Assignment No. 4

**Name: Prasad Kute**

**Roll No: 331034**

**Div: TY-A1**



**Implementation of Min-Max Search Procedure with alpha beta pruning for finding the solutions of games**

**Aim:**

To implement the Min-Max search procedure with alpha-beta pruning to find the optimal solution for the Tic-Tac-Toe game.

**Objective:**

* To design and implement a game that allows a human player to play against an AI.
* To use the Min-Max algorithm with alpha-beta pruning to optimize the AI's decision-making process, ensuring faster and more efficient gameplay.
* To evaluate board positions and determine the best move for the AI to either maximize its chances of winning or minimize its losses.

**Theory:**

The Min-Max algorithm is a recursive procedure used in game theory to optimize the decision-making process for a player. It involves minimizing the possible loss for a worst-case scenario, assuming that the opponent also plays optimally.

In this project, the game of Tic-Tac-Toe is represented as a decision tree, where each move leads to new game states. The AI evaluates all possible moves using the Min-Max algorithm. However, evaluating every possible move can be computationally expensive. To address this, **alpha-beta pruning** is used. This technique "prunes" branches of the tree that cannot possibly influence the final decision, thus reducing the number of game states the AI needs to consider.

In the game:

* The **maximizing player** (AI) aims to maximize the evaluation score.
* The **minimizing player** (human) tries to minimize the score.
* Alpha-beta pruning ensures that unnecessary nodes (moves) are not evaluated, improving the efficiency of the algorithm.

**Code :**

import java.util.ArrayList;

import java.util.List;

import java.util.\*;

public class TicTacToe {

public static final int PLAYER\_X = 1;

public static final int PLAYER\_O = -1;

public static final int EMPTY = 0;

private int[][] board;

private int currentPlayer;

public TicTacToe() {

board = new int[3][3];

currentPlayer = PLAYER\_X;

}

public void printBoard() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

char symbol = ' ';

if (board[i][j] == PLAYER\_X) {

symbol = 'X';

} else if (board[i][j] == PLAYER\_O) {

symbol = 'O';

}

System.out.print(symbol + " ");

}

System.out.println();

}

}

public boolean isGameOver() {

// Check rows, columns, and diagonals for winning combinations

for (int i = 0; i < 3; i++) {

if (board[i][0] != EMPTY && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {

return true;

}

if (board[0][i] != EMPTY && board[0][i] == board[1][i] && board[1][i] == board[2][i]) {

return true;

}

}

if (board[0][0] != EMPTY && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {

return true;

}

if (board[0][2] != EMPTY && board[0][2] == board[1][1] && board[1][1] == board[2][0]) {

return true;

}

// Check for tie

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == EMPTY) {

return false; // There are still empty spaces

}

}

}

return true; // The board is full, it's a tie

}

public int evaluateBoard() {

if (isGameOver()) {

if (checkWin(PLAYER\_X)) {

return 10; // Player X wins

} else if (checkWin(PLAYER\_O)) {

return -10; // Player O wins

} else {

return 0; // Tie

}

}

return 0; // Evaluation for non-terminal states

}

public boolean checkWin(int player) {

for (int i = 0; i < 3; i++) {

if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {

return true;

}

if (board[0][i] == player && board[1][i] == player && board[2][i] == player) {

return true;

}

}

if (board[0][0] == player && board[1][1] == player && board[2][2] == player) {

return true;

}

if (board[0][2] == player && board[1][1] == player && board[2][0] == player) {

return true;

}

return false;

}

public void makeMove(int row, int col, int player) {

board[row][col] = player;

currentPlayer = -currentPlayer;

}

public void undoMove(int row, int col) {

board[row][col] = EMPTY;

currentPlayer = -currentPlayer;

}

public int findBestMove() {

int bestValue = Integer.MIN\_VALUE;

int bestRow = -1;

int bestCol = -1;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == EMPTY) {

makeMove(i, j, currentPlayer);

int value = minimax(0, Integer.MIN\_VALUE, Integer.MAX\_VALUE, false);

undoMove(i, j);

if (value > bestValue) {

bestValue = value;

bestRow = i;

bestCol = j;

}

}

}

}

return bestRow \* 3 + bestCol;

}

public int minimax(int depth, int alpha, int beta, boolean maximizingPlayer) {

if (isGameOver()) {

return evaluateBoard();

}

if (maximizingPlayer) {

int bestValue = Integer.MIN\_VALUE;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == EMPTY) {

makeMove(i, j, currentPlayer);

int value = minimax(depth + 1, alpha, beta, !maximizingPlayer);

undoMove(i, j);

bestValue = Math.max(bestValue, value);

alpha = Math.max(alpha, bestValue);

if (beta <= alpha) {

break; // Alpha-beta pruning

}

}

}

}

return bestValue;

} else {

int bestValue = Integer.MAX\_VALUE;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == EMPTY) {

makeMove(i, j, currentPlayer);

int value = minimax(depth + 1, alpha, beta, !maximizingPlayer);

undoMove(i, j);

bestValue = Math.min(bestValue, value);

beta = Math.min(beta, bestValue);

if (beta <= alpha) {

break; // Alpha-beta pruning

}

}

}

}

return bestValue;

}

}

public static void main(String[] args) {

TicTacToe game = new TicTacToe();

Scanner scanner = new Scanner(System.in);

while (!game.isGameOver()) {

game.printBoard();

if (game.currentPlayer == PLAYER\_X) {

System.out.println("Player X's turn. Enter row and column (e.g., 0 1):");

int row = scanner.nextInt();

int col = scanner.nextInt();

game.makeMove(row, col, PLAYER\_X);

} else {

System.out.println("Player O's turn (AI):");

int bestMove = game.findBestMove();

int row = bestMove / 3;

int col = bestMove % 3;

game.makeMove(row, col, PLAYER\_O);

}

}

game.printBoard();

if (game.checkWin(PLAYER\_X)) {

System.out.println("Player X wins!");

} else if (game.checkWin(PLAYER\_O)) {

System.out.println("Player O wins!");

} else {

System.out.println("It's a tie!");

}

scanner.close();

}

}

**output:**

c207-10@c20710:~/Desktop$ javac TicTacToe.java

c207-10@c20710:~/Desktop$ java TicTacToe

Player X's turn. Enter row and column (e.g., 0 1):

0 0

X

Player O's turn (AI):

X

O

Player X's turn. Enter row and column (e.g., 0 1):

0 1

X X

O

Player O's turn (AI):

X X

O

O

Player X's turn. Enter row and column (e.g., 0 1):

0 2

X X X

O

O

Player X wins!

**Conclusion**:

The implementation successfully demonstrates the use of the Min-Max algorithm with alpha-beta pruning in a simple game like Tic-Tac-Toe. The AI, which uses this algorithm, can make optimal decisions, minimizing its losses and maximizing its chances of winning. The pruning technique significantly reduces the number of game states evaluated, improving the computational efficiency of the AI. This approach can be applied to more complex games and scenarios where decision-making plays a crucial role.