**Batch: C - 3 Roll No.: 16010122096,**

**16010122109**

**Experiment / assignment / tutorial No: 06**

|  |
| --- |
| **Title:** Implementation of Adversarial Search Algorithm |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Implementation of Alpha-Beta Pruning algorithm

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO 2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf**](http://www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf)
4. [**http://cs.lmu.edu/~ray/notes/asearch/**](http://cs.lmu.edu/~ray/notes/asearch/)
5. **www.cs.cornell.edu/courses/cs4700/2011fa/.../06\_adversarialsearch.pdf**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Two/Multi player Games and rules, state-space tree, searching algorithms and their analysis properties

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile: -** The game playing has been integral part of human life. The multiplayer games are competitive environment in which everyone tries to gain more points for himself and wishes the opponent to gain minimum.

The game can be represented in form of a state space tree and one can follow the path from root to some goal node, for either of the player.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Adversarial search, min-max algorithm, Alpha-Beta pruning,

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Adversarial Search Concept:-**

Adversarial search is a framework used in decision-making problems where agents compete in a zero-sum environment, such as in two-player board games. In these problems, each agent’s goal is to maximize its own advantage while minimizing that of its opponent. The adversarial search technique models the problem as a game tree, where nodes represent game states and edges represent possible moves.

The Alpha-Beta pruning algorithm is an optimization of the minimax algorithm. While minimax explores the entire game tree to determine the optimal move, Alpha-Beta pruning reduces the number of nodes evaluated by eliminating branches that cannot influence the final decision. It introduces two parameters: alpha (the best value found so far along the path to the maximizer) and beta (the best value found so far along the path to the minimizer). When the algorithm finds that a branch cannot possibly affect the outcome, it prunes (or cuts off) that branch, thus saving computation time without affecting the final result.

Key points of the theory include:

• **Game Tree and Minimax Principle:**

* The game tree is built from possible moves.
* The minimax algorithm evaluates terminal nodes (win, loss, or draw) and propagates scores upward to choose the move that maximizes the minimum gain (worst-case scenario).

**Alpha-beta pruning Algorithm:**

* **Alpha (α):** Represents the lower bound on the maximizer’s score.
* **Beta (β):** Represents the upper bound on the minimizer’s score.
* As the tree is traversed, if the algorithm finds that the current branch cannot produce a better outcome than a previously examined branch, it prunes the branch, avoiding unnecessary computations.

• **Benefits and Efficiency:**

* The pruning process can dramatically decrease the number of nodes that must be evaluated, especially in deeper trees.
* In the best case, Alpha-Beta pruning can reduce the effective branching factor, allowing the algorithm to search twice as deep as the standard minimax algorithm in the same amount of time.

**Chosen Problem:**

The chosen problem involves implementing an adversarial search algorithm by applying the Alpha-Beta pruning technique to a game scenario. The primary focus is to create a system that evaluates different game states—such as those found in Tic Tac Toe—and efficiently identifies the optimal move by exploring the game tree. This approach not only requires a proper assessment of each state but also the elimination of branches that will not affect the final decision, ensuring a more streamlined and effective decision-making process.

**Goals:**

* Demonstrate the application of adversarial search in a competitive, zero-sum environment.
* Implement the Alpha-Beta pruning algorithm to optimize the minimax approach by cutting off unnecessary branches during evaluation.
* Improve computational efficiency in decision-making by reducing the number of nodes explored in the game tree.
* Provide practical experience in goal-based agent architecture, aligning with the course outcome of analyzing and solving search and planning problems in AI.

**CODE:**

#include <iostream>

#include <vector>

#include <limits>

using namespace std;

#define X\_VAL 1

#define O\_VAL -1

#define BLANK 0

int computeScore(vector<vector<int>> &grid) {

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

if (grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2] && grid[i][0] != BLANK)

return grid[i][0];

if (grid[0][i] == grid[1][i] && grid[1][i] == grid[2][i] && grid[0][i] != BLANK)

return grid[0][i];

}

if (grid[0][0] == grid[1][1] && grid[1][1] == grid[2][2] && grid[0][0] != BLANK)

return grid[0][0];

if (grid[0][2] == grid[1][1] && grid[1][1] == grid[2][0] && grid[0][2] != BLANK)

return grid[0][2];

return 0;

