Corner Cases

covered in our game

- **Corner case 1:**
  In the check-four function, when we tried to check the borders of each move, we neglected many intersections, such that: if we had three (X)'s from the right and the same from left, now if the user enters an X between them the score will increase by 1 not 4, so we solve this problem by checking the whole board after each move to cover all cases.

For example, horizontal check function:

```
//function to check if we connected four horizontally
int Hcheck(int x,int y,char board[x][y],int row,int move){
    int n;
    int count=0;
    int score=0;
    for(int i=0;i<x;i++){
        for(int j=0;j<y;j++){
            if(board[i][j]==board[i][j+1]&&board[i][j]==board[row][move]){
                for(int k=j;k<j+3;k++){
                        if(board[i][k]==board[i][k+1]){
                            count++;
                        }else{
                        count=0;
                        break;
                        }
                }
                if(count==3){
                    score++;
                    count=0;
                }
            }
        }
    }
    return score;
```

- **Corner Case 2:**
  If the user enters a character move instead of a number, the program will crash into an infinite loop, so we solve this problem by taking the move from the user as a string, and storing it in an array of size for example 20, and check if it is alphabetical by using the built-in function isalpha(), if true, we assigned the variable correct move to 0 else, we use atoi() function to convert the char to integer and in all

cases, we return correct move.

```
}
int check_number(char move[20]){
    int correct_move=0;
        if(isalpha(move[0])!=0){
        correct_move=0;
        }else{
        correct_move=atoi(move);

        }
        return correct_move;


}
```

- ## Corner Case 3:

  In the **main menu** if the user's input wasn't in the choices we ask him to choose again and call our menu function again.

```
            printf("the goal of the game is to connect
            printf("we check the score of each player,
            printf("and the player with the higher sc
            break;
        default:
            system("cls");
            printf("not available choice,TRY AGAIN");
            menu();
    }
}
```

- ## Corner Case 4:

  In undo & redo, if we undo for the second time, the second undo will be neglected, so to solve this problem we declared a new variable and make it take the value of (k-1) when k is the current move in the array undo

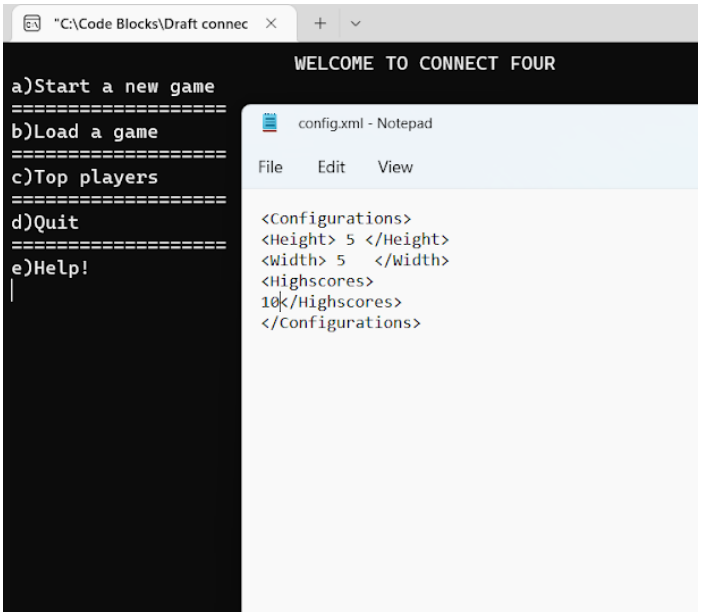
```
//undo
if(move[0]=='u'&&turn>0&&f>0){
        system("cls");
        board[row][correct_move-1]=' ';
       correct_move=undo[--f];
       row=undo1[f];
        i=i-2;
        turn--;
        v++;
        k=f-1;
        if(board[row][correct_move-1]=='X'){
       player1.score= scores_sum(x,y,board,row,correct_move-1);
    }else if(board[row][correct_move-1]=='O'){
    player2.score= scores_sum(x,y,board,row,correct_move-1);

    }
    }//redo
    else if(move[0]=='r'&&v>0){
        system("cls");
        correct_move=undo[++f];
        row=undo1[f];
    board[row][correct_move-1]=(turn%2==0? 'X':'O');
       turn++;
        v--;
        k=f+1;
        if(board[row][correct_move-1]=='X'){
       player1.score= scores_sum(x,y,board,row,correct_move-1);
    }else if(board[row][correct_move-1]=='O'){
    player2.score= scores_sum(x,y,board,row,correct_move-1);
```

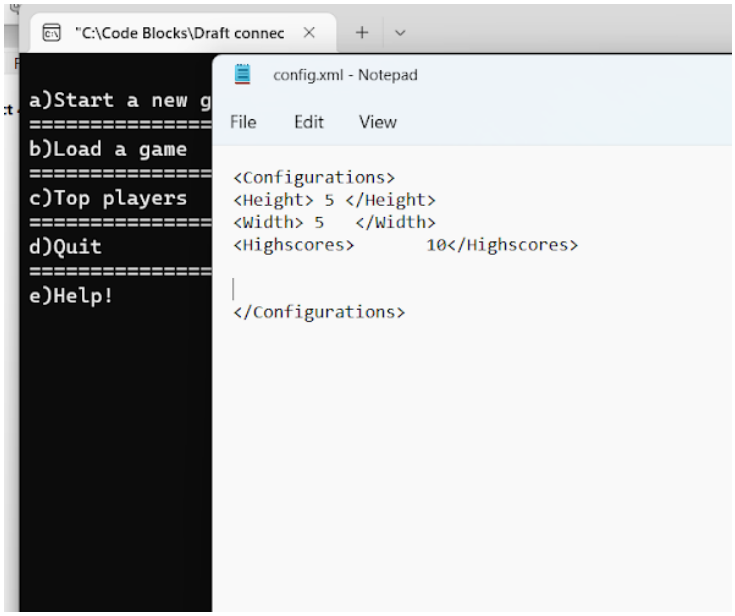
- ## Corner Case 5:

