Project 2

Title

Connect 4

Course

CSC-17C

Section

48826

Date

December 9th, 2019

Author

Brandon Sanchez

Introduction

Connect 4 is a simple but fun board game that is played with two people. There is a six row, seven column board that starts out empty. Players take turns choosing columns to place their pieces in. The piece is placed in the lowest non-filled row in that column. A player wins if they form a horizontal, vertical, or diagonal line with four of their pieces. A tie is called if there are no remaining spots left and nobody has won. In my game, the player plays against the CPU.

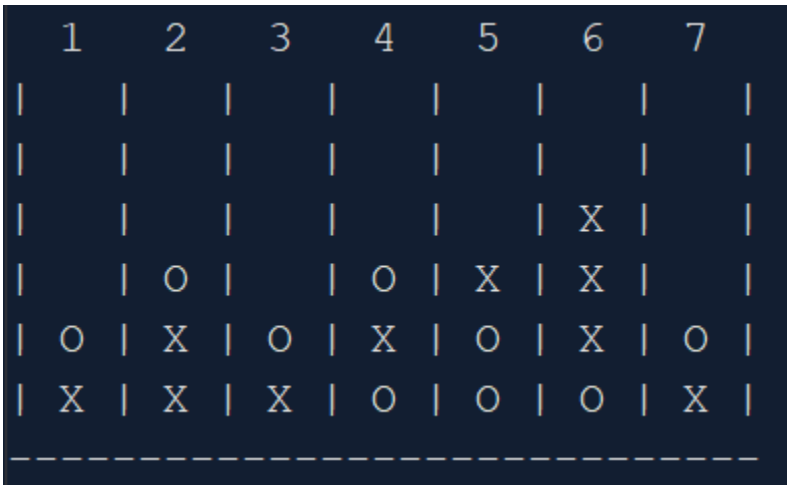
Gameplay with Sample IO

```
---------------
 ~ Connect 4 ~
---------------

Please enter your first name. Max 10 characters. Do not start with a number.
```

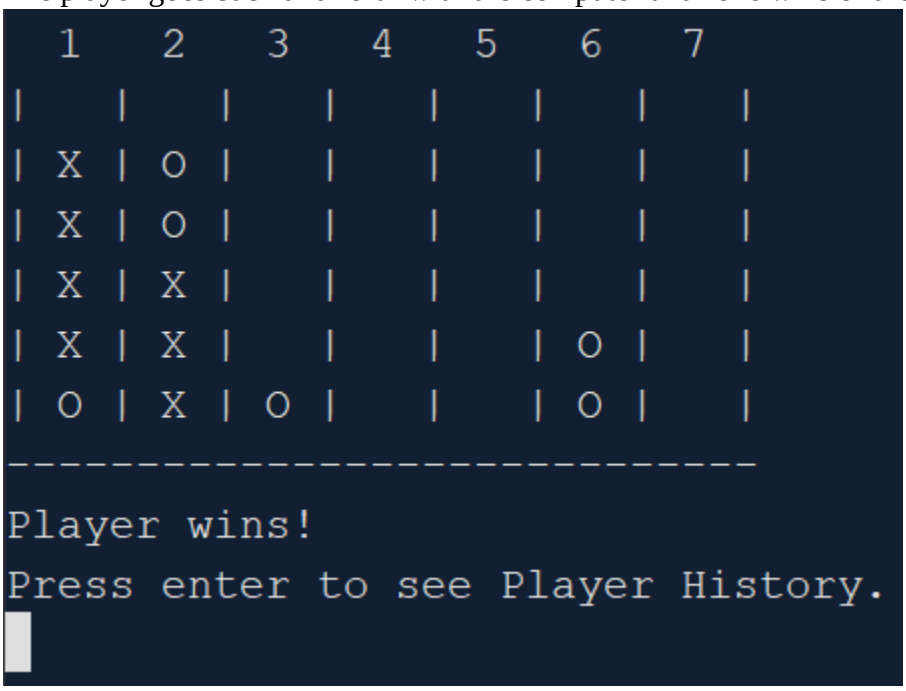
The user is greeted with this menu when they start the game.

```
Connect 4 pieces horizontally, diagonally, or vertically to win.
Player = 'X' Computer = 'O'

  1   2   3   4   5   6   7
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |

Which column would you like to place your piece? Type 0 to exit program.
```

