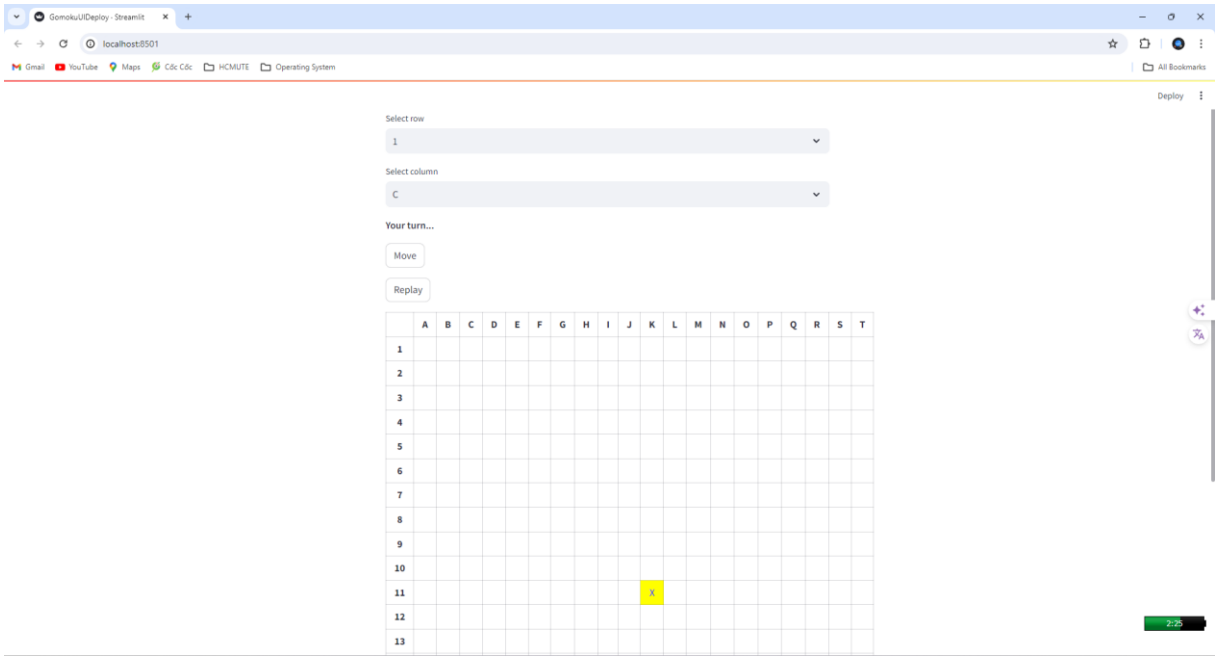
Hình 13: Giao diện trong quá trình chơi

3.1.3 Sau khi kết thúc trò chơi



Hình 14: Thông báo trò chơi kết thúc

3.2 Giao diện trên web bằng streamlit



Hình 15: Giao diện trên web

Khi chơi với giao diện web bằng streamlit, việc chọn đánh trên bản đồ bằng toạ độ theo dòng và cột

Link web được đẩy lên github: https://gomoku-hcmute-nhp-ovp.streamlit.app/

TÀI LIỆU THAM KHẢO

[1] P. Norvig, “Data files to accompany the algorithms from Norvig And Russell's "Artificial Intelligence - A Modern Approach",” 2018. [Trực tuyến]. Available: https://github.com/aimacode/aima-data. [Đã truy cập 18/5/2024].

[2] S. Russell và P. Norvig, Artificial Intelligence - A Modern Approach, 4th biên tập viên, Harlow: Pearson PLC, 2020, pp. 192-207.

[3] H. J. V. D. Herik, “Go-Moku and Threat-Space Search,” 1994. [Trực tuyến]. Available: https://www.researchgate.net/publication/2252447_Go-Moku_and_Threat-Space_Search. [Đã truy cập 19/5/2024].