```
#include <iostream>
#include <sstream>
#include <vector>
#include <fstream>
#include <stdlib.h> //rand()
#include <Windows.h>

#include "headers/Connect4.h"

using namespace std;

Connect4::Connect4() {
}

Connect4::~Connect4() {
}

//============================================================================
//  Initialize the arduino connection
//============================================================================
void Connect4::init() {
    // init the arduino and its' pins but only if USEARDUINO is set to true
#if USEARDUINO == true
    connected = ar.connect("COM4"); // connects to arduino; connected = true if succesfuly
    connected to arduino

    ar.sendDigitalPinMode(4, ARD_INPUT);    // set pinmodes on arduino
    ar.sendDigitalPinMode(5, ARD_INPUT);
    ar.sendDigitalPinMode(6, ARD_INPUT);
    ar.sendDigitalPinMode(7, ARD_INPUT);
#endif // USEARDUINO

    system("cls");
}

//============================================================================
//  This is where the main game loop is.
//  Controls the flow of the game.
//============================================================================
void Connect4::run()
{
    int col = -1;       // the collumn the user wants to play a piece in

    while (1) {     // main game loop

        switch (gameState) {
        case MENU:                          // main menu state
            int choice;
            cout << "\n\t1. New Game"
                << "\n\t2. Load Game"
                << "\n\t3. How to play"
                << "\n\t4. Exit"
                << endl;
            do {                        // get the menu option chosen by the user
                cout << "Choice:\t";
                cin >> choice;
            } while (choice < 1 || choice > 4);

            if (choice == 1) {          // init a new game and set the gameState for USER1
            to go first
                newGame();
                gameState = USER1_MOVE;
                board.displayBoard();
            }
            else if (choice == 2) {     // load the save game state. loadGame() updates the
            gameState to the proper value.
                loadFile = "save.txt";
                while (!loadGame(loadFile)) {
                    cout << "File " << loadFile << " could not be opened check filename and
                    try again" << endl << "File to load: ";
                    cin >> loadFile;
                }
                board.displayBoard();
            }
            else if (choice == 3) {     // display the instructions to the user; stay in
            MENU state
                HowToPlay();
                gameState = MENU;
            }
            else {                      // exit the game
                return;
            }
            break;

        case USER1_MOVE:                // user ones turn happens here
            playerTurn = USER1;
            cout << "\nColumn 1-7? (Enter 0 to save) ";
            do {
                col = getColumnChoice();    // get the column choice from the user
            } while (col == -1);

            if (col < 0 || col > 7)         // protect from invalid input
                continue;

            if (col == 0) {
                gameState = SAVE;           // user chose to save the game
                continue;
            }
            if (board.checkColumnFull(col - 1)) // column doesn't have room loop again
                continue;

            board.makeMove(USER1, col);

            if (board.isGameOver())             // game is done go to finished state
                gameState = FINISHED;
            else if (numPlayers == 1)       // should user2 or cpu have next move?
                gameState = CPU_MOVE;
            else
                gameState = USER2_MOVE;

            break;

        case USER2_MOVE:
            playerTurn = USER2;
            cout << "\nColumn 1-7? (Enter 0 to save) ";
            do {
                col = getColumnChoice();
            } while (col == -1);

            if (col < 0 || col > 7)     // protect from invalid input
                continue;

            if (col == 0) {
                gameState = SAVE;
                continue;
            }
            if (board.checkColumnFull(col - 1)) // column doesn't have room loop again
                continue;

            board.makeMove(USER2, col);     // make the move

            if (board.isGameOver())         // check for game over and update state
            accordingly
                gameState = FINISHED;
            else
                gameState = USER1_MOVE;

            break;

        case CPU_MOVE:                  // the cpu's turn
            col = rand() % 7 + 1;
            if (board.checkColumnFull(col - 1)) //column doesn't have room loop again
                continue;

            board.makeMove(CPU, col);       // make a move

            if (board.isGameOver())         // check for game over and update states
            accordingly
                gameState = FINISHED;
            else
                gameState = USER1_MOVE;
            break;

        case SAVE:
            saveGame(saveFile);         // save the state of the game
            gameState = FINISHED;       // go to finished game state
            break;
```

```
            case FINISHED:
                cout << endl << board.getGOMsg() << endl;        // display who won
                cout << "\nPress Enter to Continue" << endl;     // give user time to read who won
                int junk;
                cin >> junk;
                system("cls");
                gameState = MENU;                                // go to the menu
                break;
            }
        }
    }

    //=============================================================================
    //  Gets the number of players from the user and what color user1 wants
    //  to be. Then clears the board to all zeros.
