**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KĨ THUẬT THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN TRÍ TUỆ NHÂN TẠO**

**🙞🕮🙜**

A logo with hands holding a book and a flame

Description automatically generated

**BÁO CÁO CUỐI KỲ**

**LẬP TRÌNH GAME CARO**

**HỌC KÌ 1 – NĂM HỌC 2024 – 2025**

**Thực hiện: Nguyễn Anh Hào: 21110823**

**Giảng viên hướng dẫn: Trần Tiến Đức**

BỘ GIÁO DỤC VÀ ĐÀO TẠO **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KĨ THUẬT Độc lập - Tự do – Hạnh phúc**

**THÀNH PHỐ HỒ CHÍ MINH \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN TRÍ TUỆ NHÂN TẠO** *TP.HCM, ngày …. tháng … năm 2024*

**\_\_\_\_\_\_\_\_\_**

**DANH SÁCH NHÓM BÁO CÁO CUỐI KỲ**

**MÔN TRÍ TUỆ NHÂN TẠO**

**HỌC KỲ 1 NĂM HỌC 2024-2025**

1. **Mã lớp môn học:** Nhom 06CLC
2. **Giảng viên hướng dẫn:** Trần Tiến Đức
3. **Tên đề tài:** Lập trình game Caro
4. **Danh sách nhóm viết tiểu luận cuối kỳ:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ và tên**  **sinh viên** | **Mã số sinh viên** | **Số điện thoại** | **Kí tên** |
| 01 | Nguyễn Anh Hào | 21110823 | 0356534762 |  |

**Nhận xét của giáo viên**

Điểm:

Ngày …. tháng …. năm 2024

**LỜI CẢM ƠN**

Chúng em xin bày tỏ lòng biết ơn sâu sắc đến thầy Trần Tiến Đức về sự đồng hành và hỗ trợ tận tâm trong suốt quá trình thực hiện đồ án lập trình game caro.

Thầy không chỉ là người hướng dẫn chúng em về kiến thức chuyên ngành mà còn là người thầy mang đến cho chúng em những trải nghiệm quý báu về sự sáng tạo, khả năng giải quyết vấn đề, và tư duy lập trình. Bằng sự am hiểu sâu sắc về trí tuệ nhân tạo, thầy đã giúp chúng em khám phá và ứng dụng những khái niệm này một cách sáng tạo trong đồ án của mình.

Chúng em rất trân trọng vì thầy luôn dành thời gian để lắng nghe ý kiến, giải đáp thắc mắc, và hướng dẫn chúng em vượt qua những khó khăn trong quá trình nghiên cứu và phát triển game caro. Nhờ những góp ý xây dựng từ thầy, giúp chúng em hiểu rõ hơn về cách tiếp cận vấn đề, tối ưu hóa mã nguồn, và tận dụng tối đa khả năng của trí tuệ nhân tạo trong lĩnh vực lập trình game.

Đồ án không chỉ là một bài kiểm tra cuối kỳ mà còn là hành trình học tập đáng nhớ, nhờ có sự hướng dẫn và chia sẻ của thầy. Chúng em tin tưởng rằng những kiến thức và kỹ năng chúng em đã học được không chỉ mang lại thành công trong đồ án mà còn là nền tảng vững chắc cho sự phát triển trong tương lai.

Một lần nữa, chúng em xin bày tỏ lòng biết ơn sâu sắc đến thầy và hy vọng có thêm cơ hội được học hỏi và làm việc cùng thầy trong những dự án sắp tới.

MỤC LỤC

[PHẦN MỞ ĐẦU 1](#_Toc183898252)

[1. Lý do chọn đề tài 1](#_Toc183898253)

[2. Các chức năng của đề tài 1](#_Toc183898254)

[PHẦN NỘI DUNG 2](#_Toc183898255)

[CHƯƠNG 1: KIẾN THỨC LIÊN QUAN 2](#_Toc183898256)

[1. Thuật toán Minimax 2](#_Toc183898257)

[2. Alpha-Beta Pruning 3](#_Toc183898258)

[3. Thuật toán Tham Lam (Greedy) 4](#_Toc183898259)

[4. Monte Carlo Tree Search (MCTS) 5](#_Toc183898260)

[CHƯƠNG 2: THIẾT KẾ THUẬT TOÁN 8](#_Toc183898261)

[1. Hàm btsConver 8](#_Toc183898262)

[2. Hàm getCoordsAround 9](#_Toc183898263)

[3. Patterns 10](#_Toc183898264)

[4. Hàm points 10](#_Toc183898265)

[5. Hàm minimax 11](#_Toc183898266)

[6. Hàm computer 12](#_Toc183898267)

[7. Hàm greedyAlgorithm 13](#_Toc183898268)

[8. Hàm mcts\_computer 14](#_Toc183898269)

[CHƯƠNG 3: THIẾT KẾ GIAO DIỆN HỆ THỐNG 15](#_Toc183898270)

[1. Giao diện mặc định 15](#_Toc183898271)

[2. Thuật toán Minimax 15](#_Toc183898272)

[3. Thuật toán Minimax with Alpha-Beta cutoff 16](#_Toc183898273)

[4. Thuật toán Greedy 16](#_Toc183898274)

[5. Thuật toán Monte Carlo Tree Search 17](#_Toc183898275)

[PHẦN KẾT LUẬN 18](#_Toc183898276)

[1. Kết quả đạt được 18](#_Toc183898277)

[2. Ưu điểm 19](#_Toc183898278)

[3. Nhược điểm 20](#_Toc183898279)

[4. Hướng phát triển 20](#_Toc183898280)

[TÀI LIỆU THAM KHẢO 21](#_Toc183898281)

# PHẦN MỞ ĐẦU

## Lý do chọn đề tài

Trong những năm 1990, sau khi IBM phát triển Deep Blue để đánh bại người chơi cờ vua con người, mọi người đã cố gắng giải quyết mọi vấn đề về trò chơi trên bàn bằng cách sử dụng máy tính. Gomoku (Caro) là một trong những trò chơi trên bàn phổ biến tại châu Á và châu Âu, người ta cũng cố gắng mô phỏng và giải quyết nó thông qua thuật toán máy tính.

Chiến lược truyền thống và hiệu quả cho trí tuệ nhân tạo Gomoku là thuật toán tìm kiếm cây. Thuật toán minimax là một trong những cây trò chơi phổ biến nhất được sử dụng trong chiến lược trí tuệ nhân tạo. Nhưng rõ ràng, vấn đề lớn nhất với nó là khi số lượng quân trên bàn và số lớp tìm kiếm tăng lên, thậm chí máy tính cũng phải mất rất nhiều thời gian tính toán từng bước. Số lượng nút với sự tăng lên mũi số một cách cấp số nhân thực sự khó đạt được trong thực tế. Trong đồ án, chúng em sẽ lập trình game Caro với thuật toán minimax cơ bản nhất với phương pháp phổ biến nhất được sử dụng hiện nay là cắt bỏ rối trong cây tìm kiếm thông qua Alpha-Beta pruning.

