CS 380 Artificial Intelligence, Fall 2021 Sect: 002

Homework 6 - Adversarial Search, Part B: Complete

Yena Lee
yl3385
ID : 14531969

Programming Assignment

Pentago™ (75 points)

1) heuristic function

```
def yl3385_h(self, board):
    #-----------------------------------------------------------------------
    # Heuristic evaluation of board, presuming it is player's move.
    # Student code needed here.
    # Heuristic should not do further lookahead by calling miniMax.  This
    # function estimates the value of the board at a terminal node.
    #-----------------------------------------------------------------------

        for i in range(6):
            for j in range(6):
                if(board[i][j] == board[i][j+1] == "w"):
                    value += math.pow(2, horval)
                    horval+=1
                else:
                    if (board[i][j] == board[i][j+1] == "b"):
                        value -= math.pow(2, horval)
                        horval+=1
                if(board[j][i] == board[j+1][i] == "w"):
                    value += math.pow(2, verval)
                    verval+=1
                else:
                 if(board[j][i] == board[j+1][i] == "b"):
                        value -= math.pow(2, verval)
                        verval+=1
                if (board[i][j] == board[i+1][j+1] == "w"):
                    value += math.pow(2,fordiag)
                    fordiag+=1
                else:
                    if (board[i][j] == board[i+1][j+1] == "b"):
                        value -= math.pow(2,fordiag)
                        fordiag+=1
                if(board[i][5-j] == board[i+1][4-j] == "w"):
                    value += math.pow(2,revdiag)
                    revdiag+=1
                else:
                    if (board[i][5-j] == board[i+1][4-j] == "b"):
                        value -= math.pow(2,revdiag)
                        revdiag+=1

        return value
```

This code is what I made for my heuristic. I wanted to compute heuristic by noticing what

tokens are places on the board related to their neighbors. First, I wanted to analysis horizo ntal line. In horizontal line, if there is same token with it, I made the count of horizontal v alue, and made pow to add in total value for board states. If it was white token, I tried to add for high value, but if it was black token, I tried to subtract for low value. As same way, secondly , I checked in vertical line and if there are white tokens continuously in vertical line, value will be added and if its black tokens, value will be minus. Third way is in diagonal line and last way is in reverse of diagonal line. With all that calculation, this function return value of board states whenever it is high value or low value.

I expected this would be a meaningful value because it will have high possibility to have a good chance for win if the tokens are in a row. If the same color tokens are next to each other regardless of horizontal, vertical, diagonal lines, we are able to change the block of the grid, so it will make easier to reach for the win condition.

2) win(Board) function

```
def win(self,board): #complete
#------------------------------------------------------------------------------
# Determines if player has won, by finding '5 in a row'.
# Student code needed here.
#------------------------------------------------------------------------------

  pb = PentagoBoard()
  dx=[0,1,1,1]
  dy=[1,0,1,-1]
  for j in range(pb.BOARD_SIZE):
    for i in range(pb.BOARD_SIZE):
      if board[i][j]:
        for d in range(4):
          xmove=i
          ymove=j
          cnt = 1
          while (board[i][j] == board[xmove+dx[d]][ymove+dy[d]]):
            xmove+=dx[d]
            ymove+=dy[d]
            cnt+=1
          if cnt==5:
            xmove=j
            ymove=i
            if(board[i][j] == board[xmove+dx[d]][ymove+dy[d]]):
              break
          else:
            return True
  return False
```

This is a win function for judge the move is winning condition or not by returning value of True or False. Winning condition in Pentago is when there is a same color of five token in a row, it wins. So with this function, I tried to find five token in a row using direction (0,1), (1,0), (1,1), (1,-1). Inside the board size, check one of the token and another token next to it using for statement and direction. If there is same color token next to it, it add 1 in cnt variable, and when the cnt == 5, it returns true.

3) Minimax function

```
def miniMax(self, board, min, depth, maxDepth):
#-------------------------------------------------------------------------
# Use MiniMax algorithm to determine best move for player to make for given
# board.  Return the chosen move and the value of applying the heuristic to
# the board.
# To examine each of player's moves and evaluate them with no lookahead,
# maxDepth should be set to 1.  To examine each of the opponent's moves,
#  set maxDepth=2, etc.
# Increase depth by 1 on each recursive call to miniMax.
# min is the minimum value seen thus far by
#
# If a win is detected, the value returned should be INFINITY-depth.
# This rates 'one move wins' higher than 'two move wins,' etc.  This ensures
# that Player moves toward a win, rather than simply toward the assurance o
# a win.
#
# Student code needed here.
# Alpha-Beta pruning is recommended for Extra Credit.
# Argument list for this function may be altered as needed.
#
# successive calls to MiniMax should swap the self and opponent arguments.
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
# This code just picks a random move, and needs to be replaced.
#-------------------------------------------------------------------------
  moveList = board.getMoves()  # find all legal moves
  move = moveList[0]
  winning = win(board)
  newBoard = copy.deepcopy(board)
  newBoard = board.applyMove(move,self.token)
  value = self.yl3385_h(newBoard)
  INFINITY = 10000
  if depth == 0:
    return value
  if depth >= 0 and player==player[0]:
    maxnode = -INFINITY
    for child in moveList:
      child = child[0]
      value = max(value,miniMax(board,maxnode,depth-1,maxDepth))
    return value
  elif depth >= 0 and player==player[1]:
    minnode = INFINITY
    for child in moveList:
      child = child[0]
      value = min(value,miniMax(board,minnode,depth-1,maxDepth))
    return value
  # k = random.randrange(0,len(moveList))   # pick one at random
  # move = moveList[k]
  return move, value                # return move and backed-up value
```

This is the function of minimax to calculate the best move. I used minimax algorithm that I learned in AI class. With evaluating the value at each leaves, I tried to find the best move of the token's value. I used infinity = 10000, and -infinity to put the value in max node and min value. If depth is over zero, I tried to call this function again and again to update the value and find the best value.