Namal University Mianwali
Department of Electrical Engineering

Quantitative And Computational Reasoning (QCR-II) Lab
Project Report (CEP)

Noughts and Crosses Game

Name	Hassan Ali Ahmad	Kinza Wajid
Roll No.	NUM-BSEE-2023-03	NUM-BSEE-2023-15
Submission Date	June 26,2024	

Course Instructor: Ms. Zulaikha Kiran

Lab Instructor: Ms. Maria Rehman

Table Of Content

Introduction

In this project we made a noughts and crosses game which provides a good console output and it is easy to understandable for every user .

Functionalities Of The Project

- Human vs Human gameplay
- Human vs PC gameplay
- Auto-save game option
- Load game option
- Leaderboard

Making a good Console Output

- In this project we have used some ASSCI characters to make the console output good.
- The use of sleep function allows the user to understand each step of the project that what is happening mostly it is used where the random function is used to check the number generated by this function.

Libraries

```
#include<iostream>
#include<fstream>
#include <cstdlib>
```

In this project three libraries have been used,

#include< iostream > :for taking the input and providing output.

#include< fstream > : for reading and writing in files

#include<cstdlib> : for random number generation

Structures

In this game totally 2 structures have been used,

1) **Players structure**

```
struct players
{
    string player1;
    string player2;
    string temp_player;
};
```

This structure is used for the name of players of the game and the temp_player actually stores the name of player who is performing his turn.

2) **Leaderboard structure**

```
struct Leaderboard
{
    vector<string> names;
    vector<int> wins;
};
```

This structure is used to save the names and wins of plyers in the leaderboard. Names array stores the names of winners and the wins array stores their number of wins.

Functions

In this project total 12 functions have been used

1)Sleep Function

```
void sleep(int hasa)
{
```

This function is used for providing a good console output on screen by showing the output slowly.

2)DisplayBoard Function

```
void displayBoard(char A[3][3])
{
```

This function is used to display the 3×3 board of tic tac toe game.

3)Winner Function

```
bool winner(char A[3][3], char player)
{
```

This function is used to decide the winner between two players and this function is called whenever a player makes a move.

4)Draw Function

```
bool draw(char A[3][3])
{
```

This function is used to decide that whether the game between two players has been draw or not and this function is called whenever a player makes a move.

5)ComputerMove Function

```
void computerMove(char A[3][3])
{
```

This function is used whenever a computer wants to make a move in a Human vs PC gameplay mode.

6)Save_game Function

```
void save_game(char A[3][3], char Player, string player1, string player2, string temp_player)
{
```

This function is used to save the game of Human vs Human gameplay mode because if the players want to play the game later on so they can play easily by loading the previous game.

7)Save_gamepc Function

```
void save_gamepc(char A[3][3], char Player, string player1, string player2, string temp_player)
{
```

This function is used to save the game of Human vs PC gameplay mode because if the player want to play the game later on so he/she can play easily by loading the previous game.

8)Display Function

```
void display(string temp_player, const vector<string>& names, const vector<int>& wins)
```

This function is used to display the leaderboard in descending order of wins of players.

9)savetoFile Function

```
void saveToFile(vector<string>& names, vector<int>& wins)
```

This function is used to save the winners of Human vs Human gameplay mode in leaderboard. This function also sort the players according to their number of wins in descending order.

10)savetoFile1 Function

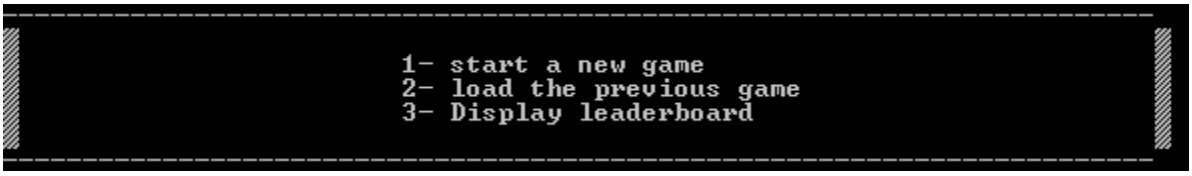
```
void saveToFile1(vector<string>& names, vector<int>& wins)
```

This function is used to save the winners of Human vs PC gameplay mode in leaderboard. This function also sort the players according to their number of wins in descending order.

