Artificial Intelligence

Practical 9

19BCE248

**AIM:** Implement Alpha-beta pruning

**Source Code:**

import java.awt.Point;

import java.util.ArrayList;

import java.util.Scanner;

public class Prac9 {

 public static final int INFINITY = 99999;

 public static final int MAX\_TRAVERSAL\_DEPTH = 6;

 Scanner scanner = new Scanner(System.in);

 int getIntInput()

 {

 int i = -1;

 try{

 i = scanner.nextInt();

 scanner.nextLine();

 }catch(Exception e){

 scanner.nextLine();

 }

 return i;

 }

 void clear()

 {

 for(int i = 0; i<25; i++)

 System.out.println();

 }

 public Prac9(){

 while(true){

 startGame();

 scanner.nextLine();

 }

 }

 void startGame()

 {

 Node root = new Node();

 clear();

 boolean playerMovesFirst = isPlayerMovingFirst(root);

 if(playerMovesFirst)

 {

 root.nextPlayer = "O";

 moveForPlayer(root);

 }

 else

 {

 root.nextPlayer = "X";

 moveForBot(root);

 }

 }

 boolean isPlayerMovingFirst(Node node)

 {

 printBoard(node);

 String input = "";

 do

 {

 System.out.println("Do you want to move first? [y/n] ['e' to exit]: ");

 input = scanner.nextLine();

 if(input.equalsIgnoreCase("e"))

 System.exit(0);

 }while(!input.equalsIgnoreCase("y") && !input.equalsIgnoreCase("n"));

 if(input.equalsIgnoreCase("y"))

 return true;

 return false;

 }

 void moveForPlayer(Node node)

 {

 Node newNode = new Node();

 ArrayList<Point> availableMoves = getAvailableMoves(node);

 Point input = getInputFromPlayer(node, availableMoves);

 newNode = getSuccessor(node, input);

 printBoard(newNode);

 System.out.println("Computer's turn, press enter to continue");

 if(checkWin(newNode))

 {

 System.out.println("You Won! Press enter to play again");

 }

 else if(isLeafNode(newNode))

 {

 System.out.println("Draw Game! Press enter to play again");

 }

 else

 {

 scanner.nextLine();

 moveForBot(newNode);

 }

 }

 Point getInputFromPlayer(Node node, ArrayList<Point> availableMoves)

 {

 Point move = new Point();

 do

 {

 int inputX, inputY;

 do

 {

 System.out.print("Input X: [0-2]: ");

 inputX = getIntInput();

 }while(inputX < 0 || inputX > 2);

 do

 {

 System.out.print("Input Y: [0-2]: ");

 inputY = getIntInput();

 }while(inputY < 0 || inputY > 2);

 move.x = inputX;

 move.y = inputY;

 if(!isValidMove(move, availableMoves))

 System.out.println("Your move is invalid!");

 }while(!isValidMove(move, availableMoves));

 return move;

 }

 boolean isLeafNode(Node node){

 return checkWin(node) || getAvailableMoves(node).size() == 0;

 }

 boolean checkWin(Node node){

 return checkWinRow(node) || checkWinColumn(node) || checkWinDiagonal(node);

 }

 boolean checkWinRow(Node node){

 String[][] board = node.board;

 return (board[0][0] != null && board[0][0] == board[0][1] && board[0][1] == board[0][2]) ||

 (board[1][0] != null && board[1][0] == board[1][1] && board[1][1] == board[1][2]) ||

 (board[2][0] != null && board[2][0] == board[2][1] && board[2][1] == board[2][2]);

 }

 boolean checkWinColumn(Node node){

 String[][] board = node.board;

 return (board[0][0] != null && board[0][0] == board[1][0] && board[1][0] == board[2][0]) ||

 (board[0][1] != null && board[0][1] == board[1][1] && board[1][1] == board[2][1]) ||

 (board[0][2] != null && board[0][2] == board[1][2] && board[1][2] == board[2][2]);

 }

 boolean checkWinDiagonal(Node node){

 String[][] board = node.board;

 return (board[0][0] != null && board[0][0] == board[1][1] && board[1][1] == board[2][2]) ||

 (board[0][2] != null && board[0][2] == board[1][1] && board[1][1] == board[2][0]);

 }

 boolean isValidMove(Point move, ArrayList<Point> availableMoves)

 {

 for(int i = 0; i<availableMoves.size(); i++)

 {

 Point curr = availableMoves.get(i);

 if(move.x == curr.x && move.y == curr.y)

 return true;

 }

 return false;

 }

 void moveForBot(Node node)

 {

 Node newNode = new Node();

 newNode.board = node.board;

 newNode.nextPlayer = "X";

 newNode = nextMove(newNode);

 printBoard(newNode);

 System.out.println("Your turn, press enter to continue");

 if(checkWin(newNode))

 {

 System.out.println("You lost! Press enter to play again");

 }

 else if(isLeafNode(newNode))

 {

 System.out.println("Draw Game! Press enter to play again");

 }

 else

 {

 scanner.nextLine();

 moveForPlayer(newNode);

 }

 }

 Node nextMove(Node node){

 getMiniMaxAlphaBeta(node, getAlpha(node), getBeta(node));

 Node newNode = getMaxNodeFromPossibleMoves();

 possibleNextMoves.clear();

 return newNode;

 }

 Node getMaxNodeFromPossibleMoves()

 {

 Node maxNode = possibleNextMoves.get(0);

 for(int i = 0, l = possibleNextMoves.size(); i<l; i++)

 {

 if(maxNode.heuristicValue < possibleNextMoves.get(i).heuristicValue)

 {

 maxNode = possibleNextMoves.get(i);