## Các chức năng của đề tài

* Xây dựng trò chơi Caro có giao diện
* Chế độ người – máy với thuật toán được chọn
* Chế độ người – người

# PHẦN NỘI DUNG

## CHƯƠNG 1: KIẾN THỨC LIÊN QUAN

### Thuật toán Minimax

Minimax là một thuật toán máy tính được sử dụng rộng rãi trong các trò chơi trên bàn. Miễn là đó là một trò chơi cạnh tranh với hai người chơi, các hướng dẫn cơ bản của minimax là phù hợp. Trong trò chơi cạnh tranh hai người chơi, mọi người đều muốn tối đa hóa lợi ích của mình trong mỗi quyết định. Vì vậy, giá trị tối đa cho người chơi hiện tại là giá trị cao nhất mà anh ta có thể đạt được khi đối thủ luôn thực hiện các hành động tốt nhất.

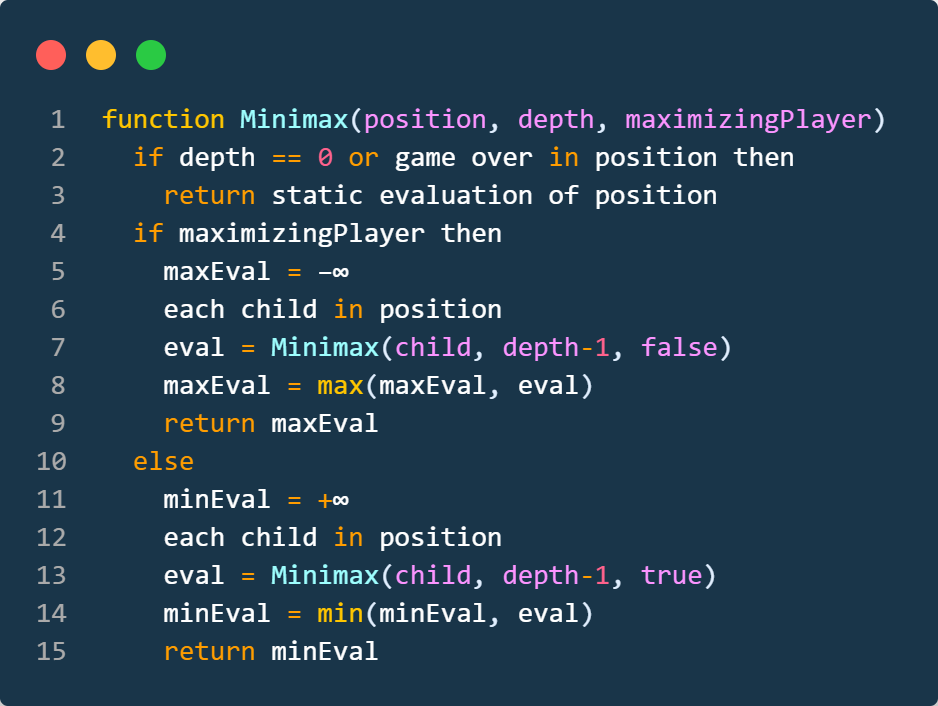
Trong Minimax, chúng ta thực tế đã đặt tên cho hai người chơi là maximizer và minimizer. Maximizer luôn muốn có được điểm số cao nhất trong khi minimizer cố gắng có được điểm số thấp nhất ngược lại. Điều này cũng hoàn toàn phù hợp với kinh nghiệm thực tế của chúng ta. Do đó, trong mỗi tìm kiếm, thuật toán đứng bên cạnh việc tối đa hóa lợi ích của người chơi hiện tại. Mỗi "hành động tốt nhất" đầu ra từ thuật toán không phải là giá trị tối đa dưới tình hình hiện tại, mà là sự lựa chọn tốt nhất để thực hiện sau khi cân nhắc, xem xét rằng đối thủ sẽ tối thiểu hóa lợi ích cho người chơi hiện tại.

**Mã giả:**

Đây là một hàm đệ quy, gồm ba phần.

* Phần đầu tiên là đầu ra dưới cùng, điều kiện là số độ sâu tìm kiếm đạt đến mục tiêu hoặc trò chơi đã kết thúc.
* Phần thứ hai và thứ ba giải quyết giá trị tối đa của người chơi hiện tại và giải pháp tối ưu của đối thủ tương ứng.

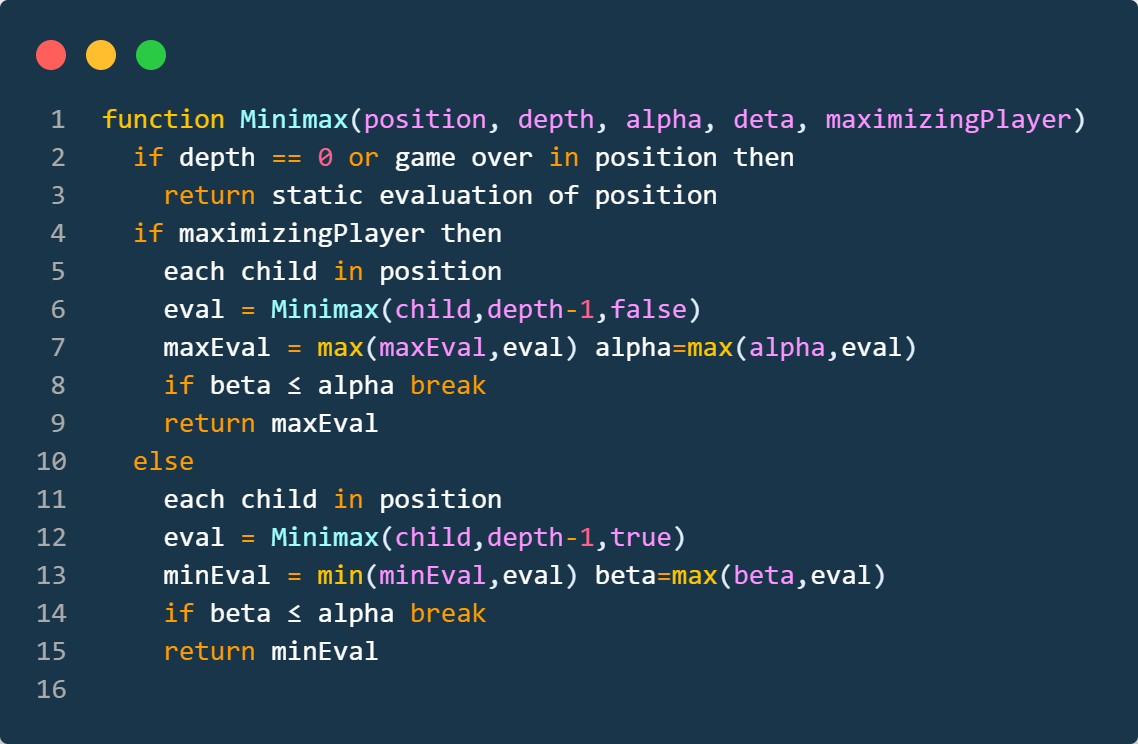
Trong mã giả, maximizingPlayer là một biến boolean. Khi độ sâu hiện tại là lẻ (maximizingPlayer == true), có nghĩa là đó là nước đi cho người chơi đầu tiên, nó sẽ được đánh giá dựa trên phương trình đánh giá bảng đã thiết lập. Ngược lại, nếu đó là độ sâu chẵn (maximizingPlayer == false), tức là nước đi cho người chơi thứ hai, nó sẽ tìm nút trong tình huống hiện tại mà là không lợi nhất cho người chơi đầu tiên.