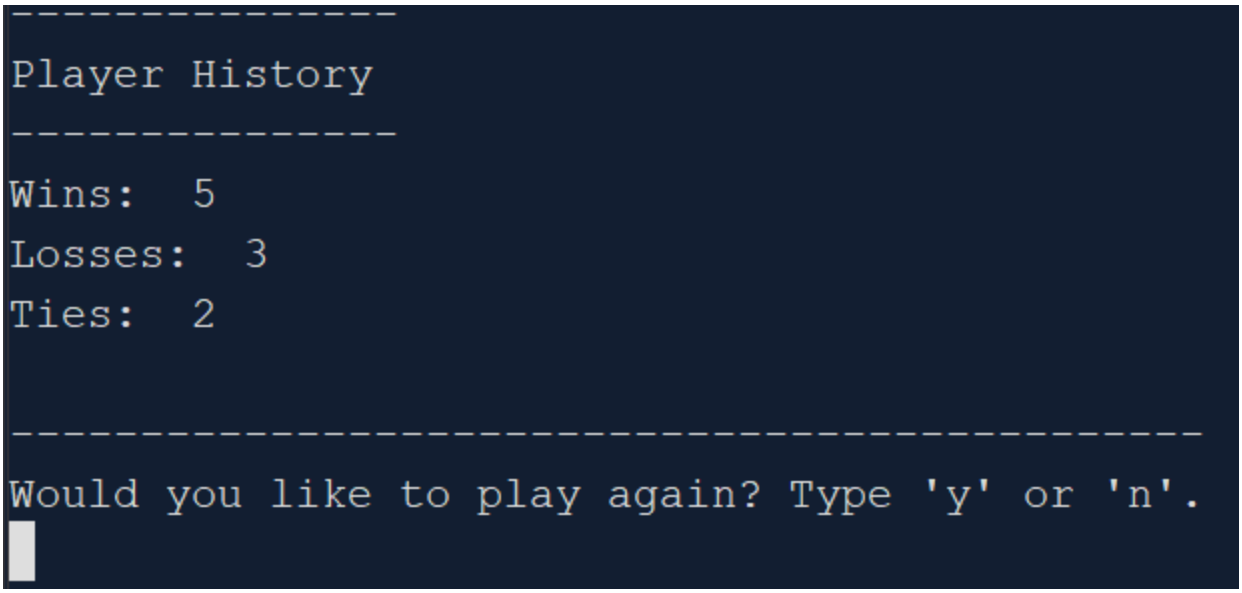
The program then outputs the rules, game board, and prompts the user to select which column they want to place their piece.



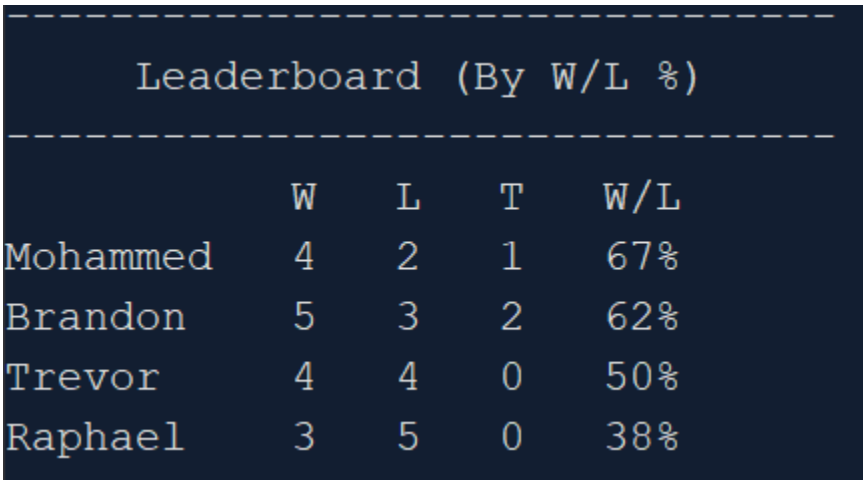
The player goes back and forth with the computer until one wins or they tie.



