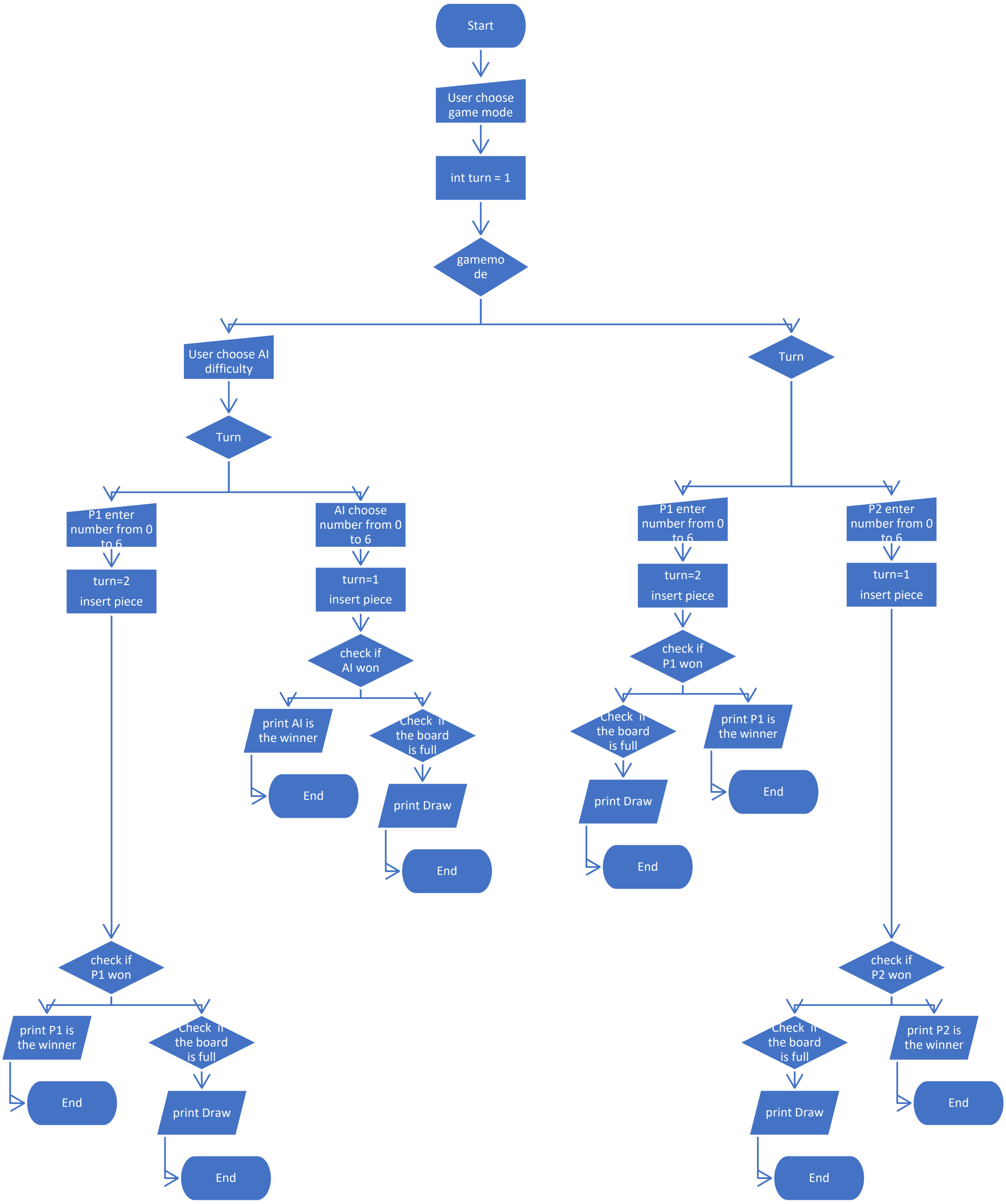
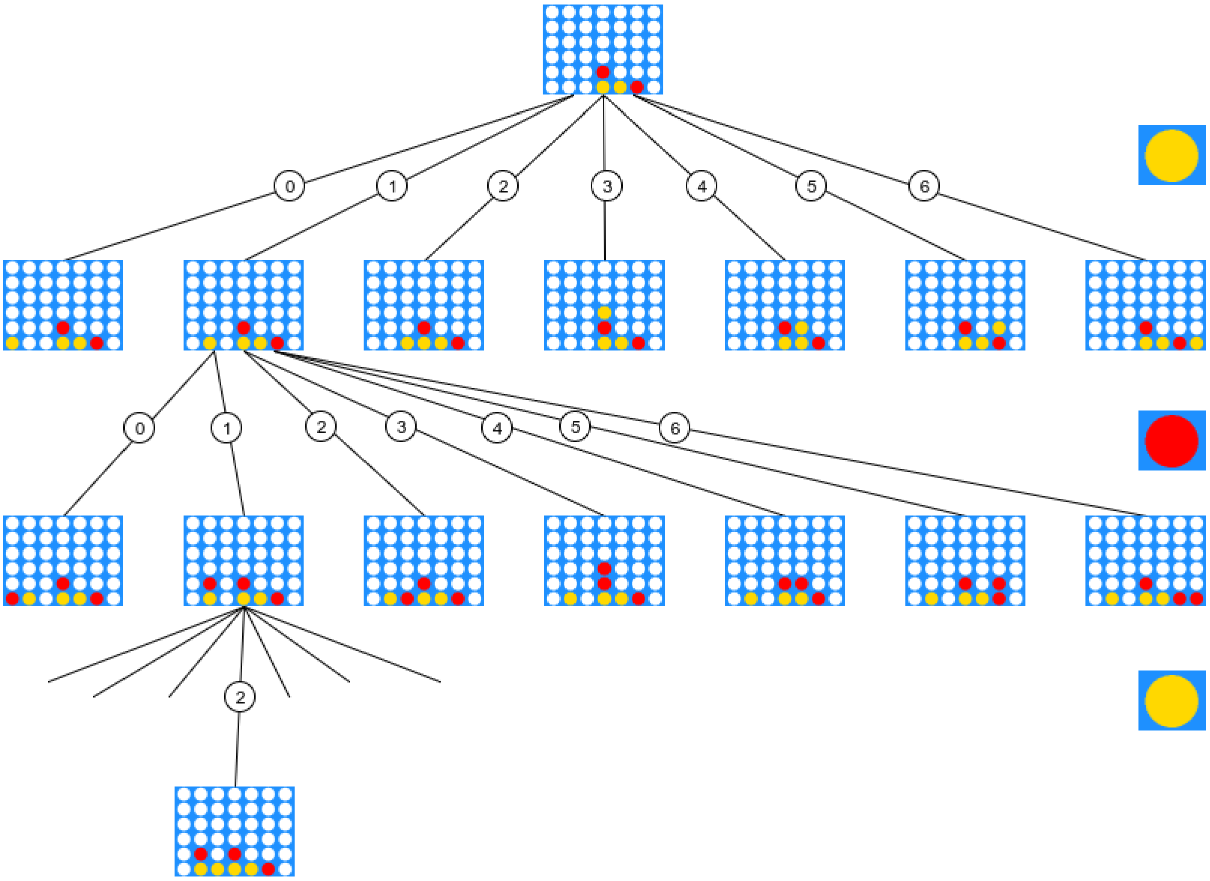

