MATH 8650-FINAL PROJECT

Quarto Board Game: Computer Vs Human, Using Minimax Algorithm

Presented By:

Mihir Phatak
Netra Inamdar

Overview:

1. Introduction to problem
2. Strategy and approach for optimal play
3. Benchmarks
4. Results and Performance analysis

What is Quarto?

- Zero-sum 2 player game with perfect information
- Solved Game
- Copyrighted by Gigamic Games
- Piece Attributes:
- Short/Tall,
- Flat/Hollow,
- Black/White,
- Round/Square
- Opponent selects piece for the other player
- Condition For Win / Draw
- Copyrighted by Gigamic Games

Binarized Mapping

```
def renderMappingBinaryToString(self,BinaryString):
    '''
    Quarto Piece Dictionary :  Size, HollowFlat, Color, Shape
    s - small
    T - tall
    H - Hollow
    F - Flat
    B - Black
    W - White
    R - Round
    S - square
    '''
    MappingDictionary = { "0000" : 'sHBR',"1111":'TFWS',"0001" :
"sHBS","1101":"TFBS", "0010":"sHWR", "0100": "sFBR", "1000":"THBR",
"1100":"TFBR","0011":"sHWS","0111": "sFWS","1010":"THWR","0101":"sFBS","1001":
"THBS","0110":"sFWR","1110" : "TFWR","1011": "THWS"}
```

Strategy

1. Brute force approach:
   Disadvantage- Large no. of possible states ($16!^{2}$)

2. Implementation of Minimax Algorithm

3. Calculating Heuristic Evaluation Function

4. Using Alpha Beta Pruning for Minimax Optimization

CLEMSON UNIVERSITY

Approaches

- Implementation of Optimal Play
- Goal: Beat Brute Force Performance by Implementation of Backtracking
- Further Improve Backtracking by adding AlphaBeta Pruning

Classes

1) Class Quarto :

- def main()
- playUntilExit():
- def getFirstPlayer()
- def playQuarto(firstPlayer)
- def userTurn(state)
- def computerTurn(state)

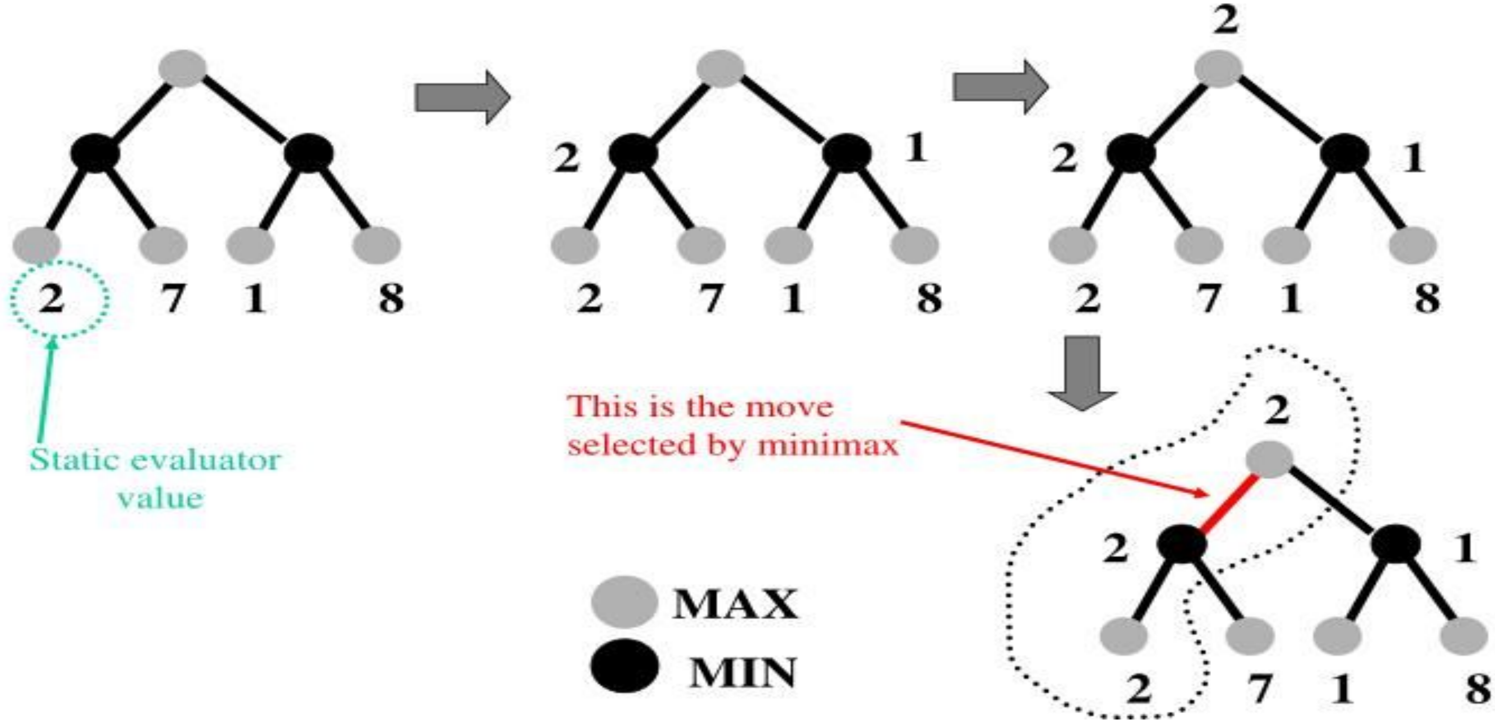
2) Class GameState :

