Project 1

Connect 4

Board Game

Course

CIS-17A

Section

48593

November 14, 2021

Author

Domenico Venuti

Contents

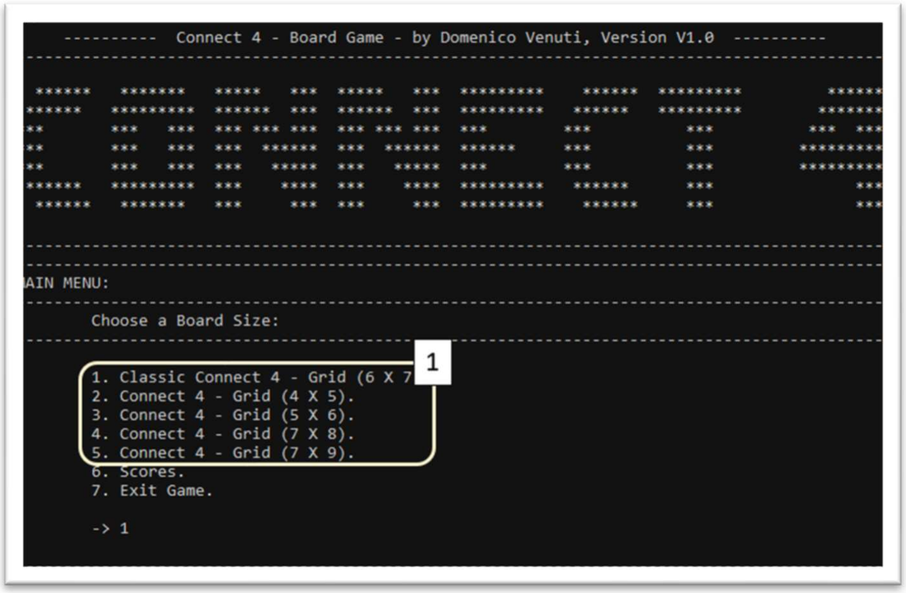
1. Introduction

Connect 4 is a 47-year-old Board Game that I enjoyed a lot when I was a child, I always played with my sister, she usually won, well, she is 3 years older than me, it is understandable when I was only 7 years old.

The Classic board was made by Hasbro, it has 6 rows and 7 columns but you can find different variations in the market. For the project, the program allows you to choose between 5 different board sizes.

The game is simple, there are 2 players, each player chooses a color (for the project this was modified and you only choose to be player 1 or 2 when writing your name), player 1 starts the game by choosing the column where he/she wants to place his/her piece (player 1's pieces are identified with number 1 and player 2's with number 2, empty spaces are identified with number 0).

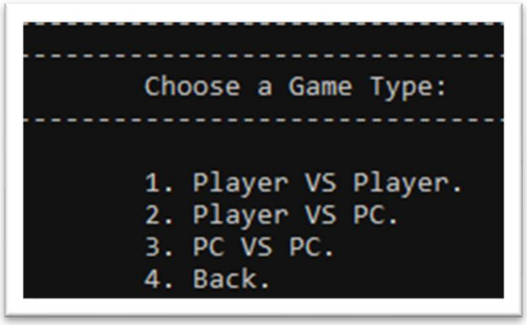
1. **The Program Allow you to Choose between 5 Different Board Sizes (6 X 7,4 X 5,5 X 6,7 X 8 and 7 X 9).**



```
    ----------  Connect 4 - Board Game - by Domenico Venuti, Version V1.0  ----------
------------------------------------------------------------------------------------------

 ******   *******   *****   ***  *****   ***  *********    ******  *********       ******
******   *********  ******  ***  ******  ***  *********   ******   *********      *******
**       ***   ***  *** *** ***  *** *** ***  ***         ***          ***        ***  ***
**       ***   ***  ***  ******  ***  ******  ******      ***          ***       *********
**       ***   ***  ***   *****  ***   *****  ***         ***          ***       *********
******   *********  ***    ****  ***    ****  *********   ******       ***             ***
 ******   *******   ***     ***  ***     ***  *********    ******      ***             ***

------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
AIN MENU:
------------------------------------------------------------------------------------------
       Choose a Board Size:
------------------------------------------------------------------------------------------

       1. Classic Connect 4 - Grid (6 X 7
       2. Connect 4 - Grid (4 X 5).
       3. Connect 4 - Grid (5 X 6).
       4. Connect 4 - Grid (7 X 8).
       5. Connect 4 - Grid (7 X 9).
       6. Scores.
       7. Exit Game.

       -> 1
```

Beside from being able to choose 5 different board sizes, to add more complexity I decided for this project to give the user the freedom to choose 3 Game Types:

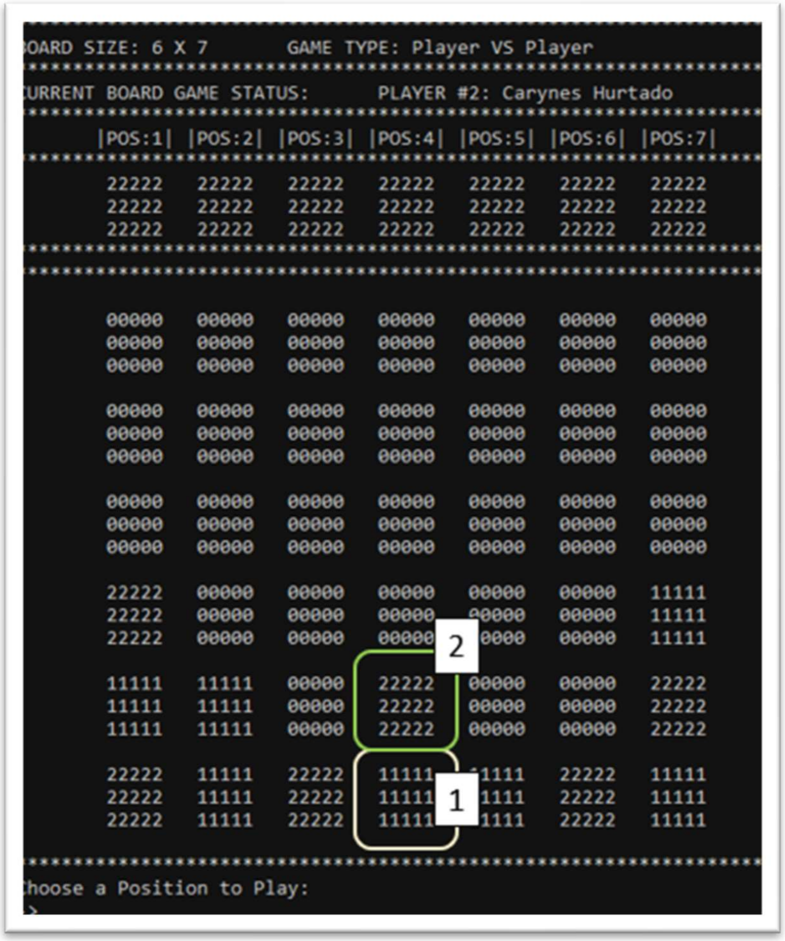
- Player Vs Player.
- Player VS PC.
- PC VS PC.



```
------------------------------
------------------------------
       Choose a Game Type:
------------------------------

       1. Player VS Player.
       2. Player VS PC.
       3. PC VS PC.
       4. Back.
```

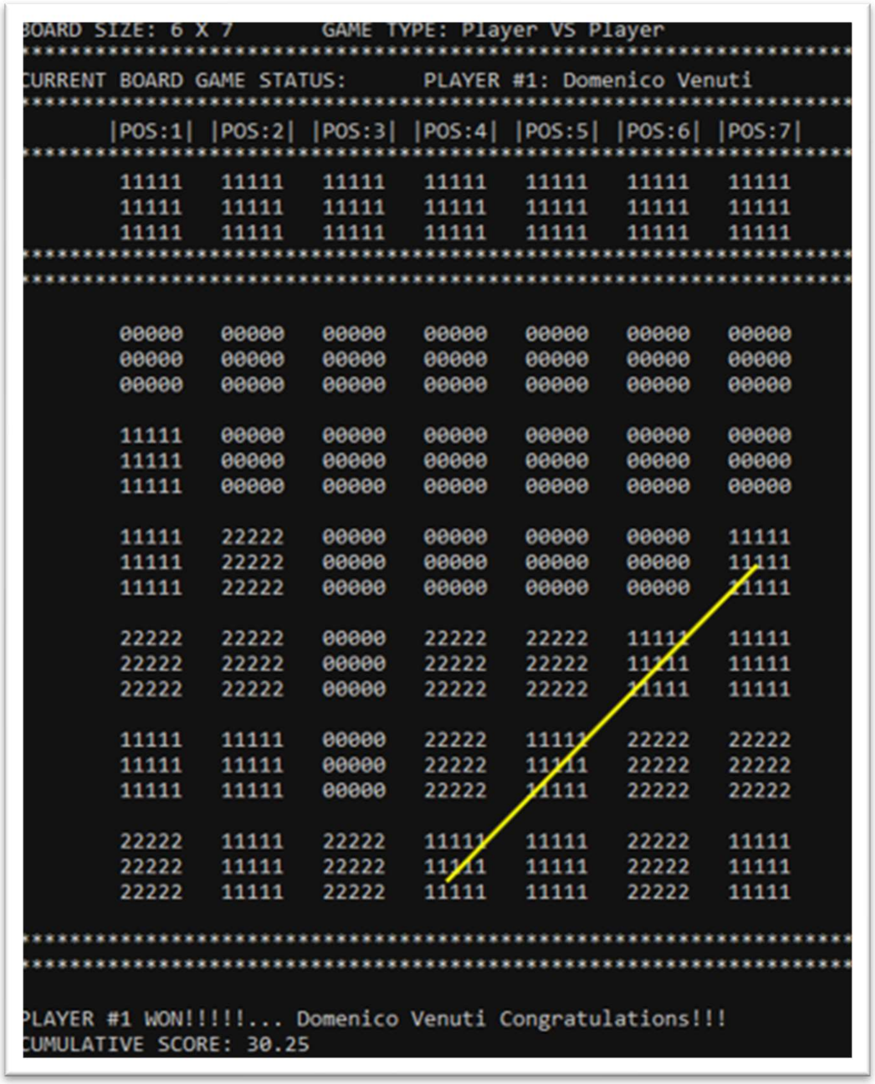
The piece must fall to the lowest place on the board corresponding to that column, in the case of a 6 x 7 board, the first piece will fall to row 6, if later a player chooses to place his/her piece on the same row, the new piece will fall to row 5 and so on, after player 1 does drop a piece, it is the turn of player 2 and so on.