### Alpha-Beta Pruning

Alpha-beta pruning là một thuật toán tiên tiến có thể giảm số lượng nút trong Minimax. Trong thuật toán minimax, chúng ta đã nói về cách cây trò chơi tìm kiếm tất cả các khả năng để đạt được kết quả cuối cùng, nhưng trong quá trình này, nhiều nút thực sự không cần phải tìm kiếm sâu. Alpha-beta pruning sẽ dừng việc đánh giá một bước khi nó đến một nút tệ hơn so với trước đó đã xem xét. Sau đó, quá trình này thường xuyên tính toán các nút còn lại qua các nút hiện đang bị bỏ qua. Khi chúng ta áp dụng thuật toán minimax tiêu chuẩn, alpha-beta trả về kết quả giống như minimax, nhưng nó cắt bớt một số nhánh không thể ảnh hưởng đến quyết định cuối cùng.

**Mã giả:**

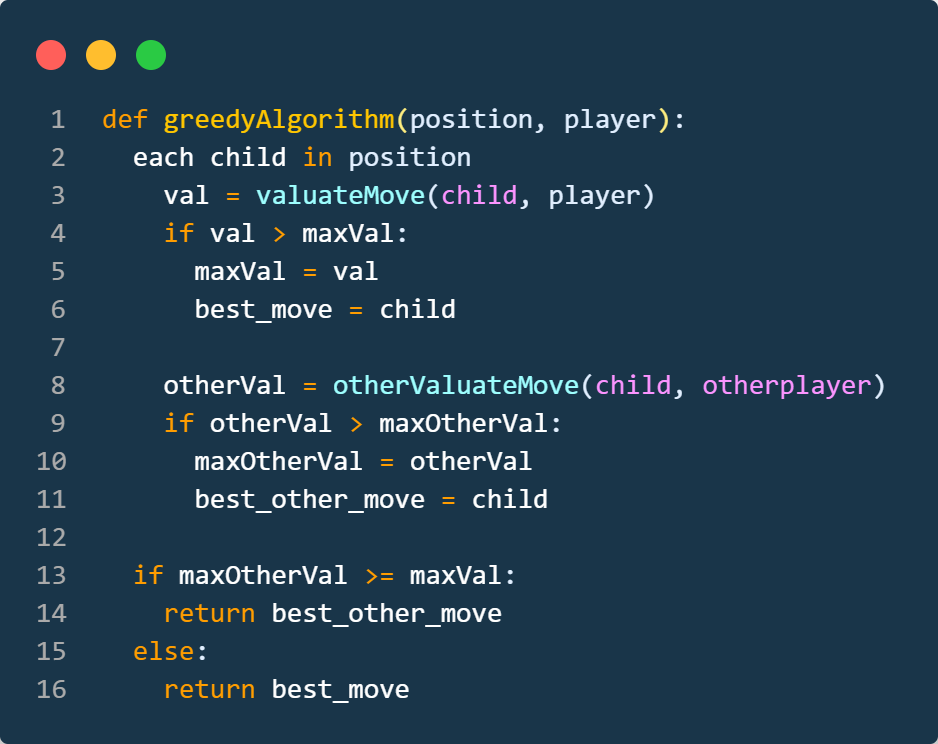


Trong thuật toán này, hai giá trị alpha và beta được giới thiệu. Chúng đại diện cho điểm số tối đa của người chơi hiện tại cần di chuyển và điểm số tối thiểu của đối thủ. Giá trị ban đầu của alpha là âm vô cực, trong khi beta là dương vô cực. Khi điểm số tối thiểu của đối thủ thấp hơn điểm số tối đa của người chơi hiện tại, chúng ta có thể dừng quá trình vì chúng sẽ không bao giờ xuất hiện trong trò chơi thực tế.

### Thuật toán Tham Lam (Greedy)

Chúng em đã thêm một thuật toán tham lam đơn giản. Nó sẽ đánh giá điểm cho cả hai bên đen và trắng riêng biệt trong khi đọc điểm hiện tại của bảng. Ví dụ, nếu lượt đi hiện tại được biết là đen, điểm cho cả đen và trắng sẽ được đánh giá. Nếu điểm đen lớn hơn so với bên trắng, vị trí của bên đen có lợi nhất sẽ được tìm thấy trên bảng. Nếu điểm trắng lớn hơn so với bên đen, người chơi hiện tại đang ở thế thua. Vị trí điểm số tối đa cho bên trắng tương ứng sẽ được tìm thấy, sau đó đặt quân đen ở đây. Đơn giản như vậy, thuật toán chọn giữa tấn công và phòng thủ, rồi tìm vị trí có lợi nhất cho người chơi hiện tại.

**Mã giả**



### Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) nổi bật như một thuật toán đột phá, làm thay đổi quá trình ra quyết định trong môi trường phức tạp thông qua việc xây dựng cẩn thận cây tìm kiếm. Thuật toán này đánh giá một cách tỉ mỉ các bước chơi tiềm năng, đạt được sự cân bằng hài hòa giữa hai nguyên tắc cơ bản là thăm dò (khám phá những bước chưa biết) và khai thác (tận dụng những bước có lợi đã biết trước). Sự tích hợp của Mạng Chính Sách và Mạng Giá Trị vào khuôn khổ này mang lại cho nó sự sâu sắc và chính xác không ngờ:

* Mạng Chính Sách (Policy Network): Thông qua đầu ra tinh tế của nó, đóng vai trò như đèn hướng dẫn định hình sự mở rộng của cây tìm kiếm. Thay vì tạo nhánh một cách bừa bãi, nó đặt điểm sáng vào những bước hứa hẹn và tiềm năng, đảm bảo quá trình thăm dò luôn có chiến lược và tập trung.
* Mạng Giá Trị (Value Network): Hoạt động như một người đánh giá tài năng, kiểm tra một cách tỉ mỉ các nút lá trong cây. Mạng này giảm sự phụ thuộc truyền thống vào các lượt chơi ngẫu nhiên để đánh giá, đưa vào quá trình một mức độ chính xác tăng cao. Khả năng này không chỉ giúp tăng tốc quá trình đánh giá mà còn trang bị cho nó cái nhìn sâu sắc hơn về động lực của trò chơi.