- def place(self, cell):
- def getRemainingPieces(self):
- def formatPieces(self, pieces):
- def getUnoccupiedCells(self):
- def calculateNextMove(self):
- def isUnoccupied(self, cell):def isUnplaced(self, piece):
- def render(self):
- def renderAsBitPatterns(self, cell):
- def WinningCheckRows(self, board):
- def WinningCheckColumns(self, board):
- def WinningcheckLeftDiagonal(self, board):
- def WinningcheckRightDiagonal(self, board):
- def DrawGame(self):
- def gameOver(self):
- def renderMappingBinaryToString(self,BinaryString):

Alpha Beta Pruning Functions

- def alphaBetaPrune(state):
- def TreeSearchAlphaBeta(state, depth, alpha, beta, player):
- def PlayMove(move, state):
- def GetPossibleMoves(state):
- def GetRowCount(board, opponentPiece):
- def getDiagonalCount(board, opponentPiece):
- def MakeEstimateOfCost(state, player):

Minimax Algorithm

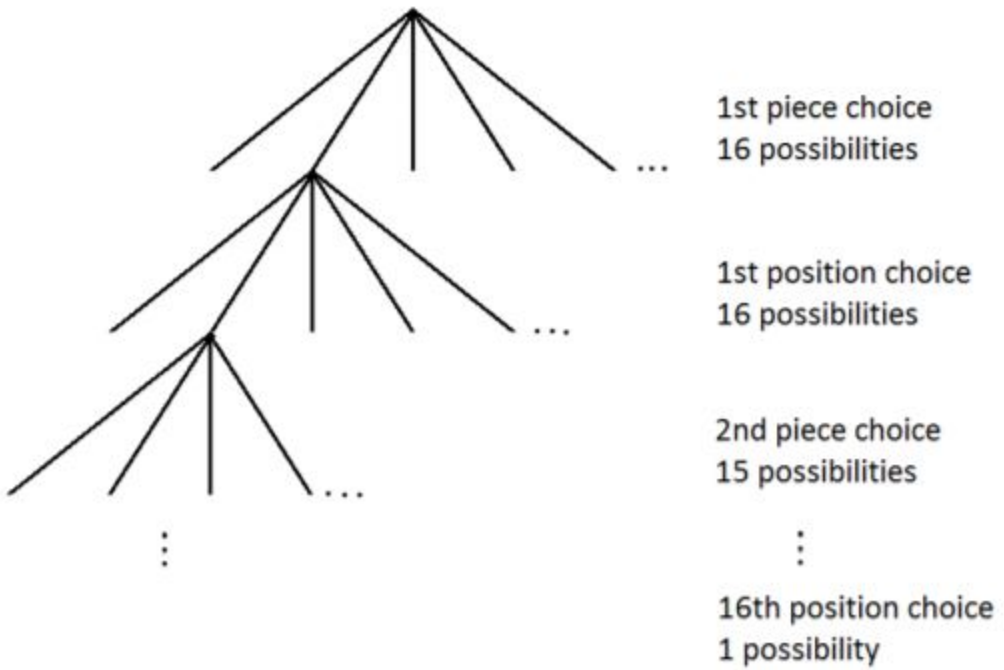


Source : https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html

Pseudocode DF search decision tree using Minimax:

```
function minimax(node, depth, Player)  is
     if depth = 0 or node is a terminal node then
          return the heuristic value of node
     if maximizing Player then
          value := −∞
          for each child of node do
               value := max(value, minimax(child, depth − 1, not Player))
          return value
     else (* minimizing player *)
          value := +∞
          for each child of node do
               value := min(value, minimax(child, depth − 1, TRUE))
            return value
```

