Project 1

<Connect 4>

CIS-17A 43396

Name: Koksal, Attila

Date: 5/12/2021

Introduction

This project revolves around the popular family game, Connect 4. I am doing this game because it was one of the many games I enjoyed playing since when I was a kid. In addition, with this digital version of the game, you are no longer required to purchase a physical copy.

Summary

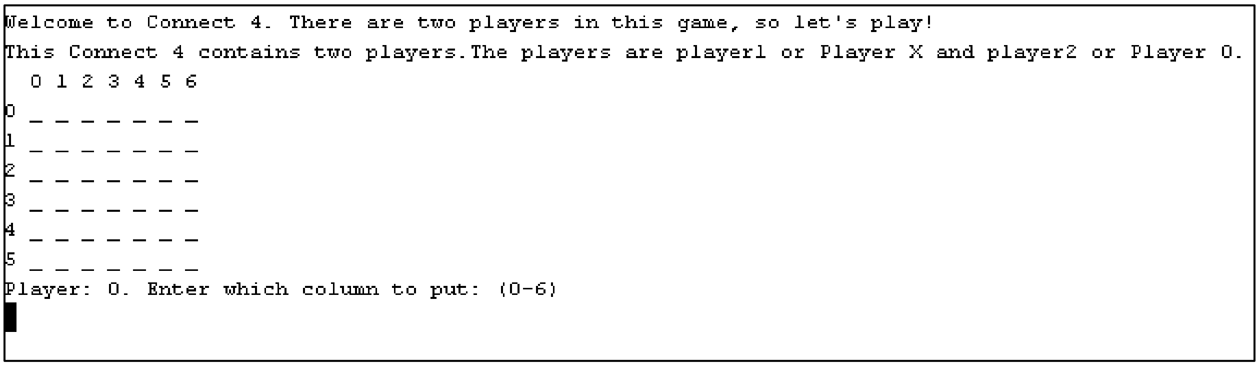
The program is 337 lines of code. This project meets the criteria of the first project because of how it utilizes various concepts described in this class such as structures, pointers, arrays, dynamic allocation/deallocation, nesting, enumeration, etc. 22 variables were utilized in this project. 42 constructs were utilized in this project. It was challenging for me, especially in when checking win conditions such as vertical, horizontal, diagonal (in both directions), and tie (when players filled the board without victory). It took 15 hours, including research, review, debugging, and coding, in total to complete this project.

Description

With the goal of creating a playable connect 4 game, there are various sections and challenges to be completed. These steps include initialize board and set all location to empty (represented through enum). Then implement player movement, which allows users to input their desired column to drop their piece in their desired column it is it a legal move (i.e the column is not filled by pieces). On the other hand, the user can press ‘X’ to save the board if desired, or press ‘L’ to load the last saved board. This step would be repeated until there is a winner. Then, after every move, the board would be displayed to players to analyze and understand their current positions. After each player moves and the board displays, the most significant part would be to check for any victory that can be caused by this most current player move. This includes checking the column, row, and two diagonals for pieces that are connected to the most recent move that causes 4 pieces to be connected. When 4 of them are connected, player wins and game ends.

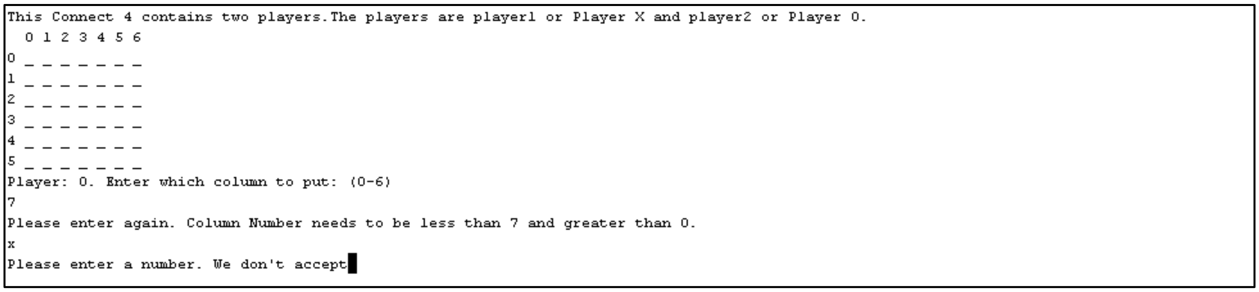
Sample Input/Output

Initialize board, display the board and ask for user input:



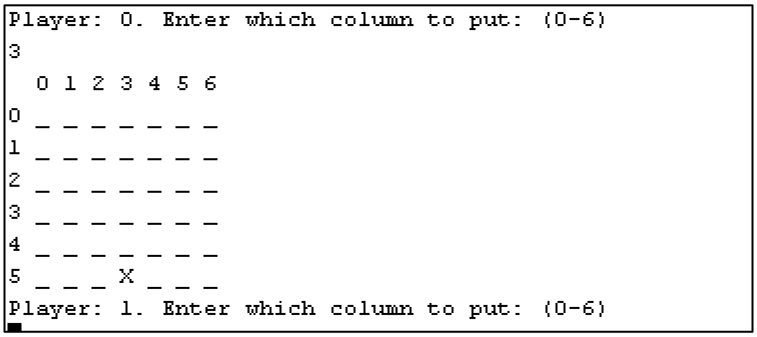
```
Welcome to Connect 4. There are two players in this game, so let's play!
This Connect 4 contains two players.The players are player1 or Player X and player2 or Player O.
  0 1 2 3 4 5 6
0 _ _ _ _ _ _ _
1 _ _ _ _ _ _ _
2 _ _ _ _ _ _ _
3 _ _ _ _ _ _ _
4 _ _ _ _ _ _ _
5 _ _ _ _ _ _ _
Player: O. Enter which column to put: (0-6)
```