Start
User choose game mode
int turn = 1
gamemo de
User choose AI difficulty
Turn
Turn
P1 enter number from 0 to 6
turn=2
insert piece
AI choose number from 0 to 6
turn=1
insert piece
check if AI won
print AI is the winner
End
Check if the board is full
print Draw
End
P1 enter number from 0 to 6
turn=2
insert piece
check if P1 won
Check if the board is full
print Draw
End
print P1 is the winner
End
P2 enter number from 0 to 6
turn=1
insert piece
check if P1 won
print P1 is the winner
End
Check if the board is full
print Draw
End
check if P2 won
Check if the board is full
print Draw
End
print P2 is the winner
End

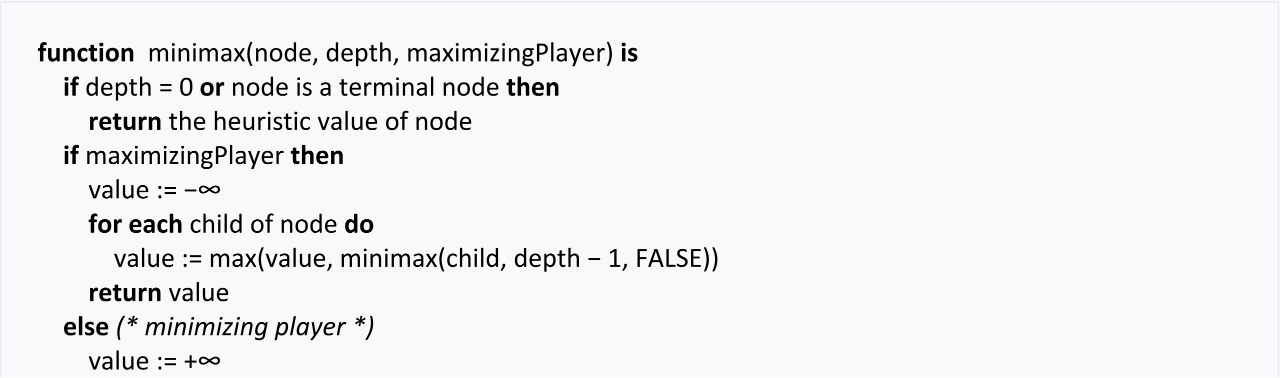
Minimax algorithm:

-A minimax algorithm is a recursive program written to find the best gameplay that minimizes any tendency to lose a game while maximizing any opportunity to win the game.

-Graphically, we can represent minimax as an exploration of a game tree's nodes to discover the best game move to make.



Implementation:



```
function  minimax(node, depth, maximizingPlayer) is
   if depth = 0 or node is a terminal node then
      return the heuristic value of node
   if maximizingPlayer then
      value := −∞
      for each child of node do
         value := max(value, minimax(child, depth − 1, FALSE))
      return value
   else (* minimizing player *)
      value := +∞
```

```
for each child of node do
   value := min(value, minimax(child, depth − 1, TRUE))
return value
```

Connect 4 Game

➢ Why did we choose this game?:

- Because it’s easy to implement on the console, we didn’t need the GUI. We found it’s easy to implement all the rules of it and programming a game like this was very interesting for us.

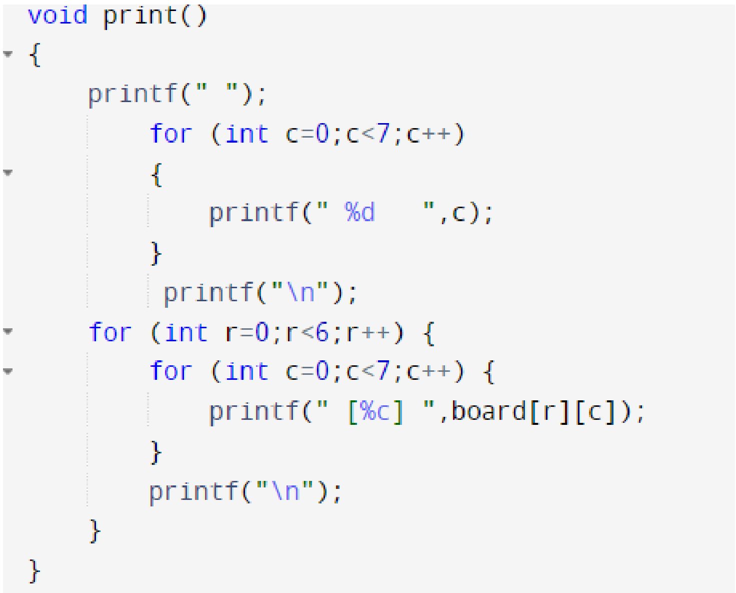
➢ Problem Analysis:

- Initializing the board of the game.
- Choosing the playing disc.
- Putting playing disc on the board.
- Check the cases of the board before putting the disc (empty, full or out of the range).
- Checking if the any player win (Row – Vertical – Diagonal).
- Modes of playing (Player vs Player, Player vs PC).
- Choosing the difficulty of the game for the PC.
- Check if the column is full in PC mode.
- Asking for another user for another game when the game is finished.

➢ Functions:

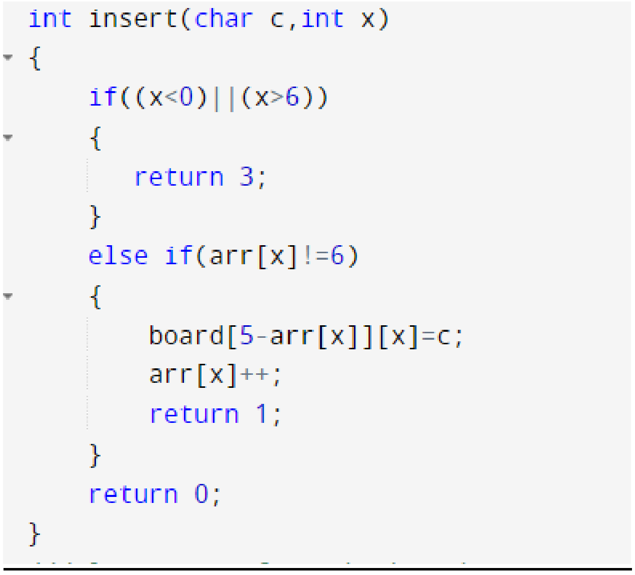
- void print();
- int insert(char c,int x);
- void player(char c);
- void reinsert(int x);
- int checkboard();
- int diff();
- int search(char a,char b, char c, char d, char e);
- int points();
- long long minimax(int depth,int isMaxmizing, int firstTime);

