ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Xây dựng chương trình chơi Cờ toán Việt Nam

Nhóm: 5

Mã lớp: 136802

Giáo viên hướng dẫn: PGS.TS Thân Quang Khoát

Nhóm sinh viên thực hiện:

1. Phan Minh Anh Tuấn 20205227
2. Nguyễn Thị Hoài Linh 20205231
3. Đồng Gang Thép 20205130
4. Vũ Tuấn Kiệt 20200308
5. Nguyễn Hoàng Long 20200364

HÀ NỘI, 02/2023

Lời nói đầu

Cờ toán Việt Nam là một trò chơi đối kháng mang tinh thần "Thân thiện, trí tuệ và sáng tạo" của người Việt Nam. Cờ toán ra đời mang theo khát vọng chinh phục thế giới, thể hiện trí tuệ Việt Nam và triết lý sâu sắc về cuộc đời.

Trong quá trình học môn Nhập môn Trí tuệ nhân tạo, nhóm chúng em đã được giới thiệu về trò chơi này. Nhận thấy đây là một trò chơi hay nhưng chưa được phổ biến rộng rãi, nhóm đã quyết định chọn đề tài bài tập lớn "Xây dựng chương trình chơi Cờ toán Việt Nam". Nhóm thực hiện đề tài này nhằm mục đích vận dụng các kiến thức đã học về tìm kiếm có đối thủ vào giải quyết một bài toán thực tế, đồng thời phổ biến một trò chơi trí tuệ bổ ích tới rộng rãi mọi người.

Tài liệu này trình bày thiết kế của phần mềm trò chơi Cờ toán và các thuật toán tìm kiếm có đối thủ đã được nhóm ứng dụng trong phần mềm này. Nhóm chúng em xin cảm ơn sự hướng dẫn của thầy Thân Quang Khoát về cả chuyên môn và định hướng trong quá trình thực hiện đề tài. Vì kiến thức trong lĩnh vực Trí tuệ nhân tạo của nhóm còn hạn hẹp nên không tránh khỏi những sai sót khi thực hiện. Chúng em mong nhận được sự góp ý và đánh giá từ thầy để đề tài được hoàn thiện hơn và chúng em có thể trau dồi thêm những kiến thức quý giá. Chúng em xin chân thành cảm ơn.

Mục lục

Chương 1

Giới thiệu bài toán

Cờ toán Việt Nam là sản phẩm sáng tạo của ông Vũ Văn Bảy (Vũ Bảy), một nghệ nhân nặn tượng ở thành phố Bắc Ninh, tỉnh Bắc Ninh, Việt Nam. Cờ toán Việt Nam đã được Cục Bản quyền tác giả văn học nghệ thuật, Bộ Văn hóa - Thông tin (nay là Bộ Văn hóa, Thể thao và Du lịch) chính thức công nhận sản phẩm trí tuệ vào tháng 5/2005.

Trò chơi có 2 người chơi thay phiên nhau đưa ra nước đi của mình theo quy luật của trò chơi như sau:

Bàn cờ toán Việt Nam có hình chữ nhật, bao gồm 99 ô, được chia ra làm 9 cột và 11 hàng. Mỗi ô hàng dọc sẽ tương ứng với các chữ cái từ a đến i, và hàng ngang là từ 1 đến 11.

Quân cờ toán Việt Nam có hình tròn. Tổng số quân trên bàn cờ là 20 được chia đều ra 2 đội. Mỗi quân cờ của các đội sẽ có màu sắc khác nhau. Ngoài ra, các dấu chấm trên các quân cờ cũng chính là con số tương ứng của quân cờ đó. Khi tham gia, người chơi sẽ sắp xếp các quân cờ vào các ô nằm ở hàng ngang đầu tiên của bàn cờ theo thứ tự từ trái sang phải. Đặc biệt, quân cờ số 0 sẽ nằm ở ô cột 5 hàng 2 và cột 5 hàng 10.

Nếu như các quân từ 1 đến 9 có thể tự do di chuyển theo hàng ngang, cột dọc và đường chéo thì riêng quân số 0 lại tuyệt đối không được phép di chuyển ra khỏi vị trí. Dựa vào số lượng dấu chấm trên các quân cờ, người chơi có thể tiến hành triển khai các bước đi trong phạm vi số chấm đó. Lưu ý không dùng quân cờ nhảy qua các ô đã có quân cờ khác đứng. Ví dụ: quân số 1 chỉ được di chuyển đúng sang các ô sát bên cạnh, quân số 2 thì có thể di chuyển trong phạm vi 2 ô trống...

Để ăn quân cờ của đối thủ, người chơi phải có hai quân cờ của mình đứng ở 2 ô liền nhau theo chiều dọc, ngang hoặc chéo. Sau đó, dùng các phép tính cộng – trừ – nhân – chia để cho ra một con số cụ thể. Con số này phải vừa bằng với số ô cần di chuyển tính từ quân đứng phía trước của mình và quân muốn ăn. Ví dụ: Người chơi có quân 2 với quân 3 đứng gần nhau, quân 3 đứng trước. Trong khi quân của đối thủ cách vị trí của quân 3 là 5 ô thì người chơi nên dùng phép tính cộng để ra kết quả như mong muốn.

