Agenda

● Containment Diagrams
● Modeling the Board
● Checking for Sandwiches
● Minimax Algorithm
● Wrap-Up

Non CS Friend: "What's on the agenda today?"
Me:
Containment/Inheritance

Our Design:

- What is the Move class?
- MiniMax class vs method?
- Do we need a GameBoard class?
- Where should the 2D Array(s) go?
- Do we need a referee class?
Association

- What is the association between Game and Controls class?
- What about the Referee and Players?
  - Do the players need to know about each other?
- How do the different Board implementations impact association?

Minimize association as much as possible!
The Board

- How will you represent the board and the pieces?

- When will the board need to graphically AND logically change?
  - When any player adds a piece
  - When a sandwich is created & pieces change color
  - When possible moves are highlighted on a HumanPlayer’s turn

- When will the board NOT need to graphically change?
  - As AI makes recursive potential moves during MiniMax!
  - Instead, make a copy of the board that exists ONLY logically! How can we implement this?
    - Copy Constructor & an isVisible flag!!!
boolean checkSandwich(row, col, player):
    sandwich = false
    for all 8 directions around board[row][col]:
        flag = false
        while true:
            if board[row][col] is empty OR out of bounds:
                move to next direction
            if board[row][col] is opponent’s piece:
                update row and col to move 1 square in direction
                flag = true
            else if board[row][col] is own piece:
                if flag is true:
                    sandwich = true
                move to next direction

    return sandwich
MINIMAX!

Tic-Tac-Toe Demo

COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

Your move is given by the position of the largest red symbol on the grid. When your opponent picks a move, zoom in on the region of the grid where they went. Repeat.

MAP FOR X:
Move getBestMove(board, intelligence, current player):
  if game is over and current player wins:
    return high value move
  else if game is over and current player loses:
    return low value move
  else if game is over and it is a tie:
    return neutral move
  if there are no valid moves:
    if intelligence is 1:
      return low value move
    else:
      return -getBestMove(board, intelligence - 1, opp. player)
  for each valid move
    make move on copy board
    if intelligence is 1:
      value = evaluate copy board
    else:
      value = -getBestMove(copy board, intelligence - 1, opp. player)
  return best value move of valid moves
And That’s It!
Start early & **CODE INCREMENTALLY** !!!!!
You got this!!!!!