A diagram of a diagram of a policy

Description automatically generated with medium confidence

Quá trình cơ bản của MCTS:

* "Lựa chọn”: MCTS bắt đầu từ nút gốc và áp dụng một cách chọn nút con (được biết đến là Tree Policy) một cách đệ quy để đi qua cây cho đến khi đạt đến nút mở rộng cấp bách nhất.
* "Mở rộng": MCTS có thể thêm vào một hoặc nhiều nút con để mở rộng cây dựa trên các hành động khả dụng. Ở đồ án này, chúng em thêm các nút con là các nút xung quang điểm đánh cũ
* "Mô phỏng": cho phép thực hiện một mô phỏng từ nút lá dựa trên chính sách đã định (còn được biết đến là Chính sách Mặc định) để tạo ra một kết quả.
* "Lùi lại truyền thông", MCTS lùi lại kết quả mô phỏng qua các nút đã chọn để cập nhật giá trị trạng thái của chúng.

A math equation with numbers and a square root

Description automatically generated

Thuật toán UCB (Upper Confidence Bound) thường được sử dụng để giải quyết vấn đề giữa việc thăm dò (exploration) và khai thác (exploitation)

## CHƯƠNG 2: THIẾT KẾ THUẬT TOÁN

### Hàm btsConver

Hàm này lấy tất cả các cột, hàng và đường chéo của mảng numpy và chuyển đổi chúng thành mảng chuỗi để tính toán dễ dàng hơn trong hàm points

|  |
| --- |
| def btsConvert(self, board, player):    temp\_board = np.array(board)    cList, rList, dList = [], [], []    board\_col = len(temp\_board[0])    bdiag = [temp\_board.diagonal(i) for i in range(board\_col - 5, - board\_col + 4, -1)]    fdiag = [temp\_board[::-1, :].diagonal(i) for i in range(board\_col - 5, - board\_col + 4, -1)]    for dgd in bdiag:        bdiagVals = ""        for point in dgd:            if point == 0:                bdiagVals += "0"            elif point == player:                bdiagVals += "1"            else:                bdiagVals += "2"        dList.append(bdiagVals)    for dgu in fdiag:        fdiagVals = ""        for point in dgu:            if point == 0:                fdiagVals += "0"            elif point == player:                fdiagVals += "1"            else:                fdiagVals += "2"        dList.append(fdiagVals)    boardT = temp\_board.copy().transpose()    for col in boardT:        colVals = ""        for point in col:            if point == 0:                colVals += "0"            elif point == player:                colVals += "1"            else:                colVals += "2"        cList.append(colVals)    for row in board:        rowVals = ""        for point in row:            if point == 0:                rowVals += "0"            elif point == player:                rowVals += "1"            else:                rowVals += "2"        rList.append(rowVals)      return dList+cList+rList |

### Hàm getCoordsAround

Hàm được tạo để giảm thời gian tính toán và giảm yếu tố nhánh đệ quy của thuật toán Minimax. Nó trả về tọa độ của các điểm xung quanh các đá trên bảng được đánh giá thay vì đánh giá từng điểm trên bảng

|  |
| --- |
| def getCoordsAround(self, board):    temp\_board = np.array(board)    board\_size = len(temp\_board)    outTpl = np.nonzero(temp\_board)  # return tuple of all non zero points on board    potentialValsCoord = {}    for i in range(len(outTpl[0])):        y = outTpl[0][i]        x = outTpl[1][i]        for dy in [-1, 0, 1]:            for dx in [-1, 0, 1]:                new\_x, new\_y = x + dx, y + dy                if 0 <= new\_x < board\_size and  0 <= new\_y < board\_size and temp\_board[new\_y][new\_x] == 0:                    potentialValsCoord[(new\_x, new\_y)] = 1    finalValsX, finalValsY = [], []    for key in potentialValsCoord:        finalValsY.append(key[1])        finalValsX.append(key[0])      return finalValsX, finalValsY |

### Patterns

Đây là một từ điển của tất cả các mẫu nước đi với trọng số điểm được gán theo cấu trúc FiveInRow; LiveFour; DeadFour; LiveThree; DeadThree; LiveTwo; DeadTwo

|  |
| --- |
| {      '11111': 30000000,      '22222': -30000000,      '011110': 20000000,      '022220': -20000000,      '011112': 50000,      '211110': 50000,      '022221': -50000,      '122220': -50000,      '01110': 30000,      '02220': -30000,      '011010': 15000,      '010110': 15000,      '022020': -15000,      '020220': -15000,      '001112': 2000,      '211100': 2000,      '002221': -2000,      '122200': -2000,      '211010': 2000,      '210110': 2000,      '010112': 2000,      '011012': 2000,      '122020': -2000,      '120220': -2000,      '020221': -2000,      '022021': -2000,      '01100': 500,      '00110': 500,      '02200': -500,      '00220': -500  } |

### Hàm points

Hàm này đánh giá bảng trò chơi dựa trên một nước đi tiềm năng được đề xuất thông qua hàm minimax và trả về giá trị là số điểm tương ứng dựa trên các mẫu mà nó thấy trên bảng trò chơi

|  |
| --- |
| def points(self, board, player):  # evaluates      val = 0      player1StrArr = self.btsConvert(board, player)      for i in range(len(player1StrArr)):          len1 = len(player1StrArr[i])          for j in range(len1):              n = j+5              if(n <= len1):                  st = player1StrArr[i][j:n]                  if st in self.patterns:                      val += self.patterns[st]          for j in range(len1):              n = j+6              if(n <= len1):                  st = player1StrArr[i][j:n]                  if st in self.patterns:                      val += self.patterns[st]      return val |

### Hàm minimax

Hàm áp dụng thuật toán Mini-Max với cắt tỉa alpha beta. Nó sử dụng hàm points để xác định nước đi nào là lợi ích và trả về điểm lợi ích. Điều này sau đó được sử dụng trong hàm computer, nơi AI sẽ chọn đặt “đá” của mình ở đâu có điểm lợi ích cao nhất.

|  |
| --- |
| def minimax(self, board, isMaximizer, depth, alpha, beta, player):      point = self.b.points(board, player)      if depth == 2 or point >= 20000000 or point <= -20000000:          return point      if isMaximizer:  # THE MAXIMIZER          best = self.MIN          potentialValsX, potentialValsY = self.getCoordsAround(board)          for i in range(len(potentialValsX)):              if board[potentialValsY[i]][potentialValsX[i]] == 0:                  board[potentialValsY[i]][potentialValsX[i]] = player                  score = self.minimax(board, False, depth+1, alpha, beta, player)                  best = max(best, score)                  alpha = max(alpha, best)  # best AI Opponent move                  board[potentialValsY[i]][potentialValsX[i]] = 0  # undoing                  if beta <= alpha:                      break          return best      else:  # THE MINIMIZER          best = self.MAX          potentialValsX, potentialValsY = self.getCoordsAround(board)          for i in range(len(potentialValsX)):              if board[potentialValsY[i]][potentialValsX[i]] == 0:                  otherplayer = self.otherPlayerStone(player)                  board[potentialValsY[i]][potentialValsX[i]] = otherplayer                  score = self.minimax(board, True, depth+1, alpha, beta, player)                  best = min(best, score)                  beta = min(beta, best)  # best AI Opponent move                  board[potentialValsY[i]][potentialValsX[i]] = 0  # undoing                  if beta <= alpha:                      break          return best |

