**COURSE: ARTIFICIAL INTELLIGENCE**

**ASSIGNMENT 1: MAGIC CUBE AND 3D TIC TAC TOE**

**AVANTIKA SHARMA: 1910110104**

**DIYA SACHDEV: 1910110140**

**KHUSHI CHAWLA:1910110203**

**Language Used : C++**

**MAGIC CUBE FUNCTIONS AND ALGORITHMS:**

1. Main Function  : User enters row, col and height surface of the first element and call function generateCube

int main()

{

   int n = 3;

   int row=0, col=0, height=0;

   cout<<"enter row ";   cin>>row;                // enter row no for 1st element

   cout<<endl<<"enter column";   cin>>col;   // enter column no for 1st element

   cout<<endl<<"enter height";   cin>>height;    // enter height no for 1st element

   generateCube(n,row,col,height);

   return 0;

}

1. GenerateCube ( n dimension = 3) : Generate the cube around the coordinate given by the user for first element and then print the final 3D magic cube

**Logic** : Magic cube contains nos from 1-27 . We have the first element ( 1 ) at the position specified by the user.

Case 1: Position is empty. Element (num) is added to the position

Case 2: Position is filled. We shift the height by - 1 and column by 1 and then add the element

Next element coordinates:

Case 1 : Element is divisible by dimension (3) ie, we have shifted the row and col 3

     times i.e. dimension size. In this case we do row +1, column + 2 and height + 1

Case 2: Element is not divisible. In this case we just shift column + 2 and row + 2

void generateCube(int n,int r, int c, int h) {

    int cube[n][n][n] ;

    memset(cube,0,sizeof(cube));   // all position = 0

    int row = r , col = c, hei = h;

    for(int num = 1; num<=(n\*n\*n); num++)

    {

        if(cube[hei][row][col]==0)      // If the space is empty

              cube[hei][row][col] = num;

        else                                     //If the space is already filled

        {  hei = (hei -1)%3;   // shift the height by 1

            col = (col+1)%3;    // shift the column by 1

           if(hei<0)

                   hei = 2;

  cube[hei][row][col] = num;   }

        // coordinates for next element

        col = (col+2)%3;     // shift column

        if(num%n==0)

        { row = (row+1)%3;   // shift row

            hei = (hei+1)%3;    // shift the surface by 1        }

        else

               row = (row+2)%3;  // shift row by 2

   } }

**3D TIC TAC TOE FUNCTIONS AND ALGORITHMS:**

**1.**       **void generateCube(int n) –** Generates a cube and transforms and prints it into a magic cube that acts as 3 square boards for the tic tac toe game.

**2.**       **void humanWinLine( char c[3][3][3] , char human, int position) –** the global variable human line  gets updated after every winning line and prints the users score on that move.

**3.**       **void printList()-** the global variable human line and computer line gets updated after every winning line and enters this function to print users and computers previous moves.

**4.**       **int checkFork(char c[3][3][3],char computer) –** after failing computerPossibleWin, checkFork condition is called in which it returns 0 if computer has no possible user’s line to block, returns 1 if computer can block 1 user’s winning line and returns 2 or more if there is a fork condition. In case of fork == 0 randomMove function is called, if fork ==1 then humanPossibleWin is called and if fork == 2 all the possible moves to block fork conditions are stored in a vector called forkList.

**5.**       **void forkMove(char c[3][3][3], char computer) –** if fork variable returns two or more that means that there are two or more winning lines of user and now the computer must choose the most preferable move. This is when this function comes in and checks possibilities of the computer's next move.

**6.**       **void print (char c[3][3][3], int n)-** prints the whole cube in an updated manner after every valid move of user and computer.

**7.**       **int randomMove(char c[3][3][3], char computer)-** . In case of fork == 0 randomMove function is called, this function iterates all the elements in the computer's list and array of all elements to find an element which has computers existing move and an unplayed position in collinear line condition with sum 42. Once the element is found it stores computer’s symbol in the character array at element’s dimensions and pushes the element in computer’s list

**8.**       **int humanPossibleWin(char c[3][3][3], char computer)-** In case of fork == 1 humanPossibleWin function is called, this function iterates all the elements in human’s list and last element of it’s list to find an element which has an unplayed position in collinear line condition with sum 42. Once the element is found it stores the computer's symbol in the character array at element’s dimensions and pushes the element in the computer's list.

**9.**       **int computerPossibleWin(char c[3][3][3], char computer)-** When computerPossibleWin function is called, this function iterates all the elements in computer’s list and last element of its list to find an element which has an unplayed position in collinear line condition with sum 42. Once the element is found it stores the computer's symbol in the character array at element’s dimensions and pushes the element in the computer's list. In case there are no possible moves it returns 0.

**10.**   **int blockUser(int num1 , int num2, char c[3][3][3], char computer)-** when counter == 4 this function is called to block the first possible line. Similar to humanPossibleWin, this function iterates all the elements in human’s list and last element of it’s list to find an element which has an unplayed position in collinear line condition with sum 42. Once the element is found it stores the computer's symbol in the character array at element’s dimensions and pushes the element in the computer's list.

**11.**   **void humanMoves (int x[3][3][3], char c[3][3][3], char human, int khushi, int arr1[30]) –** this function takes input from the user to play their next move and pushes the element in human moves list.

**12.**   **void computerMoves (char c[3][3][3], char computer, int counter)-** this function majorly has if and else if blocks counter wise. When counter == 1 that means the computer gets to choose first and it chooses the centre. For counter == 2 if the user hasn’t taken the centre it plays there else it plays position i=0, j=0, k = 0. For counter == 3 if after taking the centre when the computer is X it takes one of the suitable corners. For counter == 4 blockUser function is called and for counter 5 and above functions computerPossibleWin, checkFork, humanPossibleWin, randomMove and forkMove are called sequentially.

**13.**   **void ticTacToeGame (int x[3][3][3], char human)** – This is the root function for the game in which a character array is formed which stores all the symbols according to the cube’s dimensions turn by turn after every move. In this function itself the option to play first or let computer play is given and orderwise turns are given to user and computer with the help of a counter variable.

**14.**   **int main () –** generate cube function is called to make the base square boards along with introductory statements and necessary inputs are taken in the beginning of the game.

**CONTRIBUTION**

**Diya Sachdev - Magic Cube Code and Documentation (32%)**

**Avantika Sharma - Tic Tac Toe Code (7 functions) (34%)**

**Khushi Chawla - Tic Tac Toe Code (7 functions)(34%)**