11)human_gameplay Function

```
human_gameplay()
{
```

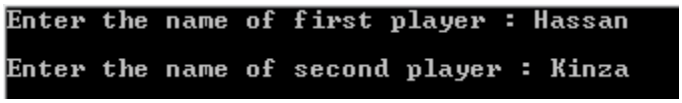
This function is used when we want to play the game between Human vs Human. When this function is called so the console shows 3 options:



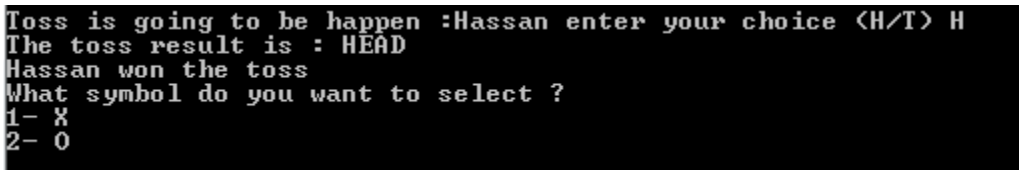
- if the user selects 1 so it starts the new game.
- if the user selects 2 so it loads the previous game.
- if the user selects 3 so it displays the leaderboard.

Option=1

The console shows that enter the name of players.

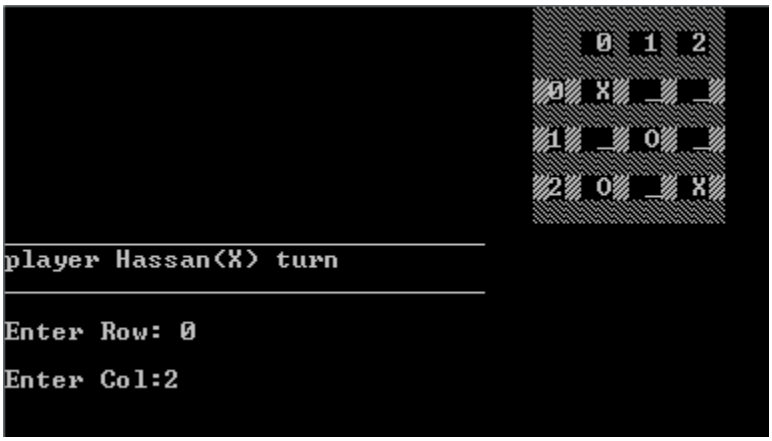


After entering the name there is a TOSS option the game asked to first player to enter his chice Head/Tail. If he will win the the toss so the game will ask to select your symbol (✕/O) .



After that the game starts and ask the players to enter their choice of row and column to place their symbol on that point on board.

After selecting the choice by one player the game changes the turn and the other player has to select his move in row and column.



Option=2

After the move of each player the save_game function is called to save the game. This function is used to save the game of Human vs Human gameplay mode because if the players want to play the game later on so they can play easily by loading the previous game.

```
save_game(A,Player,p.player1,p.player2,p.temp_player);
```

So if the user selects option 2 so it loads the previous game

```
else if (option==2)             // if the user selects 2 so it loads the previous game
{
    ifstream saliha("humanvspc.txt");
    if (saliha.is_open())
    {
        for (int i = 0; i < 3; ++i)
        {
            for (int j = 0; j < 3; ++j)
            {
                saliha >> A[i][j];
            }
        }
        saliha >> Player;
        saliha.ignore(); // Ignore newline character
        getline(saliha, p.temp_player);
        getline(saliha, p.player1);
        getline(saliha, p.player2);
        cout << "Game loaded successfully.\n";
        saliha.close();
    }
}
```

Option=3

Whenever the game is started so the leaderboard is automatically loaded.

```
Leaderboard leader;
leader.size1 = 0;
ifstream file("2human_leaderboard.txt");    // It open the file where the winners and their number of wins is stored
if (file.is_open())
{
    string name;
    int win;
    while (file >> name >> win)           // loop to load the players and their number of wins
    {
        leader.names[leader.size1] = name;
        leader.wins[leader.size1] = win;
        leader.size1++;
    }
    file.close();
}
else
{
    cout << "No saved leaderboard found." << endl;
}
```

When a player is a winner so his/her name is checked in the leaderboard if his name is already present so the game does increment in his/her wins. But if his/her name is not present so the game writes his name with his number of wins =1.

```
cout << "Player "<<p.temp_player<<"("<<Player<< ") is Winner\n";
for (int i = 0; i < leader.size1; i++)
{
    x=leader.wins[i];
    y=leader.wins[i];
    if (leader.names[i] == p.temp_player)    // condition to check if the winner is already present in leader board or not
    {
        leader.wins[i]++;
        y=leader.wins[i];
        cout<<"Player "<<p.temp_player<<" exists in leaderboard "<<endl;
        break;
    }
}
if(x==y)                                  // if the player is not present in the leader board so his name is placed in the leader board with wins
{

    for(int i=leader.size1; i<=leader.size1; i++)
    {
        leader.names[i] = p.temp_player;
        leader.wins[i] = 1;
        cout<<"Player "<<p.temp_player<<" does not exist in leaderboard "<<endl;
        break;
    }
}
```

