ĐẠI HỌC QUỐC GIA TP HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

Nhập môn trí tuệ nhân tạo

Bài tập lớn 2
Game playing - Cờ vua

GVHD: Vương Bá Thịnh
SV: Huỳnh Tấn Luân - 1914054
Nguyễn Thanh Lưu - 1914084
Nguyễn Trần Quốc Uy - 1915866

TP. HỒ CHÍ MINH, THÁNG 5/2022

Mục lục

1 Giới thiệu bài toán

Cờ vua là một trò chơi board game đối kháng dành cho hai người. Trò chơi sử dụng một bàn cờ hình vuông chia thành 64 ô vuông nhỏ hơn với 8 hàng ngang và 8 hàng dọc. Mỗi người chơi sẽ bắt đầu với 16 quân cờ, bao gồm 8 con tốt, 2 mã, 2 tượng, 2 xe, 1 hậu và 1 vua. Mục tiêu của người chơi là cố gắng chiếu hết vua đối phương. Vua được gọi bị "chiếu hết" khi đang bị chiếu mà không có cách nào thoát ra. Khi một người chơi bị chiếu hết, trò chơi kết thúc hay nói cách khác người chơi đó đã thua. Cũng có một số trường hợp mà trò chơi có thể kết thúc với tỷ số hòa. Trong suốt ván cờ, hai người chơi thay phiên nhau di chuyển một quân cờ của mình đến một vị trí khác trên bàn cờ. Một người chơi sẽ cầm quân cờ màu trắng và người còn lại sẽ cầm quân cờ đen. Có các quy tắc nhất định về việc di chuyển các quân cờ cũng như việc ăn quân của đối thủ. Người chơi cầm quân cờ trắng sẽ đi trước.

2 Luật chơi

2.1 Luật chơi cơ bản

Ván cờ được bắt đầu giữa hai đấu thủ bằng cách luân phiên nhau di chuyển các quân cờ trên một bàn hình vuông co kich thươc 8x8 ô gọi là bàn cờ. Thông thường bên quân cờ màu trắng sẽ được đi trước.

Mục tiêu của mỗi đấu thủ là tấn công vua của đối phương sao cho vua của đối phương không thể di chuyển đúng luật bước nào để thoát khỏi thế chiếu gọi là "chiếu bí". Ngươi chơi "chiêu bi" đươc đôi phương la ngươi chơi chiên thăng.

Nếu xuất hiện thế cờ mà không đấu thủ nào có thể thực hiện được chiếu bí thì ván cờ hòa.

Không được di chuyển một quân cờ tới ô có quân cùng màu đang đứng. Nếu quân cơ đi tới một ô đang có quân khac mau đứng thì se băt đươc quân đo cua đôi phương (không ap dung vơi quân tôt do tôt đi thăng, ăn cheo).

2.2 Luật di chuyển quân cờ

- Vua: được di chuyển theo đường cheo, hang ngang hoăc côt doc nhưng môi lươt đi chỉ đi được 1 ô.

- Hậu: được di chuyển nhiều ô trên ban cơ theo đường cheo hang ngang hoăc côt doc.
- Xe: có thể di chuyển nhiêu ô trên bàn cờ chỉ cần cùng cột dọc hoặc hang ngang.
- Tượng: có thể di chuyển nhiêu ô trên cùng đường chéo mà nó đứng.
- Mã: tư vi tri hiên tai, ma có thể đi đến 1 trong nhưng ô gần nhất nhưng không nằm trên cùng hàng ngang cột dọc, hay đường cheo với ô nó đang đứng.
- Tốt: mỗi bên có 8 quân tốt. Ở nước đi đầu tiên tốt có thể tiến 1 hoặc 2 nước trên cùng cột dọc. Tư lân di chuyên thư 2, con tôt đo chỉ thể tiến 1 ô. Tốt ăn theo đường chéo trong pham vi 1 ô hương vê phia trươc. Sức mạnh của quân tốt được phát huy cao nhất khi tốt được phong cấp thành 1 trong các quân hậu, xe, tượng, mã. Trong cơ vua, trươc khi phong câp, tôt không thê lui.