Dưới đây là một số trường hợp có thể xảy ra khi sử dụng các phép toán mà người mới chơi nên biết:

- Nếu phép tính cho kết quả lớn hơn 10 thì người chơi sẽ lấy số hàng đơn vị để đi.
- Nếu dùng phép chia thì người chơi có thể dùng số dư của phép tính để ăn quân.
- Nếu sau khi tính toán, kết quả thu lại có số 0 ở hàng đơn vị thì được xem như không có giá trị.

Trò chơi sẽ chính thức kết thúc khi có một trong hai bên bị bắt mất quân số 0. Ngoài ra, kết quả trận đấu có thể được phân định dựa vào việc tính điểm. Mỗi chấm trên mặt quân cờ tương ứng 1 điểm. Hai bên có thể thỏa thuận điểm kết thúc cho mỗi ván đấu là 10, 20 hay 30... điểm. Trước khi chơi, 2 bên cần thỏa thuận rõ ràng các cách tính điểm, cách áp dụng các phép tính, cách phân định thắng thua...

Chương 2

Các kỹ thuật tìm kiếm có đối thủ

Các ký hiệu sử dụng trong phần này:

- node: nút hiện tại đang được tìm kiếm.
- depth: độ sâu mà chúng ta muốn tìm kiếm.
- alpha và beta: giá trị alpha và beta tại thời điểm xét.
- maximizingPlayer: biến cho biết người chơi hiện tại có phải là người chơi muốn giá trị hàm ước lượng là cực đại hay không.
- player: biến cho biết người chơi hiện tại.
- evaluate(node): hàm ước lượng của nút.
- child: node con của node hiện tại.
- children(node): hàm trả về danh sách các nút con của nút hiện tại.

2.1. Giải thuật Minimax

Minimax là một giải thuật cho lớp bài toán tìm kiếm có đối thủ (adversarial search). Nó được sử dụng rộng rãi trong trò chơi 2 người như cờ caro, cờ vua... dùng để xác định quyết định tốt nhất trong các trò chơi 2 người.

Các bước thực hiện giải thuật Minimax:

1. Xây dựng cây biểu diễn trò chơi, mô tả tất cả các trạng thái có thể của trò chơi.
2. Đánh giá mỗi trạng thái bằng cách tính toán giá trị minimax đối với mỗi nút trong cây biểu diễn trò chơi. Giá trị minimax của một nút là giá trị hàm mục tiêu để đánh giá các trạng thái của trò chơi, giả sử rằng cả hai người chơi đều chơi nước đi tối ưu từ đó đến cuối trò chơi.
3. Duyệt cây từ trạng thái gốc đến từng nút con và chọn giá trị hàm mục tiêu tối đa hoặc tối thiểu tùy theo vai trò của mỗi người chơi.
4. Trả về quyết định tốt nhất cho người chơi hiện tại

Mã giả giải thuật Minimax như sau:

```
def minimax(node, depth, maximizingPlayer):
    if depth == 0 or node is leaf:
        return evaluate(node)

    if maximizingPlayer:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, False)
            bestValue = max(bestValue, value)
        return bestValue
    else:
        bestValue = inf
        for child in children(node):
            value = minimax(child, depth - 1, True)
            bestValue = min(bestValue, value)
        return bestValue
```

Hình 2.1: Mã giả giải thuật minimax

2.2. Cắt tỉa alpha-beta

Vấn đề của giải thuật Minimax là số trạng thái trò chơi mà nó phải đánh giá tăng theo hàm mũ của độ sâu cây biểu diễn trò chơi. Do vậy, giải thuật trên không phù hợp với nhiều bài toán trò chơi thực tế. Ta có thể cắt bỏ một số nhánh tìm kiếm không đem lại kết quả tối ưu cho giải thuật.

Giải thuật Minimax kết hợp cắt tỉa alpha-beta là một cải tiến để tối ưu hóa không gian tìm kiếm. Nó sử dụng các giá trị alpha và beta để giới hạn tìm kiếm, trả về cùng nước đi tìm được bởi giải thuật Minimax, đồng thời cắt bỏ những nút không cần thiết mà không làm ảnh hưởng đến kết quả cuối cùng.

Trong mỗi nút, giá trị alpha là giá trị tối đa mà người chơi MAX có thể đạt được trên con đường từ nút gốc đến nút hiện tại, giá trị beta là giá trị tối thiểu mà người chơi MIN có thể đạt được trên con đường từ nút gốc đến nút hiện tại. Nếu giá trị alpha lớn hơn hoặc bằng beta, chúng ta có thể bỏ qua các nút con của nút hiện tại mà không cần tính toán tiếp. Ví dụ, khi tìm kiếm từ nút cha, nếu giá trị alpha của cha lớn hơn hoặc bằng giá trị beta của con, chúng ta có thể dừng tìm kiếm trên nút con đó và chuyển sang nút con khác.

Mã giả của giải thuật Minimax kết hợp cắt tỉa như sau:

```
def minimax(node, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or node is a leaf:
        return evaluate(node)

    if maximizingPlayer:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, alpha, beta, False)
            bestValue = max(bestValue, value)
            alpha = max(alpha, bestValue)
            if alpha ⩾ beta:
                break
        return bestValue
    else:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, alpha, beta, True)
            bestValue = min(bestValue, value)
            beta = min(beta, bestValue)
            if alpha ⩾ beta:
                break
        return bestValue
```

Hình 2.2: Mã giải giải thuật Minimax cắt tỉa alpha-beta

2.3. Giải thuật Negamax

Negamax là một biến thể của giải thuật Minimax, trong đó giá trị của cả hai người chơi được chuyển đổi thành số dương hoặc số âm (dựa trên tính chất zero-sum của trò chơi hai người chơi).