}

bool availableMoves(vector<vector<int>> &grid) {

for (auto &row : grid)

for (auto cell : row)

if (cell == BLANK) return true;

return false;

}

void displayGrid(vector<vector<int>> &grid, int level, int alpha, int beta) {

cout << "\nLevel " << level << " : Alpha = " << alpha << ", Beta = " << beta << "\n";

for (auto &row : grid) {

for (auto cell : row)

cout << (cell == X\_VAL ? "X " : cell == O\_VAL ? "O " : ". ");

cout << "\n";

}

cout << "----------------\n";

}

int searchMinimax(vector<vector<int>> &grid, int depth, bool maximize, int alpha, int beta) {

int score = computeScore(grid);

if (score == X\_VAL) return 10 - depth;

if (score == O\_VAL) return depth - 10;

if (!availableMoves(grid)) return 0;

displayGrid(grid, depth, alpha, beta);

if (maximize) {

int best = numeric\_limits<int>::min();

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

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

if (grid[i][j] == BLANK) {

grid[i][j] = X\_VAL;

best = max(best, searchMinimax(grid, depth + 1, false, alpha, beta));

grid[i][j] = BLANK;

alpha = max(alpha, best);

if (beta <= alpha) {

cout << "Cutoff at level " << depth << " : Alpha = " << alpha << ", Beta = " << beta << "\n";

return best;

}

}

return best;

} else {

int best = numeric\_limits<int>::max();

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

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

if (grid[i][j] == BLANK) {

grid[i][j] = O\_VAL;

best = min(best, searchMinimax(grid, depth + 1, true, alpha, beta));

grid[i][j] = BLANK;

beta = min(beta, best);

if (beta <= alpha) {

cout << "Cutoff at level " << depth << " : Alpha = " << alpha << ", Beta = " << beta << "\n";

return best;

}

}

return best;

}

}

pair<int, int> optimalMove(vector<vector<int>> &grid) {

int bestScore = numeric\_limits<int>::min();

pair<int, int> bestMove = {-1, -1};

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

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

if (grid[i][j] == BLANK) {

grid[i][j] = X\_VAL;

int moveScore = searchMinimax(grid, 0, false, numeric\_limits<int>::min(), numeric\_limits<int>::max());

grid[i][j] = BLANK;

if (moveScore > bestScore) {

bestMove = {i, j};

bestScore = moveScore;

}

}

return bestMove;

}

int main() {

vector<vector<int>> grid = {

{X\_VAL, O\_VAL, BLANK},

{O\_VAL, X\_VAL, BLANK},

{BLANK, BLANK, BLANK}

};

displayGrid(grid, 0, numeric\_limits<int>::min(), numeric\_limits<int>::max());

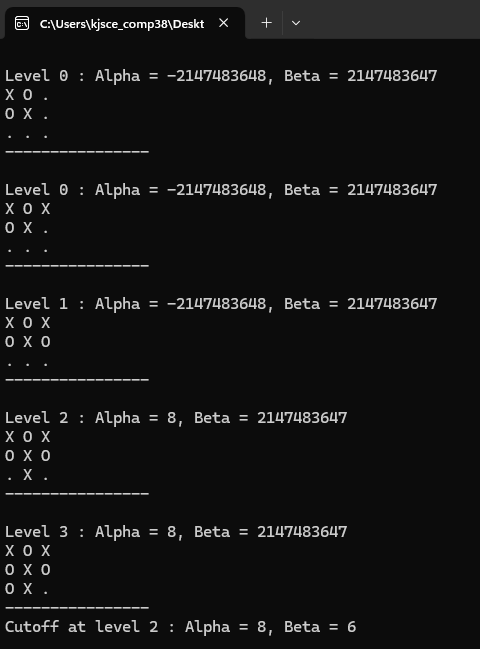
pair<int, int> move = optimalMove(grid);