Game tree of Quarto:



Source Credits : https://semanticscholar.com

Heuristic Evaluation Function

- Function costs for each move: Leaf nodes in the decision tree
- Gives values to non-final game states without considering all possible complete sequences
- No. of lines with 3 or 2 existing pieces on the board, of identical attributes are used for calculating the cost. Cost += 200, cost -= 50
- Possible Winning condition states on board- row, column and diagonal (4 + 4 +2 = 10 lines)

Heuristic Evaluation Function

- Score is assigned by checking the winning conditions as follows

```
def WinningCheckRows(self, board):
    for row in board:
        emptyCell = False
        commonOnes = empty
        commonZeroes = 0
        for cell in row:
            if cell == empty: emptyCell = True; break
            else:
  # If win, either commonOnes != 000 or commonZeros = bit string of all 1s

                commonOnes = commonOnes & cell
                commonZeroes = commonZeroes | cell
        if not emptyCell:
            if commonOnes > 0 or commonZeroes != ((1<<len(board))-1):
                return True
    return False
```

Drawbacks of Minimax Algorithm

- Increased complexity with more number of available states
- For ‘b’ legal moves and ‘m’ max-depth, time complexity with minimax algorithm- $O(b^m)$
- All branches of the decision sub-tree need to be traversed, even those not contributing to the final evaluation cost
- Enhancement to Minimax Algorithm: Alpha Beta Pruning

Alpha (œ) : minimal score that player **MAX** is guaranteed to attain.

Beta (ß) :maximum score that player **MAX** can hope to obtain against a sensible opponent.