3 Biểu diễn trạng thái bàn cờ trong mã nguồn

Trong file *pieces.py*, class Piece và các class kế thừa từ Piece quy định đặc tính, vị trí hiện tại, luật di chuyển và ăn quân của các quân cờ. Để phục vụ cho hàm lượng giá, mỗi quân cờ có thuộc tính $VALUE$ quy định độ quan trọng của quân cờ, $VALUE$ càng lớn thì quân đó càng quan trọng:

- Xe: 500
- Mã: 320
- Tượng: 330
- Hậu: 900
- Vua: 90000
- Tốt: 100

Class Board trong file *Board.py* chứa thông tin hiện tại của bàn cờ. Trong đó, attribute $chess_pieces$ trong class Board là 1 ma trận 8x8 đặc trưng cho bàn cờ. Mỗi phần tử trong ma trận là 1 đối tượng của class quân cờ đó (nếu có quân cờ tại vị trí đó).

File *ai.py* quy định hàm lượng giá và giải thuật minimax alpha beta cutoff nhóm áp dụng. Trong đó, class Heuristic quy định thêm ma trận vị trí của từng loại quân cờ. Điểm vị trí của 1 quân cờ có giá trị tương đương 1 phần tử có tọa độ tương ứng trong ma trận này.

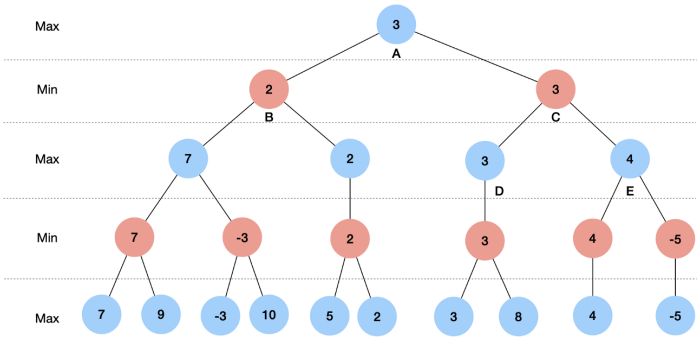
```
PAWN_TABLE = numpy.array([
    [ 0,  0,  0,  0,  0,  0,  0,  0],
    [ 5, 10, 10,-20,-20, 10, 10,  5],
    [ 5, -5,-10,  0,  0,-10, -5,  5],
    [ 0,  0,  0, 20, 20,  0,  0,  0],
    [ 5,  5, 10, 25, 25, 10,  5,  5],
    [10, 10, 20, 30, 30, 20, 10, 10],
    [50, 50, 50, 50, 50, 50, 50, 50],
    [ 0,  0,  0,  0,  0,  0,  0,  0]
])
```

Hình 1: Ma trận quy định điểm vị trí của quân tốt

4 Giải thuật Minimax

4.1 Mô tả giải thuật

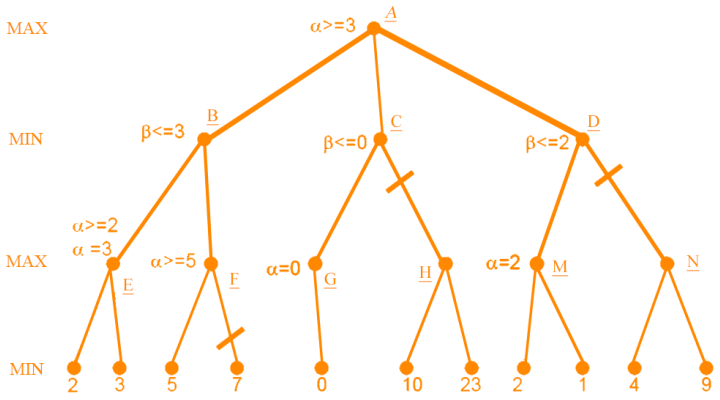
Minimax là giải thuật là một thuật toán đệ quy lựa chọn bước đi kế tiếp trong một trò chơi có hai người bằng cách định giá trị cho các Node trên cây trò chơi sau đó tìm Node có giá trị phù hợp để đi bước tiếp theo. Giả sử đối thủ của bạn cũng sử dụng kiến thức về không gian trạng thái giống bạn. Giải thuật Minimax áp dụng giả thuyết này để tìm kiếm không gian trạng thái của trò chơi.
Cây trò chơi (Game tree) - Đại khái là một sơ đồ hình cây thể hiện từng trạng thái, từng trường hợp của trò chơi theo từng nước đi. Mỗi node biểu diễn 1 trạng thái của trò chơi hiện tại trên cây trò chơi. Node được gọi nút lá là tại đó trò chơi kết thúc (trạng thái trò chơi lúc đó có thể thắng, thua hoặc hòa).
Giải thuật Minimax Hai người chơi trong game được đại diện là MAX và MIN. MAX đại diện cho người chơi luôn muốn chiến thắng và cố gắng tối ưu hóa ưu thế của mình còn MIN đại diện cho người chơi cố gắng cho người MAX giành số điểm càng thấp càng tốt. Giải thuật Minimax thể hiện bằng cách định trị các Node trên cây trò chơi: Node thuộc lớp MAX thì gán cho nó giá trị lớn nhất của con Node đó. Node thuộc lớp MIN thì gán cho nó giá trị nhỏ nhất của con Node đó. Từ các giá trị này người chơi sẽ lựa chọn cho mình nước đi tiếp theo hợp lý nhất.



Hình 2: Minh họa giải thuật minimax

4.2 Alpha–beta pruning

Giải thuật cắt tỉa Alpha-beta từng được nhiều nhà khoa học máy tính đề xuất ý tưởng và không ngừng được cải tiến cho đến ngày nay. Giải thuật này thường sử dụng chung với thuật toán tìm kiếm Minimax nhằm hỗ trợ giảm bớt các không gian trạng thái trong cây trò chơi, giúp thuật toán Minimax có thể tìm kiếm sâu và nhanh hơn. Giải thuật cắt tỉa Alpha-beta có nguyên tắc đơn giản "Nếu biết là trường hợp xấu thì không cần phải xét thêm".
Nút Max có một giá trị alpha (lớn hơn hoặc bằng alpha – luôn tăng), nút min có một giá trị beta (nhỏ hơn hoặc bằng beta – luôn giảm). Khi chưa có alpha và beta xác định thì thực hiện tìm kiếm sâu (depth-first) để xác định được alpha, beta, và truyền ngược lên các nút cha.



Hình 3: Minh họa Alpha–beta pruning

4.3 Áp dụng vào AI

Cụ thể phần giải thuật alphabeta sẽ được implement như bên dưới:

```
def alphabeta(chessboard, depth, a, b, maximizing):
    if (depth == 0):
        return Heuristics.evaluate(chessboard)

    if (maximizing):
        best_score = -AI.INFINITE
        for move in chessboard.get_possible_moves(pieces.Piece.WHITE):
            copy = board.Board.clone(chessboard)
            copy.perform_move(move)

            best_score = max(best_score, AI.alphabeta(copy, depth-1
            , a, b, False))
            a = max(a, best_score)
            if (b <= a):
                break
        return best_score
    else:
        best_score = AI.INFINITE
        for move in chessboard.get_possible_moves(pieces.Piece.BLACK):
            copy = board.Board.clone(chessboard)
            copy.perform_move(move)

            best_score = min(best_score, AI.alphabeta(copy, depth-1
            , a, b, True))
            b = min(b, best_score)
            if (b <= a):
                break
        return best_score
```

Trong đó phần hàm lượng giá được implement như giải thích ở phần trên:

```
def evaluate(board):
    material = Heuristics.get_material_score(board)

    position = 0
    for x in range(8):
        for y in range(8):
            piece = board.chesspieces[x][y]
```

```
8               if piece != 0:
9                   if piece.piece_type == pieces.Pawn.PIECE_TYPE:
10                      table = Heuristics.PAWN_TABLE
11                  if piece.piece_type == pieces.Knight.PIECE_TYPE:
12                      table = Heuristics.KNIGHT_TABLE
13                  if piece.piece_type == pieces.Bishop.PIECE_TYPE:
14                      table = Heuristics.BISHOP_TABLE
15                  if piece.piece_type == pieces.Rook.PIECE_TYPE:
16                      table = Heuristics.ROOK_TABLE
17                  if piece.piece_type == pieces.Queen.PIECE_TYPE:
18                      table = Heuristics.QUEEN_TABLE
19                  if piece.piece_type == pieces.King.PIECE_TYPE:
20                      table = Heuristics.KING_TABLE
21                  position += Heuristics.get_piece_position_score(piece
22                  , table, x, y)
23      return material + position
24
25  def get_piece_position_score(piece, table, x, y):
26      if (piece.color == pieces.Piece.WHITE):
27          return table[x][y]
28      else:
29          return -(table[7 - x][y])
30
31  def get_material_score(board):
32      white = 0
33      black = 0
34      for x in range(8):
35          for y in range(8):
36              piece = board.chesspieces[x][y]
37              if (piece != 0):
38                  if (piece.color == pieces.Piece.WHITE):
39                      white += piece.value
40                  else:
41                      black += piece.value
42
43      return white - black
```

5 Source code và video thuyết trình

https://github.com/codeorafk/chess_AI
https://drive.google.com/drive/folders/1dF15HN32Jk0Lv-p1BkzE3vDzvjNkJVuB?usp=sharing