ARTIFICIAL INTELLIGENCE

ROLL NO. : 18BCE191
DATE: 15/11/2021

Aim

Implement tic-tac-toe using minimax algorithm with alpha-beta pruning

Code

```
#include<bits/stdc++.h>

#define WIN 1000
#define DRAW 0
#define LOSS -1000

#define AI_MARKER 'O'
#define PLAYER_MARKER 'X'
#define EMPTY_SPACE '-'

#define START_DEPTH 0

using namespace std;

// Print game state
void print_game_state(int state)
{
   if (WIN == state) { cout << "WIN" << endl; }
   else if (DRAW == state) { cout << "DRAW" << endl; }
   else if (LOSS == state) { cout << "LOSS" << endl; }
}

// All possible winning states
vector<vector<pair<int, int>>> winning_states
{
   // Every row
```

```
    { {0, 0}, {0, 1}, {0, 2} },
    { {1, 0}, {1, 1}, {1, 2} },
    { {2, 0}, {2, 1}, {2, 2} },

    // Every column
    { {0, 0}, {1, 0}, {2, 0} },
    { {0, 1}, {1, 1}, {2, 1} },
    { {0, 2}, {1, 2}, {2, 2} },

    // Every diagonal
    { {0, 0}, {1, 1}, {2, 2} },
    { {2, 0}, {1, 1}, {0, 2} }

};

// Print the current board state
void print_board(char board[3][3])
{
    cout << endl;
    cout << board[0][0] << "\t| " << board[0][1] << "\t| " <<
board[0][2] << endl;
    cout << "-------------------------" << endl;
    cout << board[1][0] << "\t| " << board[1][1] << "\t| " <<
board[1][2] << endl;
    cout << "-------------------------" << endl;
    cout << board[2][0] << "\t| " << board[2][1] << "\t| " <<
board[2][2] << endl << endl;
}

// Get all available legal moves (spaces that are not occupied)
vector<pair<int, int>> get_legal_moves(char board[3][3])
{
    vector<pair<int, int>> legal_moves;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (board[i][j] != AI_MARKER && board[i][j] !=
PLAYER_MARKER)
            {
```

```
                legal_moves.push_back({i, j});
            }
        }
    }

    return legal_moves;
}

// Check if a position is occupied
bool position_occupied(char board[3][3], pair<int, int> pos)
{
    vector<pair<int, int>> legal_moves = get_legal_moves(board);

    for (int i = 0; i < legal_moves.size(); i++)
    {
        if (pos.first == legal_moves[i].first && pos.second ==
legal_moves[i].second)
        {
            return false;
        }
    }

    return true;
}

// Get all board positions occupied by the given marker
vector<pair<int, int>> get_occupied_positions(char board[3][3], char
marker)
{
    vector<pair<int, int>> occupied_positions;

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (marker == board[i][j])
            {
                occupied_positions.push_back({i, j});
            }
        }
```

```
    }

    return occupied_positions;
}

// Check if the board is full
bool board_is_full(char board[3][3])
{
    vector<pair<int, int>> legal_moves = get_legal_moves(board);

    if (0 == legal_moves.size())
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Check if the game has been won
bool game_is_won(vector<pair<int, int>> occupied_positions)
{
    bool game_won;

    for (int i = 0; i < winning_states.size(); i++)
    {
        game_won = true;
        vector<pair<int, int>> curr_win_state = winning_states[i];
        for (int j = 0; j < 3; j++)
        {
            if (!(find(begin(occupied_positions),
end(occupied_positions), curr_win_state[j]) !=
end(occupied_positions)))
            {
                game_won = false;
                break;
            }
        }
```

```
        if (game_won)
        {
            break;
        }
    }
    return game_won;
}

char get_opponent_marker(char marker)
{
    char opponent_marker;
    if (marker == PLAYER_MARKER)
    {
        opponent_marker = AI_MARKER;
    }
    else
    {
        opponent_marker = PLAYER_MARKER;
    }

    return opponent_marker;
}

// Check if someone has won or lost
int get_board_state(char board[3][3], char marker)
{

    char opponent_marker = get_opponent_marker(marker);

    vector<pair<int, int>> occupied_positions =
get_occupied_positions(board, marker);

    bool is_won = game_is_won(occupied_positions);

    if (is_won)
    {
        return WIN;
    }
```

```
    occupied_positions = get_occupied_positions(board,
opponent_marker);
    bool is_lost = game_is_won(occupied_positions);

    if (is_lost)
    {
        return LOSS;
    }

    bool is_full = board_is_full(board);
    if (is_full)
    {
        return DRAW;
    }

    return DRAW;

}

// Apply the minimax game optimization algorithm
pair<int, pair<int, int>> minimax_optimization(char board[3][3], char
marker, int depth, int alpha, int beta)
{
    // Initialize best move
    pair<int, int> best_move = {-1,-1};
    int best_score = (marker == AI_MARKER) ? LOSS : WIN;

    // If we hit a terminal state (leaf node), return the best score
and move
    if (board_is_full(board) || DRAW != get_board_state(board,
AI_MARKER))
    {
        best_score = get_board_state(board, AI_MARKER);
        return {best_score, best_move};
    }

    vector<pair<int, int>> legal_moves = get_legal_moves(board);

    for (int i = 0; i < legal_moves.size(); i++)
    {
```

```
        pair<int, int> curr_move = legal_moves[i];
        board[curr_move.first][curr_move.second] = marker;

        // Maximizing player's turn
        if (marker == AI_MARKER)
        {
            int score = minimax_optimization(board, PLAYER_MARKER,
depth + 1, alpha, beta).first;

            // Get the best scoring move
            if (best_score < score)
            {
                best_score = score - depth * 10;
                best_move = curr_move;

                // Check if this branch's best move is worse than the
best
                // option of a previously search branch. If it is, skip
it
                alpha = max(alpha, best_score);
                board[curr_move.first][curr_move.second] = EMPTY_SPACE;
                if (beta <= alpha)
                {
                    break;
                }
            }

        } // Minimizing opponent's turn
        else
        {
            int score = minimax_optimization(board, AI_MARKER, depth +
1, alpha, beta).first;

            if (best_score > score)
            {
                best_score = score + depth * 10;
                best_move = curr_move;

                // Check if this branch's best move is worse than the
best
```

```
                // option of a previously search branch. If it is, skip
it
                beta = min(beta, best_score);
                board[curr_move.first][curr_move.second] = EMPTY_SPACE;
                if (beta <= alpha)
                {
                    break;
                }
            }

        }

        board[curr_move.first][curr_move.second] = EMPTY_SPACE; // Undo
move

    }

    return {best_score, best_move};
}

// Check if the game is finished
bool game_is_done(char board[3][3])
{
    if (board_is_full(board))
    {
        return true;
    }

    if (DRAW != get_board_state(board, AI_MARKER))
    {
        return true;
    }

    return false;
}


int main()
{
    char board[3][3] = { EMPTY_SPACE };
```

```
    cout << "Player = X\t AI Computer = O" << endl << endl;

    print_board(board);

    while (!game_is_done(board))
    {
        int row, col;
        cout << "Enter coordintates: ";
        cin >> row >> col;
        cout << "\n\n";

        if (position_occupied(board, {row,col}))
        {
            cout << "The position (" << row << ", " << col << ") is
occupied. Try another one..." << endl;
            continue;
        }
        else
        {
            board[row][col] = PLAYER_MARKER;
        }

        pair<int, pair<int, int>> ai_move = minimax_optimization(board,
AI_MARKER, START_DEPTH, LOSS, WIN);

        board[ai_move.second.first][ai_move.second.second] = AI_MARKER;

        print_board(board);
    }

    cout << "********** GAME OVER **********" << endl << endl;

    int player_state = get_board_state(board, PLAYER_MARKER);

    cout << "PLAYER "; print_game_state(player_state);

    return 0;

}
```

Output

```
(base) rajat@rajat-VivoBook-S14-X430UA:/Rajat1/Books/Artificial Intelligence/Practicals$ cd "/Rajat1/Books/Artificial Intelligence/Practicals/" && g++ alphaBetaTicTacToe.cpp -o alphaBetaTicTacToe && "/Rajat1/Books/Artificial Intelligence/Practicals/"alphaBetaTicTacToe
Player = X       AI Computer = O


-       |       |
----------------------------
        |       |
----------------------------
        |       |

Enter coordintates: 1 1



O       | -     | -
----------------------------
-       | X     | -
----------------------------
-       | -     | -

Enter coordintates: 0 2



O       | -     | X
----------------------------
-       | X     | -
----------------------------
O       | -     | -

Enter coordintates: 1 0
```

```
O       | -     | X
----------------------------
X       | X     | O
----------------------------
O       | -     | -

Enter coordintates: 2 1



O       | O     | X
----------------------------
X       | X     | O
----------------------------
O       | X     | -

Enter coordintates: 2 2



O       | O     | X
----------------------------
X       | X     | O
----------------------------
O       | X     | X

********** GAME OVER **********

PLAYER DRAW
(base) rajat@rajat-VivoBook-S14-X430UA:/Rajat1/Books/Artificial Intelligence/Practicals$
```

```
(base) rajat@rajat-VivoBook-S14-X430UA:/Rajat1/Books/Artificial Intelligence/Practicals$ cd "/Rajat1/Books/Artificial Intelligence/Prac
ticals/" && g++ alphaBetaTicTacToe.cpp -o alphaBetaTicTacToe && "/Rajat1/Books/Artificial Intelligence/Practicals/"alphaBetaTicTacToe
Player = X       AI Computer = O


-       |       |
-------------------------
        |       |
-------------------------
        |       |

Enter coordintates: 0 0



X       | -     | -
-------------------------
-       | O     | -
-------------------------
-       | -     | -

Enter coordintates: 1 0



X       | -     | -
-------------------------
X       | O     | -
-------------------------
O       | -     | -

Enter coordintates: 0 1
```

```
 X       | X     | O
 -------------------------
 X       | O     | -
 -------------------------
 O       | -     | -

 ********** GAME OVER **********

 PLAYER LOSS
 (base) rajat@rajat-VivoBook-S14-X430UA:/Rajat1/Books/Artificial Intelligence/Practicals$
```