Once an outcome is decided (win/loss/tie), the message is shown and the player is prompted to press enter to continue.



The player will get to see their total wins/losses/ties. Afterwards, they are prompted if they would like to play again.



Finally after the player presses ‘n’ to stop playing, the leaderboard will be shown sorted by win/loss percentage. The program then exits.

Development Summary

Lines of Code: 1010 (Main) / 1412 (with classes)

I decided to continue with my Connect 4 game from the first project. All my implementations of the new concepts such as recursive sorts, hashing, and trees occur in the leaderboard. The hardest concept to implement was the hashing due to numerous bugs. The link list header file was created by the professor Mark Lehr and I simply utilized it. Alp sent me a hashing function since my previous one was not working properly.

New Concepts Usage

Hashing

Hashing is used to search for the name of the player in leaderboard file. All the player's names from the file are hashed into a link list. I then search through the link list to find the name. If found, the players wins/losses/ties are updated to reflect what is in the leaderboard file. If not, the player loses.

Recursive Sorting

I created a recursive sorting function to sort the leaderboard by win/loss when it is being stored in a vector. It stops once it hits the end of the vector. This was by far the easiest concept to implement because I changed my iterative sort into a recursive sort.

Tree

I rewrote the tree and node classes to work with my pair data type. The pair data type contains a string of the players name and a structure holding wins/losses/ties. I send the leaderboard data from a queue into the tree. Then, I output the leaderboard from the tree.

STL Library Usage

List

The board itself is a nested list (a list of lists). Each row is a simple list of integers with ‘0’ as empty, ‘1’ as player piece, and ‘2’ as computer piece. The rows are tied together with the nested list.

Map

Each player’s name and data from the leaderboard (and inputted) is stored as a map. Their name is the key and the value is a structure of their wins/losses/ties as sets.

Set

Each win/loss/tie has its own set. Every time a player wins/loses/ties, the set is then pushed back and incremented from the previous one. The last value is used for each set.

Pairs

Pairs are used to transfer my map into a vector so I am able to sort easily with the algorithm library.

Vector

Each pair of name/structure is pushed into a vector. Then using the algorithms library, I sort it based off win/loss percentage.

Algorithm

I utilized the algorithm library to sort my leaderboard based off win loss ratio. The sort function was very helpful in minimizing the complexity of my code.

Stack

The vector is then pushed into a stack.

Queue

The stack is then popped off into a queue to retain the proper order. The queue is then popped off to display the leaderboard.

Iterator

The iterator was used throughout to access most of the STL containers. It was particularly useful to iterate through the list so I can check if the player or computer won.

Future Goals

- Better computer AI, more difficult to beat
- Better board output, very confusing at times with current board

Pseudocode

Set random number seed.

Declares variables/containers.

Fill list with 0’s indicating empty space.

Output title.

Get player name.

Hash player data from file.

See if name is in leaderboard file using hash, update wins/losses/ties.

Output rules/print board

Get player choice of column.

Input validate.

Place player piece.

Computer’s turn.

If the player has 3 in a row, place piece to block it.

Else randomly choose a column.

Repeat until user quits or game ends.

Read from file players/totals into map.

Push pairs into vector.

Recursively sort vector based off win/loss

Push vector into stack.

Push stack into queue.

Push queue into tree.

Output the leaderboard using tree class.

End program.