# Project 2

Connect Four

CSC-5 Summer 46024

Taught by Dr. Mark Lehr

Guthrie Price

29 July 2014

# Introduction

**Game Title:** Connect Four

**Description:** Two players take turns trying to get four of their pieces lined up vertically, horizontally, or diagonally, while attempting to stop the other player from doing the same.

## Rules

- **Objectives**
  - Obtain a line of four of pieces vertically, horizontally, or diagonally.
- **Constraints**
  - Players may only choose which column they want their piece to fall.
  - Once chosen, the piece falls to the lowest non-occupied space in the chosen column.
- **Game Ending Conditions**
  - Either player gets four pieces in a row, in which case that player wins.
  - There are no more empty spaces on the board to play, in which case the game ends in a draw.

# Summary

## Outline of implementation goals

Implement a fully functional console version of Connect Four. Extra features I wanted to include: a board sizing option, choice between playing against another person or a simple AI opponent, and basic accounts that save player statistics.

## Basic project size information

Total lines: 803

Lines of code: 511

Comment lines: 232

Blank lines: 60

Main variables: 18

Functions: 21

# Concept checklist

## Data Types

- **Primitive data types used:** int, char, bool
- **Primitive data types unused:** short, float
- **Other data types used:** string, fstream
- **Modifiers used:** const

## Container Types

- **Arrays:** 2-dimensional char[][B_MAX], 1-dimensional int[]
- **Vectors:** vector<string>, vector<int>

## System Level Libraries

- **iostream:** used for I/O
- **iomanip:** used for setw() function
- **cstdlib:** used for the exit() function and EXIT_FAILURE
- **string:** used for string data type
- **vector:** used for vector object
- **fstream:** used for fstream data type
- **string:** used for string data type
- **limits:** used for bad input handling

## Operators

| Operators used (character, line #) | Operators unused (character) |
|---|---|
| Addition(+, 205) | Division (/) |
| Subtraction (-, 201) | Modulo (%) |
| Multiplication (*, 432) | Logical Xor (^) |
| Assignment (=, 444) | Subtraction and Assignment (-=) |
| Logical Not (!, 760) | Multiplication and Assignment (*=) |
| Logical And (&&, 770) | Division and Assignment (/=) |
| Logical Or (||, 147) | Modulo and Assignment (%=) |
| Greater than or Equal (>=, 252) | |
| Less than or Equal (<=, 641) | |
| Equality (==, 312) | |
| Inequality (!=, 215) | |
| Increment and Assignment (++, 333) | |
| Decrement and Assignment (--, 557) | |
| Addition and Assignment (+=, 619) | |

## Conditionals and Loop Constructs

| Conditionals and Loops used (line #) | Conditionals unused |
|---|---|
| if (70)<br>else (74)<br>else if (167)<br>switch (196)<br>while (76)<br>do-while (96)<br>for (160)<br>ternary operator (?:, 471) | |

## Function usage checklist

- ✓ Passing arrays between functions (2-dimensional 668, 1-dimensional 464)
- ✓ Pass by value (765, *b_size*, *p_col*, and *plr* are passed by value)
- ✓ Pass by reference (765, *board* and *p_row* are passed by reference)
- ✓ Defaulted parameters (30, inRange declaration)
- ✓ Returning primitive data types (108, getNum returns an int)

## Searching and sorting

- ✓ Searching is implemented by the function linSrch(defined at line 310), which finds a string in a vector of strings (example usage at line 150)
- ✓ Sorting is implemented by the function sort(defined at line 328). See the function explanation for more details (example usage at line 464)

## Reading/Writing from Files

User account names and account data are read into the vectors *plrs* and *stats* on lines 69 through 93 from “players.csv” and “stats.csv”, respectively. All new accounts and statistics are put into *plrs* and *stats* while the game is running. Once the user chooses to quit, the information in *plrs* and *stats* are written to the files “players.csv” and “stats.csv”.

## Other Comments

The AI implementation is extremely simple, and as such it is very easy to defeat. I would have liked more time to research and create a more challenging computer opponent.

The board sizing options are sloppy at best, more time and effort could be put in to these options.

The Reading/Writing of files could mostly likely be done much more efficiently.

# Main Variables

| Type | Identifier | Description | Line # for main usage (* initialized at this line) |
|---|---|---|---|
| int | B_MAX | Maximum size for the board plus 1 (const) (global) | 22*,31,36 to 43, 200, 203, 405,517,541,554,575,594, 611,634,662,694,731 |
| | B_MIN | Minimum board size minus 1 (const) | 48*,203 |
| | COMP_ID | Computers ID number (const) | 50*,214,216,265 |
| | GUEST_ID | Guest ID number (const) | 51*,62,63,220 |
| | S_NUM | Number of statistics per player (const) | 52*,159 |
| | DEF_SIZE | Default board size (const) | 53*,200,206 |
| | acct_len | Length of a temporary account name | 55*,145,146 |
| | b_size | Board size | 56*,112,183,201,203,204, 206,208 |
| | m_choice | User's menu choice | 57*,107,110,284 |
| | s_choice | User's settings and statistics choice | 58*,192,195,228,244,248, 278 |
| | p_choice | User's player choice | 59*,140,141,168,170,171 |
| | p1_id | Player 1's ID number | 62*,112,134,171,251 |
| | p2_id | Player 2's ID number | 63*,112,137,172,185,214, 216,220,258, |
| vector<int> | stats | Account statistics data | 61*,89,112,158,160,251, 258,265,271,293,294 |
| string | acct_name | Stores a temporary account name | 54*,144,145,149,153,157, 170 |

| | | | |
|---|---|---|---|
| vector<string> | plrs | Account name data | 60*,76,112,134,137,149, 156,157,251,258,265,271, 288,289 |
| fstream | plr_file | Account name file for reading and writing data | 64*,68,69,75,77,287,289, 290 |
| | stat_file | Statistics file for reading and writing data | 65*,81,82,88,91,292,294, 295 |

# Functions List

## linSrch

- **Inputs(type):** *item*(string), *list*(const vector<string>&)
- **Outputs(type):** *index*(int)
- **Description:** Searches for *item* in *list* and returns the *index* where it is located, if it can't be found, returns -1
- **Line # found(* where defined):** *309,149