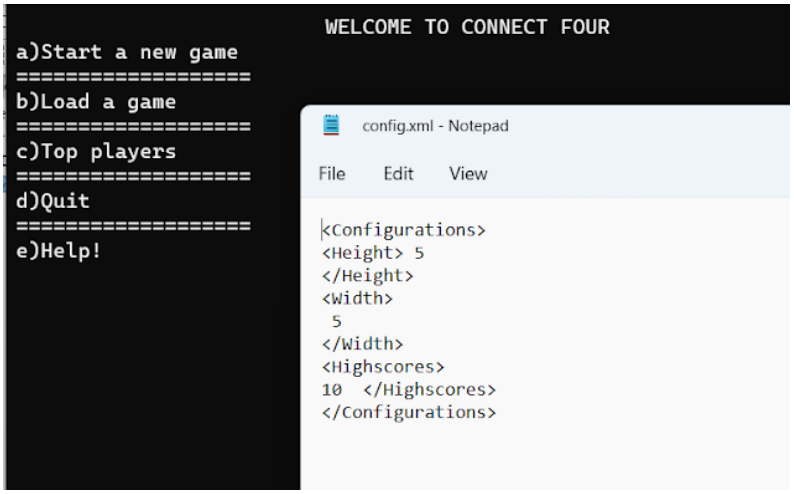
When taking the name from the player, it must be case insensitive we solve this problem by using a for loop and a function tolower() to convert the string to lowercase.

```
fflush(stdin);
game_end(choice);
for(int s=0;s<strlen(player1.name);s++){
player1.name[s]=tolower(player1.name[s]);
  }
```

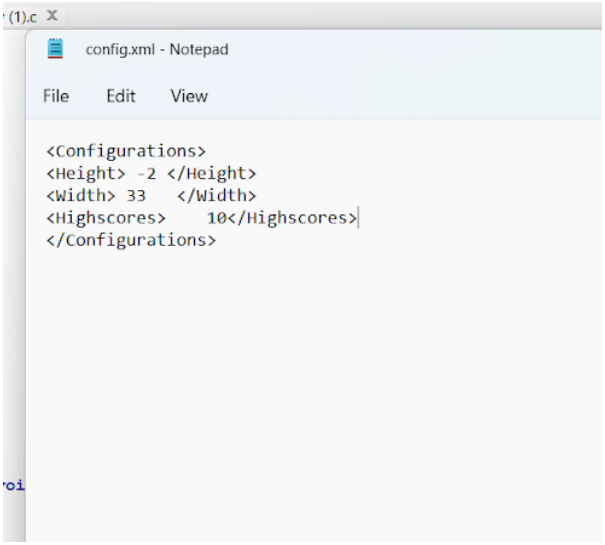
- ## Corner Case 6:
  Handling any indentation problem in XML file, (safe not corrupted)
  Cases handled:
  - Spaces between tags and numbers
  - Spaces between number and opening tag, then between number and closing tag
  - \t between number and opening tag, then between number and closing tag
  - \n between number and opening tag, then between number and closing tag
  - \n or \t between closing of a certain tag and opening of another

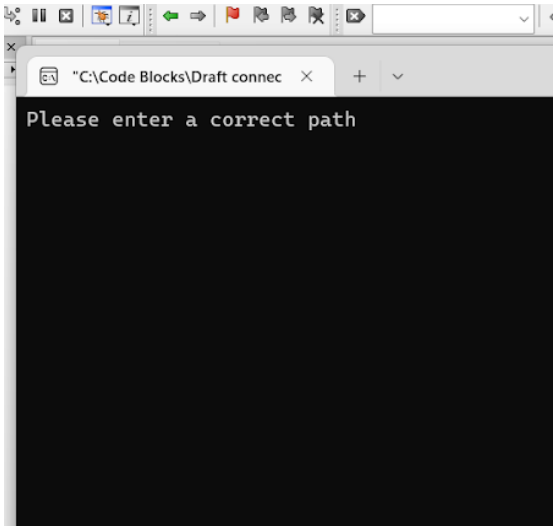






- ## Corner Case 7:
  Handling if numbers read from XML file are out of range, (less than 1) Then file is considered corrupted, user is asked for another path





- ## Corner Case 8:
  Handling wrong input of user in all 3 game modes,
  If user inputs a letter not shown in list printed or a column number out of range of columns (1:y) then loop iterations increase by 1, turn is still constant and user is asked to repeat input

```
-----------
| | | | | |
-----------
| | | | | |
-----------
| | | | | |
-----------
| | | | | |
-----------
|X|O| | | |
 1 2 3 4 5
Rowan 1's turn
Number of moves made: 1
score1=0
score2=0
Press 'u' for undo,'r' for redo
Press 's' to exit and save,'e' to exit game without saving
Time from starting the game=0:2
-2
WRONG INPUT, TRY AGAIN
77
WRONG INPUT, TRY AGAIN
r
WRONG INPUT, TRY AGAIN
w
WRONG INPUT, TRY AGAIN
```

- ## Corner Case 9:

  If user inputs a wrong number for loading a saved game,
  It directly returns to main menu with letting user know that game is unavailable

```
choose the saved game to load

Choose which game to load(1/2/3)?6

Game Unavailable
                          WELCOME TO CONNECT FOUR
a)Start a new game
===================
b)Load a game
===================
c)Top players
===================
d)Quit
===================
e)Help!
```

Those were the main highlight, we're sure many more are covered in our code, but we had to include the most important ones.