Cụ thể, thuật toán này vận dụng tính chất $min(a, b) = -max(-b, -a)$ để đơn giản hóa việc triển khai thuật toán Minimax. Điều này giúp chúng ta tránh việc phải tính toán hai trường hợp riêng biệt cho mỗi người chơi. Cách hoạt động của giải thuật Negamax vẫn giống như Minimax, ngoại trừ việc giá trị của mỗi nút được đảo ngược khi đến lượt người chơi khác.

Mã giả giải thuật Negamax như sau:

```
def negamax(node, depth, player):
  if depth == 0 or node is a leaf:
    return player * evaluate(node)

  bestValue = -inf
  for child in children(node):
    value = -negamax(child, depth-1, -player)
    bestValue = max(bestValue, value)
  return bestValue
```

Hình 2.3: Mã giả giải thuật Negamax

Do Negamax hoạt động tương tự như Minimax, nên có thể áp dụng cắt tỉa Alpha-beta vào giải thuật Negamax. Mã giả giải thuật Negamax kết hợp cắt tỉa alpha - beta như sau:

```
def negamax(node, depth, player, alpha, beta):
  if depth == 0 or node is a leaf:
    return player * evaluate(node)

  bestValue = -inf
  for child in children(node):
    bestValue = max(bestValue, -negamax(child, depth-1,
                -player, -beta, -alpha))
    alpha = max(alpha, bestValue)
    if alpha ⩾ beta:
      break
  return bestValue
```

Hình 2.4: Mã giả giải thuật Negamax cắt tỉa alpha-beta

2.4. Giải thuật Negascout

Đối với các trò chơi có không gian trạng thái lớn như cờ toán Việt Nam thì phương pháp cắt tỉa alpha-beta vẫn không phù hợp vì không gian tìm kiếm khi đã kết hợp cắt tỉa vẫn rất lớn. Một phương pháp hiệu quả hơn giúp hạn chế không gian tìm kiếm của bài toán là giải thuật Negascout.

Negascout là một giải thuật tìm kiếm alpha-beta cải tiến để tăng tốc độ tìm kiếm. Trong Negascout, khi tìm kiếm một nút, chúng ta sẽ tính toán giá trị của nút đó với một khoảng từ α đến β (window). Nếu giá trị của nút nằm trong khoảng đó, chúng ta sẽ tiếp tục mở rộng nút đó để tìm kiếm giá trị chính xác hơn. Nếu giá trị nằm ngoài khoảng đó, chúng ta sẽ giảm khoảng tìm kiếm từ $[\alpha, \beta]$ thành $[-\alpha - 1, -\alpha]$ và tính toán lại giá trị của nút đó.

Mã giả giải thuật Negascout như sau:

```
def negascout(node, depth, alpha, beta):
    if depth == 0 or node is a terminal node:
        return evaluate(node)
    for child in node:
        if child == node[0]:
            score = -negascout(child, depth-1, -beta, -alpha)
        else:
            score = -negascout(child, depth-1, -alpha-1, -alpha)
            if alpha < score < beta:
                score = -negascout(child, depth-1, -beta, -score)
        alpha = max(alpha, score)
        if alpha ⩾ beta:
            break
    return alpha
```

Hình 2.5: Mã giả giải thuật Negascout

2.5. Tìm kiếm Quiescence

Tìm kiếm Quiescence là thuật toán được sử dụng để mở rộng tìm kiếm tại các nút có thể có những nước đi gây biến động đáng kể giá trị hàm ước lượng trong trò chơi. Nó sẽ đánh giá đến khi vị trí đó đủ ổn định để được đánh giá là tĩnh (quiescene), nghĩa là không thể có nước đi tấn công nào từ nút đó được nữa. Nó giảm thiểu tác động về hiệu ứng đường chân trời (Horizon Effect), là một hiện tượng xảy ra khi giới hạn độ sâu của thuật toán tìm kiếm trong trò chơi ở một mức cố định. Điều này dẫn đến các mối đe dọa và cơ hội nằm ngoài độ sâu tìm kiếm sẽ không được phát hiện. Mục đích của tìm kiếm Quiescence chính là cố gắng giảm thiểu vấn đề này.

```
def quiescence(alpha, beta):
    best_score = evaluate()
    if best_score ⩾ beta:
        return beta
    if alpha < best_score:
        alpha = best_score

    for move in all_captures():
        make_capture(move)
        score = -quiescence(-beta, -alpha)
        undo_capture(move)

        if score ⩾ beta:
            return beta
        if score > alpha:
            alpha = score
    return alpha
```

Hình 2.6: Mã giả tìm kiếm Quiescence

Chương 3

Cài đặt trò chơi

Trò chơi Cờ toán Việt Nam được cài đặt bằng ngôn ngữ Python và chia thành 3 module chính như sau:

1. Engine: biểu diễn trạng thái bàn cờ và các nước đi theo luật của trò chơi
2. AI: triển khai các giải thuật tìm kiếm có đối thủ vào trò chơi
3. UI: tạo giao diện cho trò chơi

3.1. Biểu diễn bàn cờ và luật chơi

Trạng thái của bàn cờ và các nước đi có thể của mỗi quân cờ được biểu diễn bởi lớp GameState và lớp Move trong module Engine.