1. **Pieces from Player 1 are Identified with Number 1.**
2. **Pieces from Player 2 are Identified with Number 2.**



```
OARD SIZE: 6 X 7        GAME TYPE: Player VS Player
********************************************************************
URRENT BOARD GAME STATUS:       PLAYER #2: Carynes Hurtado
********************************************************************
       |POS:1| |POS:2| |POS:3| |POS:4| |POS:5| |POS:6| |POS:7|
********************************************************************
        22222   22222   22222   22222   22222   22222   22222
        22222   22222   22222   22222   22222   22222   22222
        22222   22222   22222   22222   22222   22222   22222
********************************************************************
********************************************************************

        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000

        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000

        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000

        22222   00000   00000   00000   00000   00000   11111
        22222   00000   00000   00000   00000   00000   11111
        22222   00000   00000   00000   00000   00000   11111

        11111   11111   00000   22222   00000   00000   22222
        11111   11111   00000   22222   00000   00000   22222
        11111   11111   00000   22222   00000   00000   22222

        22222   11111   22222   11111   11111   22222   11111
        22222   11111   22222   11111   11111   22222   11111
        22222   11111   22222   11111   11111   22222   11111
********************************************************************
hoose a Position to Play:
>
```

To win the game, you must have at least 4 pieces of the same player in line, it can be a horizontal, vertical, or diagonal line (in the project, every time someone wins, the winner earns 30.25 points). If after filling all the empty spaces with pieces, it has not been possible to obtain 4 pieces from the same player in line, is a Draw.



```
BOARD SIZE: 6 X 7       GAME TYPE: Player VS Player
*****************************************************************
CURRENT BOARD GAME STATUS:      PLAYER #1: Domenico Venuti
*****************************************************************
       |POS:1| |POS:2| |POS:3| |POS:4| |POS:5| |POS:6| |POS:7|
*****************************************************************
        11111   11111   11111   11111   11111   11111   11111
        11111   11111   11111   11111   11111   11111   11111
        11111   11111   11111   11111   11111   11111   11111
*****************************************************************
*****************************************************************

        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000
        00000   00000   00000   00000   00000   00000   00000

        11111   00000   00000   00000   00000   00000   00000
        11111   00000   00000   00000   00000   00000   00000
        11111   00000   00000   00000   00000   00000   00000

        11111   22222   00000   00000   00000   00000   11111
        11111   22222   00000   00000   00000   00000   11111
        11111   22222   00000   00000   00000   00000   11111

        22222   22222   00000   22222   22222   11111   11111
        22222   22222   00000   22222   22222   11111   11111
        22222   22222   00000   22222   22222   11111   11111

        11111   11111   00000   22222   11111   22222   22222
        11111   11111   00000   22222   11111   22222   22222
        11111   11111   00000   22222   11111   22222   22222

        22222   11111   22222   11111   11111   22222   11111
        22222   11111   22222   11111   11111   22222   11111
        22222   11111   22222   11111   11111   22222   11111

*****************************************************************
*****************************************************************

PLAYER #1 WON!!!!!... Domenico Venuti Congratulations!!!
CUMULATIVE SCORE: 30.25
```

2. Summary

The Program in total has 1003 Lines of Code.

It seemed to me a very appropriate game for this project because although it does not seem to be very complex, the logic required to know if a player has won is more difficult than what it seems to be, plus the option to choose dynamically different board sizes and the capacity to play against the Computer and Computer against Computer were very challenging. Only the basic logic to evaluate the winning conditions of having 4 pieces in line on a variable size board like this one took almost 200 lines of code.

In total, the Program has the following statistics:

- 1003 x Lines of code.
- 19 x Functions.
- 2 x Structures.
- 3 x Enum.
- 5 x Constants.
- 7 x Libraries used.
- 1 x .dat file (Binary)
- 2 x .csv files (In / Out)

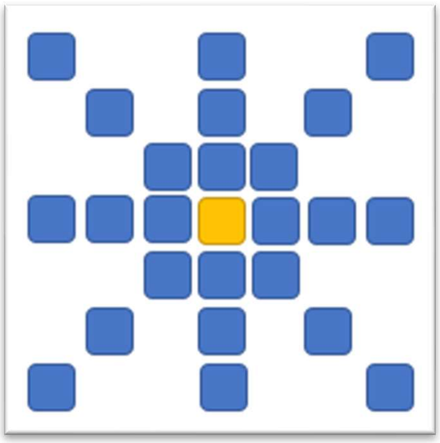
It took me about 16 hours of work to complete the program.

3. Description

I am going to define the complexity of this program in 3 main areas. How to define winner. When is a Draw. How to make the computer play Automatically without having to create a separate code just for that.

a. Define a Winner:

To define a winner, I decided to create 4 functions that will review the 4 closest positions to each position on the board and determine if there are at least 4 in a horizontal, vertical, 45-degree diagonal or 135-degree diagonal with the same value, if player 1 wins it is because at least 4 have value 1, if player 2 wins it is because at least 4 have value 2.



The 3 Functions to define a winner are:

```
//EVALUATE HORIZONTAL
horizontal = evaluateHorizontal(positions, grid, option);
//EVALUATE VERTICAL
vertical = evaluateVertical(positions, grid, option);
//EVALUATE DIAGONAL 45 DEGREES
d45d = evaluate45Degrees(positions, grid, option);
//EVALUATE DIAGONAL 135 DEGREES
d135d = evaluate135Degrees(positions, grid, option);
```

The Basic of the Logic for Each Function is this (All 4 Function use the same logic with small changes):

```
for (int pos1 = 0; pos1 < positions; pos1++) {
        col = grid.position[pos1].y;
        row = grid.position[pos1].x;
        value = grid.position[pos1].value;
        if (value > 0) {
                for (int x = row; x > row - 4; x--) {
                        for (int pos2 = 0; pos2 < positions; pos2++) {
                                if (grid.position[pos2].x == x && grid.position[pos2].y == col &&
                                grid.position[pos2].value == value) {
                                        control1++;
                                }
```

```
                    }
            }
    }
    if (control1 >= 4) {
            pos1 = positions;
            won = true;
    }
    else {
            control1 = 0;
    }
}
```

b. Draw:

To know when a Draw occurred, I used 2 variables and a Function:

- Int positions: **Total Positions available in a Board.**
- int movements: **Counts how many positions were played already in the Board**
- int winLogic(int positions, Grid grid, int option): **If someone Won, return 1, if nobody won Return 0.**

If (movements >= positions && won == 0): **It is a Draw.**

c. Computer Plays by Itself:

When the option Player VS PC or PC Vs PC is chosen, the first thing the program does is assign the name of the player or players that are PCs as COMPUTER.

