**FIRST ROUND -FINAL CODE ASSESMENT DOCUMENT**

**TECHNICAL ASSESSMENT**

**Problem Statement:**

Perform Alpha-beta pruning search on random100 number.The brut force method will be accepted.Neat Documentation is expected with from scratch implementation with c++.

**Alpha-beta pruning Search:**

Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. **We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time.** Complexity in ideal ordering is O(bm/2).

Let’s define the parameters alpha and beta.  
**Alpha** is the best value that the **maximizer** currently can guarantee at that level or above.  
**Beta** is the best value that the **minimizer** currently can guarantee at that level or above.

**Algorithm:**

**function** alphabeta(node, depth, α, β, Player)

• **if** depth = 0 **or** node is a terminal node

• **return** the heuristic value of node

• **if** Player = MaxPlayer

• **for each** child of node

• α := max(α, alphabeta(child, depth-1, α, β, not(Player) ))

• **if** β ≤ α

• **break** ; Prune

• **return** α

• **else**

• **for each** child of node

• β := min(β, alphabeta(child, depth-1, α, β, not(Player) ))

• **if** β ≤ α **break** ; Prune

• **return** β

• **end**

• Initial call

• alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)

**Logic**:

* First of all, we are generating 100 random numbers using while loop

cout<<"The random numbers are:";

while(i++ < 100) {

int values = (rand() % 100) + 1;

cout << values << " ";

* While calling this function

minimax(0, 0, true, values, MIN, MAX)

* This part of code is executed

int minimax(int depth, int nodeIndex, bool maximizingPlayer, int values[], int alpha, int beta)

{

if (depth == 2)

return values[nodeIndex];

}

To Returns optimal value for current player(Initially called for root and maximizer)

* This part of code is for traversing the values.

if (depth == 2)

return values[nodeIndex];

if (maximizingPlayer)

{

int best = MIN;

// Recur for left and

// right children

for (int i = 0; i < 2; i++)

{

int val = minimax(depth + 1, nodeIndex \* 2 + i,

false, values, alpha, beta);

best = max(best, val);

alpha = max(alpha, best);

// Alpha Beta Pruning

if (beta <= alpha)

break;

}

return best;

}

else

{

int best = MAX;

// Recur for left and

// right children

for (int i = 0; i < 2; i++)

{

int val = minimax(depth + 1, nodeIndex \* 2 + i, true, values, alpha, beta);

best = min(best, val);

beta = min(beta, best);

* Then alpha beta condition is checked

if (beta <= alpha)

break;

**Code:**

// C++ program to demonstrate

// working of Alpha-Beta Pruning

#include<bits/stdc++.h>

using namespace std;

// Initial values of

// Aplha and Beta

const int MAX = 100;

const int MIN = -100;

// Returns optimal value for

// current player(Initially called

// for root and maximizer)

int minimax(int depth, int nodeIndex, bool maximizingPlayer, int values[], int alpha,

int beta)

{

// Terminating condition. i.e

// leaf node is reached

if (depth == 2)

return values[nodeIndex];

if (maximizingPlayer)

{

int best = MIN;

// Recur for left and

// right children

for (int i = 0; i < 2; i++)

{

int val = minimax(depth + 1, nodeIndex \* 2 + i,

false, values, alpha, beta);

best = max(best, val);

alpha = max(alpha, best);

// Alpha Beta Pruning

if (beta <= alpha)

break;

}

return best;

}

else

{

int best = MAX;

// Recur for left and

// right children

for (int i = 0; i < 2; i++)

{

int val = minimax(depth + 1, nodeIndex \* 2 + i,

true, values, alpha, beta);

best = min(best, val);

beta = min(beta, best);

// Alpha Beta Pruning

if (beta <= alpha)

break;

}

return best;

}

}

// Driver Code

int main()

{

int values[0];

int i = 0;

cout<<"The random numbers are:";

while(i++ < 100) {

int values = (rand() % 100) + 1;

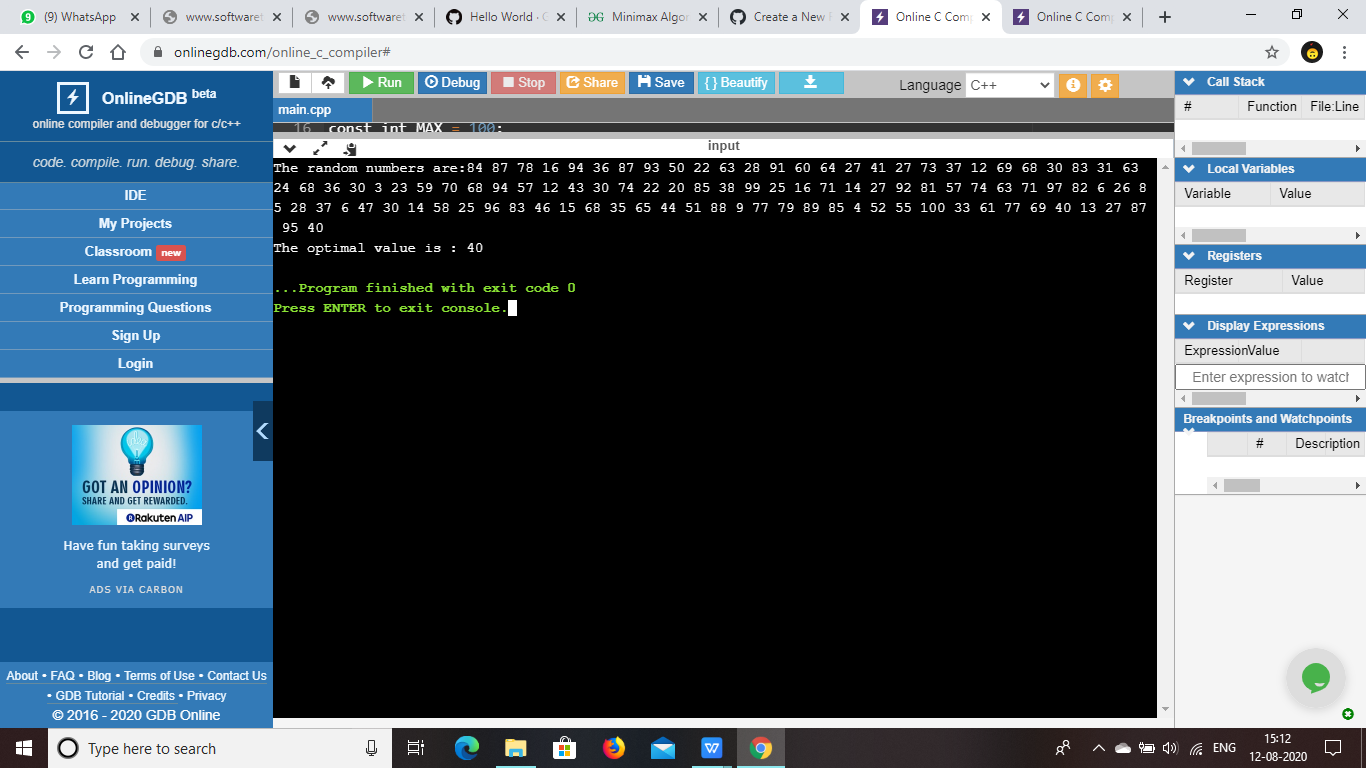
cout << values << " ";

}

cout <<"\n"<<"The optimal value is : "<< minimax(0, 0, true, values, MIN, MAX);

return 0;

}



**SRUTHI.R**

**18BCS051**

**Bachelor of Computer Science**

**Due Date:August 12,2020**