Alpha Beta Pruning

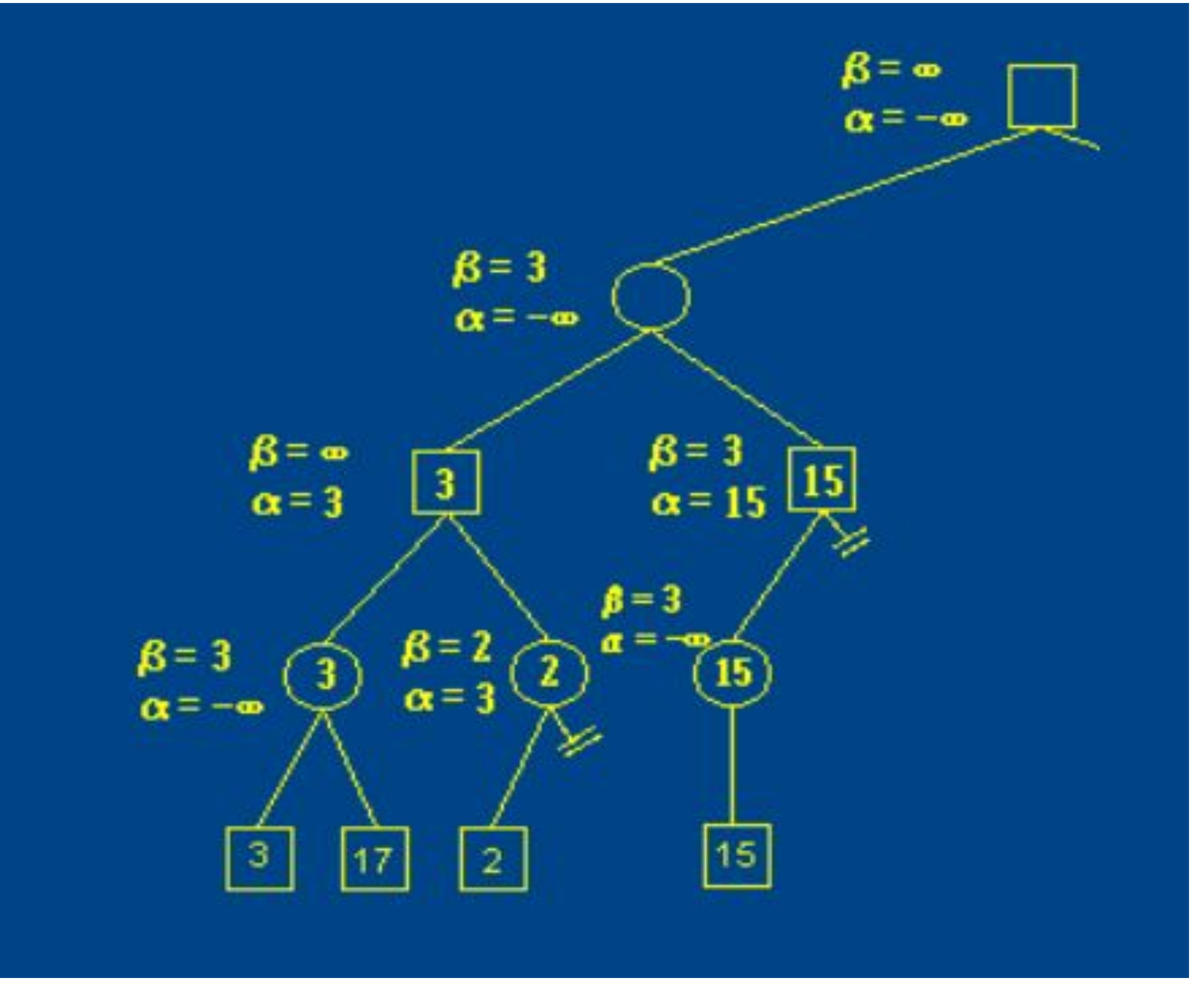
```
function alphaBeta(state, depth, alpha, beta, player):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = alphaBeta(state, depth, alpha, beta, player)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = alphaBeta(state, depth, alpha, beta, player)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

Alpha Beta Pruning Example



Picture Credits : http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html

Advantages of Alpha Beta Pruning

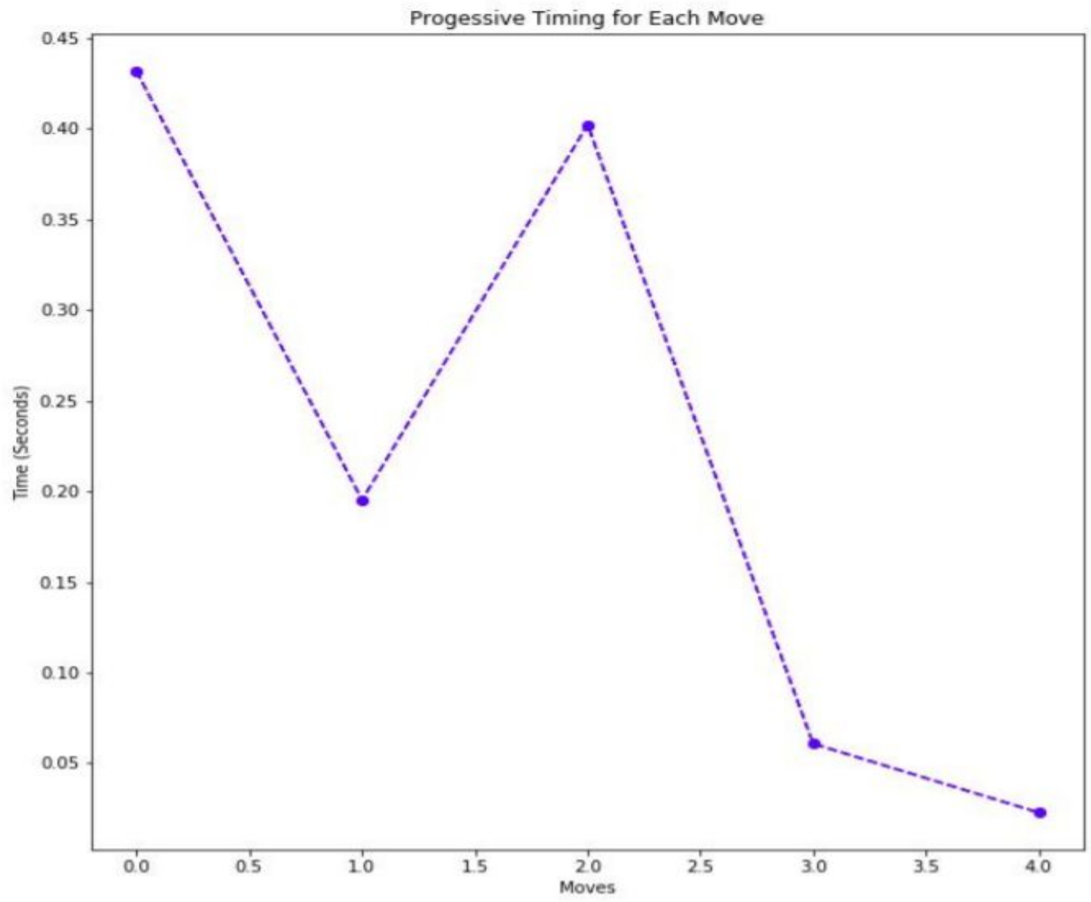
- With ‘b’ legal moves and ‘m’ max-depth, the time complexity of minimax algorithm reduces from $O(b^m)$ to $O(b^{m/2})$

- With increase in depth, optimal move can be found out in less number of steps as it prunes the states not affecting the final outcome

- Effective branching factor is $b^{1/2}$ instead of b.

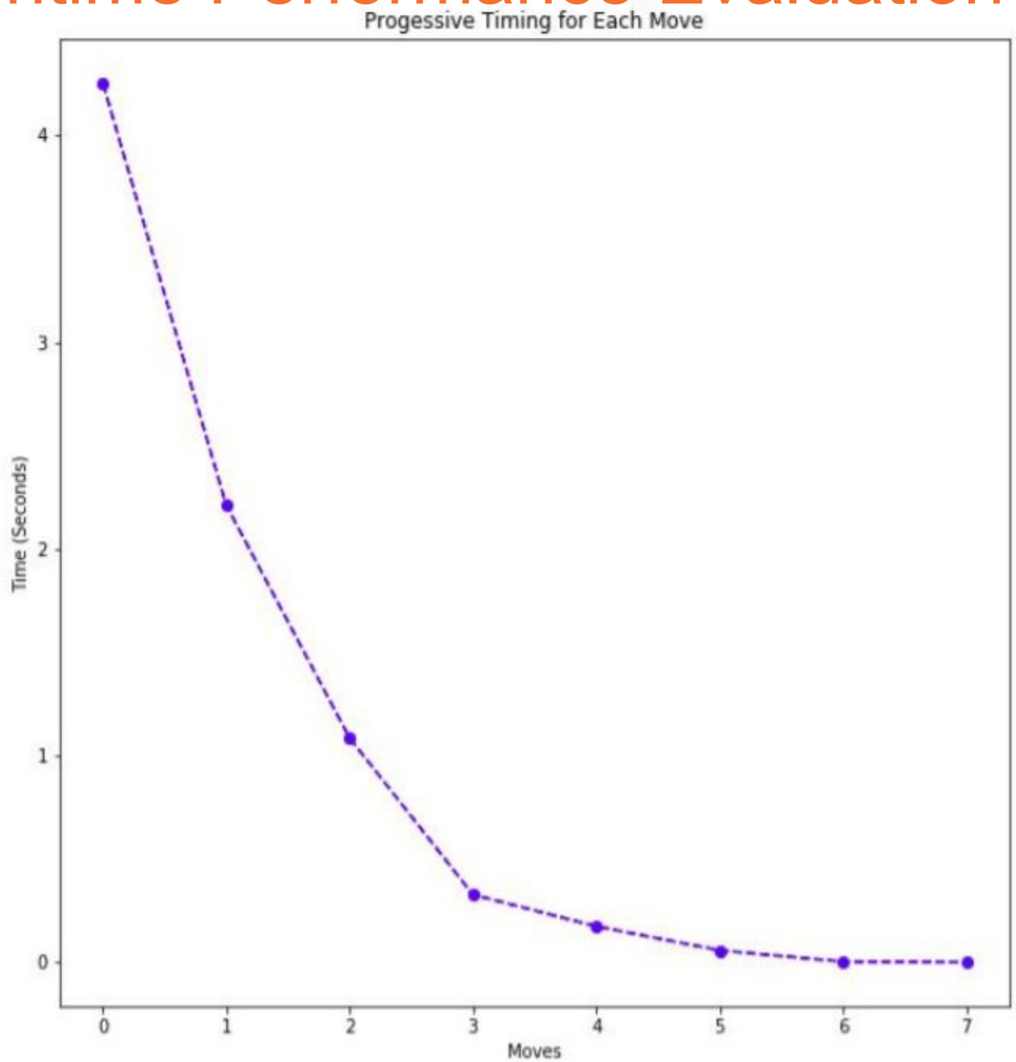
Computation Time Analysis with depth = 2

- Time vs Nth Move



Runtime Performance Evaluation

- Time vs nth Move
- Depth = 3



AI - Depth and Winnability Relationship

Sr. No.	Search Depth	Wins	Draws	Losses	Played Games
1	1	1	0	4	5
2	2	3	1	1	5
3	3	5	0	0	5

References

- Acknowledgement to Jochen Mohrmann, Michael Neumann, David Suendermann
  An Artificial Intelligence for the Board Game ‘Quarto!’ in Java

- https://en.wikipedia.org/wiki/Quarto
- http://suendermann.com/su/pdf/pppj2013.pdf

THANK YOU!