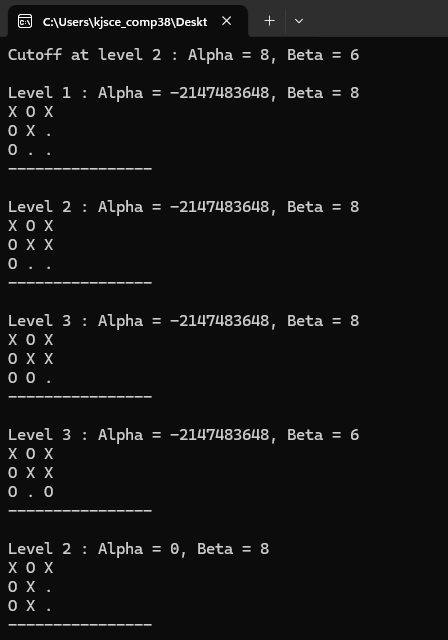
cout << "\nOptimal move for X is at position (" << move.first << ", " << move.second << ")\n";

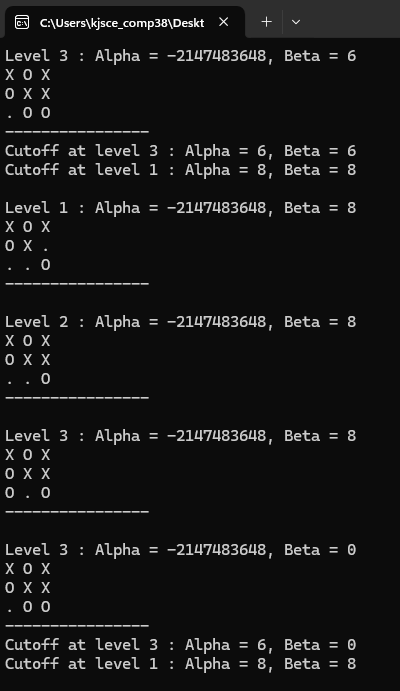
return 0;