Player 1 move, with incorrect input:



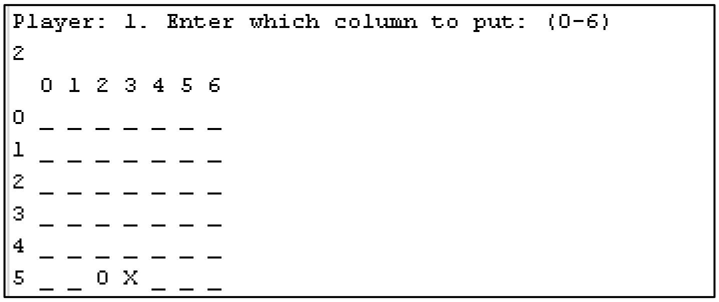
```
This Connect 4 contains two players.The players are player1 or Player X and player2 or Player 0.
  0 1 2 3 4 5 6
0 _ _ _ _ _ _ _
1 _ _ _ _ _ _ _
2 _ _ _ _ _ _ _
3 _ _ _ _ _ _ _
4 _ _ _ _ _ _ _
5 _ _ _ _ _ _ _
Player: 0. Enter which column to put: (0-6)
7
Please enter again. Column Number needs to be less than 7 and greater than 0.
x
Please enter a number. We don't accept
```

Player 1 move successful:



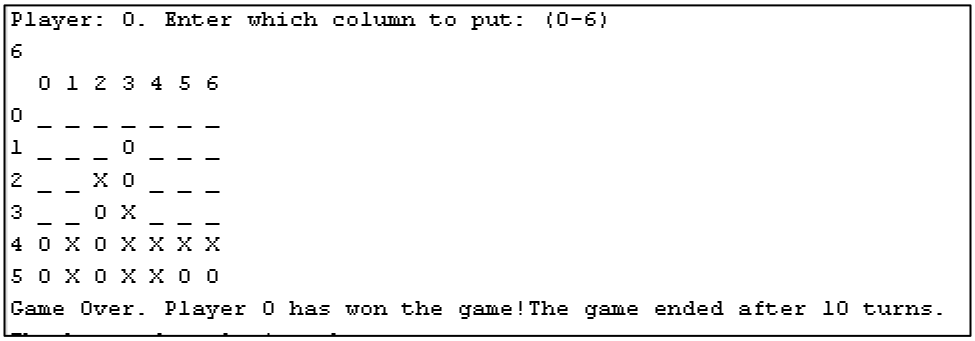
```
Player: 0. Enter which column to put: (0-6)
3
  0 1 2 3 4 5 6
0 _ _ _ _ _ _ _
1 _ _ _ _ _ _ _
2 _ _ _ _ _ _ _
3 _ _ _ _ _ _ _
4 _ _ _ _ _ _ _
5 _ _ _ X _ _ _
Player: 1. Enter which column to put: (0-6)
```

Player 2 moves successful:



```
Player: 1. Enter which column to put: (0-6)
2
  0 1 2 3 4 5 6
0 _ _ _ _ _ _ _
1 _ _ _ _ _ _ _
2 _ _ _ _ _ _ _
3 _ _ _ _ _ _ _
4 _ _ _ _ _ _ _
5 _ _ 0 X _ _ _
```