1. **Print**: This function is used to print the board



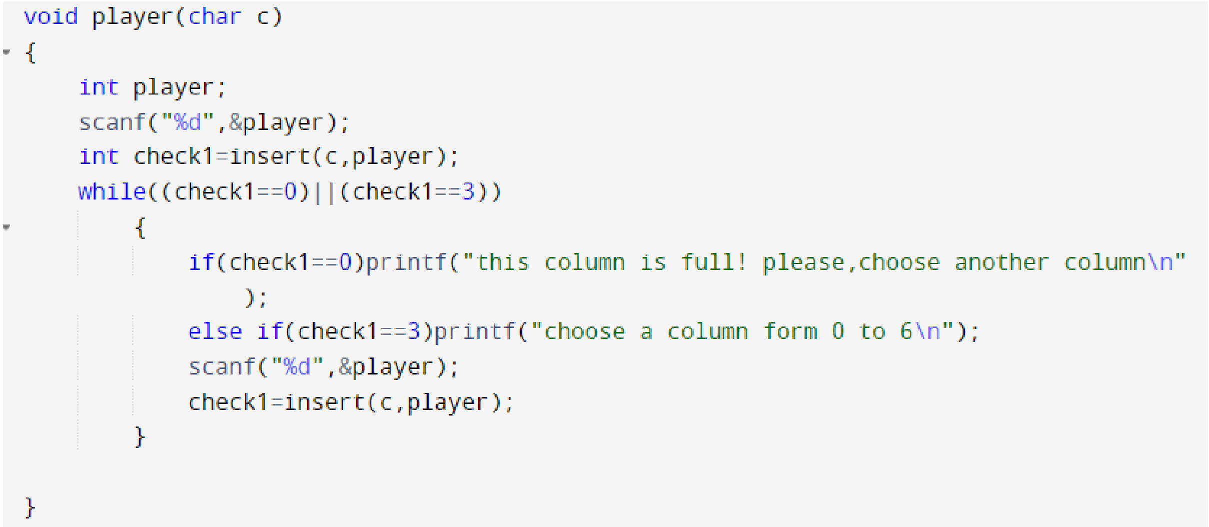
```
void print()
{
    printf(" ");
        for (int c=0;c<7;c++)
        {
            printf(" %d   ",c);
        }
         printf("\n");
    for (int r=0;r<6;r++) {
        for (int c=0;c<7;c++) {
            printf(" [%c] ",board[r][c]);
        }
        printf("\n");
    }
}
```

2. **Insert**: This function is used to receive the input from the player. The player chooses the column he wants to play and it checks if it’s possible to play in this column or not. It has **3 returns**:
    - **return 0**: The column is full .
    - **return 1**: The input is received safely.
    - **return 3**: Wrong input if the range isn’t from 0 to 6.



```c
int insert(char c,int x)
{
    if((x<0)||(x>6))
    {
       return 3;
    }
    else if(arr[x]!=6)
    {
        board[5-arr[x]][x]=c;
        arr[x]++;
        return 1;
    }
    return 0;
}
```

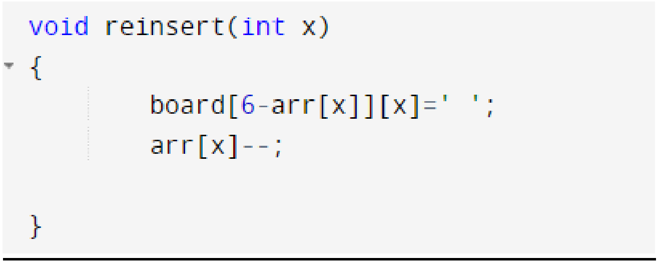
3. **Player:** This function is made for the player. It passes the piece of the player to the insert function. It has **2 checks** in a **while loop** and it doesn't get out of the loop till the player gives the possible column to play in it**:**
    - **check 0:** The column is full.
    - **check 3:** Wrong input (the range from 0 to 6).



```c
void player(char c)
{
    int player;
    scanf("%d",&player);
    int check1=insert(c,player);
    while((check1==0)||(check1==3))
        {
            if(check1==0)printf("this column is full! please,choose another column\n"
                );
            else if(check1==3)printf("choose a column form 0 to 6\n");
            scanf("%d",&player);
            check1=insert(c,player);
        }

}
```

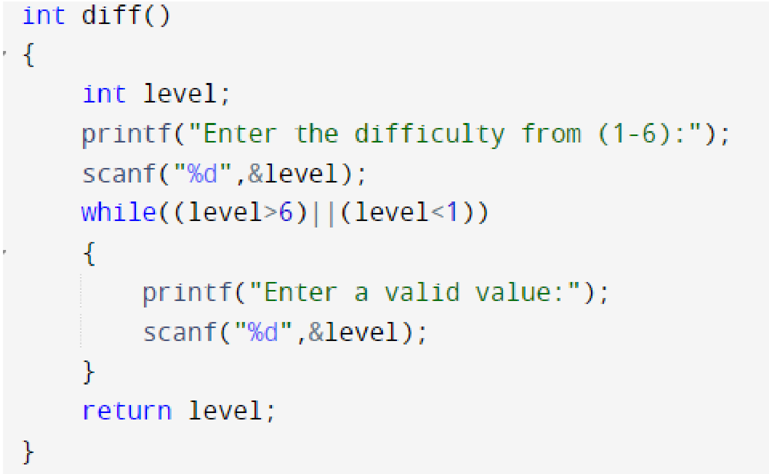
4. **reinsert:** This function is made to delete the top piece from chosen column.



```
void reinsert(int x)
{
        board[6-arr[x]][x]=' ';
        arr[x]--;

}
```

5. **diff:** It's used to choose the difficulty of the game if you play with the PC. Levels are made from **1 to 6**. Definitely we cared about if the user choose a number **out of this range**, it will give him a message to enter a valid value.