## sort

- **Inputs(type):** *array*(int[]), *size*(int)
- **Outputs(type):** *array*(int[])(returned by reference)
- **Description:** A special implementation of selection sort. *array* must be of *size**2. *array* is partitioned into two sections, the first section is sorted from high to low. The swaps that occur while sorting the first partition are also applied to the second partition.
- **Line # found(* where defined):** 327*,463 to 465

## isYes

- **Inputs(type):** None
- **Outputs(type):** (bool)
- **Description:** Prompts the user for input and determines if it is 'y' or 'Y' indicating yes.
- **Line # found(* where defined):** *354,155

## getNum

- **Inputs(type):** None
- **Outputs(type):** *choice*(int)
- **Notable internal variables:** *choice*(int) (User's input)
- **Description:** Gets a single number from the user.
- **Line # found(* where defined):** *367,107,140,192,201,244,758

## waitIpt

- **Inputs(type):** None
- **Outputs(type):** None
- **Description:** Waits for the user to press Enter before continuing.
- **Line # found(* where defined):** *380,128,508,581

## inRange

- **Inputs(type):** *ipt*(int),*ub*(int),*lb*(int)=0
- **Outputs(type):** (bool)
- **Description:** Determines if *ipt* is in the range given by (*lb,ub*). Note that this range does not include the end points. The lower bound *lb* has a default argument of 0.
- **Line # found(* where defined):** *394,203,228,278,284,643,759

## empty

- **Inputs(type):** *arr*(char[][B_MAX],*size*(int)
- **Outputs(type):** *arr*(char[][B_MAX])(returned by reference)
- **Description:** Fills a two dimensional array of *size* by *size*+1 with the character '.'
- **Line # found(* where defined):** *405,739

## acctOut

- **Inputs(type):** *plrs*(const vector<string>&),*id*(int)
- **Outputs(type):** None
- **Description:** Outputs the current user's account name to the screen.
- **Line # found(* where defined):** *417,134,137

## statOut

- **Inputs(type):** *plrs*(const vector<string>&),*stats*(const vector<int>&,*id*(int)
- **Outputs(type):** None
- **Description:** Outputs a user's statistics to the screen.
- **Line # found(* where defined):** *428,251,258,265

## rankOut

- **Inputs(type):** *plrs*(const vector<string>&),*stats*(const vector<int>&)
- **Outputs(type):** None
- **Notable internal variables:** *wins*(int[]),*loss*(int[]),*draw*(int[]) (arrays of win, loss and draw statistics and corresponding account IDs.
- **Description:** Outputs a sorted list of the top 10 rankings for wins, losses, and draws.
- **Line # found(* where defined):** 441*,271

## catgryOut

- **Inputs(type):** *plrs*(const vector<string>&),*c*_arr(const int[]),size(int),*cat*(string)
- **Outputs(type):** None
- **Description:** Outputs a single category (wins, losses, draws) to the screen.
- **Line # found(* where defined):** *483,470 to 472

## boardOut

- **Inputs(type):** *board*(const char[][B_MAX],*size*(int)
- **Outputs(type):** None
- **Description:** Outputs the current board state to the screen.
- **Line # found(* where defined):** *517,579,749

## isLegal

- **Inputs(type):** *board*(const char[][B_MAX],*p_move*(int)
- **Outputs(type):** (bool)
- **Description:** Given the current board state and the players attempted move, determines if the move is legal.
- **Line # found(* where defined):** *541,759

## update

- **Inputs(type):** *board*(char[][B_MAX]),*size*(int),*move*(int),*plr*(bool),*row*(int&)
- **Outputs(type):** *board*(char[][B_MAX])(The new board state updated with the current move)(returned by reference),*row*(int&)(The row the player's move landed)(returned by reference).
- **Description:** Updates the current board state given the current player's move. Returns the new board state and the row the player's move landed.
- **Line # found(* where defined):** *554,764

## endGame

- **Inputs(type):** *board*(const char[][B_MAX]),*size*(int),*running*(bool&)
- **Outputs(type):** *running*(bool&)(The status of whether the function connectFour should be running or not)(returned by reference)
- **Description:** Cleanup for the game. Outputs the final position of the board to the screen, waits for user to press Enter, and sets *running* to false.
- **Line # found(* where defined):** *575,776,793

## isDraw

- **Inputs(type):** *board*(const char[][B_MAX]),*size*(int)
- **Outputs(type):** (bool)
- **Description:** Given the current board position, determines if the game is a draw.
- **Line # found(* where defined):** *594,784

## didWin

- **Inputs(type):** *board*(const char[][B_MAX]),*size*(int),*row*(int),*col*(int)
- **Outputs(type):** (bool)
- **Description:** Given the current board position and the position of the last move, determines if the current player won.
- **Line # found(* where defined):** *611,767

## getMatch

- **Inputs(type):** *board*(char[][B_MAX]),*size*(int),*r*(int),*c*(int),*c_pce*(char),*r_d*(int), *c_d*(int)
- **Outputs(type):** *matches*(int)
- **Notable internal variables:** *matches*(int)(the number of pieces of type *c_pce* in a line vertically, horizontally, or diagonally, (depending on *r_d* and *c_d*) from a reference point given by *board*[r][c])
- **Description:** Calculates and return how many pieces in a row the current piece creates vertically, horizontally, and diagonally.
- **Line # found(* where defined):** *634,667,669,670,674,675,679,680

## getValue

- **Inputs(type):** *board*(const char[][B_MAX]),*size*(int),*pce*(char),*row*(int),*col*(int)
- **Outputs(type):** *highest*(int)
- **Notable internal variables:** *highest*(int)(calculate the highest number of matches of piece type *pce* around a given reference point *board*[row][col]),dir(enum{NONE=0,UP=-1,DOWN=1,LEFT=-1,RIGHT=1})(Used for input to getMatch for determining which way to search for matches)
- **Description:** Calculates the greatest number of pieces in a row for a given piece type at a specific point on the board.
- **Line # found(* where defined):**

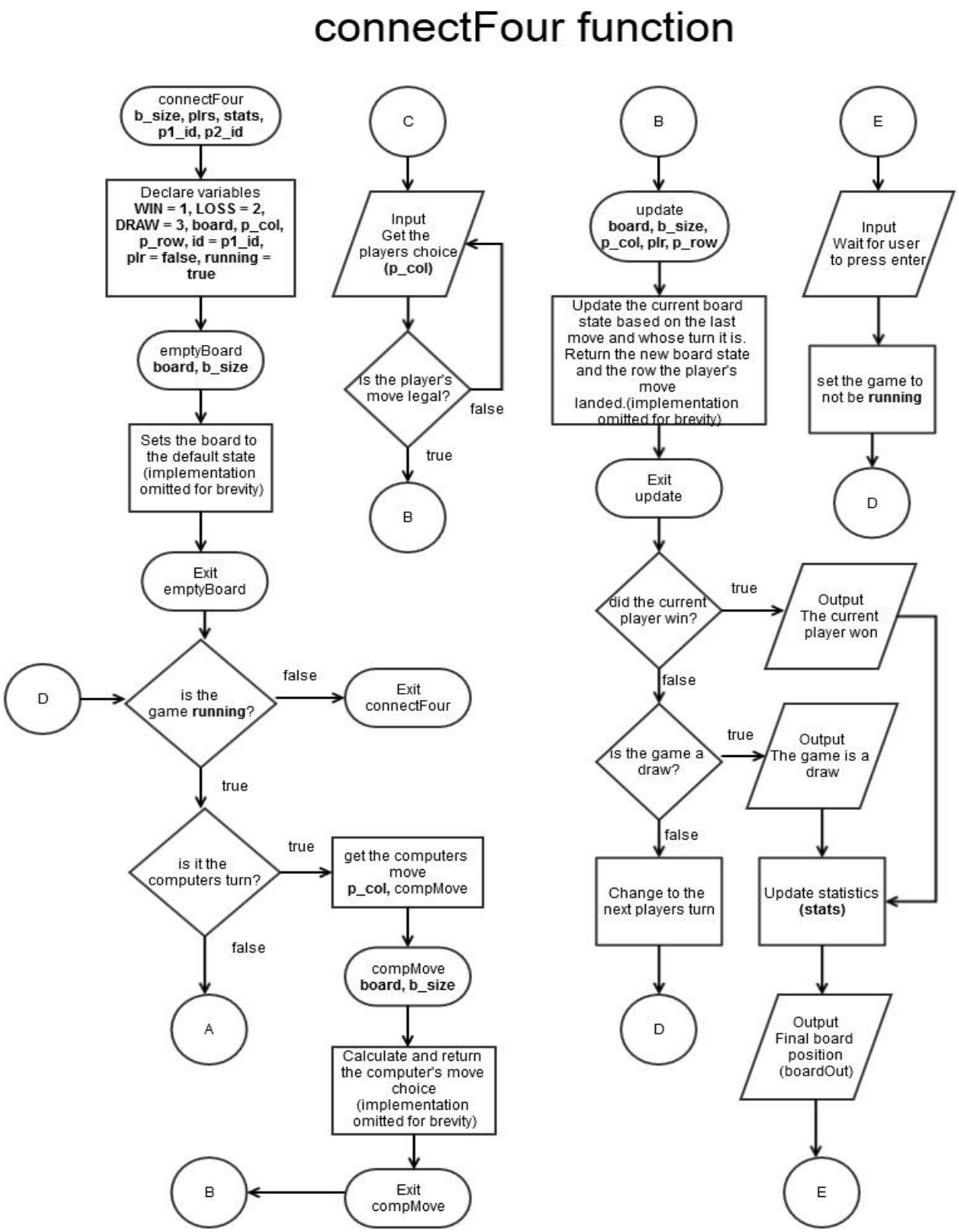
## compMove

- **Inputs(type):** *board*(const char[][B_MAX]),*size*(int)
- **Outputs(type):** *move*(int)
- **Notable internal variables:** *move*(int)(The computer's choice of move)
- **Description:** Calculate the computer's "best" move based on the getValue function.
- **Line # found(* where defined):**

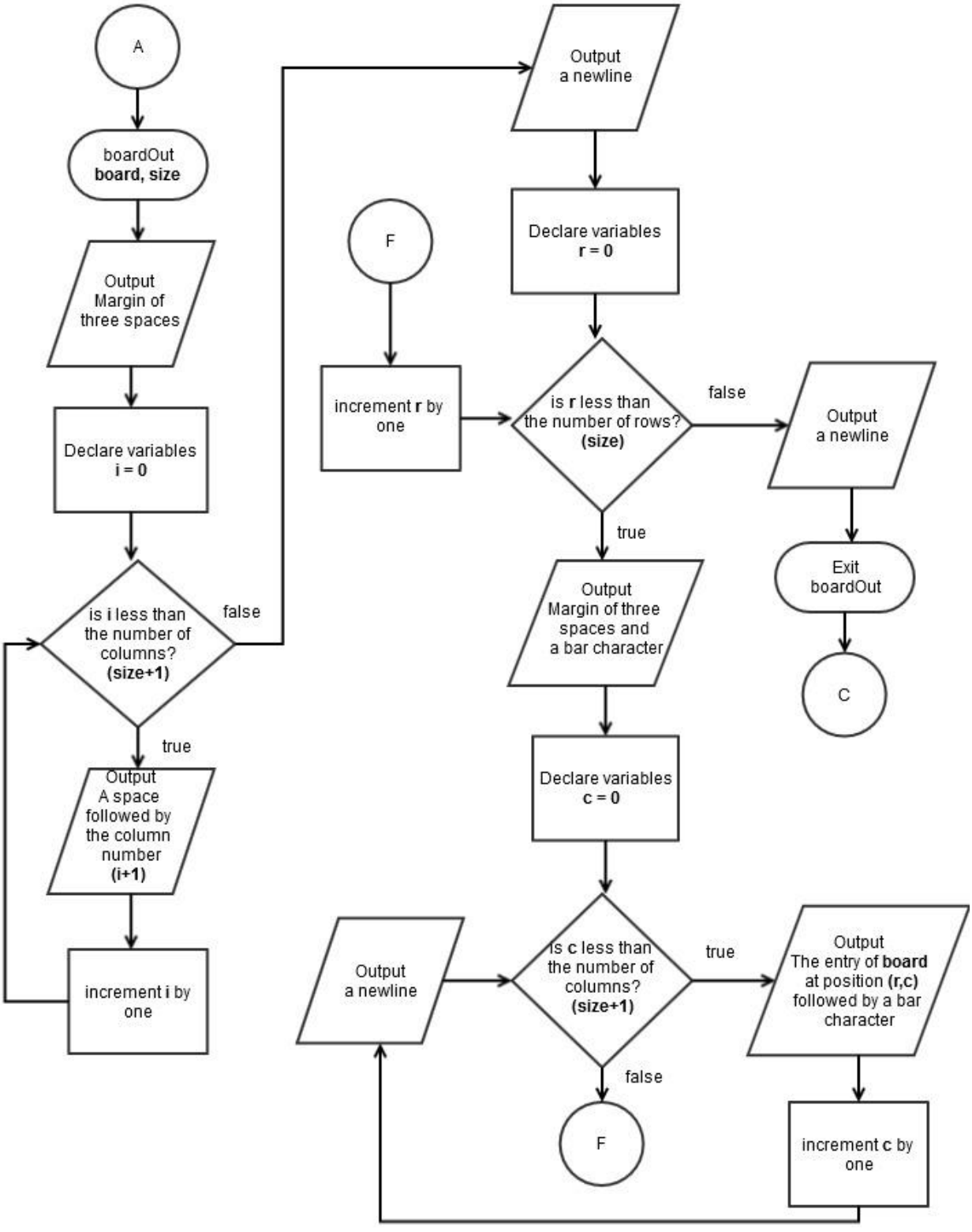
## connectFour

- **Inputs(type):** *b_size*(int),*plrs*(const vector<string>&),*stats*(const vector<int>&), *p1_id*(int),*p2_id*(int)
- **Outputs(type):** None
- **Notable internal variables:** *board*[b_size][B_MAX](The board of variable size used for the Connect Four game)
- **Description:** Implementation of Connect Four.
- **Line # found(* where defined):** (*729,111)

# Flowcharts for select functions



# boardOut function



# Pseudocode

```
Initialize
Load account and statistics data
While the users main menu choice is from 1 to 5
    Output the main menu
    Ask the user for a main menu option
    If the user's main menu choice is 1
        Enter Connect Four game
        Create the board for the game
        Set the status of the game to be running
        While the game is running
            If the computer is playing and it’s the computer’s turn
            Get the computer’s move
            Else
                Output the current status of the board
                Get the user’s move
            Update the board
            If the last move got four in a row or the game is a draw
                Output that the current player wins or that the game is a draw
                Update statistics
                Show the final board position
                Wait for user input
                Set the game to not be running
            Else
                Set the current player to the next player
    If the user's main menu choice is 2
        Output the rules for Connect Four
    If the user's main menu choice is 3
        Ask the user if s/he wants to change Player 1's or Player 2's account
            If the user chooses 1 or 2
                Ask the user for an account name
                If the account name exists
                    Set the users player choice to that account name
                Else
                    Ask the user if s/he wishes to register the account
                    If the user indicates yes
                        Register the account
                    Else
                        Set the account to Guest
    If the user's main menu choice is 4
        While the users settings menu choice is from 1 to 2
```

```
        Output the settings menu
        Ask the user for a settings menu choice
        If the settings menu choice is 1
            Ask the user for a board size choice
            If the user's board size choice is within range
                Set the board size to the user's board size choice
            Else
                Set the board size to the default size
        If the settings menu choice is 2
            Toggle the computer on or off
    If the user's main menu choice is 5
        While the user's statistics menu choice is from 1 to 4
            Output the statistics menu
            Ask the user for a statistics menu choice
            If the user's statistics menu choice is 1
                If Player 1's account is not set to Guest
                    Output the statistics for Player 1's account
            If the user's statistics menu choice is 2
                If Player 2's account is not set to Guest
                    Output the statistics for Player 2's account
            If the user's statistics menu choice is 3
                Output the statistics for the Computer account
            If the user's statistics menu choice is 4
                Output the top 10 rankings for the statistics
Save account and statistics data
```

# Actual Code

```
/*
 * File:   main.cpp
 * Author: Guthrie Price
 * Created on July 24, 2014, 10:39 PM
 * Purpose: Project 2 Summer CSC-5 46024
 *          Connect Four
 */

//System Level Libraries
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <string>
#include <vector>
#include <fstream>
#include <limits>
using namespace std;

//User Defined Libraries

//Global Constants
const int B_MAX = 10;//Maximum size for a board plus 1

//Function Prototypes
int linSrch(string,const vector<string>&);
void sort(int[],int);
bool isYes();
int getNum();
void waitIpt();
bool inRange(int,int,int=0);
void empty(char[][B_MAX],int);
void acctOut(const vector<string>&,int);
void statOut(const vector<string>&,const vector<int>&,int);
void rankOut(const vector<string>&,const vector<int>&);
void catgryOut(const vector<string>&,const int[],int,string);
void boardOut(const char[][B_MAX],int);
bool isLegal(const char[][B_MAX],int);
void update(char[][B_MAX],int,int,bool,int&);
void endGame(const char[][B_MAX],int,bool&);
bool isDraw(const char[][B_MAX],int);
bool didWin(const char[][B_MAX],int,int,int,int);
int getMatch(const char[][B_MAX],int,int,int,char,int,int);
int getValue(const char[][B_MAX],int,char,int,int);
int compMove(const char[][B_MAX],int);
void connectFour(int,const vector<string>&,vector<int>&,int,int);
//Begin Execution

int main(int argc, char** argv) {
    //Main menu setup and output
    const int B_MIN = 4;//Minimum board size minus 1
    const int COMP_ID = 0;//Computer accounts ID
    const int GUEST_ID = -1;//Guest ID
    const int S_NUM = 3;//The number of statistics kept per player
    const int DEF_SIZE = 6;//The default board size
    string acct_name;//User's account name choice
    int acct_len;//Length of the users account name
    int b_size = 6;//Size of the connect four game board (default is 6)
    int m_choice;//User's menu choice
    int s_choice;//User's settings and statistics choice
    int p_choice;//User's player choice
    vector<string> plrs;//A list of registered players
    vector<int> stats;//A list of statistics for each player
    int p1_id = GUEST_ID;//ID for player one (default loaded at startup)
    int p2_id = GUEST_ID;//ID for player two (default loaded at startup)
    fstream plr_file;//Stream for the player file
    fstream stat_file;//Stream for the statistics file

    //Try to load registered players
    plr_file.open("players.csv");
    if(!plr_file.is_open()){
        cout<<"Fatal Error: \"players.csv\" not found.\n";
        exit(EXIT_FAILURE);
    }
    else{
        string load;//Dummy variable for loading
        while(plr_file>>load)
            plrs.push_back(load);
```

```
    plr_file.close();
}

//Try to load statistics for registered players
stat_file.open("stats.csv");
if(!stat_file.is_open()){
    cout<<"Fatal Error: \"stats.csv\" not found.\n";
    exit(EXIT_FAILURE);
}
else{
    int load;//Dummy variable for loading
    while(stat_file>>load){
        stats.push_back(load);
    }
    stat_file.close();
}

//Enter menu loop
do{
    //Output menu
    cout<<"----------Menu-Menu----------\n";
    cout<<"1. Play Connect Four\n";
    cout<<"2. Rules for Connect Four\n";
    cout<<"3. Change account\n";
    cout<<"4. Settings Menu\n";
    cout<<"5. Statistics Menu\n";
    cout<<"6. Quit\n";
    cout<<endl;
    //Get a number from the user
    cout<<"Enter your menu choice: ";
    m_choice = getNum();
    cout<<endl;

    switch(m_choice){
        case(1):{
            connectFour(b_size,plrs,stats,p1_id,p2_id);
            break;
        }
        case(2):{
            cout<<"----------------------------Rules-for-Connect-Four--------------------------------\n";
            cout<<"Players: 2\n";
            cout<<"Description: Players take turns trying to get four of their pieces lined up\n";
            cout<<"             vertically, horizontally, or diagonally, while attempting\n";
            cout<<"             to stop the other player from doing the same.\n";
            cout<<"Objective: Obtain a line of four of pieces vertically, horizontally, or\n";
            cout<<"           or diagonally.\n";
            cout<<"Constraints: Players may only choose which column they want their piece\n";
            cout<<"             to fall. Once chosen, the piece falls to the lowest non-occupied\n";
            cout<<"             space in the chosen column.\n";
            cout<<"Ending conditions: 1. Either player gets four pieces in a row, and wins.\n";
            cout<<"                   2. There are no more places to play, and the game is a draw.\n\n";
            waitIpt();
            cout<<endl;
            break;
        }
        case(3):{
            cout<<"Player 1's current account: ";
            acctOut(plrs,p1_id);
            cout<<endl;
            cout<<"Player 2's current account: ";
            acctOut(plrs,p2_id);
            cout<<endl;
            cout<<"Change which player's account?(1 or 2): ";
            p_choice = getNum();
            if(p_choice == 1 || p_choice == 2){
                do{
                cout<<"Enter account name(from 1 to 10 characters): ";
                getline(cin,acct_name);
                acct_len = acct_name.length();
                }while(acct_len<=0 || acct_len>10);
                //See if the given account name exists already

                int id = linSrch(acct_name,plrs);//Dummy variable for id number

                //If the name isn't found, check to see if the account name is guest
                //Otherwise ask if the user wishes to register the account
                if(id<0 && !(acct_name == "Guest" || acct_name == "guest")){
                    cout<<"Account name not found, register new account? [y/n]: ";
                    if(isYes()){
                        id = plrs.size();
                        plrs.push_back(acct_name);
```

```
                stats.push_back(id);
                for(int i = 0;i<S_NUM;i++){
                stats.push_back(0);
                }
                cout<<"Account registered\n\n";
            }
            else cout<<"Setting to default account\n\n";
        }
        else if(id == 0) {
            cout<<"Computer opponent is on\n\n";
            p_choice = 2;
        }
        else cout<<"Setting Player "<<p_choice<<" account to "<<acct_name<<"\n\n";
        if(p_choice == 1) p1_id = id;
        else p2_id = id;
    }
    else
        cout<<"Invalid input\n\n";
    break;
}

case(4):{
    do{
        //Output settings menu
        cout<<"------------------Settings-Menu-----------------\n";
        cout<<"1. Change board size [currently "<<b_size<<" by "<<b_size+1<<"]\n";
        cout<<"2. Toggle computer opponent [currently ";
        if(p2_id == 0) cout<<"on";
        else cout<<"off";
        cout<<"]\n";
        cout<<"3. Exit Settings Menu\n\n";

        //Get users settings choice
        cout<<"Enter your settings choice: ";
        s_choice = getNum();
        cout<<endl;

        switch(s_choice){
            case(1):{
                cout<<"The board size is (N-1) by N, ";
                cout<<"where N is the number of columns.\n";
                cout<<"Enter the number of columns for the board";
                cout<<"("<<DEF_SIZE<<" to "<<B_MAX-1<<"): ";
                b_size = getNum()-1;
                cout<<endl;
                if(inRange(b_size,B_MAX-1,B_MIN))
                    cout<<"The board size is now "<<b_size<<" by "<<b_size+1<<endl<<endl;
                else{
                    b_size = DEF_SIZE;//Default board size
                    cout<<"Invalid input, setting board size to ";
                    cout<<b_size<<" by "<<b_size+1<<endl<<endl;
                }
                break;
            }
            case(2):{
                cout<<"Computer opponent is now ";
                if(p2_id != COMP_ID){
                    cout<<"on";
                    p2_id = COMP_ID;
                }
                else {
                    cout<<"off";
                    p2_id = GUEST_ID;
                }
                cout<<endl<<endl;
                break;
            }
            default:
                cout<<endl<<endl;
        }
    }while(inRange(s_choice,3));
    break;
}

case(5):{
    do{
        //Output statistics menu
        cout<<"-----Statistics-Menu-----\n";
        cout<<"1. Player 1 Statistics\n";
        cout<<"2. Player 2 Statistics\n";
        cout<<"3. Computer Statistics\n";
```

```
                    cout<<"4. Overall Rankings\n";
                    cout<<"5. Exit Statistics Menu\n\n";

                    //Get users choice
                    cout<<"Enter your statistics choice: ";
                    s_choice = getNum();
                    cout<<endl;

                    //Determine what to do based on the user's choice
                    switch(s_choice){
                        case(1):{
                            //Output player 1's statistics unless it is a guest player
                            if(p1_id>=0) statOut(plrs,stats,p1_id);
                            else cout<<"Guest players have no statistics";
                            cout<<endl<<endl;
                            break;
                        }
                        case(2):{
                            //Output player 2's statistics unless it is a guest player
                            if(p2_id>=0) statOut(plrs,stats,p2_id);
                            else cout<<"Guest players have no statistics";
                            cout<<endl<<endl;
                            break;
                        }
                        case(3):{
                            //Output the computer's statistics
                            statOut(plrs,stats,COMP_ID);
                            cout<<endl;
                            break;
                        }
                        case(4):{
                            //Output the overall rankings
                            rankOut(plrs,stats);
                            cout<<endl;
                            break;
                        }
                        default:
                            cout<<endl<<endl;
                    }
                }while(inRange(s_choice,5));
                break;
            }
            default:
                cout<<"Exiting game.\n";
        }
    }while(inRange(m_choice,6));//Check to see if the input is in a range

    //Write to files
    plr_file.open("players.csv",ios::out|ios::trunc);
    for(int i = 0;i<plrs.size();i++)
        plr_file<<plrs[i]<<endl;
    plr_file.close();

    stat_file.open("stats.csv",ios::out|ios::trunc);
    for(int i = 0;i<stats.size();i++)
        stat_file<<stats[i]<<endl;
    stat_file.close();

    //Exit program
    return 0;
}
//Function definitions

//Implementation of a linear search for string vectors
//Returns the index of the position of the string if found, otherwise returns -1
//Inputs
//  item = the string being searched for
//  list = the list of strings being searched
//Outputs
//  index = the index of the item (or -1 if not found)
int linSrch(string item,const vector<string>& list){
    for(int i = 0;i<list.size();i++){
        if(item == list[i])
            return i;
    }
    return -1;//Guest ID is -1
}

//A special implementation of selection sort that sorts the first partition of the
//array, and corresponds the second partition to the first. (sort is from high to low)
//Example: Input array is [1,2,4,3] and the size is given as 2. Once sorted the array
```

```
//         will be [2,1,3,4] because there is an assumed relationship between
//         the first and second partitions (2 corresponds to 3, 1 to 4).
//Inputs
//  array = the partitioned array
//  size = the partition size
//Output (by reference)
//  array = the sorted array
void sort(int array[],int size){
    //Declare variables
    int max_i;//The index with the maximum value and ID
    int max_v;//The maximum value
    int max_id;//ID corresponding to the maximum value
    for(int start = 0;start<(size-1);start++){
        max_i = start;
        max_v = array[start];
        max_id = array[start+size];
        for(int i = start+1;i<size;i++){
            if(array[i]>max_v){
                max_v = array[i];
                max_id = array[i+size];
                max_i = i;
            }
        }
        array[max_i] = array[start];
        array[max_i+size] = array[start+size];
        array[start] = max_v;
        array[start+size] = max_id;
    }
}

//Determine if the user entered y or Y for yes
//No inputs
//Outputs
//  bool indicating if the user entered y or Y
bool isYes(){
    char input;//User's input
    cin>>input;
    //Clear the input buffer
    cin.clear();//Remove the error flag on bad input
    cin.ignore(numeric_limits<streamsize>::max(), '\n');//Skip to the next newline character
    return(input == 'y' || input == 'Y');
}

//Gets a number from the user (assumes input validation elsewhere)
//No inputs
//Outputs
//  choice = the users choice
int getNum(){
    //Get the choice
    int choice;
    cin>>choice;
    //Clear the input buffer
    cin.clear();//Remove the error flag on bad input
    cin.ignore(numeric_limits<streamsize>::max(), '\n');//Skip to the next newline character
    return choice;
}

//Waits for user input before continuing
//No Inputs
//No Outputs
void waitIpt(){
    cout<<"Press Enter to continue ";
    cin;
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');//Ignore all input
}

//Determines whether input is inside a range exclusive of the bounds
//Inputs
//  ipt = input to be validated
//  LB = lower bound on the input (defaults to zero))
//  UB = upper bound on the input
//Outputs
//  bool determining if the input is valid or not
bool inRange(int ipt,int ub,int lb){
    //The input is in the range if it is between the upper and lower bounds
    return ipt>lb && ipt<ub;
}

//Fills an nx(n+1) array with '.' characters
//Inputs
```

```
//  arr = a 2-dimensional array
//  size = number of rows and number of columns-1
//Outputs
//  arr = a 2-dimensional array filled with '.' characters
void empty(char arr[][B_MAX],int size){
    for(int i = 0;i<size;i++){
        for(int j = 0;j<size+1;j++)
            arr[i][j] = '.';
    }
}

//Prints an account name to the screen based on the players id number
//Inputs
//  plrs = vector of player names
//  id = players current id number
//No outputs
void acctOut(const vector<string>& plrs,int id){
    if(id<0) cout<<"Guest";
    else cout<<plrs[id];
}

//Outputs the statistics of a specific player
//Inputs
//  plrs = the list of players
//  stats = the list of statistics
//  id = the id number of the player to lookup
//No Outputs
void statOut(const vector<string>& plrs,const vector<int>& stats,int id){
    cout<<plrs[id]<<"'s Statistics\n";
    cout<<"-------------------------\n";
    cout<<"Games Won:   "<<setw(4)<<stats[(id*4)+1]<<endl;
    cout<<"Games Lost:  "<<setw(4)<<stats[(id*4)+2]<<endl;
    cout<<"Games Drawn: "<<setw(4)<<stats[(id*4)+3]<<endl;
}

//Prints the users with the most wins, losses, and draws to the screen
//Inputs
//  plrs = the list of players
//  stats = statistics for the players
//No Output
void rankOut(const vector<string>& plrs,const vector<int>& stats){
    //Declare variables
    const int SIZE = plrs.size();//Size of a specific category
    const int CATEGORIES = 3;//The number of categories
    //For the following arrays, the first partition of size SIZE has will have a
    //sorted list for a specific category of statistics (wins, losses, draws).
    //the second partition, also of size SIZE, has the corresponding IDs.
    int wins[SIZE*2];
    int loss[SIZE*2];
    int draw[SIZE*2];

    //Load IDs and categories into the arrays
    for(int i = 0;i<SIZE;i++){
        wins[i] = stats[i*4+1];
        wins[i+SIZE] = stats[i*4];
        loss[i] = stats[i*4+2];
        loss[i+SIZE] = stats[i*4];
        draw[i] = stats[i*4+3];
        draw[i+SIZE] = stats[i*4];
    }

    //Sort each array
    sort(wins,SIZE);
    sort(loss,SIZE);
    sort(draw,SIZE);

    //Print to the screen
    cout<<"---------Rankings----------\n";
    for(int i = 0;i<CATEGORIES;i++){
        (i == 0)?catgryOut(plrs,wins,SIZE,"Wins"):
        (i == 1)?catgryOut(plrs,loss,SIZE,"Losses"):
                 catgryOut(plrs,draw,SIZE,"Draws");
    }
}

//Prints a specific category to the screen
//Inputs
//  plrs = list of players
//  c_arr = the category array, holding the score for each category and the
//          corresponding IDs
//  size = the size of a single partition of the array
```

```
//  cat = the category type to be printed
void catgryOut(const vector<string>& plrs,const int c_arr[],int size,string cat){
    const int MAX_DIS = 10;//Maximum number of users displayed
    const int MAX_SPC = 11;//Maximum number of spaces between the user and his score
    int top;//The number of users displayed
    int s_num;//The number of spaces between the user and his score
    string plr;//The players account name

    //If the size number of users is less than 10, set the maximum number of users
    //to be displayed to size
    if(size<MAX_DIS) top = size;
    else top = MAX_DIS;

    //Output a specific category
    cout<<"Most "<<cat<<endl;
    cout<<"-------------------\n";
    for(int i = 0;i<top;i++){
        plr = plrs[c_arr[size+i]];
        s_num = MAX_SPC-plr.length();
        cout<<(i+1)<<". "<<plr<<" with ";
        for(int j = 0;j<s_num;j++) cout<<" ";
        cout<<setw(4)<<c_arr[i]<<" "<<cat<<endl;
    }

    //Wait for input before moving to the next category
    cout<<endl;
    waitIpt();
    cout<<endl;
}

//Print the board to the screen
//Inputs
//  board = the current board
//  size = the size of the board
//No outputs
void boardOut(const char board[][B_MAX],int size){
    //Output numbers above each column on the board
    cout<<"    ";
    for(int i = 0;i<size+1;i++)
        cout<<" "<<i+1;
    cout<<endl;
    //Output the board
    for(int i = 0;i<size;i++){
        cout<<"    |";
        for(int j = 0;j<size+1;j++)
            cout<<board[i][j]<<"|";
        cout<<endl;
    }
    cout<<endl;
}

//Determines whether a players move is legal given the current board
//A move is legal if there is at least one space empty in the players
//chosen column
//Inputs
//  board = the current state of the board
//  p_move = the players move
//Output
//  bool indicating if the move is legal
bool isLegal(const char board[][B_MAX],int p_move){
    return(board[0][p_move-1] == '.');
}

//Updates the board with the current players move
//Inputs
//  board = the current state of the board
//  size = the size of the board
//  MOVE = the current player's move
//  plr = the current player
//Outputs
//  board = the board updated with the current players move
//  row = the row where the piece landed
void update(char board[][B_MAX],int size,int move,bool plr,int& row){
    char crd;//The current coordinate being examined
    for(int i = size-1;i>=0;i--){
        crd = board[i][move-1];
        if(crd == '.'){
            if(plr) crd = 'O';
            else crd = 'X';
            board[i][move-1] = crd;
            row = i+1;
```

```
            break;
        }
    }
}

//End the game by printing the final board position and setting the game to stop
//running
//Inputs
//  board = the final board position
//  size = size of the board
//Outputs (by reference)
//  running = flag determining whether the game should keep running
void endGame(const char board[][B_MAX],int size,bool& running){
    //Output the final position
    cout<<"     Ending position\n";
    cout<<"------------------------\n";
    boardOut(board,size);
    //Wait for the user to continue
    waitIpt();
    cout<<endl;
    //Set the status of the game to not be running
    running = false;

}
//Determines if the game is a draw
//A game is a draw if the entire first row is filled with pieces
//Inputs
//  board = the current board state
//  size = size of the game board
//Output
//  flag determining if the game is a draw
bool isDraw(const char board[][B_MAX],int size){
    for(int i = 0;i<size+1;i++){
        if(board[0][i] == '.')
            return false;
    }
    return true;
}

//Determine if a player has won
//A player wins if he has four pieces in a row vertically, horizontally, or
//diagonally
//Inputs
//  board = current board status
//  size = size of the board
//  PLR = the current player
//Outputs
//  bool determining if the current player has won
bool didWin(const char board[][B_MAX],int size,int row,int col){
    //Declare variables
    const int MIN_M = 4;//The minimum number of matches for a win
    char c_pce = board[row][col];//The current type of piece
    int matches = 1;//The number of matches in a line