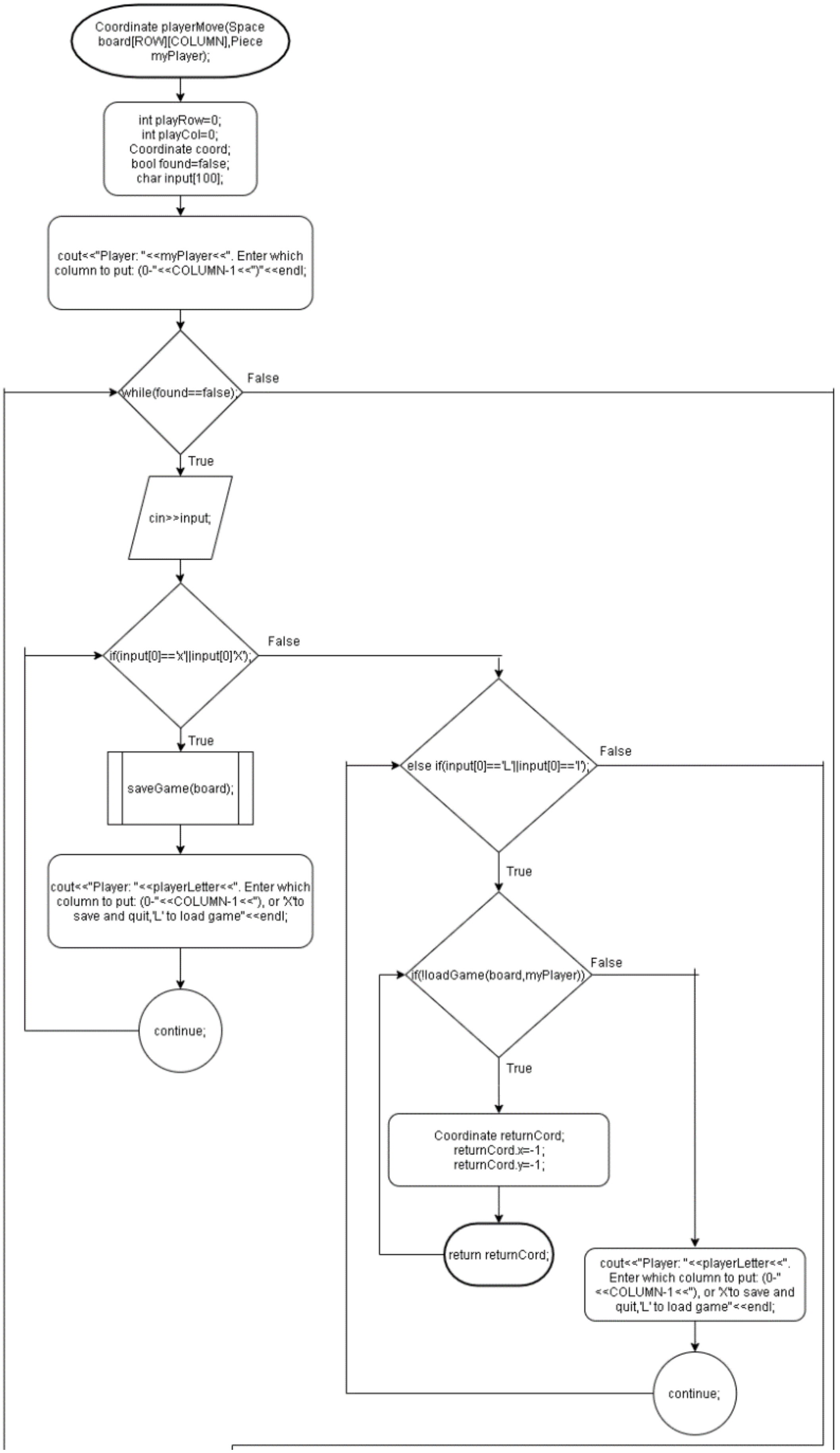
Game end, with victory of player 0, on row 4