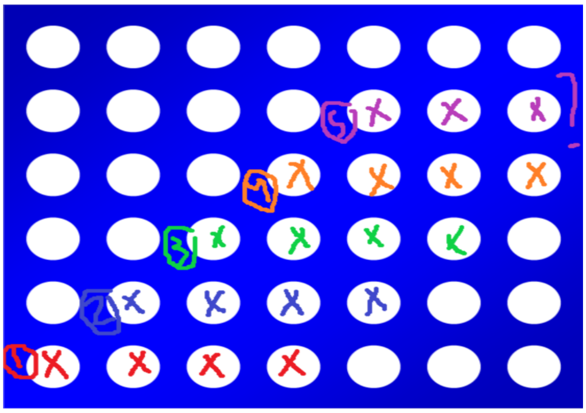
```
int diff()
{
    int level;
    printf("Enter the difficulty from (1-6):");
    scanf("%d",&level);
    while((level>6)||(level<1))
    {
        printf("Enter a valid value:");
        scanf("%d",&level);
    }
    return level;
}
```

6. **checkboard:** This is the winning's check function. There are 4 different methods to win in this game (Row – Column – Diagonal with positive slope – Diagonal with negative slope):

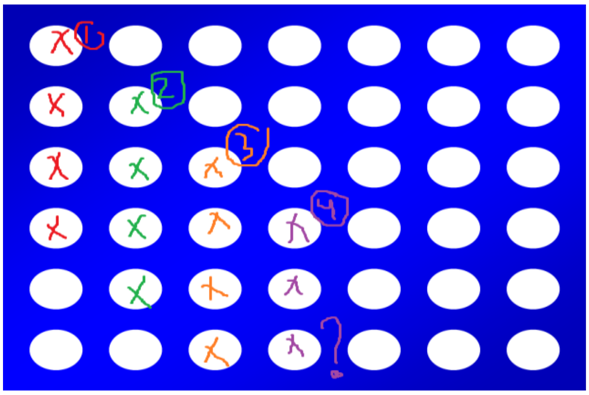
    ❖ **NOTE: Rows from 0 to 5 and Columns from 0 to 6.**

- **Row:**

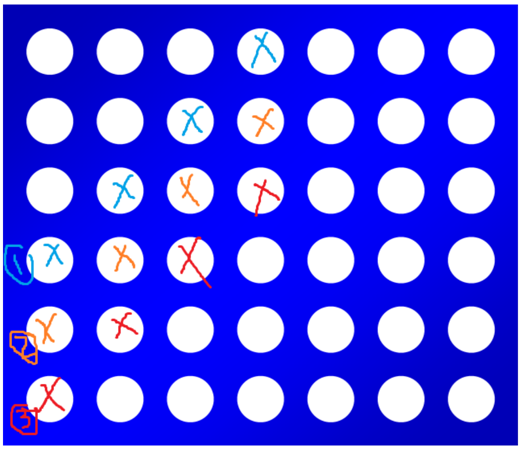
We need to check all the rows every game played if there are 4 same pieces beside each other. So we check every piece and 3 pieces beside it. As we can see in the picture, we can only check columns from **0 to 3**. **WHY?** in the **game 4**, **it's the last piece that we can check 3 beside it.**



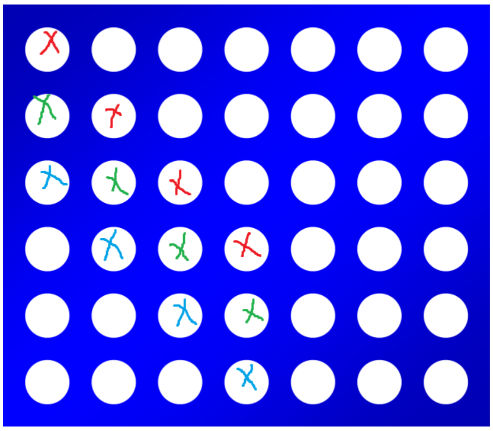
- **Column:**
  We need to check all the columns every game played if there are 4 same pieces beside each other. As we can see in the picture, we can only check the rows from **0 to 2**. **WHY?** in the **game 3**, **it's the last piece that we can check 3 under it.**

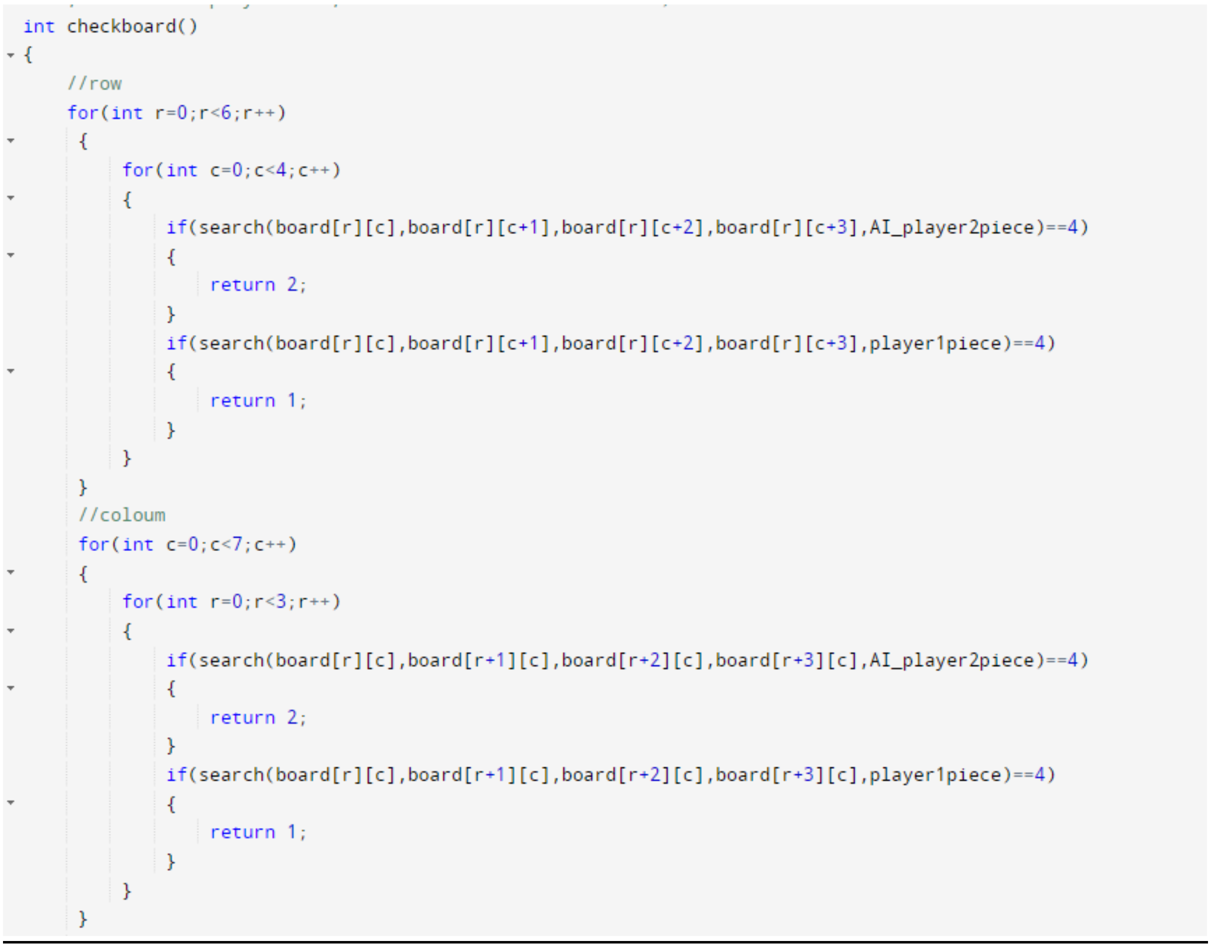


- **Diagonal with positive slope**:
  We need to find a relation between every piece and other one after it diagonally. We can see the **RELATION** is **[r-1][c+1]** between the pieces. Range of the **rows** from **3 to 6**. Range of the **columns** from **0 to 3**. We don't need to check all the rows and columns to save time and ease the processing.