    //Determine who won
    //We need only need to check if the currently placed piece creates a win
    matches += getValue(board,size,c_pce,row,col);
    if(matches >= MIN_M) return true;
    else return false;
}

//Give the number of repeated matches of the current piece from a given location
//up to 3 spaces away in a given direction
//Inputs
//  board = the current board state
//  size = the board size
//  r = the row index of the current piece
//  c = the column index of the current piece
//  r_d = the direction the row index will move
//  c_d = the direction the column index will move
//Outputs
//  matches = the number of matches in the given direction
int getMatch(const char board[][B_MAX],int size,int r,int c,char c_pce,int r_d,int c_d){
    //Declare variables
    const int D_MAX = 3;//Maximum number of spaces away from the current piece that matter
    int matches = 0;//Number of matches so far to the current piece

    //Check the matches in the given direction
    for(int i = 1;i<=D_MAX;i++){
        r += r_d;
        c += c_d;
        if(inRange(r,size,-1) && inRange(c,size+1,-1) && board[r][c] == c_pce)
```

```
            matches += 1;
        else
            break;
    }
    return matches;
}

//Examines the surrounding areas of a space on the board for unbroken lines of
//a specific type of game piece.
//Used to assign values for computer moves and determine if there is a winner
//Inputs
//  board = current board state
//  size = size of the board
//  pce = the type of piece to check for
//  row = the row of the reference space
//  col = the column of the reference space
//Outputs
//  highest = the highest value
int getValue(const char board[][B_MAX],int size,char pce,int row,int col){
    int val = 0;//The current value
    int highest;//The highest value
    enum dir{NONE = 0,UP = -1,DOWN = 1,LEFT = -1,RIGHT = 1};//Directions

    highest = getMatch(board,size,row,col,pce,DOWN,NONE);
    //Look for horizontal matches
    val += getMatch(board,size,row,col,pce,NONE,LEFT);
    val += getMatch(board,size,row,col,pce,NONE,RIGHT);
    if(val > highest) highest = val;
    val = 0;
    //Look for diagonal matches from low to high (going left to right)
    val += getMatch(board,size,row,col,pce,UP,RIGHT);
    val += getMatch(board,size,row,col,pce,DOWN,LEFT);
    if(val > highest) highest = val;
    val = 0;
    //Look for diagonal matches from high to low (going left to right)
    val += getMatch(board,size,row,col,pce,UP,LEFT);
    val += getMatch(board,size,row,col,pce,DOWN,RIGHT);
    if(val > highest) highest = val;

    return highest;
}

//Implementation of a basic computer opponent
//The computer plays entirely defensively, assigning values to each open space
//depending on how many pieces the human player has in a row surrounding that space
//Inputs
//  board = the current board position
//  size = the size of the board
//Outputs
//  move = the chosen column by the computer
int compMove(const char board[][B_MAX],int size){
    //Declare variables
    char plr = 'X';//The human player's piece
    int c_val;//The value of the current space being examined
    int h_val = 0;//The highest valued spot
    int move = 0;//The computers chosen column

    //Loop through each space of a column until the first empty one is found
    for(int col = 0;col<size+1;col++){
        for(int row = size-1;row>=0;row--){
            //If the space is empty, get the value of that space
            //Otherwise move on to the next space
            if(board[row][col] == '.'){
                c_val = getValue(board,size,plr,row,col);
                //If the current spaces value is greater than the highest value
                //set the move to equal the current column and set the highest
                //value to the current value
                if(c_val > h_val){
                    move = col;
                    h_val = c_val;
                }
                //Once the first empty space is found, move on to the next column
                break;
            }
        }
    }
    //Return the highest valued move
    return move;
}

//Implements the game Connect Four of variable board size
```

```
//Inputs
//  b_size = the size of the game board
//No outputs
void connectFour(int b_size,const vector<string>& plrs,vector<int>& stats,int p1_id,int p2_id){
    //Declare variables
    enum s_loc{WIN = 1,LOSS = 2,DRAW = 3};//For accessing player statistics
    char board[b_size][B_MAX];//An nx(n+1) board
    int p_col;//The column the current player chose to drop a piece
    int p_row;//The row where the current players move landed
    int id = p1_id;//The id of the current player
    bool plr = false;//Flag determining which player is currently playing
    bool running = true;//Flag determining if the game is over or not

    //Make the board "empty"
    empty(board,b_size);

    //Start game loop
    do{
        //Determine if its the computers turn
        if(p2_id == 0 && plr)//0 is the computers ID
            //Get the computers move
            p_col = compMove(board,b_size)+1;
        else{
            //Output the board
            boardOut(board,b_size);

            //Input and validation loop
            //The input isn't valid if it is out of the range of the boards
            //columns, or if it is an illegal move
            do{
                if(id<0) cout<<"Player "<<static_cast<int>(plr)+1;
                else cout<<plrs[id];
                cout<<", enter your move: ";
                p_col = getNum();
            }while(!inRange(p_col,b_size+2) || !isLegal(board,p_col));
            cout<<endl;
        }

        //Update the board with the current players move
        update(board,b_size,p_col,plr,p_row);

        //Check for game ending conditions
        if(didWin(board,b_size,p_row-1,p_col-1)){
            //Output who won
            if(p2_id == 0 && plr) cout<<"The computer";
            else{
                if(id<0) cout<<"Player "<<static_cast<int>(plr)+1;
                else cout<<plrs[id];
            }
            cout<<" has won!\n\n";

            //Update statistics
            if(id >= 0) stats[(id*4)+WIN]+=1;
            if(!plr && p2_id >= 0) stats[(p2_id*4)+LOSS]+=1;
            if(plr && p1_id >= 0) stats[(p1_id*4)+LOSS]+=1;

            //End the game
            endGame(board,b_size,running);
        }
        else if(isDraw(board,b_size)){
            //Output a draw message
            cout<<"   The game is a draw!\n\n";

            //Update statistics
            if(p1_id >= 0) stats[(p1_id*4)+DRAW]+=1;
            if(p2_id >= 0) stats[(p2_id*4)+DRAW]+=1;

            //End the game
            endGame(board,b_size,running);
        }
        else{
            //If the game isn't over, change the player
            plr = !plr;
            if(plr) id = p2_id;
            else id = p1_id;
        }
    }while(running);
}
```