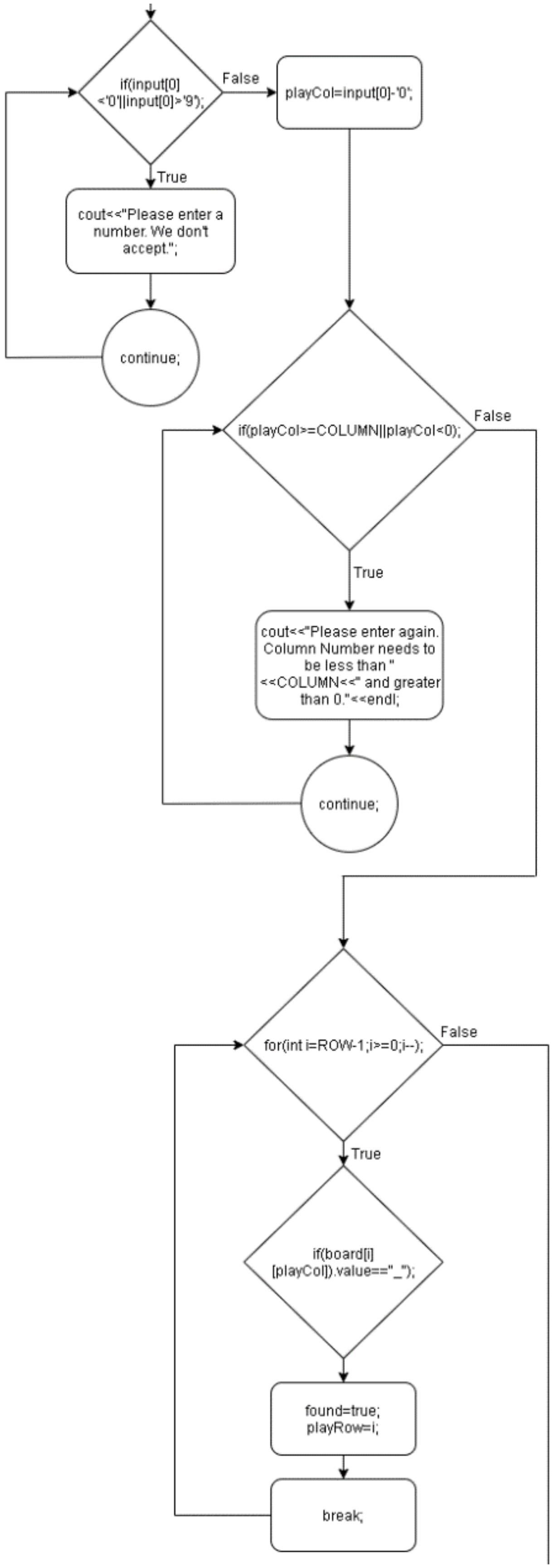


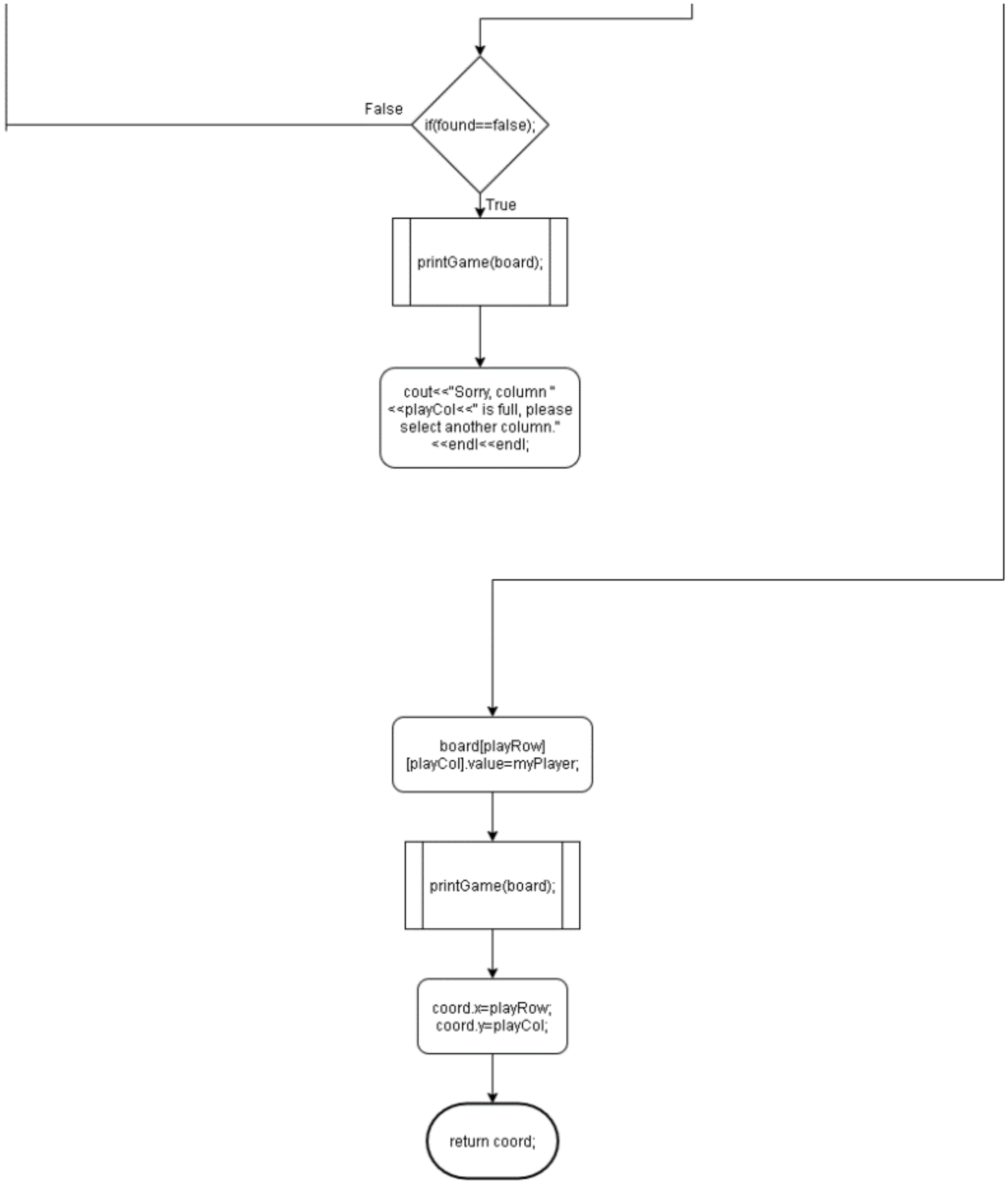
```
Player: 0. Enter which column to put: (0-6)
6
  0 1 2 3 4 5 6
0 _ _ _ _ _ _ _
1 _ _ _ 0 _ _ _
2 _ _ X 0 _ _ _
3 _ _ 0 X _ _ _
4 0 X 0 X X X X
5 0 X 0 X X 0 0
Game Over. Player 0 has won the game!The game ended after 10 turns.
```