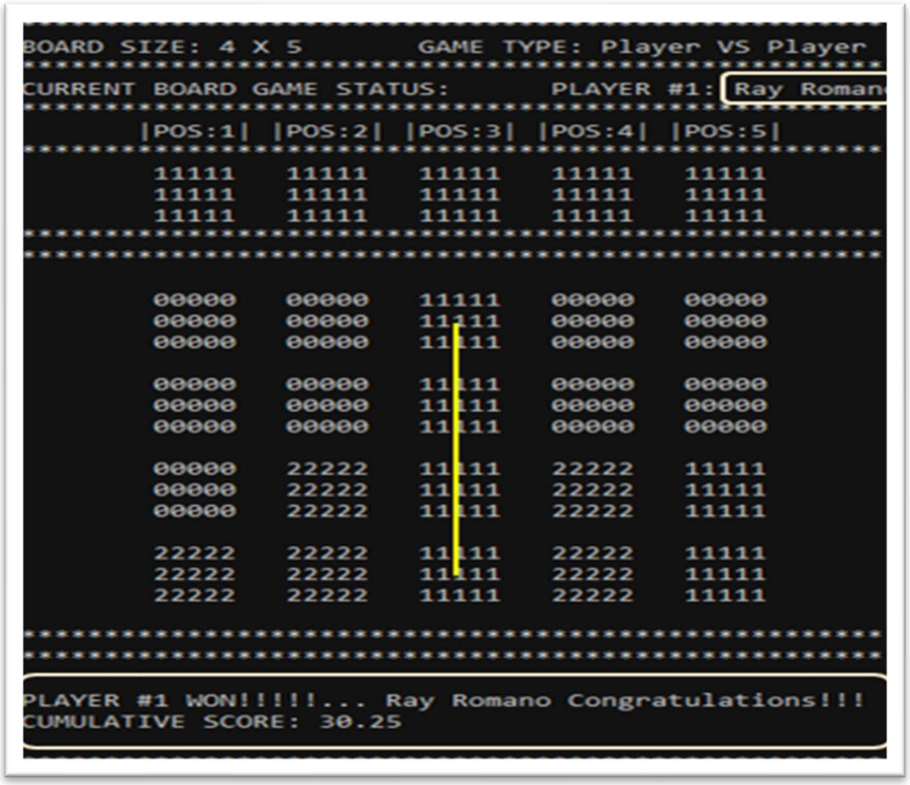
For the Computer to play by itself, it use the same functions created for a human player but, if the name of the current player is COMPUTER the program ignores the keyboard input and uses a random number between 1 and the maximum columns of the board, if the move is valid it processes it, if it is not valid, it tries again until the move is valid, so the computer plays randomly.

```
int randNum = 0; //Integer that will Generate a Random Number
stringstream ss;
while (control == 0) {
    control2 = 0;
    control3 = 0;
    control = 1;
    cout << endl;
    cout << "\t\t-> ";
    randNum = rand() % (cols - 1 + 1) + 1; //Generates a Random int Between 1 and Max Columns.
    if (playername == "COMPUTER") { //Decide if use the Random number or Read from Keyboard
            ss << randNum;
            n = ss.str(); //Assign Random Number to Variable n
            ss.str("");
            ss.clear();
            cout << n << endl;
            cout << "\t\tPress Enter Process COMPUTER SELECTION...";
            getchar();
    }
```

```
else {
        getline(cin, n); //Read from Keyboard if the Player is Human
}
```

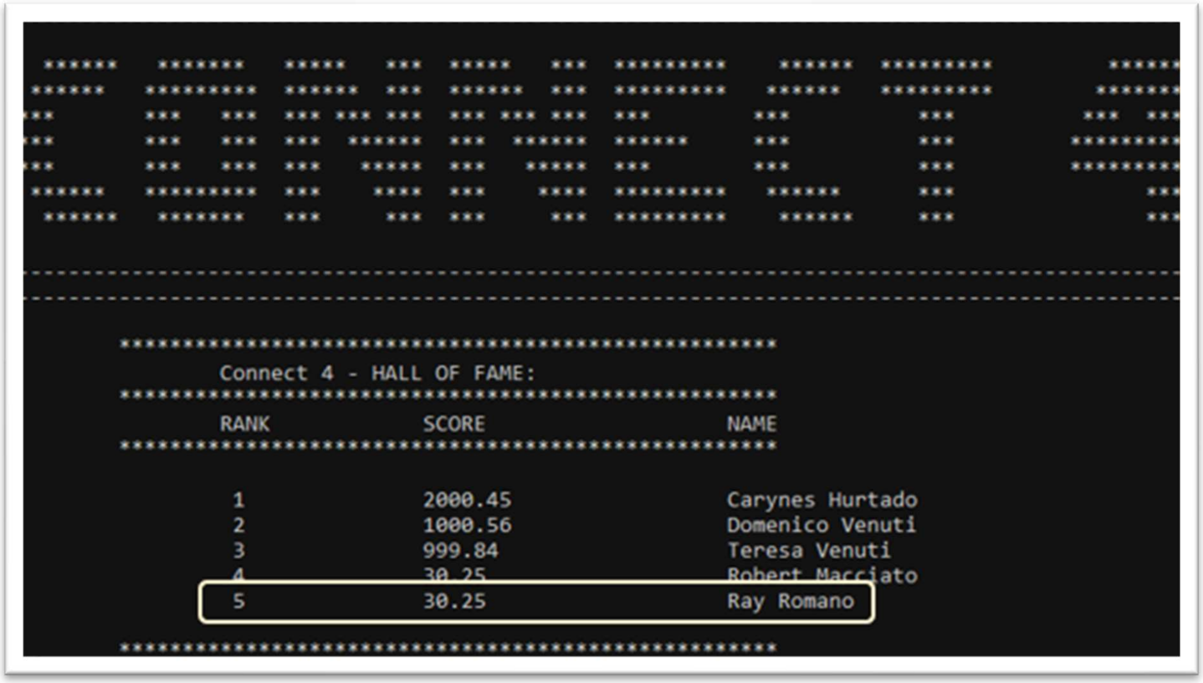
a. Sample In/Out.

Player 1 (Ray Romano) win game with a Vertical Line of 4 in Column 3 on a 4 X 5 Board Size Game. Because of that, earns 30.25 Points and Show the New Score in the Scores Table.



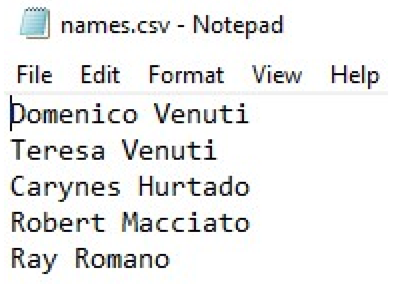
```
BOARD SIZE: 4 X 5          GAME TYPE: Player VS Player
*************************************************************
CURRENT BOARD GAME STATUS:         PLAYER #1: Ray Roman
*************************************************************
          |POS:1| |POS:2| |POS:3| |POS:4| |POS:5|
*************************************************************
           11111   11111   11111   11111   11111
           11111   11111   11111   11111   11111
           11111   11111   11111   11111   11111
*************************************************************
*************************************************************

           00000   00000   11111   00000   00000
           00000   00000   11111   00000   00000
           00000   00000   11111   00000   00000

           00000   00000   11111   00000   00000
           00000   00000   11111   00000   00000
           00000   00000   11111   00000   00000

           00000   22222   11111   22222   11111
           00000   22222   11111   22222   11111
           00000   22222   11111   22222   11111

           22222   22222   11111   22222   11111
           22222   22222   11111   22222   11111
           22222   22222   11111   22222   11111

*************************************************************
*************************************************************

PLAYER #1 WON!!!!!... Ray Romano Congratulations!!!
CUMULATIVE SCORE: 30.25
```

Ray Won 30.24 Points



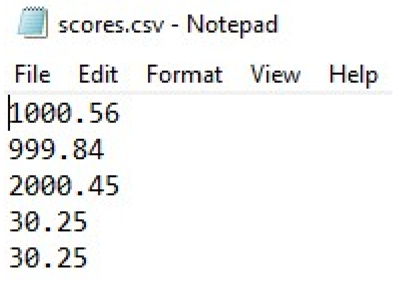
```
  ****************************************************
        Connect 4 - HALL OF FAME:
  ****************************************************
        RANK            SCORE                  NAME
  ****************************************************

        1               2000.45                Carynes Hurtado
        2               1000.56                Domenico Venuti
        3               999.84                 Teresa Venuti
        4               30.25                  Robert Macciato
        5               30.25                  Ray Romano

  ****************************************************
```

Ray is un the Score Board as 5 with 30.25 Points



```
Domenico Venuti
Teresa Venuti
Carynes Hurtado
Robert Macciato
Ray Romano
```

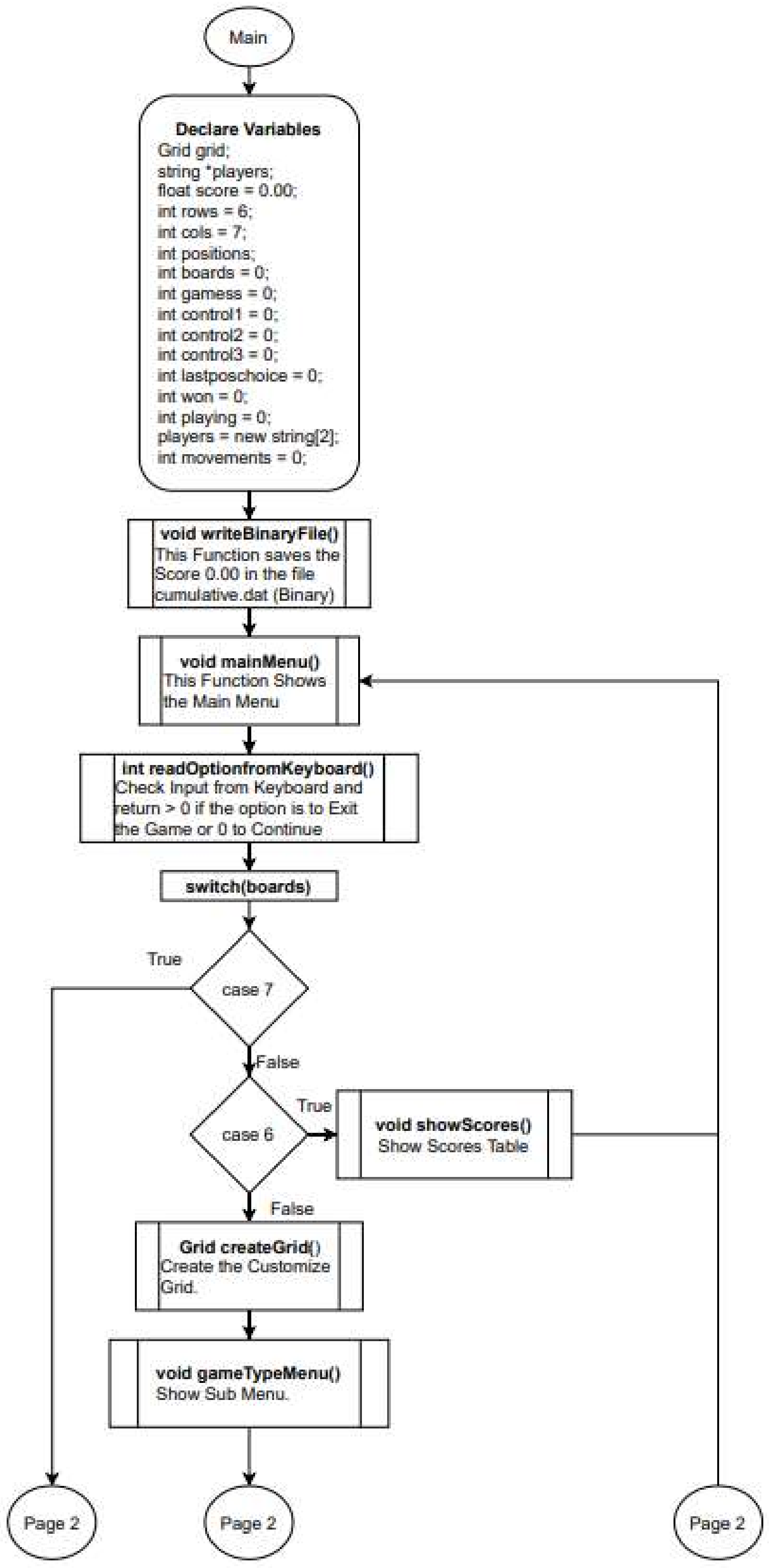
Ray’s Name is the Last one in File names.csv

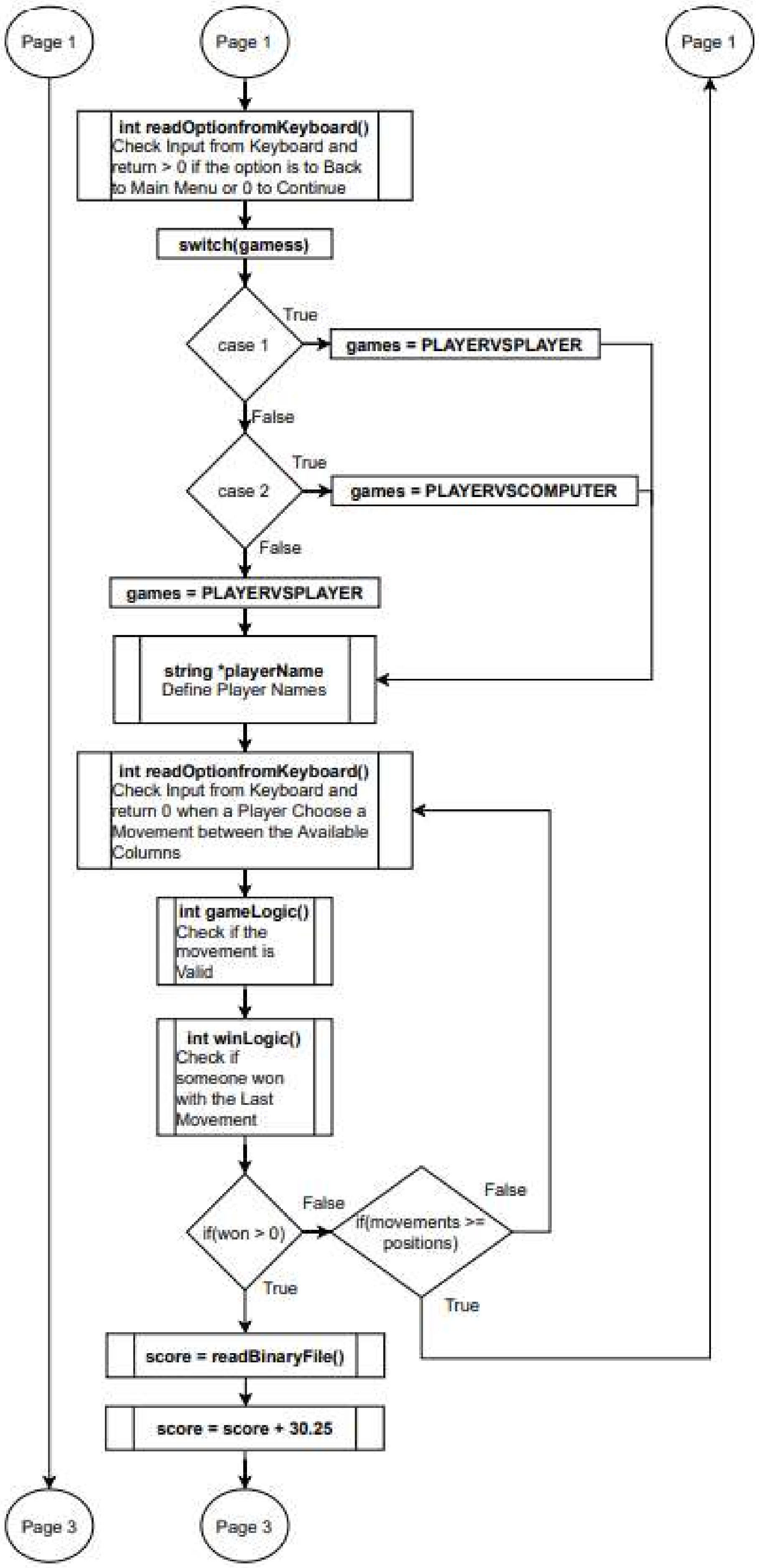


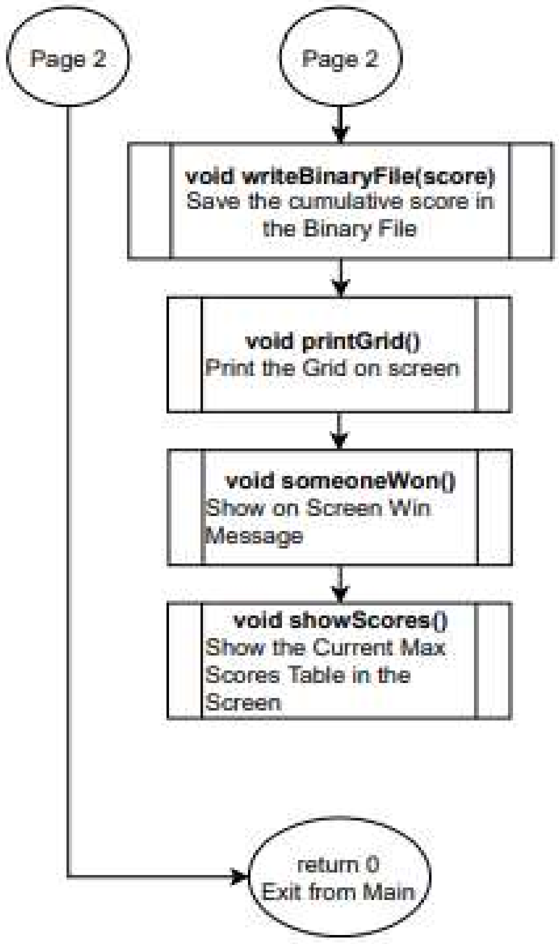
```
1000.56
999.84
2000.45
30.25
30.25
```

Ray’s Score is the Last one in File scores.csv

b. FlowChart


Main
Declare Variables
Grid grid;
string *players;
float score = 0.00;
int rows = 6;
int cols = 7;
int positions;
int boards = 0;
int gamess = 0;
int control1 = 0;
int control2 = 0;
int control3 = 0;
int lastposchoice = 0;
int won = 0;
int playing = 0;
players = new string[2];
int movements = 0;
void writeBinaryFile()
This Function saves the Score 0.00 in the file cumulative.dat (Binary)
void mainMenu()
This Function Shows the Main Menu
int readOptionfromKeyboard()
Check Input from Keyboard and return > 0 if the option is to Exit the Game or 0 to Continue
switch(boards)
case 7
True
False
case 6
True
void showScores()
Show Scores Table
False
Grid createGrid()
Create the Customize Grid.
void gameTypeMenu()
Show Sub Menu.
Page 2
Page 2
Page 2


Page 1
Page 1
Page 1
int readOptionfromKeyboard()
Check Input from Keyboard and return > 0 if the option is to Back to Main Menu or 0 to Continue
switch(gamess)
case 1
True
games = PLAYERVSPLAYER
False
case 2
True
games = PLAYERVSCOMPUTER
False
games = PLAYERVSPLAYER
string *playerName
Define Player Names
int readOptionfromKeyboard()
Check Input from Keyboard and return 0 when a Player Choose a Movement between the Available Columns
int gameLogic()
Check if the movement is Valid
int winLogic()
Check if someone won with the Last Movement
if(won > 0)
False
if(movements >= positions)
False
True
True
score = readBinaryFile()
score = score + 30.25
Page 3
Page 3


Page 2
Page 2
void writeBinaryFile(score)
Save the cumulative score in the Binary File
void printGrid()
Print the Grid on screen
void someoneWon()
Show on Screen Win Message
void showScores()
Show the Current Max Scores Table in the Screen
return 0
Exit from Main

c. Major Variables

- enum Games {PLAYERVSPLAYER, PLAYERVSCOMPUTER, RANDOM} games;

Define What Type of Game the User Chooses (Player VS Player, Player VS PC or PC VS PC).

- enum Boardsize {CLASSIC7X6, X54, X65, X87, X97} boardsize;

Define What Board Size the User Chooses (6 X 7, 4 X 5, 5 X 6, 7 X 8, 7 X 9).

- enum Player {NONE, PLAYER1, PLAYER2} player;

Define What Player is Currently Playing.

- struct Grid {

```
        int rows = 0;
        int cols = 0;
        Position *position = nullptr;
};
```

Struct that Represent the Game Board.
It has 3 Parameters, how Many Columns, how many Rows and an Array of a Structure Position.

-
  ```
  struct Position {
          int x;
          int y;
          int value;
  };
  ```
  Nested Structure that Represents each Board Position, for example, in a Board of 6 X 7 are 42 positions in the Board, each Position has 3 Parameters, x, y and the value that can be 0, 1 or 2, o means the position is empty, 1 means player 1 has a piece in that position and 2 means player 2 has a piece in that position.

- int positions;

Total of Positions in the Current Board, for example, in a Board of 6 X 7 are 42 positions.

- int lastposchoice = 0;

Last position played.

- int won = 0;

0 = Nobody Won Yet, 1 = Someone Won.

- int playing = 0;

0 = No game is Open. 1 = Playing Game.

- players = new string[2];

Array with Player's Names. players[0] = Player 1's Name. players[1] = Player 2's Name.

- int movements = 0;

How Many Movements were made, it is use to know if it is a Draw. After complete the Whole Board if won = 0 still, it is a Draw.

- ifstream infile;
  ifstream infile2;
  These 2 Variables represents the 2 .csv Files and Binary File used. (IN).

- vector<float> scorev;

Vector to Save all Scores from scores.csv

- vector<string> namesv;

Vector to Save all Scores from names.csv

- ofstream outfile;
  ofstream outfile2;
  These 2 Variables represents the 2 .csv Files and Binary File used. (OUT).

d. Concepts

✓ **Pointer Variables/Memory Allocation:**

- int positions; (Line 828)
  grid = createGrid(rows, cols, &positions); (Line 887)
  Variable positions will change value inside the Function createGrid because you can choose different Board Sizes and depending of the size, the value of postions changes, this value needs to be available in a lot of places, that's why I decided to use a Pointer for it.

- string *players; (Line 824)
  Players is a Pointer Array, was defined like that because will be initialized later from a function. And will take all the characteristics from the Array returned from that function. players = playerName(); (Line 909)

- void showScores(int option, float cumulativescore, int cols, string playername) (Line 698)
  Function Parameters Example. In total you will find 19 Funtions in this Program

- Position* position = nullptr; //Define Position Structure (Line 42)
  position = new Position[MAX_SIZE]; (Line 45)
  Pointer Array Dinamically Created.

- return value; (Line 403)
  Return Parameter Example. Integer Returned from Function readOptionfromKeyboard.

✓ **Char Arrays and Strings:**

- cumulative = stof(cumulativet); (Line 693)
  Convert String to Float.

- stringstream ss; //Concatenate all Characters to Show the Grid (Line 177)
  Stringstream to Concatenate Grid Values and Show as a Graphic Interface.

✓ **Structured Data:**

-
  ```
  struct Position { (Line 25)
          int x;
          int y;
          int value;
  };
  ```

  ```
  Position* position = nullptr; //Define Position Structure (Line 42)
  ```

  ```
  for (int x = 0; x < rows; x++) { (Line 49)
          for (int y = 0; y < cols; y++) {
  ```

```
                position[pos].x = x;
                position[pos].y = y;
                position[pos].value = 0;
                pos++;
        }
}
```
Pointer of Array of Structure, position is used as a array inside Grid Struct.

-
  ```
  struct Grid { (Line 31)
          int rows = 0;
          int cols = 0;
          Position *position = nullptr;
  };
  ```
  Structure that Represents the Board Grid and is Nested to Position Structure.

-
  ```
  printGrid(&positions, &grid, players, won);
  ```
  Grid Structure passed as an argument to Function printGrid.

-
  ```
  Grid createGrid(int rows, int cols, int* positions) { (Line 40)
  ```
  Return Grid Structure to Create the Grid.

-
  ```
  enum Games {PLAYERVSPLAYER, PLAYERVSCOMPUTER, RANDOM} games; (Line 20)
  enum Boardsize {CLASSIC7X6, X54, X65, X87, X97} boardsize;
  enum Player {NONE, PLAYER1, PLAYER2} player;
  ```
  Enums.

✓ <u>Binary Files - In/Out Files:</u>

-
  ```
  string filename = "scores.csv"; (Line 703)
  string filename2 = "names.csv";
  if (option == 1) { (Line 717)
          outfile.open(filename, ios::app);
          outfile2.open(filename2, ios::app);
          if (outfile.is_open() && outfile2.is_open()) {
                  outfile << endl;
                  outfile2 << endl;
                  outfile << fixed << setprecision(2) << cumulativescore;
                  outfile2 << playername;
                  y++;
          }
          if (y == 0) {
                  cout << endl;
                  cout << "\t\tFiles couldn't be Created, Data was not transfered..." << endl;
                  cout << endl;
          }
          outfile.close();
  ```

```
        outfile2.close();
}
infile.open(filename, ios::in);
infile2.open(filename2, ios::in);
if (infile.is_open() && infile2.is_open()) {
        while (getline(infile, line)) {
                scorev.push_back(stof(line));
                x++;
        }
        infile.close();
        if (x > 0) {
                x = 0;
                while (getline(infile2, line2)) {
                        namesv.push_back(line2);
                        x++;
                }
        }
        infile2.close();
```

Piece of code that Format and Append Data to 2 .csv files and Read the Content from the Files again in the same Function.

- string name = "cumulative.dat"; (Line 665)

```
ofstream outFile;

outFile.open(name, ios::out | ios::binary);
if (outFile.is_open()) {
        outFile << fixed << setprecision(2) << cumulative;
}
else {
        cout << "File was not Found or can't be opened...";
}
outFile.close();
```

Piece of code that write in a .dat Binary File.