When the user selects the option 3 so the leaderboard is displayed.

```
if(option==3)                          // if the user selects 3 so it displays the leaderboard
{
    display(p.temp_player, leader.names, leader.wins, leader.size1);
    return 1;
}
```

11)human_vs_pc_gameplay Function

```
human_vs_pc_gameplay()
{
```

This function is used when we want to play the game between Human vs PC. When this function is called so the console shows 3 options:

```
1- start a new game
2- load the previous game
3- Display leaderboard
```

- if the user selects 1 so it starts the new game.
- if the user selects 2 so it loads the previous game.
- if the user selects 3 so it displays the leaderboard.

Option=1

The console shows that enter your name.

```
Enter your name : Hassan
```

After entering the name there is a TOSS option the game asked to first player to enter his/her choice Head/Tail. If he will win the the toss so the game will ask to select your symbol (✕/O) otherwise the game will ask to PC for is symbol.

```
Toss is going to be happen :Hassan enter your choice (H/T) H
The toss result is : HEAD
Hassan won the toss
What symbol do you want to select ?
1- X
2- O
```

After that the game starts and ask the players to enter their choice of row and column to place their symbol on that point on board.

```
   0 1 2
0  _ O X
1  O _ O
2  _ X X

player Hassan(X) turn

Enter Row: 0
Enter Col:0
```

After selecting the choice by one player the game changes the turn and the other player has to select his move in row and column. In this Human vs PC gameplay mode PC selects his choice rendomely and we call a function for the PC's turn

```
void computerMove(char A[3][3])                    // function for the computer's move
{
    int row, col;
    //  srand(time(0));
    while (true)
    {
        row = rand() % 3;
        sleep(200000);
        cout<<"Row :"<<row<<endl;
        col = rand() % 3;
        sleep(200000);
        cout<<"coloumn :"<<col<<endl;
        sleep(1000000);
        if (A[row][col] == '_')
        {
            A[row][col] ='O';
            break;
        }
    }
}
```

Option=2

After the move of each player the save_game function is called to save the game. This function is used to save the game of Human vs PC gameplay mode because if the player wants to play the game later on so they can play easily by loading the previous game.

```
saveToFile1(p.temp_player, leader1.names, leader1.wins, leader1.size1);
```

So if the user selects option 2 so it loads the previous game

```
else if (option==2)             // if the user selects 2 so it loads the previous game
{
    ifstream saliha("humanvspc.txt");
    if (saliha.is_open())
    {
        for (int i = 0; i < 3; ++i)
        {
            for (int j = 0; j < 3; ++j)
            {
                saliha >> A[i][j];
            }
        }
        saliha >> Player;
        saliha.ignore(); // Ignore newline character
        getline(saliha, p.temp_player);
        getline(saliha, p.player1);
        getline(saliha, p.player2);
        cout << "Game loaded successfully.\n";
        saliha.close();
    }
}
```

Option=3

Whenever the game is started so the leaderboard is automatically loaded.

```
Leaderboard leader1;
leader1.size1 = 0;
ifstream file("human_vs_pc_leaderboard.txt");              // It open the file where the winners and their number of wins is stored
if (file.is_open())
{
    string name;
    int win;
    while (file >> name >> win)                            // loop to load the players and their number of wins
    {
        leader1.names[leader1.size1] = name;
        leader1.wins[leader1.size1] = win;
        leader1.size1++;
    }
    file.close();
}
else
{
    cout << "No saved leaderboard found." << endl;
}
```

When a player is a winner so his/her name is checked in the leaderboard if his name is already present so the game does increment in his/her wins. But if his/her name is not present so the game writes his name with his number of wins =1.

```
cout << "Player "<<p.temp_player<<"("<<Player<< ") is Winner\n";
for (int i = 0; i < leader.size1; i++)
{
    x=leader.wins[i];
    y=leader.wins[i];
    if (leader.names[i] == p.temp_player)    // condition to check if the winner is already present in leader board or not
    {
        leader.wins[i]++;
        y=leader.wins[i];
        cout<<"Player "<<p.temp_player<<" exists in leaderboard "<<endl;
        break;
    }
}
if(x==y)                                  // if the player is not present in the leader board so his name is placed in the leader board with wins
{

    for(int i=leader.size1; i<=leader.size1; i++)
    {
        leader.names[i] = p.temp_player;
        leader.wins[i] = 1;
        cout<<"Player "<<p.temp_player<<" does not exist in leaderboard "<<endl;
        break;
    }
}
```

When the user selects the option 3 so the leaderboard is displayed.

```
if(option==3)                          // if the user selects 3 so it displays the leaderboard
{
    display(p.temp_player, leader.names, leader.wins, leader.size1);
    return 1;
}
```

Functionalities making an effective project

The Tic Tac Toe game in C++ features an enhanced console output, making it more user-friendly and visually appealing. This simple yet classic game, often one of the first that many people learn, has been carefully crafted to provide an engaging and interactive experience. The game begins by presenting a 3x3 grid, which serves as the game board. This labeling system allows players to easily identify and select their moves, eliminating any confusion about which cells are available for play.