 }

 }

 return maxNode;

 }

 int getMiniMaxAlphaBeta(Node node, int alpha, int beta)

 {

 if(isLeafNode(node) || node.traversalDepth >= MAX\_TRAVERSAL\_DEPTH)

 return miniMaxLeafNode(node);

 else if(node.nextPlayer.equals("O"))

 return minimaxAlphaBetaForMinimizer(node, alpha, beta);

 else

 return minimaxAlphaBetaForMaximizer(node, alpha, beta);

 }

 int minimaxAlphaBetaForMinimizer(Node node, int alpha, int beta)

 {

 ArrayList<Node> allSuccessors = getAllSuccessors(node);

 for(int i = 0, l = allSuccessors.size(); i<l; i++)

 {

 Node s = allSuccessors.get(i);

 int currMin = getMiniMaxAlphaBeta(s, alpha, beta);

 beta = Math.min(beta, currMin);

 node.heuristicValue = Math.min(node.heuristicValue, beta);

 if(alpha >= beta)

 break;

 }

 if(possibleNextMoves(node) != null)

 possibleNextMoves.add(node);

 return beta;

 }

 int minimaxAlphaBetaForMaximizer(Node node, int alpha, int beta)

 {

 ArrayList<Node> allSuccessors = getAllSuccessors(node);

 for(int i = 0, l = allSuccessors.size(); i<l; i++)

 {

 Node s = allSuccessors.get(i);

 int currMax = getMiniMaxAlphaBeta(s, alpha, beta);

 alpha = Math.max(alpha, currMax);

 node.heuristicValue = Math.max(node.heuristicValue, alpha);

 if(alpha >= beta)

 break;

 }

 if(possibleNextMoves(node) != null)

 possibleNextMoves.add(node);

 return alpha;

 }

 int getAlpha(Node node)

 {

 if(isLeafNode(node))

 return evaluateHeuristicValue(node);

 return -INFINITY;

 }

 int getBeta(Node node)

 {

 if(isLeafNode(node))

 return evaluateHeuristicValue(node);

 return INFINITY;

 }

 ArrayList<Node> possibleNextMoves = new ArrayList<Node>();

 int miniMaxLeafNode(Node node)

 {

 if(possibleNextMoves(node) != null)

 possibleNextMoves.add(node);

 return evaluateHeuristicValue(node);

 }

 ArrayList<Node> getAllSuccessors(Node node)

 {

 ArrayList<Node> successors = new ArrayList<Node>();

 ArrayList<Point> availableMoves = getAvailableMoves(node);

 for(int i = 0, l = availableMoves.size(); i<l; i++)

 {

 successors.add(getSuccessor(node, availableMoves.get(i)));

 }

 return successors;

 }

 Node possibleNextMoves(Node node){

 if(node.atDepth == 1)

 return node;

 else

 return null;

 }

 ArrayList<Point> getAvailableMoves(Node node)

 {

 ArrayList<Point> availableMoves = new ArrayList<Point>();

 for(int i = 0; i<3; i++)

 {

 for(int j = 0; j<3; j++)

 {

 if(node.board[i][j] == null)

 {

 availableMoves.add(new Point(j, i));

 }

 }

 }

 return availableMoves;

 }

 Node getSuccessor(Node node, Point p) {

 if(isLeafNode(node)) return null;

 return new Node(updateBoard(node, p), node, evaluateHeuristicValue(node), node.atDepth+1, node.nextPlayer.equals("X") ? "O" : "X", node.traversalDepth+1);

 }

 int evaluateHeuristicValue(Node node)

 {

 if(node.nextPlayer == "X" && this.checkWin(node)==true) return -1;

 if(node.nextPlayer == "O" && this.checkWin(node)==true) return 1;

 return 0;

 }

 String[][] updateBoard(Node node, Point p) {

 String[][] newBoard = copyBoard(node.board);

 newBoard[p.y][p.x] = node.nextPlayer;

 return newBoard;

 }

 String[][] copyBoard(String[][] aBoard) {

 int boardSize = aBoard.length;

 String[][] newBoard = new String[boardSize][boardSize];

 for(int row = 0; row < boardSize; row++) {

 for(int column = 0; column < boardSize; column++)

 newBoard[row][column] = aBoard[row][column];

 }

 return newBoard;

 }

 void printBoard(Node node){

 String board[][] = node.board;

 clear();

 System.out.println("==========================");

 System.out.println("| Tic-Tac Toe |");

 System.out.println("==========================");

 System.out.println("-----------");

 System.out.println("|y\\x| 0 | 1 | 2 |");

 for(int i = 0; i<3; i++)

 {

 System.out.println("-----------");

 System.out.print("| " + i + " ");

 for(int j = 0; j<3; j++)

 {

 System.out.print("| ");

 if(board[i][j] != null)

 System.out.print(board[i][j]);

 else

 System.out.print(" ");

 System.out.print(" ");

 }

 System.out.println("|");

 }

 System.out.println("-----------");

 }

 public static void main(String[] args) {

 new Prac9();

 }

 public class Node {

 String[][] board;

 String nextPlayer;

 Node parent;

 int heuristicValue = 0;

 int atDepth = 0;

 int traversalDepth = 0;

 public Node(){

 board = new String[3][3];

 parent = null;

 }

 public Node(String[][] board, Node parent, int heuristicValue, int atDepth, String nextPlayer, int traversalDepth){

 this.board = board;

 this.parent = parent;

 this.heuristicValue = heuristicValue;

 this.atDepth = atDepth;

 this.nextPlayer = nextPlayer;

 this.traversalDepth = traversalDepth;

 }

}

}

**Output:**