3.1.1 Lớp GameState

Lớp GameState lưu trữ trạng thái của bàn cờ bởi các thuộc tính:

- board: một mảng 2 chiều có kích thước 11x9 trong đó mỗi phần tử đại diện cho ô vuông trên bàn cờ. Trong một ô vuông có thể không có quân cờ nào ở đó, kí hiệu: "--". Một ô vuông có chứa một quân cờ sẽ được kí hiệu bởi màu và trị số của quân cờ đó, ví dụ ô vuông có chứa quân đỏ số 9 được ký hiệu là "r9".
- red_to_move: một biến kiểu boolean cho biết hiện tại là lượt đi của bên nào, bằng True nếu đến lượt quân đỏ đi và ngược lại.
- move_log: một mảng lưu trữ tất cả các nước đi đã được thực hiện trong trò chơi.

Ngoài ra còn có 2 thuộc tính phụ:

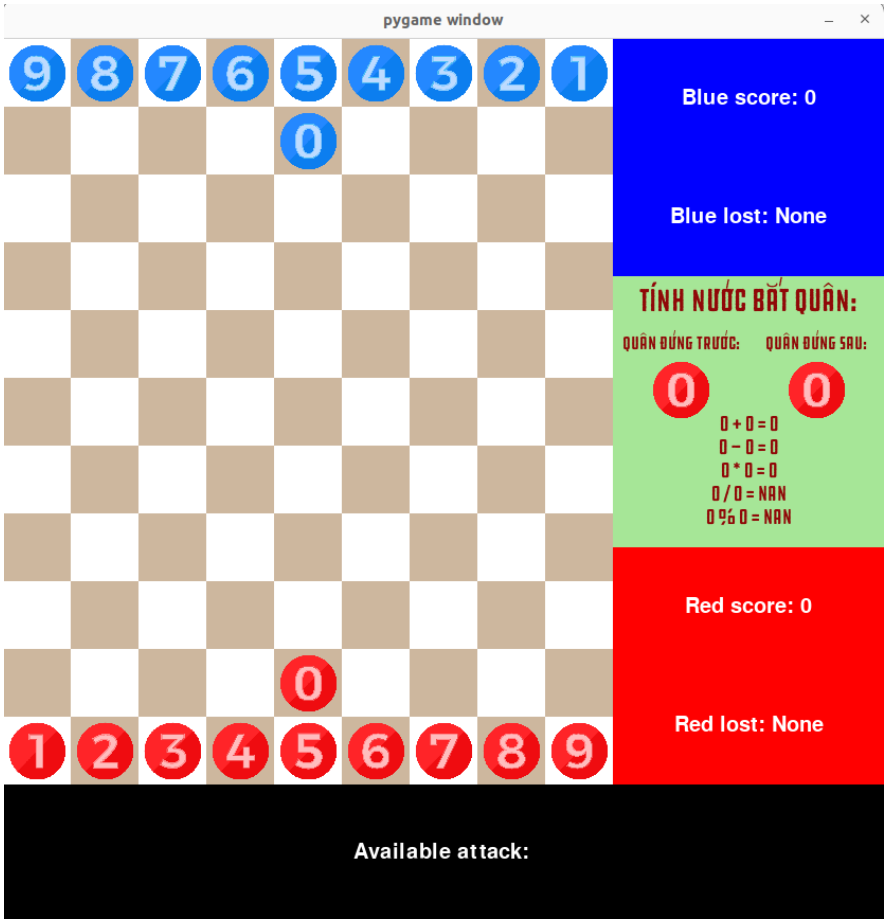
- zeroRed: vị trí quân đỏ số 0 trên bàn cờ.
- zeroBlue: vị trí quân xanh số 0 trên bàn cờ.

Để xử lý cơ chế trò chơi và trả lại các bước di chuyển hợp lệ, lớp này bao gồm các phương thức:

- check(): kiểm tra xem một trong hai quân 0 đã bị ăn hay chưa
- makeMove(move): thực hiện một nước đi move được truyền vào, lưu nước đi đó vào nhật ký di chuyển move_log và hoán đổi lượt chơi
- undoMove(): hoàn tác nước đi cuối cùng đã được thực hiện
- getValidMoves(): trả về danh sách nước đi hợp lệ, bao gồm cả nước đi quân thông thường (nước đi mà không ăn quân) và nước ăn quân
- getAllPossibleMoves(): trả về danh sách nước đi quân thông thường hợp lệ
- getAllPossibleAttacks(): trả về danh sách nước ăn quân hợp lệ

Trong đó, phương thức getAllPossibleMoves() và getAllPossibleAttacks() đều được hỗ trợ bởi 2 phương thức khác để thực hiện nhiệm vụ của mình. getAllPossibleMoves() được hỗ trợ bởi getPieceMove(r, c, piece) và getMoveWithDirection(r, c, piece, direction, moves). getAllPossibleAttacks() được hỗ trợ bởi getAttackMove(r, c, piece) và getAttackWithDirection(r, c, piece, direction, moves).