### Hàm computer

Hàm này xử lý nước đi của máy tính. Nó là Maximizer đầu tiên của hàm minimax. Sự khác biệt duy nhất là thay vì chỉ trả về điểm lợi ích như hàm minimax, hàm này cũng trả về nước đi tốt nhất

|  |
| --- |
| def computer(self, board, isComputerFirst):    mostPoints = float('-inf')    alpha,beta = self.MIN, self.MAX    bestMoveRow = bestMoveCol = -1      potentialValsX, potentialValsY = self.getCoordsAround(board)    for i in range(len(potentialValsX)):        if board[potentialValsY[i]][potentialValsX[i]] == 0:            board[potentialValsY[i]][potentialValsX[i]] = isComputerFirst            movePoints = max(mostPoints, self.minimax(                board, False, 1, alpha, beta, isComputerFirst))            alpha = max(alpha, movePoints)            board[potentialValsY[i]][potentialValsX[i]] = 0            if beta <= alpha:               break            if movePoints > mostPoints:                bestMoveRow = potentialValsY[i]                bestMoveCol = potentialValsX[i]                mostPoints = movePoints                if movePoints >= 20000000:                    break    global AI\_turn    AI\_turn = False    return bestMoveRow, bestMoveCol |

### Hàm greedyAlgorithm

Hàm áp dụng thuật toán tham lam. Nó cũng sử dụng hàm points để xác định nước đi nào là lợi ích và trả về điểm tốt nhất.

|  |
| --- |
| def greedyAlgorithm(self, board, player):    bestMoveRow = bestMoveCol = -1    bestOtherMoveRow = bestOtherMoveCol = -1      mostPoints = float('-inf')    mostOtherPoints = float('-inf')      potentialValsX, potentialValsY = self.getCoordsAround(board)    for i in range(len(potentialValsX)):        if board[potentialValsY[i]][potentialValsX[i]] == 0:            board[potentialValsY[i]][potentialValsX[i]] = player            movePoints = self.b.points(board, player)            board[potentialValsY[i]][potentialValsX[i]] = self.otherPlayerStone(player)            moveOtherPoints = self.b.points(board, self.otherPlayerStone(player))              board[potentialValsY[i]][potentialValsX[i]] = 0              if moveOtherPoints > mostOtherPoints:                bestOtherMoveRow = potentialValsY[i]                bestOtherMoveCol = potentialValsX[i]                mostOtherPoints = moveOtherPoints              if movePoints > mostPoints:                bestMoveRow = potentialValsY[i]                bestMoveCol = potentialValsX[i]                mostPoints = movePoints    global AI\_turn    AI\_turn = False    if mostOtherPoints >= mostPoints:        return bestOtherMoveRow, bestOtherMoveCol    else:        return bestMoveRow, bestMoveCol |