4. Program

```
/*
* File: main.cpp
* Author: Domenico Venuti
* Created on November 14, 2021 at 8:00 AM
* Purpose: Project 1 - CIS-17A - C++ Objects - Board Game - Connect 4
*/

//LIBRARIES
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <fstream>
#include <vector>
#include <random>

//NAMESPACE STD
using namespace std;

//DEFINE CONSTANTS
const int MAX_ROWS = 10;
const int MAX_COLS = 7;
const int MAX_SIZE = 70;
const int MAX_ARRAY = 50;

//DEFINE ENUMERATION
enum Games { PLAYERVSPLAYER, PLAYERVSCOMPUTER, RANDOM } games;
enum Boardsize { CLASSIC7X6, X54, X65, X87, X97 } boardsize;
enum Player { NONE, PLAYER1, PLAYER2 } player;

//DEFINE STRUCTURES
struct Position { //Each Position in the Board have some Parameters as x, y and the Value
        int x;
        int y;
        int value;
};

struct Grid { //The Board Game Grid Structure, have some parameters as rows, cols and an Dinamic array of
Structure named Position for each position in the Grid (Nested Structures).
        int rows = 0;
        int cols = 0;
        Position* position = nullptr;
};
```

```
//FUNCTIONS
Grid createGrid(int, int, int*);
string* playerName();
void printPlayerOptions(Grid*);
void printGrid(int*, Grid*, string*, int);
void mainMenu();
int readOptionfromKeyboard(int, int*, int, string);
void gameTypeMenu();
int gameLogic(int, Grid, int*);
bool evaluateHorizontal(int, Grid, int);
bool evaluateVertical(int, Grid, int);
bool evaluate45Degrees(int, Grid, int);
bool evaluate135Degrees(int, Grid, int);
int winLogic(int, Grid, int);
void someoneWon(string*, int, float);
void nobodyWon(int);
void writeBinaryFile(float);
float readBinaryFile();
void showScores(int, float, int, string);

//MAIN
int main(int argc, char** argv)
{
	Grid grid;
	string* players;
	float score = 0.00;
	int rows = 6;
	int cols = 7;
	int positions;
	int boards = 0;
	int gamess = 0;
	int control1 = 0;
	int control2 = 0;
	int control3 = 0;
	int lastposchoice = 0;
	int won = 0;
	int playing = 0;
	players = new string[2];
	int movements = 0;

	writeBinaryFile(0.00);
	while (control1 == 0) {
		movements = 0;
		control2 = 0;
		control3 = 0;
		player = NONE;
		mainMenu();
```

```
control1 = readOptionfromKeyboard(0, &boards, cols, "NONE");
switch (boards)
{
case 1:
        boardsize = CLASSIC7X6;
        rows = 6;
        cols = 7;
        break;
case 2:
        boardsize = X54;
        rows = 4;
        cols = 5;
        break;
case 3:
        boardsize = X65;
        rows = 5;
        cols = 6;
        break;
case 4:
        boardsize = X87;
        rows = 7;
        cols = 8;
        break;
case 5:
        boardsize = X97;
        rows = 7;
        cols = 9;
        break;
case 6:
        showScores(0, 0, 7, "");
        cout << endl;
        cout << "\t\tPress Enter to Go to the Main Menu...";
        getchar();
        cout << endl;
        cout << endl;
        break;
default:
        break;
}
if (boards != 6) {
        grid = createGrid(rows, cols, &positions);
        if (control1 == 0) {
                if (playing == 0) {
                        gameTypeMenu();
                        control2 = readOptionfromKeyboard(1, &gamess, cols, "NONE");
                        switch (gamess)
                        {
```

```
                        case 1:
                            games = PLAYERVSPLAYER;
                            break;
                        case 2:
                            games = PLAYERVSCOMPUTER;
                            break;
                        case 3:
                            games = RANDOM;
                            break;
                        default:
                            break;
                        }
                    }
                    if (control2 == 0) {
                        if (playing == 0) {
                            players = playerName();
                        }
                        playing = 1;
                        while (control2 == 0) {
                            if (player == NONE) {
                                player = PLAYER1;
                            }
                            else if (player == PLAYER1) {
                                player = PLAYER2;
                            }
                            else {
                                player = PLAYER1;
                            }
                            if (control3 == 0) {
                                printGrid(&positions, &grid, players, won);
                                control3 = 2;
                                while (control3 > 0) {
                                    control3 = readOptionfromKeyboard(2,
&lastposchoice, cols, players[player - 1]);

                                    control3 = gameLogic(positions, grid,
&lastposchoice);

                                    won = winLogic(positions, grid, lastposchoice);
                                    if (control3 == 0) {
                                        movements++;
                                    }
                                    if (won > 0) {
                                        score = readBinaryFile(); //READ
CUMULATIVE SCORE TO BE READY FOR A NEW

                                        score = score + 30.25;
                                        writeBinaryFile(score);        //WRITE
SCORE IN A BINARY FILE

                                        printGrid(&positions, &grid, players, won);
                                        control3 = 0;
                                        control2 = 1;
                                        control1 = 0;
                                        someoneWon(players, grid.cols, score);
                                        showScores(1, score, grid.cols,
players[player - 1]);
                                        won = 0;
                                        cout << endl;
                                        cout << "\t\tPress Enter to Go to the Main
Menu...";
                                        getchar();
                                        cout << endl;
                                        cout << endl;
                                    }
                                    else {
                                        if (movements >= positions) {
                                            nobodyWon(grid.cols);
                                            control3 = 0;
                                            control2 = 1;
                                            control1 = 0;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        cout << endl;
        cout << "\t\t------------------------------------------------------------------------------------------" << endl;
        cout << "\t\tSee Your Later!!!... Thanks for Play Connect 4... Come Back Soon." << endl;
        cout << "\t\t------------------------------------------------------------------------------------------" << endl;
        return 0;
}

//Function that Returns Structure to Initialize the Board Game Grid
Grid createGrid(int rows, int cols, int* positions) {
        Grid grid; //Define Grid structure
        Position* position = nullptr; //Define Position Structure
        int pos = 0;
        *positions = rows * cols;
        position = new Position[MAX_SIZE];
        grid.rows = rows;
        grid.cols = cols;
```

```
	for (int x = 0; x < rows; x++) { //Initialize all Positions in the Board Game Grid as 0 (No movement made
yet by any player).
		for (int y = 0; y < cols; y++) {
			position[pos].x = x;
			position[pos].y = y;
			position[pos].value = 0;
			pos++;
		}
	}
	grid.position = position;
	return grid; //Return Board Game Grid Initialized.
}

//Define Player Names
string* playerName() {
	static string name[2];
	string n;
	int control = 0;
	int control2 = 0;
	int control3 = 0;

	if (games != RANDOM) {
		while (control == 0) {
			control = 1;
			control2 = 0;
			control3 = 0;
			cout << endl;
			cout << "\t\tPlease, Insert the Name for Player #1:" << endl;
			cout << "\t\t-> ";
			getline(cin, n);
			for (char& c : n) {
				control2 = 1;
				if (isalpha(c) || isdigit(c)) {
					control3 = 1;
				}
				else {
					if (control3 != 1) {
						control = 0;
					}
				}
			}
			if (control == 1 && control2 == 1) { //At least 1 Character was type
				name[0] = n;
			}
			else {
				cout << "\t\tYou must Type At Least 1 Character, no Special Characters Allowed...
Please, try Again." << endl;
```

```
                    control = 0;
                }
                cin.clear();
                cin.sync();
                n = "";
            }
            control = 0;
            control2 = 0;
            control3 = 0;
        }
        else {
            name[0] = "COMPUTER";
        }
        if (games == PLAYERVSPLAYER) {
            while (control == 0) {
                control = 1;
                control2 = 0;
                control3 = 0;
                cout << endl;
                cout << "\t\tPlease, Insert the Name for Player #2:" << endl;
                cout << "\t\t-> ";
                getline(cin, n);
                for (char& c : n) {
                    control2 = 1;
                    if (isalpha(c) || isdigit(c)) {
                        control3 = 1;
                    }
                    else {
                        if (control3 != 1) {
                            control = 0;
                        }
                    }
                }
                if (control == 1 && control2 == 1) { //At least 1 Character was type
                    name[1] = n;
                }
                else {
                    cout << "\t\tYou must Type At Least 1 Character, no Special Characters Allowed...
Please, try Again." << endl;
                    control = 0;
                }
                cin.clear();
                cin.sync();
                n = "";
            }
        }
        else {
```

```
		name[1] = "COMPUTER";
	}
	return name; //Return Player Name
}

//FUNCTION TO PRINT OPTIONS FOR CURRENT PLAYER
void printPlayerOptions(Grid* grid) {
	int cols = grid->cols;
	int colhelp = cols;
	string printChar[2] = { "11111","22222" };

	cout << endl;
	for (int x = 0; x < 3; x++) {
		cout << "\t\t      ";
		for (int y = 0; y < colhelp; y++) {
			if (player == PLAYER1) {
				cout << " " << printChar[0] << "  ";
			}
			else {
				cout << " " << printChar[1] << "  ";
			}
		}
		cout << endl;
	}
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
}

//FUNCTION TO PRINT THE BOARD GRID
void printGrid(int* positions, Grid* grid, string* players, int won) {
	stringstream ss; //Concatenate all Characters to Show the Grid simulatinga a Graphic Interface.
	string coutstring; //Each line to Print in Screen
	int rows = grid->rows;
	int cols = grid->cols;
	int pos = 0;
	int colhelp = cols;
	string printChar[3] = { "00000","11111","22222" };
	string boardstring = "";
	string gametypestring = "";
```

```
//BOAR SIZE TITLE
switch (boardsize)
{
case CLASSIC7X6:
        boardstring = "6 X 7";
        break;
case X54:
        boardstring = "4 X 5";
        break;
case X65:
        boardstring = "5 X 6";
        break;
case X87:
        boardstring = "7 X 8";
        break;
case X97:
        boardstring = "7 X 9";
        break;
default:
        boardstring = "NONE";
        break;
}
//TYPE GAME TITLE
switch (games)
{
case PLAYERVSPLAYER:
        gametypestring = "Player VS Player";
        break;
case PLAYERVSCOMPUTER:
        gametypestring = "Player VS PC";
        break;
case RANDOM:
        gametypestring = "PC VS PC";
        break;
default:
        gametypestring = "NONE";
        break;
}
ss.str("");
ss.clear();
cout << endl;
cout << "\t\t***";
for (int x = 0; x < colhelp + 2; x++) {
        cout << "*******";
}
cout << endl;
cout << "\t\tBOARD SIZE: " << boardstring << "\tGAME TYPE: " << gametypestring;
```

```
	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
	cout << "\t\tCURRENT BOARD GAME STATUS:\t" << "PLAYER #" << player << ": " << players[(int)player -
1] << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
	cout << "\t\t       ";
	for (int x = 0; x < colhelp; x++) {
		cout << "|POS:" << x + 1 << "| ";
	}
	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	printPlayerOptions(grid);
	cout << endl;
	cout << endl;
	for (int x = 0; x < rows; x++) {
		for (int y = 0; y < cols; y++) {
			if (grid->position[pos].value == 0) {
				ss << "\t" << printChar[0];
			}
			else if (grid->position[pos].value == 1) {
				ss << "\t" << printChar[1];
			}
			else {
				ss << "\t" << printChar[2];
			}

			if (y == cols - 1) {
				cout << "\t\t       ";
				cout << ss.str() << endl;
				cout << "\t\t       ";
				cout << ss.str() << endl;
				cout << "\t\t       ";
				cout << ss.str() << endl;
				cout << "\t";
				ss.str("");
				ss.clear();
```

```
                    }
                    pos++;
               }
               cout << endl;
          }
          cout << "\t\t***";
          for (int x = 0; x < colhelp + 2; x++) {
               cout << "*******";
          }
          if (won == 0) {
               cout << endl;
               cout << "\t\tChoose a Position to Play:";
          }
}

//MAIN MENU
void mainMenu() {
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << "\t    ----------  Connect 4 - Board Game - by Domenico Venuti, Version V1.0  ----------     " <<
endl;
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << endl;
          cout << "\t  ******   *******   *****   ***  *****   ***  *********    ******  *********       ******"
<< endl;
          cout << "\t ******   *********  ******  ***  ******  ***  *********   ******   *********
*******" << endl;
          cout << "\t***      ***   ***  *** *** ***  *** *** ***  ***      ***       ***        ***  ***" << endl;
          cout << "\t***      ***   ***  ***  ******  ***  ******  ******    ***       ***       *********" <<
endl;
          cout << "\t***      ***   ***  ***   *****  ***   *****  ***      ***       ***       *********" << endl;
          cout << "\t ******   *********  ***    ****  ***    ****  *********   ******     ***            ***" <<
endl;
          cout << "\t  ******   *******   ***    ***  ***    ***  *********    ******    ***            ***" << endl;
          cout << endl;
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << "\tMAIN MENU:" << endl;
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << "\t\tChoose a Board Size:" << endl;
          cout << "\t------------------------------------------------------------------------------------------" << endl;
          cout << endl;
          cout << "\t\t1. Classic Connect 4 - Grid (6 X 7)." << endl;
          cout << "\t\t2. Connect 4 - Grid (4 X 5)." << endl;
          cout << "\t\t3. Connect 4 - Grid (5 X 6)." << endl;
          cout << "\t\t4. Connect 4 - Grid (7 X 8)." << endl;
          cout << "\t\t5. Connect 4 - Grid (7 X 9)." << endl;
          cout << "\t\t6. Scores." << endl;
```

```
	cout << "\t\t7. Exit Game." << endl;
}

//OPTIONS FROM KEYBOARD
int readOptionfromKeyboard(int menuoption, int* option, int cols, string playername) {
	string n;
	int control = 0;
	int control2 = 0;
	int control3 = 0;
	int value = 0;
	int randNum = 0;
	stringstream ss;

	while (control == 0) {
		control2 = 0;
		control3 = 0;
		control = 1;
		cout << endl;
		cout << "\t\t-> ";
		randNum = rand() % (cols - 1 + 1) + 1;
		if (playername == "COMPUTER") {
			ss << randNum;
			n = ss.str();
			ss.str("");
			ss.clear();
			cout << n << endl;
			cout << "\t\tPress Enter Process COMPUTER SELECTION...";
			getchar();
		}
		else {
			getline(cin, n);
		}
		if (n.length() < 2 && n.length() > 0) {
			for (char& c : n) {
				control2 = 1;
				if (isdigit(c) && control == 1) {
					*option = (int)c - 48;
					if (menuoption == 0 && *option < 8 && *option > 0) {
						control3 = 1;
					}
					else if (menuoption == 1 && *option < 5 && *option > 0) {
						control3 = 1;
					}
					else if (menuoption == 2 && *option <= cols && *option > 0) {
						control3 = 1;
					}
				}
```

```
                    else {
                        if (control3 != 1) {
                            control = 0;
                        }
                    }
                }
            }
            if (control3 == 0) {
                cout << "\t\tWrong Option Typed, no Special Characters Allowed... Please, try Again." <<
endl;
                control = 0;
            }
            cin.clear();
            cin.sync();
            n = "";
        }
        if (menuoption == 0) {
            if (*option == 7) {
                value = 7;
            }
            else {
                value = 0;
            }
        }
        else if (menuoption == 1) {
            if (*option == 4) {
                value = 4;
            }
            else {
                value = 0;
            }
        }
        else {
            value = 0;
        }
        return value;
}

//GAME TYPE OPTIONS
void gameTypeMenu() {
        cout << endl;
        cout << "\t-----------------------------------------------------------------------------------------------" << endl;
        cout << "\t-----------------------------------------------------------------------------------------------" << endl;
        cout << "\t\tChoose a Game Type:" << endl;
        cout << "\t-----------------------------------------------------------------------------------------------" << endl;
        cout << endl;
        cout << "\t\t1. Player VS Player." << endl;
        cout << "\t\t2. Player VS PC." << endl;
        cout << "\t\t3. PC VS PC." << endl;
        cout << "\t\t4. Back." << endl;
}

//GAME LOGIC
int gameLogic(int positions, Grid grid, int* option) {
        int value = 0;
        int counter = 0;
        int lastx = 0;
        int playervalue = 1;
        int column = *option - 1;

        switch (player)
        {
        case PLAYER1:
                playervalue = 1;
                break;
        default:
                playervalue = 2;
                break;
        }
        for (int x = 0; x < positions; x++) {
                if (grid.position[x].y == column && grid.position[x].value == 0) {
                        lastx = x;
                        counter++;
                }
        }
        if (counter > 0) {
                grid.position[lastx].value = playervalue;
                *option = lastx;
                value = 0;
        }
        else {
                value = 1;
                cout << endl;
                cout << "\t\tYou can't make that Move, Please, Choose a Different Position." << endl;
        }
        return value;
}

//EVALUATE HORIZONTAL
bool evaluateHorizontal(int positions, Grid grid, int option) {
        int col;
        int row;
        int value;
        int control1 = 0;
```

```
	bool won = false;

	for (int pos1 = 0; pos1 < positions; pos1++) {
		col = grid.position[pos1].y;
		row = grid.position[pos1].x;
		value = grid.position[pos1].value;
		if (value > 0) {
			for (int y = col; y < col + 4; y++) {
				for (int pos2 = 0; pos2 < positions; pos2++) {
					if (grid.position[pos2].x == row && grid.position[pos2].y == y &&
grid.position[pos2].value == value) {