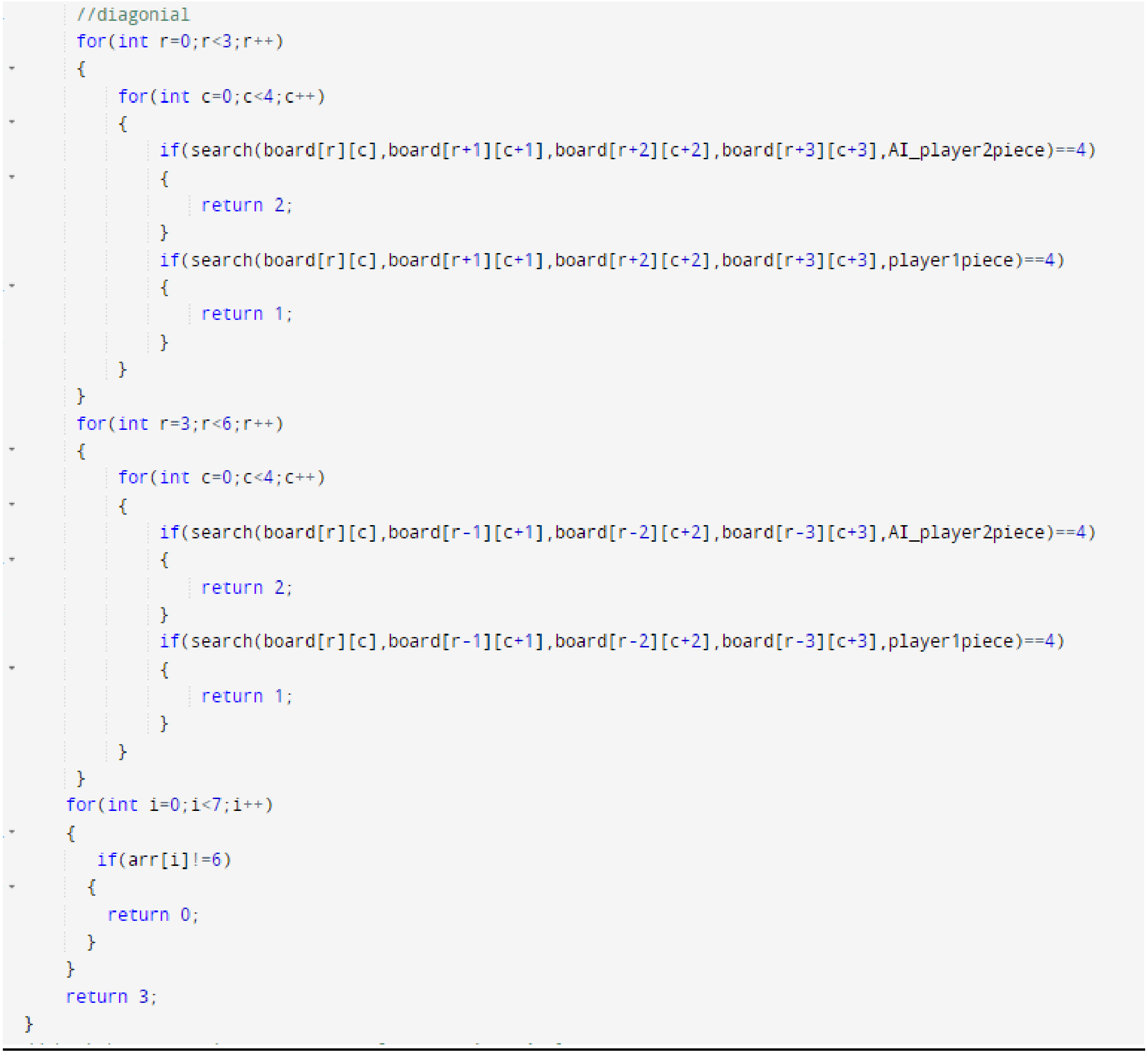


- **Diagonal with negative slope:**
  We need to find a relation between every piece and other one after it diagonally. We can see the **RELATION** is **[r+1][c+1]** between the pieces. Range of the **rows** from **0 to 2**. Range of the **columns** from **0 to 3**. We don’t need to check all the rows and columns to save time and ease the processing.



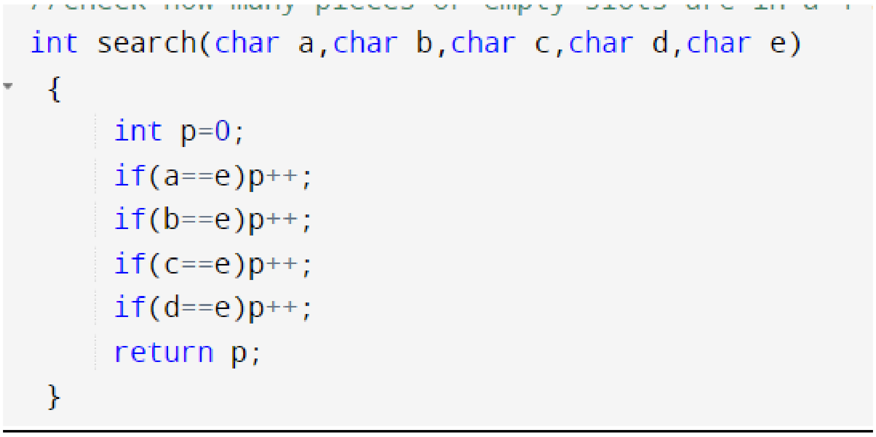


```
int checkboard()
{
    //row
    for(int r=0;r<6;r++)
     {
        for(int c=0;c<4;c++)
        {
            if(search(board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],AI_player2piece)==4)
            {
                return 2;
            }
            if(search(board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],player1piece)==4)
            {
                return 1;
            }
        }
     }
     //coloum
     for(int c=0;c<7;c++)
     {
        for(int r=0;r<3;r++)
        {
            if(search(board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],AI_player2piece)==4)
            {
                return 2;
            }
            if(search(board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],player1piece)==4)
            {
                return 1;
            }
        }
     }
```