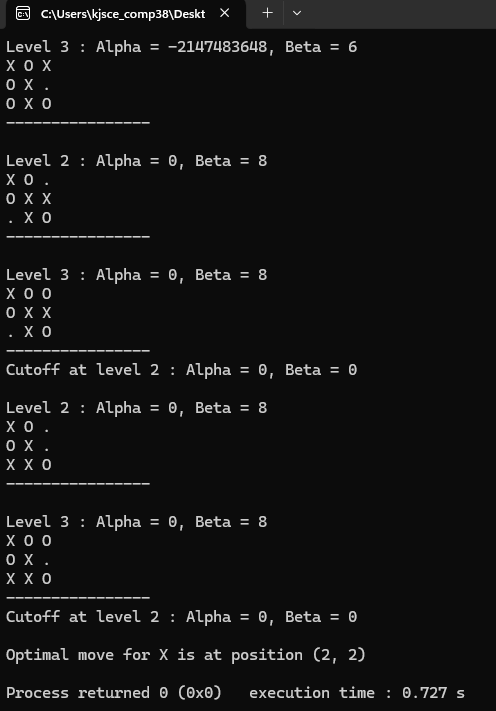
}

**OUTPUT:**

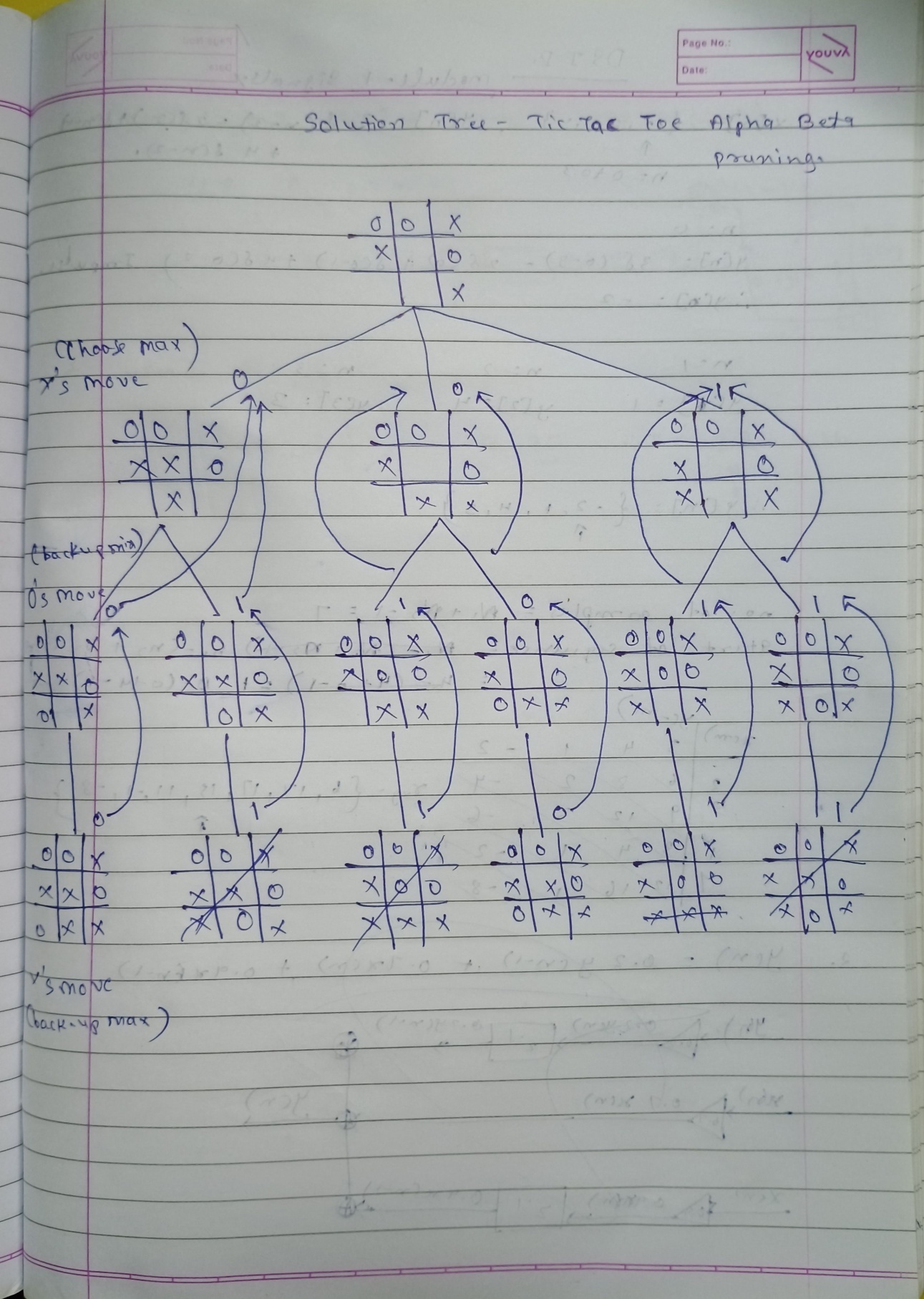
****

****

****

****

**Solution tree for chosen Problem:**



**Post Lab objective Questions:**

1. **Which search is equal to minmax search but eliminates the branches that can’t influence the final decision?**
   1. Breadth-first search
   2. Depth first search
   3. Alpha-beta pruning
   4. None of the above
2. **Which values are independent in minmax search alogirthm?**
   1. Pruned leaves x and y
   2. Every states are dependant
   3. Root is independent
   4. None of the above

**Post Lab Subjective Questions:**

1. What is the main purpose of the Alpha-Beta pruning algorithm in game trees?

The main purpose of the Alpha-Beta pruning algorithm in game trees is to reduce the number of nodes that need to be evaluated by the minimax algorithm, thereby speeding up the decision-making process in adversarial games.

1. How does Alpha-Beta pruning improve the efficiency of the minmax algorithm?

Alpha-Beta pruning improves the efficiency of the minimax algorithm by eliminating branches that cannot possibly affect the final decision. This means that when a branch is identified as being worse than a previously examined move, it is pruned, avoiding unnecessary evaluations.

1. Explain the terms **alpha** and **beta** in the context of the Alpha-Beta pruning algorithm.

In the context of the Alpha-Beta pruning algorithm, alpha represents the best (maximum) score that the maximizer can guarantee at that point, while beta represents the best (minimum) score that the minimizer can guarantee. These values are used to decide whether further exploration of a branch is necessary.

1. What condition must be met for a node to be pruned during Alpha-Beta pruning?

A node is pruned during Alpha-Beta pruning when the condition alpha ≥ beta is met. This indicates that further exploration of that branch cannot yield a better outcome than the already known possibilities.

1. Compare and contrast Alpha-Beta pruning with the standard Minmax algorithm.

Alpha-Beta pruning and the standard minimax algorithm both aim to determine the optimal move in a game tree. However, while minimax evaluates every possible move, Alpha-Beta pruning