						control1++;
					}
				}
			}
		}
		if (control1 >= 4) {
			pos1 = positions;
			won = true;
		}
		else {
			control1 = 0;
		}
	}
	return won;
}

//EVALUATE VERTICAL
bool evaluateVertical(int positions, Grid grid, int option) {
	int col;
	int row;
	int value;
	int control1 = 0;
	bool won = false;

	for (int pos1 = 0; pos1 < positions; pos1++) {
		col = grid.position[pos1].y;
		row = grid.position[pos1].x;
		value = grid.position[pos1].value;
		if (value > 0) {
			for (int x = row; x > row - 4; x--) {
				for (int pos2 = 0; pos2 < positions; pos2++) {
					if (grid.position[pos2].x == x && grid.position[pos2].y == col &&
grid.position[pos2].value == value) {
						control1++;
					}
				}
```

```
                    }
                }
                if (control1 >= 4) {
                    pos1 = positions;
                    won = true;
                }
                else {
                    control1 = 0;
                }
            }
            return won;
}

//EVALUATE DIAGONAL 45 DEGREES
bool evaluate45Degrees(int positions, Grid grid, int option) {
            int col;
            int row;
            int value;
            int control1 = 0;
            bool won = false;
            int y = 0;

            for (int pos1 = 0; pos1 < positions; pos1++) {
                col = grid.position[pos1].y;
                row = grid.position[pos1].x;
                value = grid.position[pos1].value;
                if (value > 0) {
                    y = col - 1;
                    for (int x = row; x > row - 4; x--) {
                        if (y < col + 4) {
                            y = y + 1;
                        }
                        for (int pos2 = 0; pos2 < positions; pos2++) {
                            if (grid.position[pos2].x == x && grid.position[pos2].y == y &&
grid.position[pos2].value == value) {
                                control1++;
                            }
                        }
                    }
                }
                if (control1 >= 4) {
                    pos1 = positions;
                    won = true;
                }
                else {
                    control1 = 0;
                }
```

```
	}
	return won;
}

//EVALUATE DIAGONAL 135 DEGREES
bool evaluate135Degrees(int positions, Grid grid, int option) {
	int col;
	int row;
	int value;
	int control1 = 0;
	bool won = false;
	int y = 0;

	for (int pos1 = 0; pos1 < positions; pos1++) {
		col = grid.position[pos1].y;
		row = grid.position[pos1].x;
		value = grid.position[pos1].value;
		if (value > 0) {
			y = col - 1;
			for (int x = row; x < row + 4; x++) {
				if (y < col + 4) {
					y = y + 1;
				}
				for (int pos2 = 0; pos2 < positions; pos2++) {
					if (grid.position[pos2].x == x && grid.position[pos2].y == y &&
grid.position[pos2].value == value) {
						control1++;
					}
				}
			}
		}
		if (control1 >= 4) {
			pos1 = positions;
			won = true;
		}
		else {
			control1 = 0;
		}
	}
	return won;
}

//MAIN GAME LOGIC FUNCTION
int winLogic(int positions, Grid grid, int option) {
	int won = 0;
	bool horizontal = false;
	bool vertical = false;
	bool d45d = false;
	bool d135d = false;

	//EVALUATE HORIZONTAL
	horizontal = evaluateHorizontal(positions, grid, option);
	//EVALUATE VERTICAL
	vertical = evaluateVertical(positions, grid, option);
	//EVALUATE DIAGONAL 45 DEGREES
	d45d = evaluate45Degrees(positions, grid, option);
	//EVALUATE DIAGONAL 135 DEGREES
	d135d = evaluate135Degrees(positions, grid, option);
	if (player == PLAYER1 && (horizontal == true || vertical == true || d45d == true || d135d == true)) {
		won = 1;
	}
	else if (player == PLAYER2 && (horizontal == true || vertical == true || d45d == true || d135d == true))
{
		won = 2;
	}
	else {
		won = 0;
	}
	return won;
}

//SOMEBODY WON
void someoneWon(string* players, int cols, float score) {
	int colhelp = cols;

	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
	cout << endl;
	cout << "\t\tPLAYER #" << player << " WON!!!!!... " << players[player - 1] << " Congratulations!!!" <<
endl;
	cout << setprecision(2) << fixed << "\t\tCUMULATIVE SCORE: " << score << endl;
	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
}

//DRAW
```

```
void nobodyWon(int cols) {
	int colhelp = cols;

	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
	cout << endl;
	cout << "\t\tNOBODY WON.... THE GAME ENDED IN DRAW!!..." << endl;
	cout << endl;
	cout << "\t\t***";
	for (int x = 0; x < colhelp + 2; x++) {
		cout << "*******";
	}
	cout << endl;
}

//WRITE CUMULATIVE IN A BINARY FILE
void writeBinaryFile(float cumulative) {
	string name = "cumulative.dat";
	ofstream outFile;
	string cumulativet = to_string(cumulative);
	int size1 = (cumulativet.size());

	outFile.open(name, ios::out | ios::binary);
	if (outFile.is_open()) {
		outFile.write(reinterpret_cast<char*>(&size1), sizeof(int));
		outFile.write(cumulativet.c_str(), size1);
	}
	else {
		cout << "File was not Found or can't be opened...";
	}
	outFile.close();
}

//READ CUMULATIVE FROM A BINARY FILE
float readBinaryFile() {
	string name = "cumulative.dat";
	string cumulativet = "00000000000000000000";
	ifstream outFile;
	float cumulative = 0.00;
	char* buf;
	int size1 = (cumulativet.size());

	outFile.open(name, ios::in | ios::binary);
	if (outFile.is_open()) {
		outFile.read(reinterpret_cast<char*>(&size1), sizeof(int));
		buf = new char[size1];
		outFile.read(buf, size1);
		cumulativet = "";
		cumulativet.append(buf, size1);
	}
	else {
		cout << "File was not Found or can't be opened...";
	}
	outFile.close();
	cumulative = stof(cumulativet);
	return cumulative;
}

//SHOW SCORES:
void showScores(int option, float cumulativescore, int cols, string playername) {
	vector<float> scorev;
	vector<string> namesv;
	float score = 0.00;
	string name = "";
	string filename = "scores.csv";
	string filename2 = "names.csv";
	int x = 0;
	int y = 0;
	string line;
	string line2;
	ifstream infile;
	ifstream infile2;
	ofstream outfile;
	ofstream outfile2;
	int colhelp = cols;
	int control1 = 0;

	if (option == 1) {
		outfile.open(filename, ios::app);
		outfile2.open(filename2, ios::app);
		if (outfile.is_open() && outfile2.is_open()) {
			outfile << endl;
			outfile2 << endl;
			outfile << fixed << setprecision(2) << cumulativescore;
			outfile2 << playername;
			y++;
		}
		if (y == 0) {
			cout << endl;
			cout << "\t\tFiles couldn't be Created, Data was not transfered..." << endl;
```

```
				cout << endl;
			}
			outfile.close();
			outfile2.close();
		}
		infile.open(filename, ios::in);
		infile2.open(filename2, ios::in);
		if (infile.is_open() && infile2.is_open()) {
			while (getline(infile, line)) {
				scorev.push_back(stof(line));
				x++;
			}
			infile.close();
			if (x > 0) {
				x = 0;
				while (getline(infile2, line2)) {
					namesv.push_back(line2);
					x++;
				}
			}
			infile2.close();
			if (x == 0 || (namesv.size() != scorev.size())) {
				cout << endl;
				cout << "\t\tFile is Empty or Corrupted... Can't Show Scores... Please, Fix Files and Try
Again..." << endl;
			}
			else {
				while (control1 == 0) {
					control1 = 1;
					for (int x = 0; x < namesv.size() - 1; x++) {
						if (scorev.at(x + 1) > scorev.at(x)) {
							score = scorev.at(x + 1);
							name = namesv.at(x + 1);
							scorev.at(x + 1) = scorev.at(x);
							namesv.at(x + 1) = namesv.at(x);
							scorev.at(x) = score;
							namesv.at(x) = name;
							control1 = 0;
						}
					}
				}
				cout << "\t---------------------------------------------------------------------------------------------" <<
endl;
				cout << endl;
				cout << "\t  ******   *******   *****   ***  *****   ***  *********    ******
*********       ******" << endl;
```

```
			cout << "\t ******   *********  ******  ***  ******  ***  *********   ******
*********       *******" << endl;
			cout << "\t***      ***   ***  *** *** ***  *** *** ***  ***      ***       ***       ***
***" << endl;
			cout << "\t***      ***   ***  ***  ******  ***  ******  ******    ***       ***
*********" << endl;
			cout << "\t***      ***   ***  ***   *****  ***   *****  ***      ***       ***
*********" << endl;
			cout << "\t ******   *********  ***    ****  ***    ****  *********   ******     ***
***" << endl;
			cout << "\t  ******   *******   ***     ***  ***     ***  *********   ******    ***
***" << endl;
			cout << endl;
			cout << "\t-----------------------------------------------------------------------------------------" <<
endl;
			cout << "\t-----------------------------------------------------------------------------------------" <<
endl;
			cout << endl;
			cout << "\t\t***";
			for (int x = 0; x < colhelp + 2; x++) {
				cout << "*******";
			}
			cout << endl;
			cout << "\t\t\tConnect 4 - HALL OF FAME:" << endl;
			cout << "\t\t***";
			for (int x = 0; x < colhelp + 2; x++) {
				cout << "*******";
			}
			cout << endl;

			cout << "\t\t\tRANK\t\tSCORE\t\t\tNAME" << endl;
			cout << "\t\t***";
			for (int x = 0; x < colhelp + 2; x++) {
				cout << "*******";
			}
			cout << endl;
			cout << endl;
			for (int x = 0; x < namesv.size(); x++) {
				cout << fixed << setprecision(2);
				cout << "\t\t\t " << x + 1 << "\t\t" << scorev.at(x) << "\t\t\t" << namesv.at(x) <<
endl;
			}
			cout << endl;
			cout << "\t\t***";
			for (int x = 0; x < colhelp + 2; x++) {
				cout << "*******";
			}
```

```
                        cout << endl;
                }
        }
        else {
                cout << endl;
                cout << "\t\tFiles In didn't Exist...";
        }
        cout << endl;
}
```