Flowchart

Flowchart of playerMove() function


Coordinate playerMove(Space board[ROW][COLUMN],Piece myPlayer);
int playRow=0;
int playCol=0;
Coordinate coord;
bool found=false;
char input[100];
cout<<"Player: "<<myPlayer<<". Enter which column to put: (0-"<<COLUMN-1<<")"<<endl;
while(found==false);
False
True
cin>>input;
if(input[0]=='x'||input[0]'X');
False
True
saveGame(board);
cout<<"Player: "<<playerLetter<<". Enter which column to put: (0-"<<COLUMN-1<<"), or 'X'to save and quit,'L' to load game"<<endl;
continue;
else if(input[0]=='L'||input[0]=='l');
False
True
if(!loadGame(board,myPlayer))
False
True
Coordinate returnCord;
returnCord.x=-1;
returnCord.y=-1;
return returnCord;
cout<<"Player: "<<playerLetter<<". Enter which column to put: (0-"<<COLUMN-1<<"), or 'X'to save and quit,'L' to load game"<<endl;
continue;


if(input[0] <'0'||input[0]>'9');
False
playCol=input[0]-'0';
True
cout<<"Please enter a number. We don't accept.";
continue;
if(playCol>=COLUMN||playCol<0);
False
True
cout<<"Please enter again. Column Number needs to be less than " <<COLUMN<<" and greater than 0."<<endl;
continue;
for(int i=ROW-1;i>=0;i--);
False
True
if(board[i][playCol]).value=="_");
found=true; playRow=i;
break;


False
if(found==false);
True
printGame(board);
cout<<"Sorry, column "<<playCol<<" is full, please select another column."<<endl<<endl;
board[playRow][playCol].value=myPlayer;
printGame(board);
coord.x=playRow; coord.y=playCol;
return coord;

Pseudocode

```
Initialize the board
Display board on screen
While game is not over
    Input player1's column to be played
    If user entered 'X'
    Save board to binary file
If user entered  ‘L’
    Load board from binary file
    If loaded board indicates it is player2’s turn
        Skip rest of player1’s move
Calculate the row on the corresponding column where the piece to be played
Update board
Display board
If player1’s move caused horizontal victory
    Print player1 win
    End the game
If player1’s move caused vertical victory
    Print player1 win
    End the game
If player1’s move caused left bottom to top right diagonal victory
    Print player1 win
    End the game
If player1’s move caused right bottom to top left  diagonal victory
    Print player1 win
    End the game
Repeat steps above for player2
```

Variables

const int ROW = 6;

- The name of the variable is called ROW.
- Its type is a constant integer.
- It represents the number of rows in the Connect 4 game.
- It is located on line 28.

const int COLUMN = 7;

- The name of the variable is called COLUMN.
- Its type is a constant integer.

- It represents the number of columns in the Connect 4 game.
- It is located on line 29.

Space board[ROW][COLUMN];

- The name of the variable is called board.
- Its type is a Space 2-dimensional array, which is a structure in the program located on line 19.
- It represents the number of columns in the Connect 4 game.
- It is located on line 29.

Piece player1 = X;

- The name of the variable is called player1.
- Its type is a Piece, which is an enumerator in the program located on line 13.
- It represents player1 of the two-player Connect 4 game.
- It is located on line 40.

Piece player2 = O;

- The name of the variable is called player2.
- Its type is a Piece, which is an enumerator in the program located on line 13.
- It represents player2 of the two-player Connect 4 game.
- It is located on line 41.

int turnCnt=0;

- The name of the variable is called turnCnt.
- Its type is an integer.
- It represents the number of turns taken for the Connect 4 game to reach an end.
- It is located on line 42.

Coordinate coord1;

- The name of the variable is called coord1.
- Its type is a Coordinate, which is a structure in the program located on line 23.
- It represents the location player1 played.
- It is located on line 43.

Coordinate coord2;

- The name of the variable is called coord2.
- Its type is a Coordinate, which is a structure in the program located on line 23.
- It represents the location player2 played.
- It is located on line 44.

int playRow = 0;

- The name of the variable is called playRow.
- Its type is an integer.

- It stores which row a user’s placed piece drops down to.
- It is located on line 105.

int playCol=0;

- The name of the variable is called playCol.
- Its type is an integer.
- It stores which column the user decides to place a piece.
- It is located on line 106.

Coordinate coord;

- The name of the variable is called coord.
- Its type is a Coordinate, which is a structure in the program located on line 23.
- It later stores the two above, playRow and playCol info with x and y info.
- It is located on line 107.

bool found=false;

- The name of the variable is called found.
- Its type is a boolean.
- It checks if it finds the first empty from the bottom up, as connect 4 piece drops to top of pile.
- It is located on line 108.

char input[100];

- The name of the variable is called input.
- Its type is a character array.
- It stores user input before verifying it.
- It is located on line 109.

int counter=0;

- The name of the variable is called counter.
- Its type is an integer.
- It counts how much in a row we have.
- It is located on line 152.

Concepts

I used enum Piece to represent all possible values of a spot on a board can be, in traditional board, its spot can be either red or yellow, or empty. It is located on line 13. I utilized structure Space to represent whether if a spot on the board is filled or not. I used structure Coordinate to represent a spot on the game board. It is located on line 23. I utilized while and for loops in various parts of the program such as on lines 53, 73, 74, 83, 88, etc. In addition, I used if and else statements in various sections of the program such as on lines 56, 61, 116, 123, 130, etc. I utilized a character array called char input[100] to store the user’s input before verifying it, which was located on line 109. I used a switch statement, switch(board[i][j].value) to check the value of the piece and return the corresponding string. I utilized a pointer variable, Coordinate* newCoord = new Coordinate(), which acts as an index pointer that we use to scan through the matrix. I used a binary file called “save.dat” to allow users to save and load the board at any given time.