    //=============================================================================
    void Connect4::newGame(){
        //initialize new game
        system("cls");

        do {    // get the number of players
            cout << "\n1 or 2 Players?\t";
            cin >> numPlayers;
        } while (numPlayers != 1 && numPlayers != 2);

        do {    // get Player 1 color
            cout << "\nPlayer 1 Color? R or B?\t";
            cin >> playerColor;
            playerColor = toupper(playerColor);
            board.setP1Color(playerColor);
        } while (playerColor != 'R' && playerColor != 'B');

        for (int r = 0; r < 6; r++) {       // clear the board
            for (int c = 0; c < 7; c++) {
                board.setPiece(c, r, EMPTY);
                board.zeroColHeight(c);
            }
        }
    }

    //====================================================================================
    //  loads the game state from save.txt and sets all the state variables acordingly
    //  save.txt format: [playerTurn] [numPlayers] [playerColor] [board layout]
    //====================================================================================
    int Connect4::loadGame(string loadFile){
        ifstream myfile;
        myfile.open(loadFile, ios::in);

        if (!myfile.is_open()) {
            return 0;
        }

        myfile >> playerTurn >> numPlayers >> playerColor;
        gameState = (States)playerTurn;
        board.setP1Color(playerColor);

        for (int c = 0; c < 7; c++) {
            board.zeroColHeight(c);
        }

        for (int r = 0; r < 6; r++) {
            for (int c = 0; c < 7; c++) {
                int val;
                myfile >> val;
                board.setPiece(c, r, val);
                if (board.getPiece(c, r) != 0) {
                    board.incColHeight(c);          // keep track of how many are in each column
                }
            }
        }

        cout << loadFile << " has been read\n";
        myfile.close();


        return 1;
    }

    //============================================================================
    //  save.txt format: [playerTurn] [numPlayers] [playerColor] [board layout]
    //============================================================================
    void Connect4::saveGame(string saveFile){
        ofstream myfile;
        myfile.open(saveFile, ios::out | ios::trunc);

        if(!myfile.is_open()){
            cout << saveFile << " could not be accessed\n";
            return;
        }

        myfile << playerTurn << " " << numPlayers << " " << playerColor << " ";

        for (int r = 0; r < 6; r++) {
            for (int c = 0; c < 7; c++) {
                myfile << board.getPiece(c, r) << " ";
            }
        }

        cout << saveFile << " has been written\n";
        myfile.close();
        return;
    }

    //============================================================================
    //  reads Instructions.txt and displays its contents to the user
    //============================================================================
    void Connect4::HowToPlay(){
        string filename = "Instructions.txt";
        ifstream read;
        read.open(filename);

        if(!read.is_open()){
            cout << filename << " could not be accessed\n";
            return;
        }

        cout << endl;

        string line;
        while (getline(read, line)) {
            cout << line << endl;
        }

        read.close();

        return;
    }

    //===========================================================================
    //  Takes an ofArduino object to read the pin states from
    //  Return the int value of the button that was pressed
    //  buttons need to be wired to pins sequntially increasing
    //===========================================================================
    int Connect4::checkButtons() {

    #if USEARUINO == true
        static int prev[4] = { -1 };    // buffer to hold previous states of pins
        static int curr[4] = { -1 };    // buffer to hold current states of pins

        for (int i = 0; i < numButtons; i++) {          // go through all the buttons and get
        their current states
            ar.update();                                // update the values read from the
            arduino
            curr[i] = ar.getDigital(startPin + i);      // pins must be sequntial

            if (prev[i] == 1 && curr[i] != prev[i]) {   // check for button state change from
            pressed to released
                prev[i] = curr[i];
                return i + 1;           // return the button that was released
            }
            else {
                prev[i] = curr[i];      // update previous state
            }
        }
    #endif // USEARUINO == true
        return -1;      // return -1 if no button was pressed

    }
```

```
    //=========================================================================
    //  Return the int value of the number key that was pressed
    //=========================================================================
    int Connect4::checkNumPress() {
        static int prev[10] = { -1 };   // buffer to hold previous state of keys
        static int curr[10] = { -1 };   // buffer to hold current state of keys

        for (int i = 0; i < 10; i++) {  // go through the keys 0-9 and get their current states
            curr[i] = ((GetKeyState(48 + i) >> 16) & 0x1);  // send ascii value of key; returns
            16 bit number; leftmost bit signifies state of key.

            if (prev[i] == 1 && curr[i] != prev[i]) {   // check for key state change from
            pressed to released
                prev[i] = curr[i];
                return i;                   // return the key that was released
            }
            else {
                prev[i] = curr[i];          // update previous state
            }
        }

        return -1;      // return -1 if no key was pressed
    }

    //=========================================================================
    //  Return the int value of the column selected
    //=========================================================================
    int Connect4::getColumnChoice() {

        int numPress = checkNumPress();
        int buttonPushed = -1;
        if (connected)
            buttonPushed = checkButtons();

        if (numPress != -1)
            return numPress;
        else if (buttonPushed != -1)
            return buttonPushed;
        else
            return -1;
    }
```