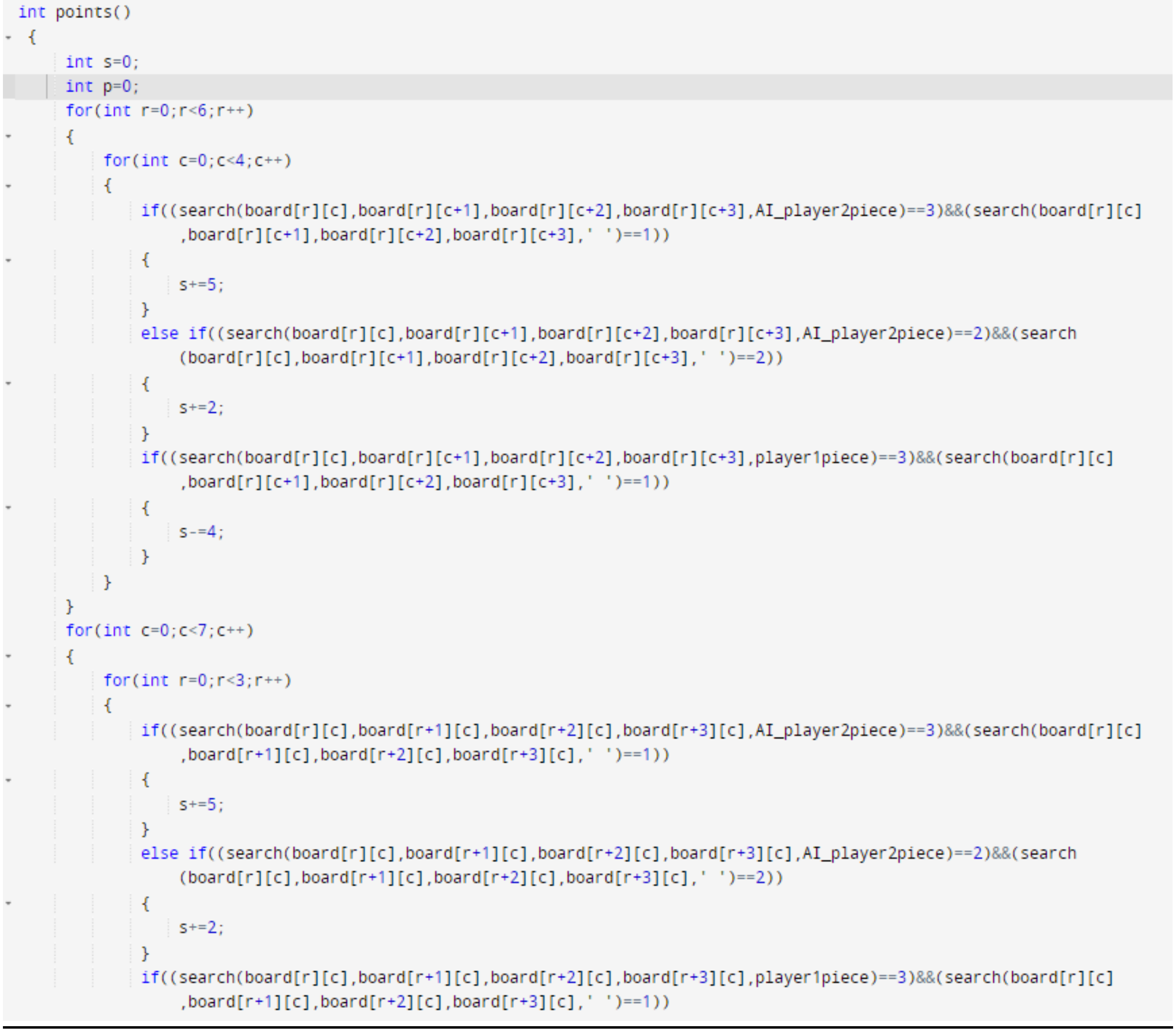
```
    //diagonial
    for(int r=0;r<3;r++)
    {
        for(int c=0;c<4;c++)
        {
            if(search(board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],AI_player2piece)==4)
            {
                return 2;
            }
            if(search(board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],player1piece)==4)
            {
                return 1;
            }
        }
    }
    for(int r=3;r<6;r++)
    {
        for(int c=0;c<4;c++)
        {
            if(search(board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],AI_player2piece)==4)
            {
                return 2;
            }
            if(search(board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],player1piece)==4)
            {
                return 1;
            }
        }
    }
   for(int i=0;i<7;i++)
   {
      if(arr[i]!=6)
     {
       return 0;
     }
   }
   return 3;
}
```

7. **Search:** This function is made to count how many p1 pieces or p2 pieces or empty cells are in 4 places. Char a , b , c and d are the 4 places and char e is the p1 piece or p2 piece or empty cell.

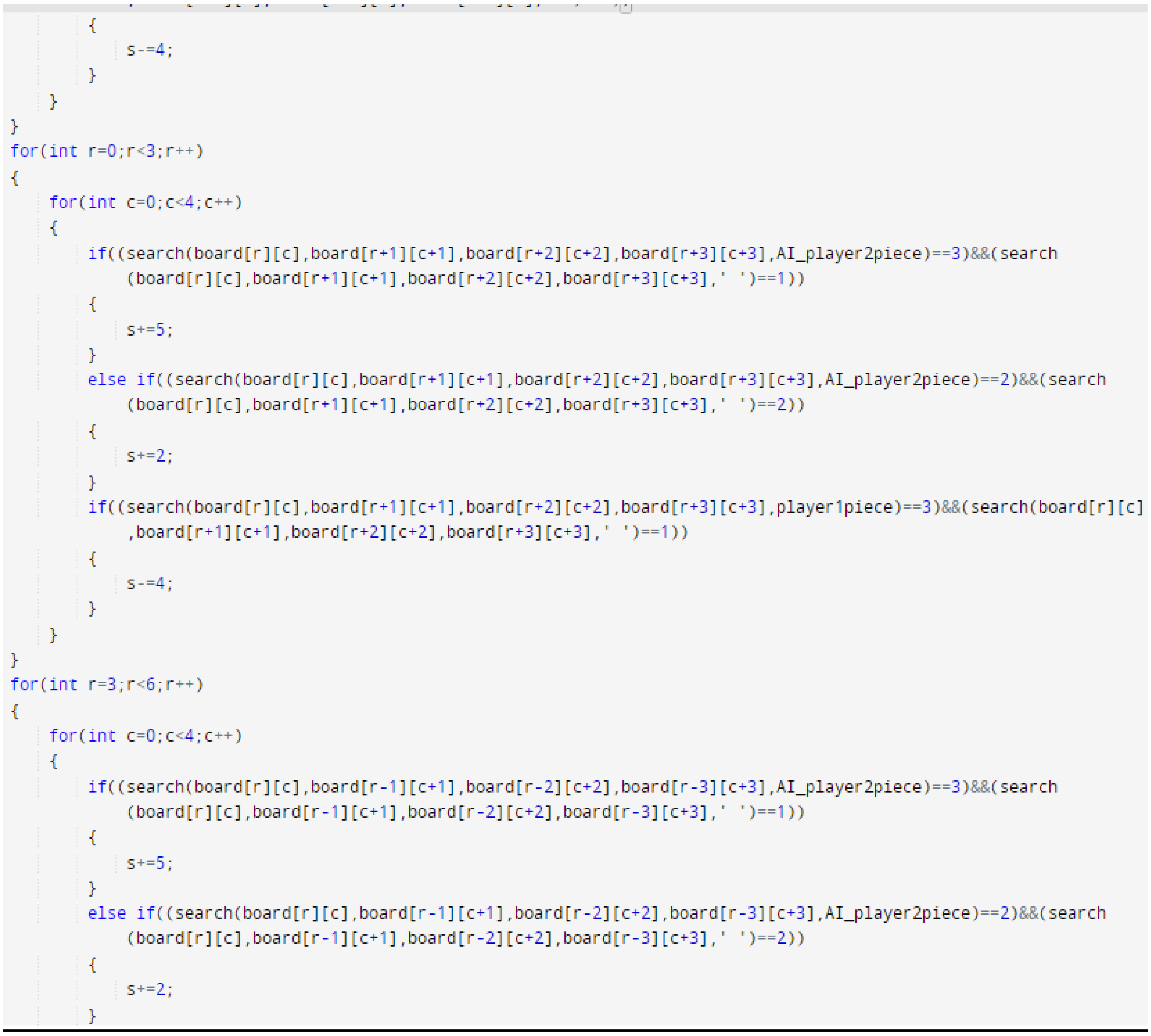


```
//check how many pieces or empty slots are in a
int search(char a,char b,char c,char d,char e)
 {
     int p=0;
     if(a==e)p++;
     if(b==e)p++;
     if(c==e)p++;
     if(d==e)p++;
     return p;
 }
```