The console output dynamically updates after each player's turn, ensuring that the current state of the game is always visible. This real-time update feature is crucial for maintaining the flow of the game and keeping both players informed about the progression of their moves. The board's clear and structured presentation helps players focus on their strategy without getting distracted by a cluttered interface.

To enhance the user experience, the game provides clear instructions at every step. As

the game begins, a welcome message greets the players, explaining the basic rules of Tic Tac Toe and how to input their moves. Players are prompted to enter their move by selecting a number corresponding to an empty cell on the board. This prompt not only guides the players but also ensures that their input is valid and within the acceptable range.

If a player attempts to make an invalid move, such as selecting a cell that is already occupied or entering an out-of-range number, the game immediately provides feedback, notifying them of the invalid input. This feedback is accompanied by a prompt to try again, which prevents any disruption in the game's flow and helps maintain a smooth and enjoyable experience for both players.

The console output also highlights whose turn it is, switching between player 'X' and player 'O' after each move. This alternating prompt keeps players engaged and attentive, as they anticipate their next opportunity to place their mark on the board. The use of distinct characters ('X' and 'O') for each player helps in easily distinguishing their moves, thereby reducing any potential confusion.

One of the key features of this implementation is the clear and straightforward presentation of the final game outcome. Once a player successfully aligns three of their marks in a row, column, or diagonal, the game announces the winner in a celebratory message. This message is prominently displayed, ensuring that the victorious moment is clearly communicated. In the case of a draw, where all cells are filled without any player achieving three in a row, the game declares a tie, recognizing the well-fought efforts of both players.

This enhanced console output not only makes the game enjoyable but also serves as an excellent example of effective programming practices in C++. The use of functions for board display, player moves, and game outcome checks demonstrates modular programming, making the code more organized and maintainable. The dynamic update of the board and immediate feedback on player input showcase real-time interaction, a crucial aspect of interactive console applications.

Moreover, the clear and structured output aligns with best practices for user interface design, emphasizing readability and user engagement. The game's ability to provide real-time updates and clear instructions ensures that even players who are new to Tic Tac Toe can easily understand and enjoy the game. The attention to detail in handling player input and providing feedback contributes to a seamless user experience, highlighting the importance of thoughtful user interface design in programming.

Overall, the Tic Tac Toe game in C++ with enhanced console output stands as a testament to the power of simple yet effective design. It demonstrates how a well-crafted console application can provide an engaging and enjoyable experience, making it a valuable project for learning and practicing programming skills. By focusing on user-

friendly design, clear instructions, and real-time interaction, this implementation not only revitalizes a classic game but also imparts important lessons in programming and user interface design.

The expanded console output serves multiple purposes. First, it ensures that players have a smooth and enjoyable experience by providing a clear and interactive interface. Second, it illustrates the application of good programming practices, such as modular code structure and user-centered design. Third, it enhances the learning experience for those new to C++ programming, offering a hands-on example of how to create a functional and user-friendly console application.

From the initial setup of the board to the final declaration of the game outcome, every aspect of the Tic Tac Toe game in C++ has been carefully designed to enhance the player experience. The clear labeling of cells, dynamic board updates, alternating player prompts, immediate feedback on invalid moves, and celebratory winner announcements all contribute to a polished and professional console application.

Conclusion

In conclusion, the enhanced console output of the Tic Tac Toe game in C++ not only makes the game more enjoyable and accessible for players but also serves as an exemplary model of good programming practices. By focusing on clear and structured output, real-time interaction, and user-friendly design, this implementation provides a rich and engaging experience that is both educational and entertaining. This Tic Tac Toe game offers valuable insights and lessons in creating effective and enjoyable console applications.

References

Ascii

https://theasciicode.com.ar/extended-ascii-code/black-square-ascii-code-254.html

clear screen

https://www.sololearn.com/en/Discuss/1897226/how-can-i-clear-screen-when-user-input-something-in-c

Code with harry

https://www.codewithharry.com/tutorial/cplusplus/