Trò chơi bắt đầu với cả hai quân màu đỏ và xanh tại vị trí như hình vẽ (insert hình vẽ xuống dưới), với người chơi quân đỏ bắt đầu trò chơi. Số điểm kết thúc trận đấu trong chương trình được quy định là 15 điểm. Trận đấu sẽ kết thúc khi một bên ăn được quân số của đối thủ, hoặc một bên đạt đến 15 điểm.



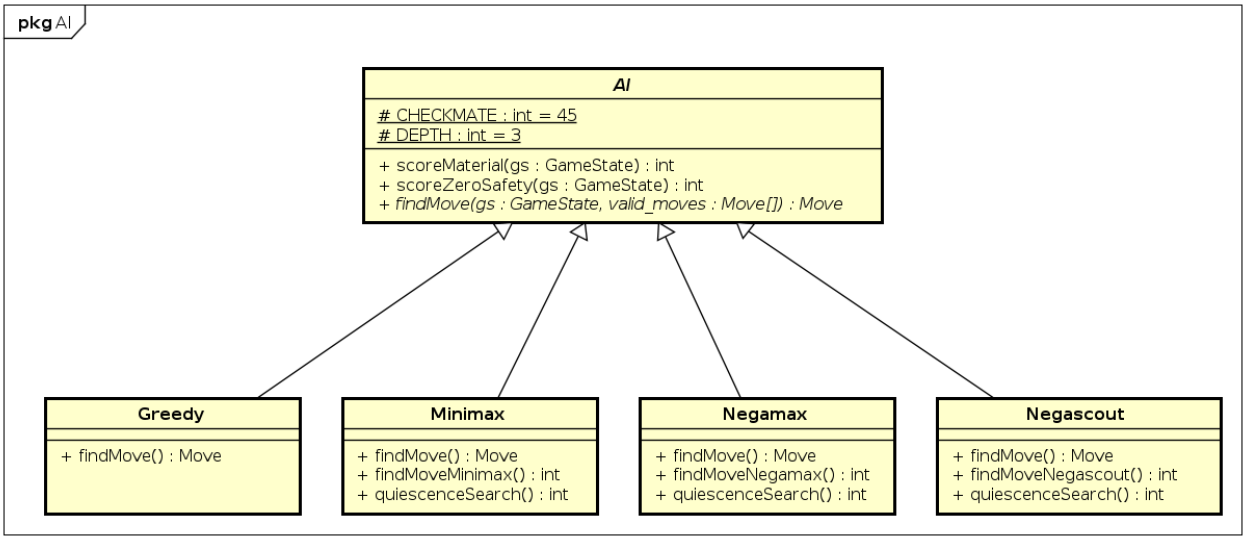
Hình 3.1: Giao diện bàn cờ

3.1.2 Lớp Move

Lớp Move biểu diễn cho các nước đi của quân cờ bởi các thuộc tính phục vụ việc ánh xạ từ ký hiệu cột (a,...,i) và hàng (1,...,11) trên bàn cờ sang chỉ số hàng và cột cho bàn cờ lưu trong chương trình. Mỗi vị trí trên bàn cờ sẽ được biểu diễn dưới dạng (hàng, cột). Mỗi nước đi của quân cờ sẽ được khởi tạo với các thông tin: vị trí bắt đầu, vị trí kết thúc, quân di chuyển, quân bị ăn và mã số (moveID) của nước đi đó phục vụ cho việc so sánh. Từ đó, ta mã hóa và lưu trữ được các nước đi trong trò chơi phục vụ việc chơi cờ của người dùng và triển khai các giải thuật tìm kiếm có đối thủ sau này.

3.2. Ứng dụng các kỹ thuật tìm kiếm có đối thủ

Các giải thuật tìm kiếm có đối thủ được triển khai trong module AI của phần mềm, trong đó bao gồm 5 lớp:



Hình 3.2: Biểu đồ lớp thể hiện module AI

3.2.1 AI

Lớp AI đại diện cho một đối tượng như một người máy AI với các thuộc tính đặc trưng CHECKMATE (giá trị đánh giá tình huống chiếu tướng), DEPTH (độ sâu tìm kiếm) và next_move (nước đi tiếp theo sẽ thực hiện). Trong lớp này định nghĩa hàm ước lượng $h(n)$ cho phép đánh giá mỗi nước đi của máy trong trò chơi. Trong quá trình thực hiện đề tài, ban đầu nhóm đã cài đặt hàm ước lượng rất đơn giản bằng cách tính điểm cho bàn cờ như sau:

```
def scoreMaterial(board):
  score = 0
  for row in board:
      for square in row:
          if square is red:
              if value(square) == 0:
                  score += 45
              else:
                  score += value(square)
          elif square is blue:
              if value(square) == 0:
                  score -= 45
              else:
                  score -= value(square)
  return score
```

Hàm này sẽ duyệt qua từng ô trên bàn cờ và tính điểm tương ứng với quân cờ nằm trên mỗi ô. Nếu ô đó chứa quân cờ màu đỏ, nó sẽ thêm vào điểm của bàn cờ một lượng bằng số điểm của quân cờ đó. Nếu ô đó chứa quân cờ màu xanh, nó sẽ thêm vào điểm của bàn cờ một lượng bằng giá trị âm của số điểm của quân cờ đó. Riêng quân 0 đóng vai trò quan trọng, nếu quân 0 bị bắt thì người chơi sẽ thua, nên điểm của quân 0 được tính bằng giá trị CHECKMATE. Cuối cùng, hàm sẽ trả về tổng số điểm của bàn cờ. Đánh giá hàm ước lượng như vậy nhằm mục đích tính xem trên bàn cờ, bên nào đang có lợi thế về chất lượng quân cờ hơn. Giả sử nếu điểm trả về là một số dương càng lớn, nghĩa là bên đỏ đang có lợi thế càng lớn và ngược lại.

Sau đó, nhóm tiếp tục phát triển hàm ước lượng bằng cách bổ sung một hàm gọi là scoreZeroSafety, được sử dụng để tính điểm an toàn của quân số 0 của người chơi như sau:

```
def scoreZeroSafety(board):
  score = 0
  red_direction = ((-1, -1), (-1, 1), (-1, 0), (0, -1), (0, 1))
  blue_direction = ((0, -1), (0, 1), (1, -1), (1, 0), (1, 1))
  for d in red_direction:
      for i in range(1, 4, 1):
          if board[d[0] * i][d[1] * i][0] == 'b':
              score -= 4 - i
      for d in blue_direction:
          for i in range(1, 4, 1):
              if board[d[0] * i][d[1] * i][0] == 'r':
                  score += 4 - i
  return score
```

Hàm này sẽ duyệt các hướng có thể tấn công quân số 0 và tính số lượng những quân đối địch nằm trên những hướng đó. Nếu có quân đỏ xung quanh quân xanh, điểm sẽ được tăng lên 1; còn nếu có quân xanh xung quanh quân đỏ, điểm sẽ được giảm xuống 1. Cuối cùng, hàm sẽ trả về điểm an toàn của quân 0. Hàm scoreZeroSafety này có tác dụng phòng ngự, giúp cho chúng ta nhận diện được các mối đe dọa tấn công quân số 0 từ đối phương.

Cuối cùng, ta thu được hàm ước lượng vận dụng các tri thức cụ thể của bài toán để đánh giá tốt mỗi trạng thái của trò chơi, giúp cho AI có thể quyết định nước đi tiếp theo tốt nhất.

```
def scoreMaterial(board):
  score = 0
  for row in board:
      for square in row:
          if square is red:
              if value(square) == 0:
                  score += 45
              else:
                  score += value(square)
          elif square is blue:
              if value(square) == 0:
                  score -= 45
              else:
                  score -= value(square)
  score += scoreZeroSafety(board)
  return score
```

3.2.2 Greedy

Lớp Greedy là lớp con của lớp AI, từ đó có thể sử dụng các thuộc tính và phương thức của lớp AI. Lớp này được xây dựng để triển khai giải thuật tham lam trong việc tìm kiếm nước đi tiếp theo trong trò chơi.

Hàm findMove trong lớp này sử dụng giải thuật tham lam để tìm nước đi tiếp theo tốt nhất. Hàm này sẽ lấy tất cả các nước đi hợp lệ và đánh giá điểm cho mỗi nước đi bằng cách sử dụng phương thức scoreMaterial từ lớp AI. Nước đi có điểm cao nhất sẽ được trả về như nước đi tiếp theo tốt nhất.

Ta có thể thấy rằng, giải thuật Greedy rất đơn giản, dễ hiểu, dễ cài đặt và tốc độ tìm ra nước đi rất nhanh. Nhưng cũng bởi vậy mà Greedy có hạn chế lớn, đó là Greedy chỉ tìm kiếm nước đi tối ưu trong mỗi bước hiện tại mà không tính toán đến những nước đi sau nên có thể chọn phải nước đi tốt trước mắt nhưng sau đó sẽ dẫn đến thất bại. Nói ngắn gọn, Greedy có thể không chọn được nước đi mang lại kết quả chung cuộc tốt nhất.

3.2.3 Minimax, Negamax và Negascout

Các lớp Minimax, Negamax và Negascout đều kế thừa lớp AI và ghi đè phương thức findMove để tìm kiếm nước đi tốt nhất tiếp theo cho một trạng thái bàn cờ. Các lớp này lần lượt được cài đặt dựa trên giải thuật Minimax, Negamax và Negascout theo tên gọi của chúng. Cách hoạt động chung của 3 lớp này là: Xuất phát từ phương thức findMove nhận giá trị tham số gồm trạng thái hiện tại của trò chơi và danh sách các nước hợp lệ, từ đây gọi đến phương thức tiếp theo (findMoveMinimax, findMoveNegamax và findMoveNegascout) triển khai các thuật toán riêng. Các thuật toán đều được

thực hiện với một độ sâu tìm kiếm giới hạn đã xác định từ trước. Đồng thời, Minimax và Negamax có sử dụng kết hợp phương pháp cắt tỉa Alpha-Beta để giảm số lượng nước đi phải xét. Khi độ sâu tìm kiếm bằng 0, cả 3 lớp đều tiếp tục gọi phương thức quiescenceSearch để đánh giá các nước đi nhằm mở rộng tìm kiếm ở những nước đi gây biến động. Toàn bộ các giải thuật ở đây được cài đặt theo những lý thuyết đã được trình bày ở phần 2.

3.3. Giao diện trò chơi

Trong giao diện trò chơi bao gồm 2 phần chính: chọn chế độ chơi và vẽ bảng.

3.3.1 Cảnh mở đầu và chọn chế độ chơi

Ở phần này, có 4 lớp để hiển thị các bước nhập chế độ chơi cho người chơi.

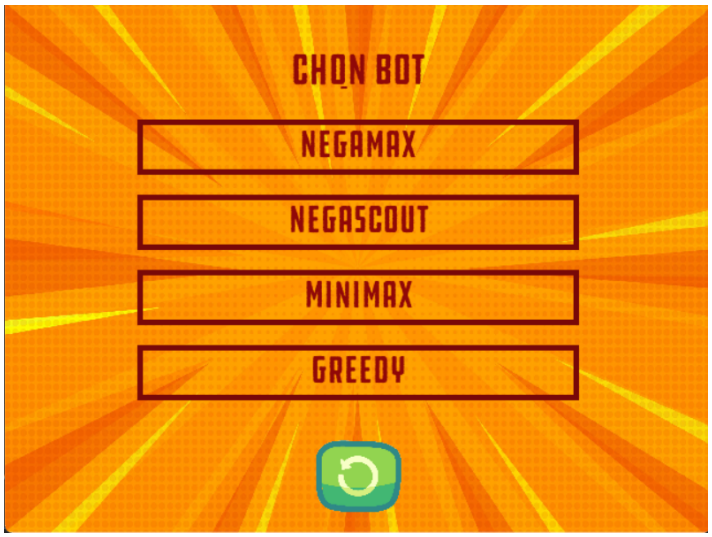
1. **SimpleScene**: Lớp SimpleScene đại diện cho cảnh mở đầu của trò chơi với các phương thức draw() để vẽ giao diện cảnh mở đầu. Phương thức update() sẽ lắng nghe các sự kiện và trả về cảnh tiếp theo nếu nút "NEXT" được nhấn.

Hình 3.3: Cảnh mở đầu trò chơi

2. **ChoooseScene**: Lớp ChooseScene đại diện cho cảnh chọn chế độ chơi của trò chơi với phương thức draw() để vẽ giao diện phần chọn chế độ chơi. Phương thức update() sẽ lắng nghe các sự kiện và trả về cảnh trước đó nếu nút "BACK" được nhấn. Phương thức element() sẽ lắng nghe các chế độ chơi mà người dùng chọn. Nếu người chơi chọn chế độ chơi "NGƯỜI VÀ NGƯỜI" thì phương thức sẽ trả về (True, True) tương ứng với hai người chơi đều là người và bắt đầu chơi. Nếu người chơi chọn chế độ chơi "NGƯỜI VÀ MÁY" thì phương thức trả về (True, False) tương ứng với một người chơi là người và người chơi còn lại là Bot và chuyển sang chọn Bot. CUối cùng, nếu người chơi chọn chế độ chơi "MÁY VÀ MÁY" thì phương thức sẽ trả về (False, False) tương ứng với hai người chơi đều là Bot và chuyển sang chọn Bot thứ nhất và Bot thứ hai.

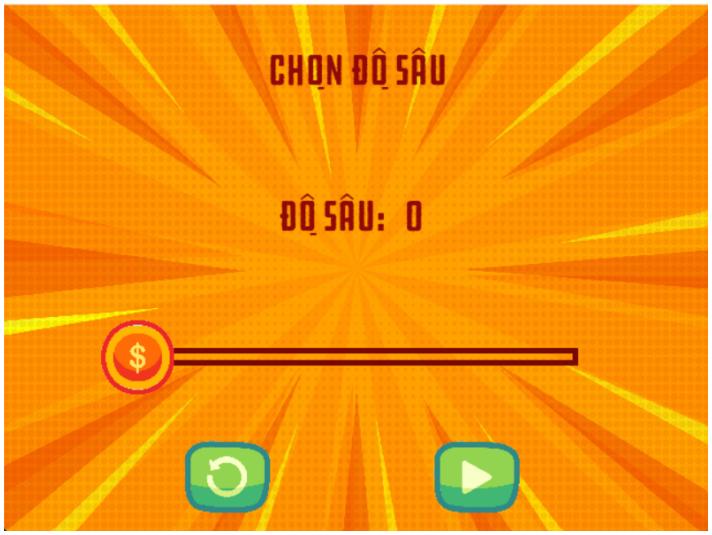
Hình 3.4: Cảnh chọn chế độ chơi

3. **ChooseBot**: Lớp ChooseBot đại diện cho cảnh chọn Bot của trò chơi với các phương thức draw() để vẽ giao diện phần chọn Bot. Phương thức update() sẽ lắng nghe các sự kiện và trả về cảnh trước đó nếu nút "BACK" được nhấn. Phương thức element() sẽ lắng nghe các sự kiện và Bot nào được chọn thì sẽ trả về lớp tương ứng với Bot đó.