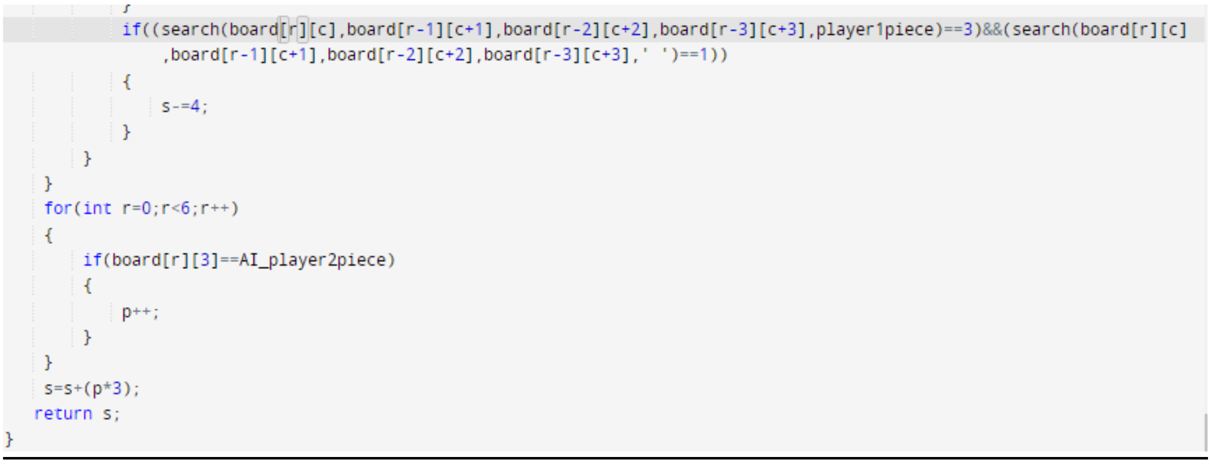
8. **Points:** This function is made to give the final board -when depth=0 or board is full-a score and this score decides how good or bad the move is.



```
int points()
{
    int s=0;
    int p=0;
    for(int r=0;r<6;r++)
    {
        for(int c=0;c<4;c++)
        {
            if((search(board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],AI_player2piece)==3)&&(search(board[r][c]
                ,board[r][c+1],board[r][c+2],board[r][c+3],' ')==1))
            {
                s+=5;
            }
            else if((search(board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],AI_player2piece)==2)&&(search
                (board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],' ')==2))
            {
                s+=2;
            }
            if((search(board[r][c],board[r][c+1],board[r][c+2],board[r][c+3],player1piece)==3)&&(search(board[r][c]
                ,board[r][c+1],board[r][c+2],board[r][c+3],' ')==1))
            {
                s-=4;
            }
        }
    }
    for(int c=0;c<7;c++)
    {
        for(int r=0;r<3;r++)
        {
            if((search(board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],AI_player2piece)==3)&&(search(board[r][c]
                ,board[r+1][c],board[r+2][c],board[r+3][c],' ')==1))
            {
                s+=5;
            }
            else if((search(board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],AI_player2piece)==2)&&(search
                (board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],' ')==2))
            {
                s+=2;
            }
            if((search(board[r][c],board[r+1][c],board[r+2][c],board[r+3][c],player1piece)==3)&&(search(board[r][c]
                ,board[r+1][c],board[r+2][c],board[r+3][c],' ')==1))
```