Program

```
// Connect 4 Game
#include <iostream>  // I/O Library
#include <fstream>   // F Stream Library
#include <string>    // String Library
using namespace std;

enum Piece{//Piece represents all possible values of a spot on a board can be, in traditional board, its spot can be either red or yellow, or empty. Here we use X and O to represent yellow and empty
    X,
    O,
    Empty
};

struct Space{ // Whether if a spot on the board is filled or not
    Piece value;
};

struct Coordinate{ // Represents a spot on the board
    int x;
    int y;
};

const int ROW=6; // Number of rows in the Connect 4 game
const int COLUMN=7; // Number of columns in the Connect 4 game
```

```
void initialize(Space[ROW][COLUMN]); // Initializing the game board to where all the spots are empty and creates a board with given number of rows and columns

void printGame(Space[ROW][COLUMN]); // Prints the game board onto the screen

Coordinate playerMove(Space[ROW][COLUMN],Piece); // Plays and prints to a specific spot on the board according to the player's input

bool gameOver(Space[ROW][COLUMN],Coordinate); // Checks if the game is over by checking win conditions (Horizontal, Vertical, Diagonal Slash and Backslash, and Draw)

void saveGame(Space[ROW][COLUMN]); // Saves the game whenever the user wants to

bool loadGame(Space[ROW][COLUMN],Piece); // Loads the saved game when the user decides to resume the unfinished game



int main(int argc, char** argv) {


    Space board[ROW][COLUMN]; // Creates a game board with the designated number of rows and columns


    Piece player1=X; // Creates player1 or Player X

    Piece player2=O; // Creates player2 or Player O

    int turnCnt=0; // Number of turns taken before game reaches conclusion

    Coordinate coord1; //Holds the location player1 played

    Coordinate coord2; //Holds the location player2 played


    cout<<"Welcome to Connect 4. There are two players in this game, so let's play!"<<endl; // Prints a welcome message before game starts

    cout<<"This Connect 4 contains two players."; // Gives info on how many players can play the game

    cout<<"The players are player1 or Player X and player2 or Player O."<<endl; // How the players are represented when playing the game
    initialize(board); // Cleans the board completely, making all spots empty
    printGame(board); // Prints the game board onto the screen

    while(true){ // Using a while loop to check if a player's move resulted in victory
        turnCnt++; // Increments turn count whenever a player moves
        coord1=playerMove(board,player1); // Assigns coord1 variable to player1's move
        if(coord1.x!=-1){ // If player1 loaded the game and if it's player2's turn, then skip win check
        if(gameOver(board,coord1)){ // Checks if player1's move on the game board resulted in
victory
            cout<<" The game ended after "<<turnCnt<<" turns.";
            break; // If victory, the program exits out of the game
        }
        }

        coord2=playerMove(board,player2); // Assigns coord2 variable to player2's move
        if(coord2.x!=-1){ // If player2 loaded the game and if it's player1's turn, then skip win check
        if(gameOver(board,coord2)){ // Checks if player2's move on the game board resulted in
victory
            cout<<"The game ended after "<<turnCnt<<" turns."; // Prints a message
            break; // If victory, the program exits out of the game
        }
        }
    }
    cout<<endl<<"Thank you for playing the game."<<endl; // Prints a message

    return 0;
}
```

```
// Initializing the game board to where all the spots are empty and creates a board with given
number of rows and columns

void initialize(Space board[ROW][COLUMN]){

    for(int i=0;i<ROW;i++){ // For loop that goes through each row of the game board

            for(int j=0;j<COLUMN;j++){ // For loop that goes through each column of the game
board

            board[i][j].value=Empty; // Prints an string "_" to each spot on the game board using the
nested for loops above

        }

    }

}


// Prints the game board onto the screen

void printGame(Space board[ROW][COLUMN]){

    cout<<"  ";

    for(int i=0;i<COLUMN;i++){ // For loop that loops through the board from COLUMN 0 to
COLUMN-1

        cout<<i<<" "; // Prints an empty space for each spot

    }

    cout<<endl; // Prints a newline


    for(int i=0;i<ROW;i++){ // For loop that goes through each row of the game board

      cout<<i<<" ";

      for(int j=0;j<COLUMN;j++){ // For loop that goes through each column of the game board

          string value;

          switch(board[i][j].value){ // Switch statement that checks the value of the piece and return
the corresponding string

              case X: value="X";break;

              case O: value="O";break;

              case Empty: value="_";break;
          }
          cout<<value<<" "; // Prints the player's representation of play (X or O) on the chosen spot of the board according to the player's input
      }
      cout<<endl; // Prints a newline
    }
}


// Do the flowchart representation of this function
Coordinate playerMove(Space board[ROW][COLUMN],Piece myPlayer){
    int playRow = 0;//used to store which row a user's placed piece drops down to
    int playCol=0; // used to store which column the user decides to place a piece
    Coordinate coord; // used to later store the two above info with x and y info
    bool found=false; // checks if it finds the first empty from the bottom up, as connect 4 piece drops to top of pile
    char input[100]; // used to store user input before verifying it
    char playerLetter;
    if (myPlayer == X){playerLetter='X';}
    else if (myPlayer == O){playerLetter='O';}



    cout<<"Player: "<<playerLetter<<". Enter which column to put: (0-"<<COLUMN-1<<"), or 'X'to save and quit,'L' to load game"<<endl; // ask for input



    while(found==false){ // while user inputs is not valid, we keep asking user to try again


        cin>>input; // gets the input from the user
        if(input[0]=='x'||input[0]=='X'){//if user input x, or X, save game
           saveGame(board); // saves the game board
           cout<<"Player: "<<playerLetter<<". Enter which column to put: (0-"<<COLUMN-1<<"), or 'X'to save and quit,'L' to load game"<<endl; // ask for input
           continue;// if it is not valid, go back to ask input again
        }else if(input[0]=='L'||input[0]=='l'){//if user input l, or L, load game
           if(!loadGame(board,myPlayer)){Coordinate returnCord;returnCord.x=-1;returnCord.y=-1;return returnCord;}//if when loaded game, it is next player's turn instead, end function and return coord with -1 for x and y
           cout<<"Player: "<<playerLetter<<". Enter which column to put: (0-"<<COLUMN-1<<"), or 'X'to save and quit,'L' to load game"<<endl; // ask for input
           continue;// if it is not valid, go back to ask input again
        }

        // Checks if input is letter, word, or number
        if(input[0]<'0'||input[0]>'9'){
           cout<<"Please enter a number. We don't accept";
           continue;// if it is not valid, go back to ask input again
        }
        playCol=input[0]-'0';

        // Checks if the number is valid or not
        if(playCol>=COLUMN||playCol<0){//check if the column is out of bounds
          cout<<"Please enter again. Column Number needs to be less than "<<COLUMN<<" and greater than 0."<<endl;
          continue;// if it is not valid, go back to ask input again
        }

        // Checks the column availability for input
        for(int i=ROW-1;i>=0;i--){
            if(board[i][playCol].value==Empty){
                found=true;
                playRow=i;
                break;// if valid location found, break and place
            }
        }
        if(found==false){//if board full
            printGame(board);//display bored
            cout<<"Sorry, column "<<playCol<<" is full, please select another column."<<endl<<endl;
        }
    }
    board[playRow][playCol].value=myPlayer;//place user's piece at the location
    printGame(board);  //display board
    coord.x=playRow; //prep to return the x info for later result check
    coord.y=playCol; //prep to return the y info for later result check
    return coord;   //Return the info for later result check
}
void saveGame(Space board[ROW][COLUMN]){// save the current board to be loaded at another time
    fstream file; //create file instance
    file.open("save.dat",ios::out|ios::binary); // open save.dat file
    for(int i=0;i<ROW;i++){ //for all rows
        for(int j=0;j<COLUMN;j++){ // for all columns
            char writeTarget; //create char to be written to
            if(board[i][j].value==X){writeTarget='X';}// if player X is at the location, write X in file
            else if(board[i][j].value==O){writeTarget='O';} // if player O is at the location, write O in file
            else if(board[i][j].value==Empty){writeTarget='_';} // if no player is at the location, write _ in file
            file.write(&writeTarget,(sizeof(char)));//write to document
        }
    }
    file.close();//close file
    cout<<"Game Saved"<<endl;//show message to confirm saving
}
bool loadGame(Space board[ROW][COLUMN],Piece myPlayer){//when user enter "L" or "l". update board to reflect saved game and then the next player would play
    fstream file;//create file
    file.open("save.dat",ios::in|ios::binary);//open file to read
    int playerXCount=0;//counter check how many x pieces are there,to see whos turn is next
    int playerOCount=0;//counter check how many o pieces are there,to see whos turn is next
    bool result=true;// if it is not "myPlayer"'s turn next, return false
    for(int i=0;i<ROW;i++){//for each row
        for(int j=0;j<COLUMN;j++){//for each column
            char ch;//character to read file with
            file.read(&ch,(sizeof(ch)));//read char from file
           if(ch=='X'){board[i][j].value=X;playerXCount++;}//if character read is X, put that value on board then add to counter
           else if(ch=='O'){board[i][j].value=O;playerOCount++;}//if character read is O, put that value on board then add to counter
           else if(ch=='_'){board[i][j].value=Empty;}//if character read is _, put that value on board then add to counter

        }
        if(myPlayer==X&&(playerXCount>playerOCount)){// if it is player X's turn but loaded game indicates that O should play next
            result=false;//return false
        }
        else if(myPlayer==O&&(playerXCount<=playerOCount)){// if it is player O's turn but loaded game indicates that X should play next
            result=false;//return false
        }
    }
        printGame(board);//show board
    file.close();//close file
    return result;//return result
}



// Checks if the game is over by checking win conditions (Horizontal, Vertical, Diagonal Slash and Backslash, and Draw)
bool gameOver(Space board[ROW][COLUMN],Coordinate myCoord){

    // Horizontal Win Test
    int counter=0;// used to count how much in a row we have
    Piece myPlayer=board[myCoord.x][myCoord.y].value;//get the piece that is just put down
    char playerLetter;
    if(myPlayer == X){playerLetter='X';}
    else if(myPlayer == O){playerLetter='O';}
    for(int i=0;i<COLUMN;i++){ //loop through the row and see if there are four in a row
        if(board[myCoord.x][i].value==myPlayer){//if the piece is what user just placed
            counter++;//counter goes up
        }
        else{//otherwise
            counter=0;//counter resets
        }
        if(counter==4){// if we have four in a row of the piece type user just put down
```

```
            cout<<"Game Over. Player (hor)"<<playerLetter<<" has won the game!"; //player won, display method and return true
            return true;
        }
    }

    // Vertical Win Test
    counter=0;// used to count how much in a row we have
    for(int i=0;i<ROW;i++){//for the whole column,check if we have 4 in piece together
        if(board[i][myCoord.y].value==myPlayer){//if the piece at the current location is same as user placed
            counter++;// counter goes up
        }
        else{//otherwise
            counter=0;//counter gets reseted
        }
        if(counter==4){//if found 4 pieces connected
            cout<<"Game Over(ver). Player "<<playerLetter<<" has won the game!";//player won, display method and return true
            return true;
        }
    }
    // Diagonal Win Test
    counter=1;// in diagonal cases, we check two diagonal directions, for each direction, we count the number of connected pieces on two sub directions up, and down
    Coordinate* newCoord=new Coordinate();// an index pointer that we use to scan through the matrix
    newCoord->x=myCoord.x; //initialize x to the coordinate user just placed
    newCoord->y=myCoord.y; //initialize y to the coordinate user just placed
    //slash direction /
    //up
    newCoord->x=newCoord->x-1; // start from one row up
    newCoord->y=newCoord->y+1;// start from one column down
    while(newCoord->x>=0&&newCoord->y>=0&&newCoord->x<ROW&&newCoord->y<COLUMN){// while we are not out of bounds
        if(board[newCoord->x][newCoord->y].value!=myPlayer){// if we see none user placed piece like empty or opponents piece
            break;//break and check other directions as there won't be 4 in a row on this direction
        }
        else{// if see player's piece
            counter++;// increase counter
            if(counter==4){//if counted 4 in a row
                cout<<"Game Over. Player "<<playerLetter<<" has won the game!";//player won, deallocate space and turn true
                delete newCoord;
                return true;
            }
            newCoord->x=newCoord->x-1;// move to the next row up in the diagonal
            newCoord->y=newCoord->y+1;// move to the next column down in the diagonal
        }
    }
    //down
    newCoord->x=myCoord.x+1; // start from one row down
    newCoord->y=myCoord.y-1; // start from one column up
    while(newCoord->x>=0&&newCoord->y>=0&&newCoord->x<ROW&&newCoord->y<COLUMN){ // while we are not out of bounds
        if(board[newCoord->x][newCoord->y].value!=myPlayer){ // if we see none user placed piece like empty or opponents piece
            break;//break and check other directions as there won't be 4 in a row on this direction
        }
        else{// if see player's piece
            counter++;// increase counter
            if(counter==4){//if counted 4 in a row
                cout<<"Game Over. Player "<<playerLetter<<" has won the game!";//player won, deallocate space and turn true
                delete newCoord;
                return true;
            }
            newCoord->x=newCoord->x+1;// move to the next row down in the diagonal
            newCoord->y=newCoord->y-1;// move to the next column up in the diagonal
        }
    }
    //backslash
    //up
    counter=1;
    newCoord->x=newCoord->x-1;// start from one row up
    newCoord->y=newCoord->y-1;// start from one column up
    while(newCoord->x>=0&&newCoord->y>=0&&newCoord->x<ROW&&newCoord->y<COLUMN){// while we are not out of bounds
        if(board[newCoord->x][newCoord->y].value!=myPlayer){// if we see none user placed piece like empty or opponents piece
            break;//break and check other directions as there won't be 4 in a row on this direction
        }
        else{// if see player's piece
            counter++;// increase counter
            if(counter==4){//if counted 4 in a row
                cout<<"Game Over. Player "<<playerLetter<<" has won the game!";//player won, deallocate space and turn true
                delete newCoord;
                return true;
            }
            newCoord->x=newCoord->x-1;//move to the next row up in the diagonal
            newCoord->y=newCoord->y-1;// move to the next column up in the diagonal
        }
    }
    //down
    newCoord->x=myCoord.x+1;// start from one row down
    newCoord->y=myCoord.y+1;// start from one column down
    while(newCoord->x>=0&&newCoord->y>=0&&newCoord->x<ROW&&newCoord->y<COLUMN){// while we are not out of bounds
        if(board[newCoord->x][newCoord->y].value!=myPlayer){// if we see none user placed piece like empty or opponents piece
            break;//break and check other directions as there won't be 4 in a row on this direction
        }
        else{// if see player's piece
            counter++;// increase counter
            if(counter==4){//if counted 4 in a row
                cout<<"Game Over. Player "<<playerLetter<<" has won the game!";//player won, deallocate space and turn true
                delete newCoord;
                return true;
            }
            newCoord->x=newCoord->x+1;// move to the next row down in the diagonal
            newCoord->y=newCoord->y+1;// move to the next column down in the diagonal
        }
    }
    delete newCoord; // Deallocates the new coordinate after usage
    return false; // Returns false automatically if no victory is in existence
```

```
}
```