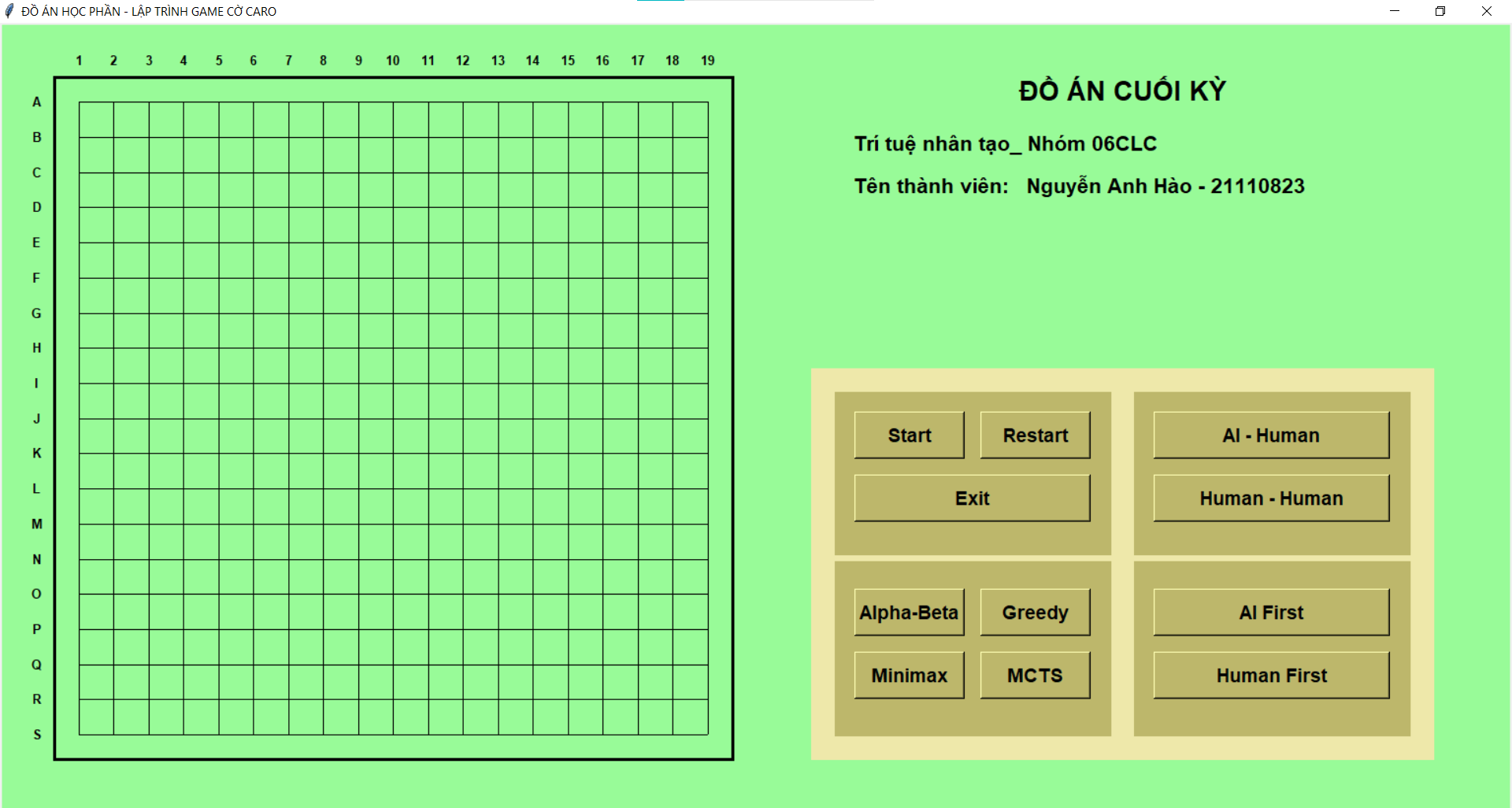
### Hàm mcts\_computer

Hàm áp dụng UCB để thực hiện thuật toán MCTS

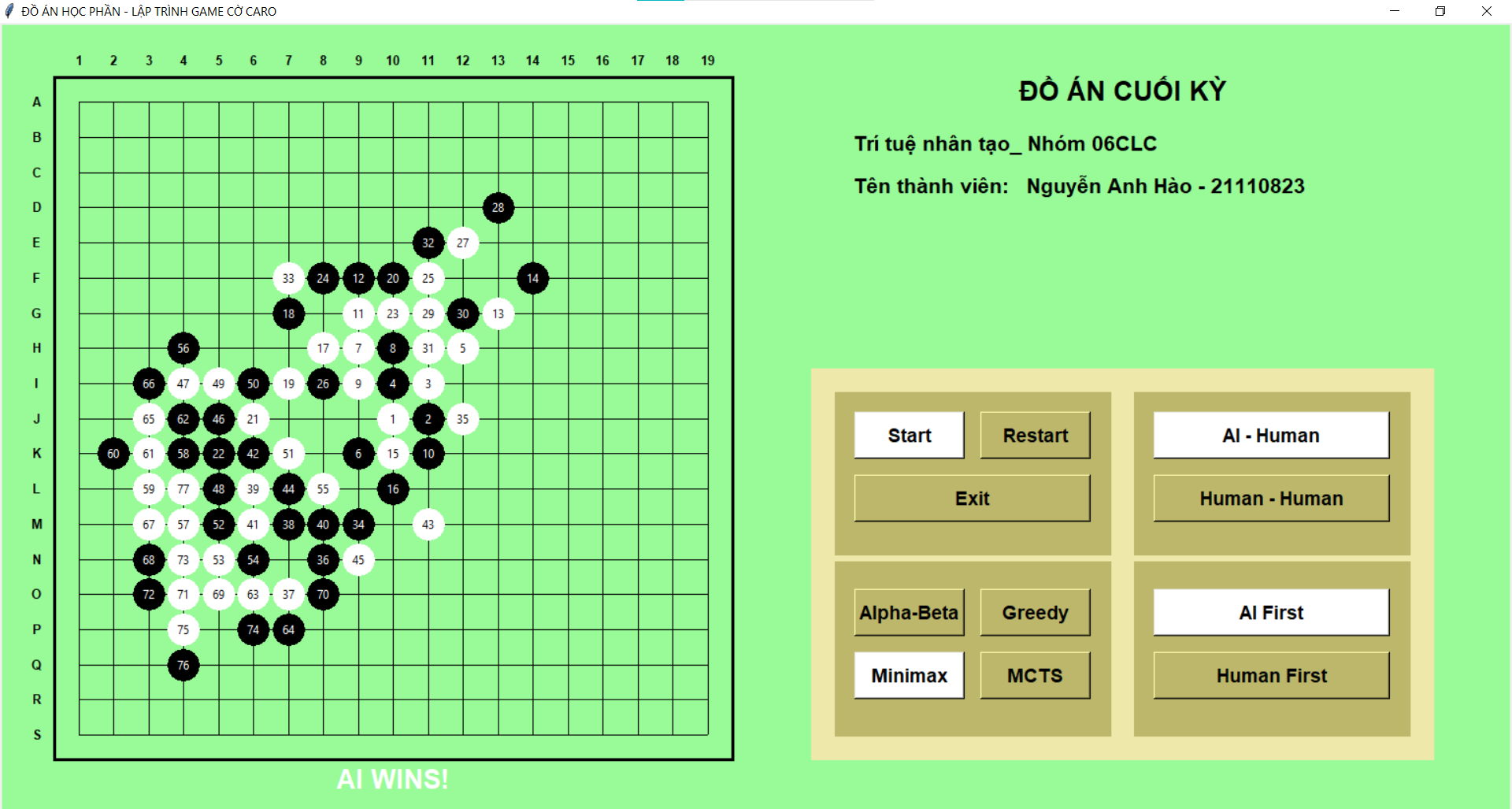
|  |
| --- |
| def mcts\_computer(self, board, isComputerFirst):      mostPoints = float('-inf')      bestMoveRow = bestMoveCol = -1        potentialValsX, potentialValsY = self.getCoordsAround(board)      save\_x = len(potentialValsX)      for i in range(len(potentialValsX)):          if board[potentialValsY[i]][potentialValsX[i]] == 0:              board[potentialValsY[i]][potentialValsX[i]] = isComputerFirst              self.save\_point = 0              movePoints = self.mcts(board, False, 1, isComputerFirst)              ucb = movePoints / self.save\_point + 1.4 \* math.sqrt(math.log(save\_x) / self.save\_point)              board[potentialValsY[i]][potentialValsX[i]] = 0              if ucb > mostPoints:                  bestMoveRow = potentialValsY[i]                  bestMoveCol = potentialValsX[i]                  mostPoints = ucb      global AI\_turn      AI\_turn = False      return bestMoveRow, bestMoveCol |

## CHƯƠNG 3: THIẾT KẾ GIAO DIỆN HỆ THỐNG

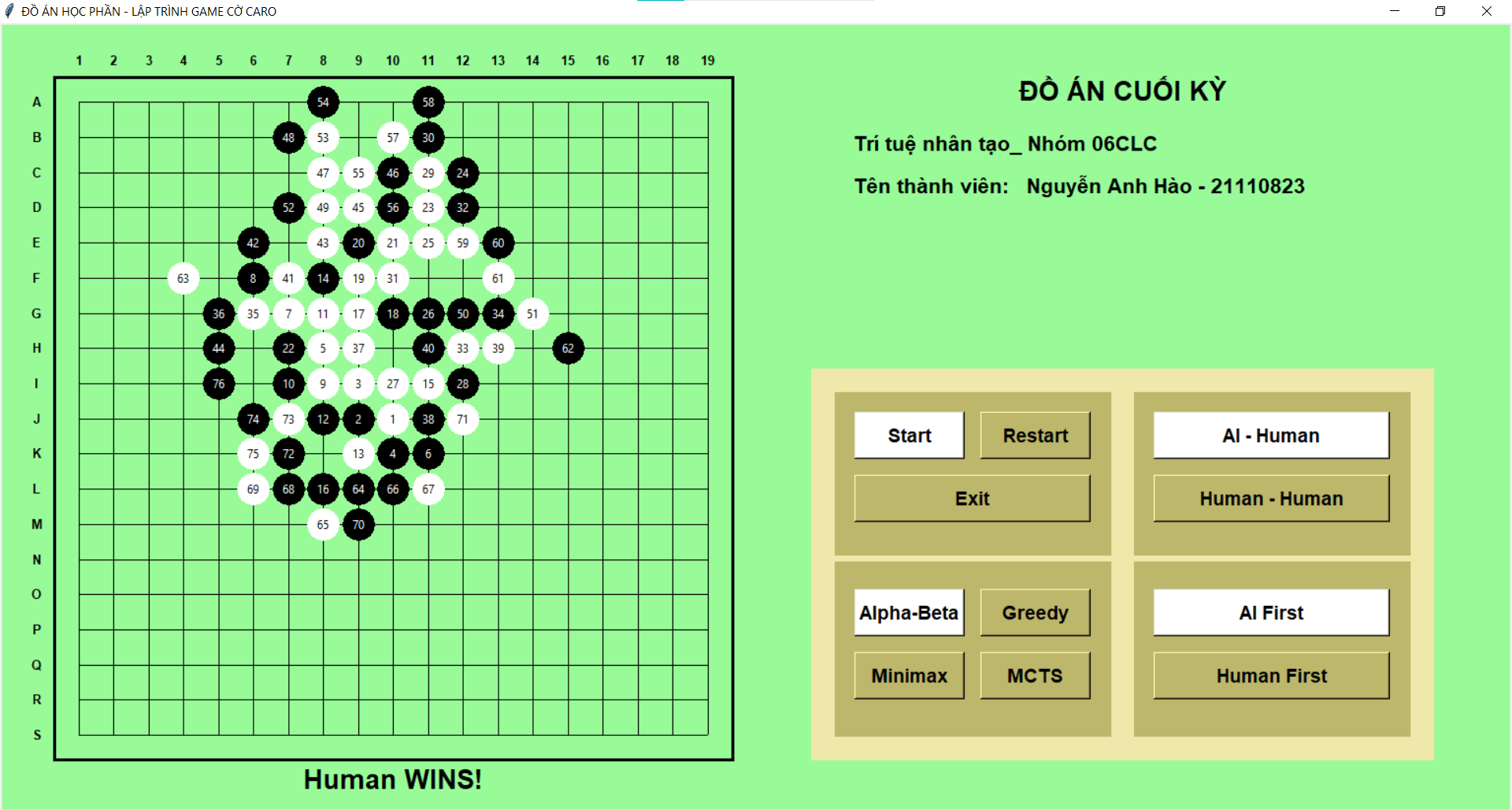
### Giao diện mặc định



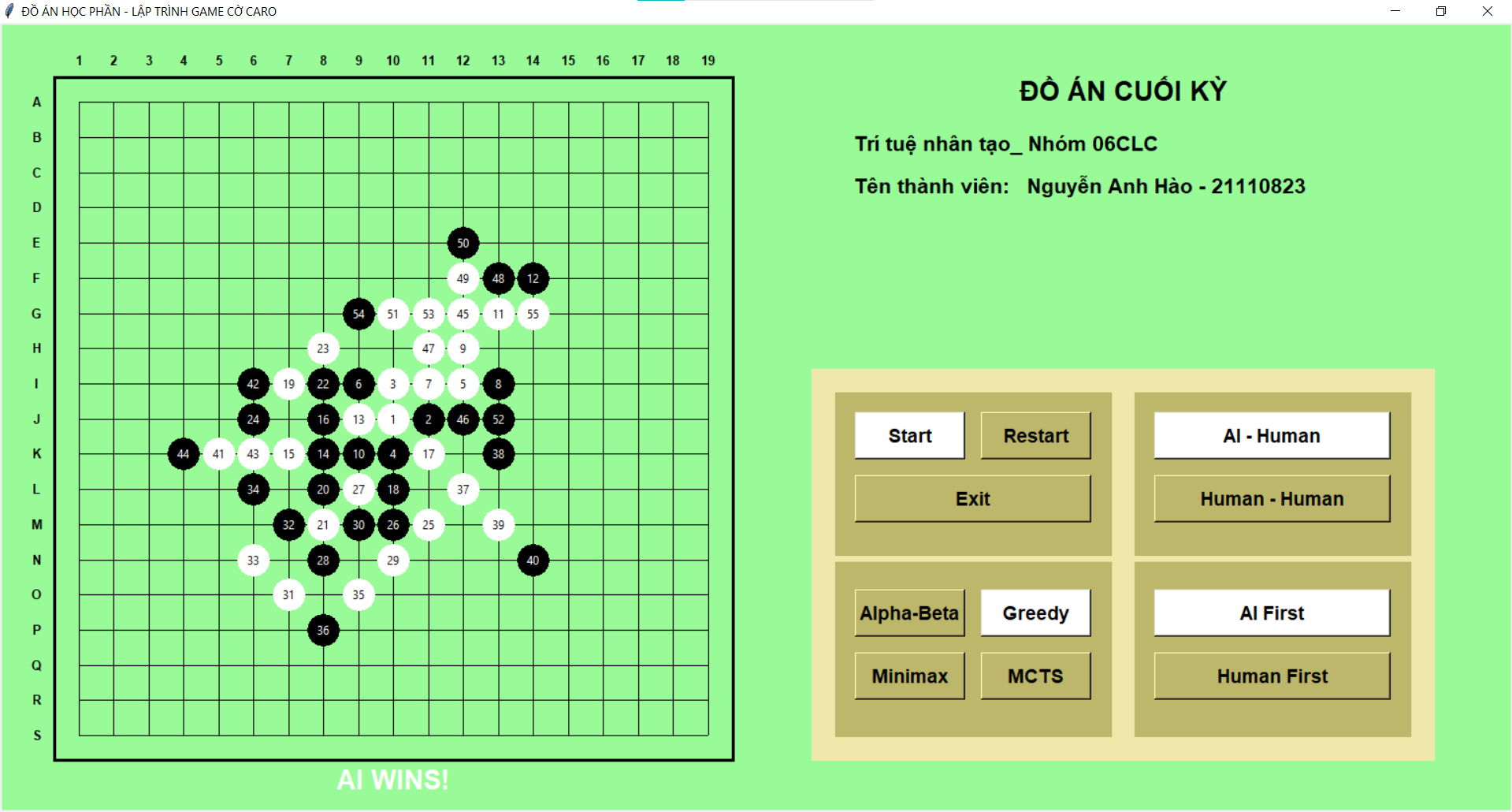
### Thuật toán Minimax



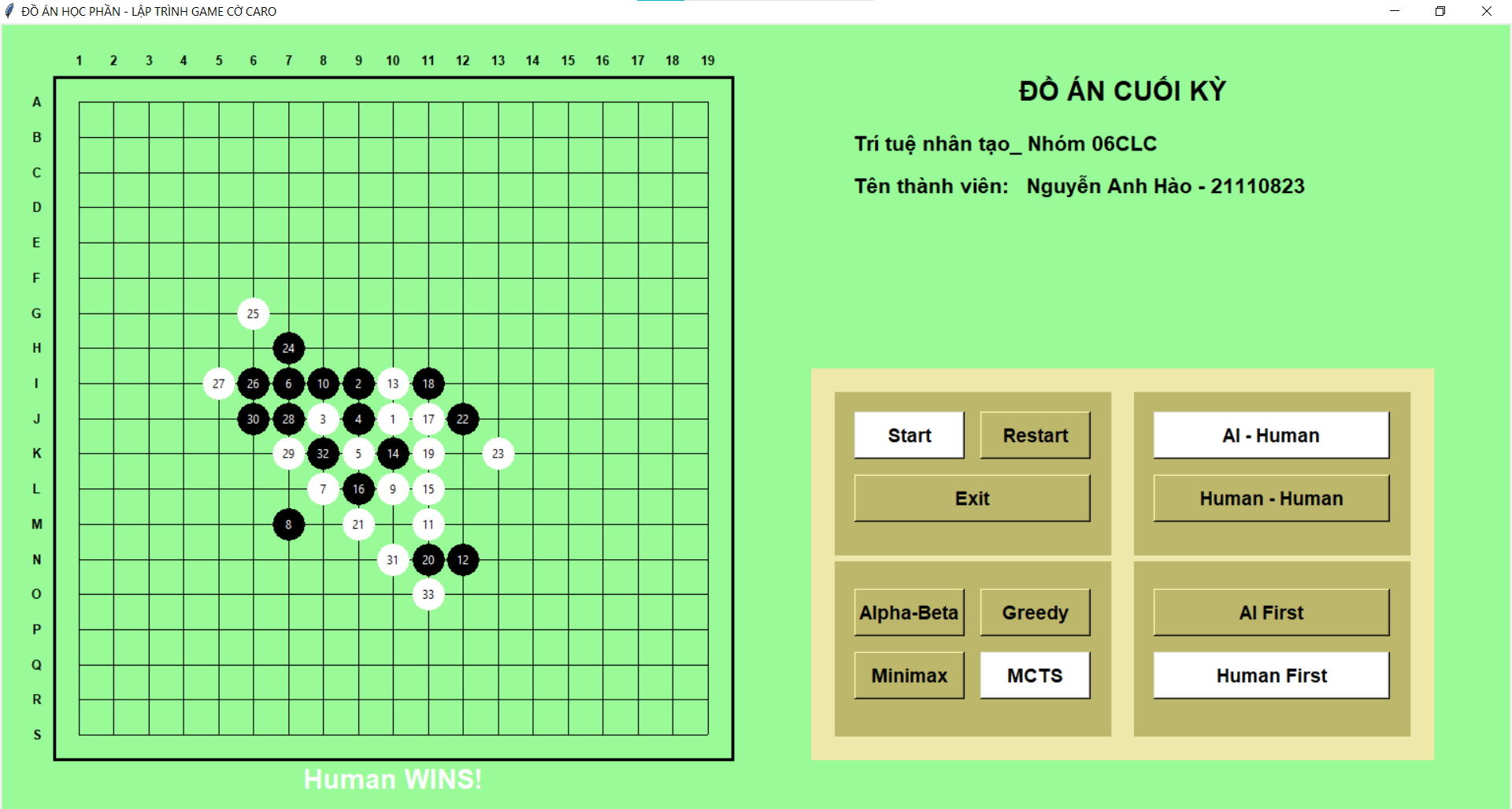
### Thuật toán Minimax with Alpha-Beta cutoff



### Thuật toán Greedy



### Thuật toán Monte Carlo Tree Search



# PHẦN KẾT LUẬN

## Kết quả đạt được

Thành công trong việc phát triển một trò chơi caro hoàn chỉnh, với giao diện đẹp mắt và trải nghiệm người chơi mượt mà. Các tính năng chính của game như chế độ đối kháng 2 người chơi, chế độ chơi với máy tính, và ghi chú điểm đã được triển khai và kiểm thử kỹ lưỡng. Vân dụng kiến thức học được từ môn trí tuệ nhân tạo, tích hợp thành công hai mô hình trí tuệ nhân tạo để cung cấp khả năng chơi của máy tính. Máy tính có thể đánh giỏi và đối đầu với người chơi một cách thông minh và thách thức. Mô hình AI đã được điều chỉnh để đảm bảo tính công bằng và giải thuật tối ưu trong quá trình đánh giữa người chơi và máy tính. Và bọn em đưa ra được các nhận xét sau:

**Minimax:**

Ưu điểm:

* Đảm bảo tìm ra nước đi tối ưu cho người chơi trong trò chơi có thông tin hoàn toàn (perfect information games).
* Hoạt động tốt đối với các trò chơi với không gian trạng thái nhỏ.

Nhược điểm:

* Yêu cầu thời gian tính toán lớn, đặc biệt là với các trò chơi có không gian trạng thái lớn.
* Không hiệu quả trong việc giải quyết các trò chơi với không gian trạng thái lớn hoặc không xác định.

**Minimax với Alpha-Beta Cut Off:**

Ưu điểm:

* Cải thiện hiệu suất so với Minimax thông thường bằng cách cắt bớt các nhánh không cần thiết.
* Phù hợp cho các trò chơi với không gian trạng thái lớn.

Nhược điểm:

* Vẫn đòi hỏi một lượng tính toán đáng kể, đặc biệt là với các trò chơi phức tạp.