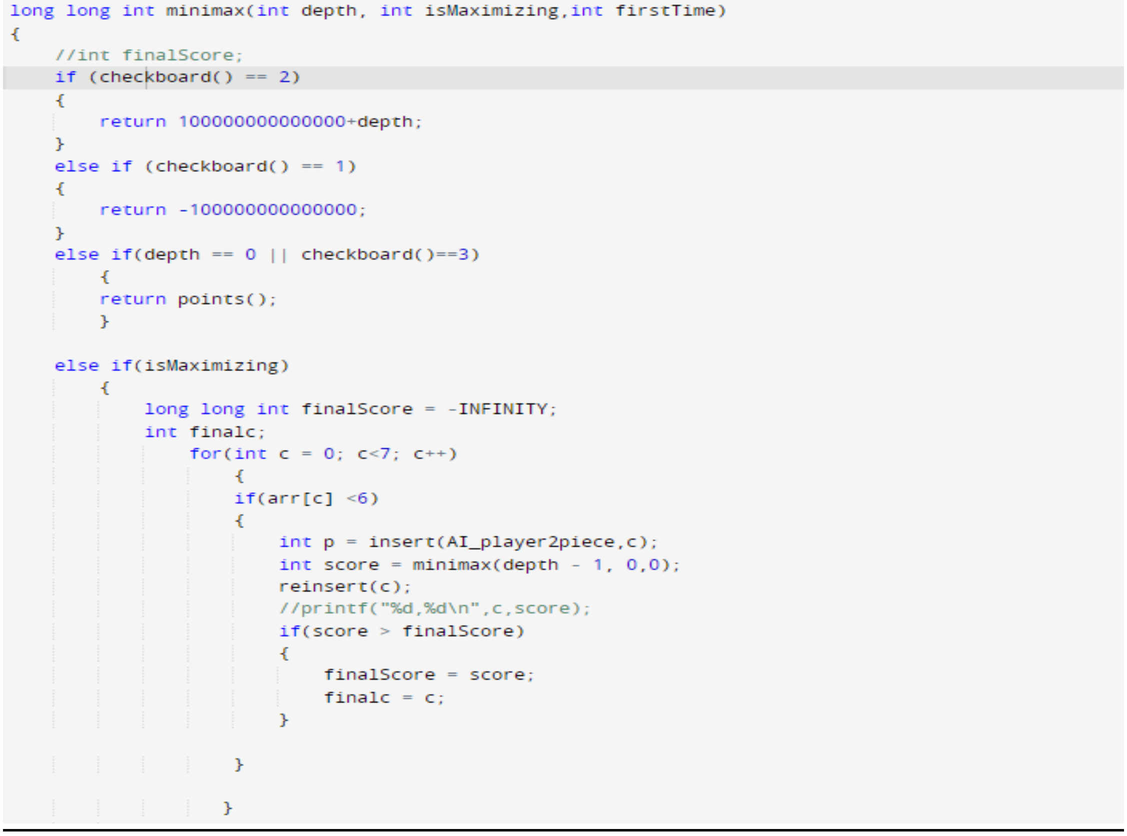


```
        {
            s-=4;
        }
    }
}
for(int r=0;r<3;r++)
{
    for(int c=0;c<4;c++)
    {
        if((search(board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],AI_player2piece)==3)&&(search
            (board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],' ')==1))
        {
            s+=5;
        }
        else if((search(board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],AI_player2piece)==2)&&(search
            (board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],' ')==2))
        {
            s+=2;
        }
        if((search(board[r][c],board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],player1piece)==3)&&(search(board[r][c]
            ,board[r+1][c+1],board[r+2][c+2],board[r+3][c+3],' ')==1))
        {
            s-=4;
        }
    }
}
for(int r=3;r<6;r++)
{
    for(int c=0;c<4;c++)
    {
        if((search(board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],AI_player2piece)==3)&&(search
            (board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],' ')==1))
        {
            s+=5;
        }
        else if((search(board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],AI_player2piece)==2)&&(search
            (board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],' ')==2))
        {
            s+=2;
        }
```



```
            if((search(board[r][c],board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],player1piece)==3)&&(search(board[r][c]
                ,board[r-1][c+1],board[r-2][c+2],board[r-3][c+3],' ')==1))
            {
                s-=4;
            }
        }
    }
    for(int r=0;r<6;r++)
    {
        if(board[r][3]==AI_player2piece)
        {
            p++;
        }
    }
    s=s+(p*3);
    return s;
}
```

9. **Minimax:** This function is made to choose the best move for AI depends on the score of the final board.

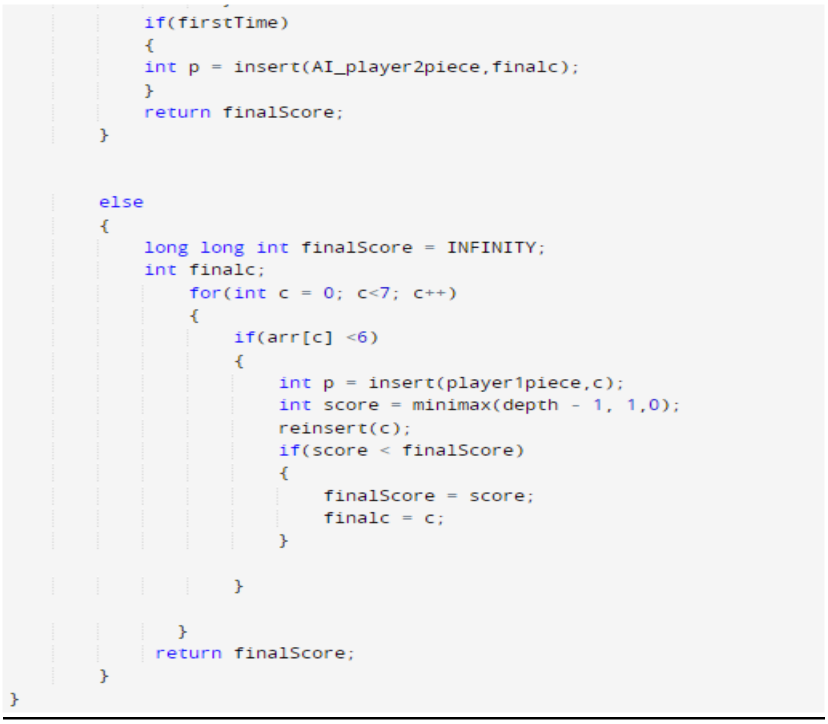


```
long long int minimax(int depth, int isMaximizing,int firstTime)
{
    //int finalScore;
    if (checkboard() == 2)
    {
        return 100000000000000+depth;
    }
    else if (checkboard() == 1)
    {
        return -100000000000000;
    }
    else if(depth == 0 || checkboard()==3)
        {
        return points();
        }

    else if(isMaximizing)
        {
            long long int finalScore = -INFINITY;
            int finalc;
                for(int c = 0; c<7; c++)
                    {
                    if(arr[c] <6)
                    {
                        int p = insert(AI_player2piece,c);
                        int score = minimax(depth - 1, 0,0);
                        reinsert(c);
                        //printf("%d,%d\n",c,score);
                        if(score > finalScore)
                        {
                            finalScore = score;
                            finalc = c;
                        }

                    }

                   }
```



```
            if(firstTime)
            {
            int p = insert(AI_player2piece,finalc);
            }
            return finalScore;
        }

        else
        {
            long long int finalScore = INFINITY;
            int finalc;
                for(int c = 0; c<7; c++)
                {
                    if(arr[c] <6)
                    {
                        int p = insert(player1piece,c);
                        int score = minimax(depth - 1, 1,0);
                        reinsert(c);
                        if(score < finalScore)
                        {
                            finalScore = score;
                            finalc = c;
                        }

                    }

               }
             return finalScore;
        }
}
```

-Problems we had faced:

1- player 1 plays every time at first so we make the first player plays at first random. By making initial value of the turn get a random number 1or2.

```
srand(time(0));

int turn=(rand()%2);
```

2-to avoid make if statement for each combination (xx x-x xx- xxx-xxx)

We implement fun called **search** to reduce 4 if statement into 1 if statement.

```
If(number of x==3  && number of empty cells==1)
```

3-when the code has two places to play and both places are guaranteed win but if it plays at the first place, it will win after this play. and if it plays at the second place, it will win immediately. The code prefers the play that comes first so to avoid this we made the score of the win depends on the depth of the play.

```
if (checkboard() == 2)

{

return 100000000000000+depth;

}
```

4-when the code has two places to play and both places are guaranteed lose but if it plays at the first place, it will lose after this play. and if it plays at the second place, it will lose immediately. The code prefers the play that comes first so to avoid this we made the score of the lose depends on the depth of the play.

```
else if (checkboard() == 1)

   {

       return -100000000000000-depth;

   }
```

-Future improvements: