```cpp
#include "headers/gameBoard.h"

gameBoard::gameBoard() {
    gameFinished = false;

    for (int i = 0; i < 7; i++) {
        boardColumn current;        //create empty column
        board.push_back(current);   //push 7 empty columns to intialize
    }
}

gameBoard::~gameBoard() {
}

//=============================================================================
//  display the board to the user
//=============================================================================
void gameBoard::displayBoard() {
    int r, c;
    system("cls");  // Windows Only
    cout << "  +---+---+---+---+---+---+---+\n";

    for (r = 0; r < 6; r++) {
        cout << "  |";
        for (c = 0; c < 7; c++) {
            cout << " " << symbol(board[c].column[5 - r].value) << " |";    // use helper
            funtion to convert from int to char
        }
        cout << "\n"
            << "  +---+---+---+---+---+---+---+\n";
    }
    cout << "    1   2   3   4   5   6   7\n";  //print column nums
}

//=============================================================================
//  Place the game piece in the chosen column
//=============================================================================
void gameBoard::makeMove(int player, int column) {
    if (column >= 1 && column <= 7) {       //error check column choice
        int c = column - 1;                 //indices (r,c)
        int r = board[c].length;            //num of pieces in column

        if (board[c].checkColumnFull() == false) {  //error check for full column
            board[c].column[r].value = player;  //change that one space
            board[c].length++;                  //one more piece added
            gameFinished = chekcForWin(r, c);   //check for game over
            displayBoard();
        }
        else
            cout << "\nColumn is full";
    }
    else
        cout << "\nColumn must be 1-7";
```

```
}

//================================================================================
//  Helper functions
//================================================================================
char gameBoard::symbol(int i) {
// converts the int in the gamePiece to char to display on the board
    switch (i) {
    case 0:
        return ' ';
    case 1:
        return playerOneColor;
    case 2:
        if (playerOneColor == 'R')
            return 'B';
        else
            return 'R';
    case 3:
        if (playerOneColor == 'R')
            return 'B';
        else
            return 'R';
    }
    return ('?');
}

bool gameBoard::checkFull() {
// check if the board is full
    for (int i = 0; i < 7; i++) {
        if (!board[i].checkColumnFull())    //if column found not full exit
            return false;
    }
    return true;    //if all column full, board is full return true
}

bool gameBoard::isGameOver()
{
    return gameFinished;
}

string gameBoard::getGOMsg()
{
    return gameOverMessage;
}

bool gameBoard::checkColumnFull(int col)
{
    return board[col].checkColumnFull();
}

int gameBoard::getPiece(int col, int row)
{
    return board[col].column[row].value;
```

```
}

void gameBoard::setPiece(int col, int row, int value)
{
    board[col].column[row].value = value;
}

void gameBoard::incColHeight(int col)
{
    board[col].length++;
}

void gameBoard::zeroColHeight(int col)
{
    board[col].length = 0;
}

void gameBoard::setP1Color(char color)
{
    playerOneColor = color;
}
//================================================================================

//================================================================================
//  checks for connect fours in the game board
//================================================================================
bool gameBoard::chekcForWin(int row, int col) {
    //four across
    if (across(row, col))
        return true;

    //four down(no need to check up)
    else if (down(row, col))
        return true;

    //four diagonal positive slope
    else if (posDiagonal(row, col))
        return true;

    //four diagonal negative slope
    else if (negDiagonal(row, col))
        return true;

    //Tie game
    else if (checkFull()) {
        gameOverMessage = "\nIt's a draw!\n";
        return true;
    }

    //otherwise
    else
        return false;
}
```

```
//=========================================================================
//  helper functions for the function checkForWin().
//=========================================================================
bool gameBoard::across(int row, int col) {
    //which player to look for win
    int player = board[col].column[row].value;
    //what area to check for win
    int columnRangeLow = (col - 3 >= 0) ? col - 3 : 0;
    int columnRangeHigh = (col + 3 <= 6) ? col + 3 : 6;

    int count = 0;  //how many in a row
    int x = 0;      //bump indicesi

    while (col + x <= columnRangeHigh && board[col + x].column[row].value == player) {
        count++;    //count piece and pieces to right
        x++;
    }
    x = 1; //start one over to not double count move
    while (col - x >= columnRangeLow && board[col - x].column[row].value == player) {
        count++;    //count pieces to left
        x++;
    }
    if (count >= 4) {
        if (player != 3) {
            gameOverMessage = "\nPlayer ";
            stringstream ss;    //sequence converts num to string
            ss << player;
            gameOverMessage += ss.str();
        }
        else
            gameOverMessage = "CPU";    //for player=3 that is not actual player
        gameOverMessage += " wins!\n";
        return true;
    }
    return false;
}

bool gameBoard::down(int row, int col) {
    //which player to look for win
    int player = board[col].column[row].value;

    //what area to check for win
    int rowRangeLow = (row - 3 >= 0) ? row - 3 : 0;
    int rowRangeHigh = (row + 3 <= 5) ? row + 3 : 5;

    int count = 0;  //how many in a column
    int x = 0;      //bump indices

    while (row - x >= rowRangeLow && board[col].column[row - x].value == player) {
        count++;    //count pieces down
        x++;
    }
```

```
    if (count >= 4) {
        if (player != 3) {
            gameOverMessage = "\nPlayer ";
            stringstream ss;    //sequence converts num to string
            ss << player;
            gameOverMessage += ss.str();
        }
        else
            gameOverMessage = "CPU";    //for player=3 that is not actual player
        gameOverMessage += " wins!\n";
        return true;
    }
    return false;
}

bool gameBoard::posDiagonal(int row, int col) {
    //which player to look for win
    int player = board[col].column[row].value;
    //what area to check for win
    int columnRangeLow = (col - 3 >= 0) ? col - 3 : 0;
    int columnRangeHigh = (col + 3 <= 6) ? col + 3 : 6;
    int rowRangeLow = (row - 3 >= 0) ? row - 3 : 0;
    int rowRangeHigh = (row + 3 <= 5) ? row + 3 : 5;

    int count = 0;  //how many in a row
    int x = 0;      //bump indices

    while (row + x <= rowRangeHigh && col + x <= columnRangeHigh && board[col + x].column[
    row + x].value == player) {
        count++;    //count piece and pieces to right
        x++;
    }
    x = 1; //start one over to not double count move
    while (row - x >= rowRangeLow && col - x >= columnRangeLow && board[col - x].column[row
    - x].value == player) {
        count++;    //count pieces to left
        x++;
    }
    //cout << count;
    if (count >= 4) {
        if (player != 3) {
            gameOverMessage = "\nPlayer ";
            stringstream ss;    //sequence converts num to string
            ss << player;
            gameOverMessage += ss.str();
        }
        else
            gameOverMessage = "CPU";    //for player=3 that is not actual player
        gameOverMessage += " wins!\n";
        return true;
    }
    return false;
}
```

```
bool gameBoard::negDiagonal(int row, int col) {
    //which player to look for win
    int player = board[col].column[row].value;
    //what area to check for win
    int columnRangeLow = (col - 3 >= 0) ? col - 3 : 0;
    int columnRangeHigh = (col + 3 <= 6) ? col + 3 : 6;
    int rowRangeLow = (row - 3 >= 0) ? row - 3 : 0;
    int rowRangeHigh = (row + 3 <= 5) ? row + 3 : 5;

    int count = 0;  //how many in a row
    int x = 0;      //bump indices

    while (row + x <= rowRangeHigh && col - x >= columnRangeLow && board[col - x].column[row
     + x].value == player) {
        count++;    //count piece and pieces to right
        x++;
    }
    x = 1; //start one over to not double count move
    while (row - x >= rowRangeLow && col + x <= columnRangeHigh && board[col + x].column[row
     - x].value == player) {
        count++;    //count pieces to left
        x++;
    }
    //cout << count;
    if (count >= 4) {
        if (player != 3) {
            gameOverMessage = "\nPlayer ";
            stringstream ss;    //sequence converts num to string
            ss << player;
            gameOverMessage += ss.str();
        }
        else
            gameOverMessage = "CPU";    //for player=3 that is not actual player
        gameOverMessage += " wins!\n";
        return true;
    }
    return false;
}
```