Hình 3.5: Cảnh chọn bot

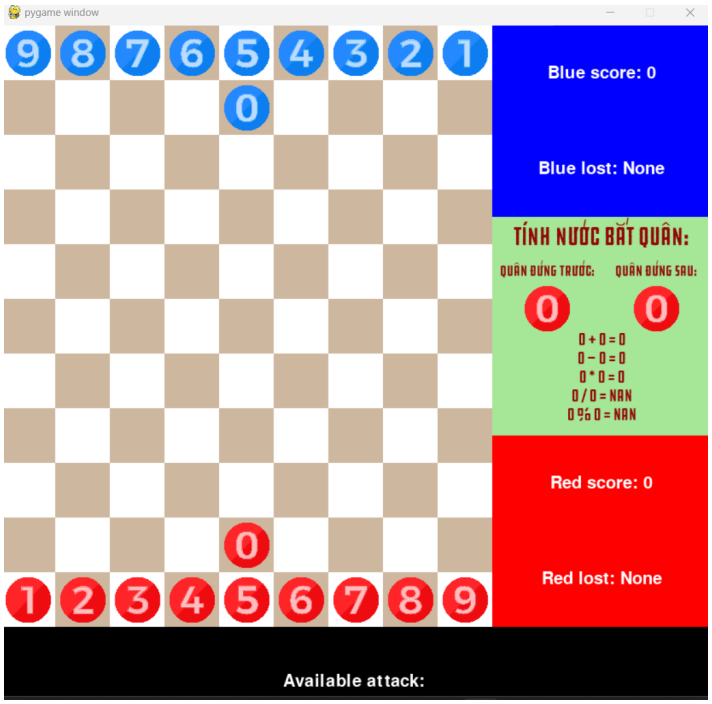
4. **ChooseDepth**: Lớp này chỉ được gọi khi ở cảnh trước, người chơi chọn các Bot sử dụng độ sâu như Minimax, Negamax, Negascout. Lớp ChooseDepth đại diện cho cảnh chọn độ sâu của trò chơi với các phương thức draw() để vẽ giao diện phần chọn độ sâu. Trong phương thức update() sẽ có một vòng alwpj để lắng nghe thanh kéo chọn độ sâu. Phương thức update() sẽ lắng nghe các sự kiện và trả về cảnh trước đó nếu nút "BACK" được nhấn, nút "NEXT" đuọc nhấn thì bắt đầu chơi khi đầy đủ các Bot đuọc chọn hoặc chọn Bot nếu Bot thứ hai chưa được chọn. Phương thức element() sẽ trả về độ sâu mà người chơi chọn.



Hình 3.6: Cảnh chọn độ sâu

3.3.2 Phần vẽ bảng

Phần vẽ bảng sẽ dùng hàm drawGameState() để vẽ bảng và sử dụng lớp CAL để vẽ bảng tính. Hàm drawGameState() sẽ gọi 3 hàm drawBoard(), highlightSquares(), drawPieces(). Hàm drawBoard() sẽ vẽ một bảng gồm 9x11 ô đan xen nhau bằng hai màu trắng và vàng. Hàm hightlightSquares() sẽ vẽ các nước được đi nếu một ô được nhấn. Hàm drawPieces() sẽ vẽ các quân cờ trên một bảng. Lớp CAL được dùng để vẽ bảng tính hỗ trợ người chơi ăn quân.



Hình 3.7: Bàn cờ

Chương 4

Kết luận

Thông qua quá trình thực hiện đề tài, chúng em đã có được kinh nghiệm hữu ích trong việc ứng dụng trí tuệ nhân tạo vào giải quyết vấn đề thực tế, cụ thể là cài đặt các thuật toán tìm kiếm có đối thủ trong trò chơi Cờ toán Việt Nam. Kết quả thu được của đề tài là một phần mềm trò chơi Cờ toán Việt Nam có ứng dụng trí tuệ nhân tạo tương tác với người chơi. Phần mềm hoạt động tốt với các chế độ chơi khác nhau sử dụng các thuật toán tìm kiếm có đối thủ kinh điển Minimax, Negamax, Negascout với độ sâu tìm kiếm từ 1 đến 4. Theo tác giả của trò chơi, số thế chiến lược của Cờ toán là lũy thừa của 87, do đó không gian trạng thái trò chơi vô cùng lớn. Nhóm nhận thấy còn một số hướng phát triển để tối ưu hoạt động của các thuật toán như ứng dụng transposition table để lưu trữ kết quả của các trạng thái đã xử lý trước đó, giúp giảm khối lượng tính toán hay ứng dụng thuật toán Monte Carlo Tree Serach, một thuật toán tìm kiếm xác suất sử dụng mẫu ngẫu nhiên để ước tính giá trị của các nước đi. Tuy nhiên, thời gian cho đề tài môn học có hạn và chưa từng có kinh nghiệm trong việc xử lý các bài toán tìm kiếm trong trò chơi trước đó nên nhóm chưa thể thực hiện giải pháp tối ưu hoàn toàn. Nếu có sai sót trong việc thực hiện đề tài, mong thầy góp ý và nhận xét giúp chúng em hoàn thiện đề tài hơn. Hy vọng đề tài của chúng em sẽ trở thành một tài liệu tham khảo hữu ích cho các bạn sinh viên học Nhập môn Trí tuệ nhân tạo sau này.

Chương 5

Tài liệu tham khảo

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson, 3 edition, 2010.

[2] Chess Programming Wiki. Negamax, 2018. URL https://www.chessprogramming.org/Negamax.

[3] Chess Programming Wiki. Negascout, 2020. URL https://www.chessprogramming.org/NegaScout.

[4] Chess Programming Wiki. Quiescence search, 2021. URL https://www.chessprogramming.org/Quiescence_Search.