**Monte Carlo Tree Search (MCTS):**

Ưu điểm:

* Hiệu quả đối với các trò chơi có không gian trạng thái lớn.
* Tích hợp tốt với các trò chơi có không gian trạng thái không xác định hoặc không hoàn toàn biết trước (imperfect information games).

Nhược điểm:

* Khả năng tiến triển không ổn định, đặc biệt là ở những giai đoạn đầu của trò chơi.
* Yêu cầu số lần thực hiện lặp (iterations) lớn để có kết quả chính xác.

**Greedy:**

Ưu điểm:

* Đơn giản và dễ triển khai.
* Hoạt động tốt trong một số trường hợp đơn giản.

Nhược điểm:

* Thường không tạo ra nước đi tối ưu.

## Ưu điểm

* Thành công trong việc tối ưu hóa thuật toán, giảm thời gian tính toán khi máy tính đưa ra quyết định trong trò chơi caro.
* Trải Nghiệm Người Chơi Tốt Hơn: Thời gian tính toán giảm đồng nghĩa với việc máy tính có thể đưa ra động đậy nhanh chóng và đáng tin cậy, tăng cường trải nghiệm người chơi khi tham gia trò chơi

## Nhược điểm

* Thiếu sự chứng minh rõ ràng về ưu thế của thuật toán so với các thuật toán khác. Điều này có thể tạo ra một thiếu sót trong việc thuyết phục về sự hiệu quả của hệ thống AI

## Hướng phát triển

* Thêm hiển thị đường đi tốt nhất mà AI đã tính toán giúp người chơi hiểu tại sao máy tính đưa ra một quyết định cụ thể

# 

# TÀI LIỆU THAM KHẢO

[1] Gomoku Calculator. Web Tính toán nước đi tốt nhất cờ caro - <https://gomocalc.com/> [Ngày truy cập: 10/11/2024]

[2] KSH Gomoku Ai. Web chơi cờ caro với nhiều thuật toán - <https://easycaro.github.io/>

[Ngày truy cập: 11/11/2024]

[3] Wikipedia | Minimax. Bách khoa toàn thư - <https://vi.wikipedia.org/wiki/Minimax> [Ngày truy cập: 20/11/2024]

[4] Wikipedia | Greedy. Bách khoa toàn thư -<https://en.wikipedia.org/wiki/Greedy_algorithm> [Ngày truy cập: 20/11/2024]