PROJECT 1: CONNECT 4 DOCUMENTATION

BY: CARLOS VELAZQUEZ
DATE: 5/7/2021

Overview
This is documentation for my Connect Four Program that was created in C++ .

1 INTRODUCTION

1.1 Why Connect Four?

My game of choice was Connect Four. I've only played a few board games throughout my life, and the first one that came to mind was Connect Four. It was my favorite board game as a child. I remember that I enjoyed playing it at school with my friends and at home with my family. While I haven't played it in years, I decided to go with Connect Four due to my familiarity with the board game.

1.2 Time Spent, Number of Lines and Classes

I spent about 2 weeks working on this project. If I had to guess how many hours I spent in total, I would say about 20 hours. In total, there's about 979 lines of code.

1.2.1 Link to GitHub

https://github.com/cavelazquez8/ConnectFour

2 APPROACH TO DEVELOPMENT

This section describes the development process of the Connect Four game.

2.1 Concepts

Before I began the coding process, I took out a whiteboard, simulated a few games, and took notes of key things that I noticed. Here are some of them:

1. The board is a 6x7.
2. The player has at most 7 columns to choose from.
3. You only need to consider the column when making a move.
4. The row moves upwards the more checkers you place into a column.
5. You cannot place more checkers into a column once there's a checker at the top-most row.
6. If all spots on the grid are full, the game ends in a tie.
7. Starting from any square doesn't guarantee a row, column, or diagonal of four consecutive, same-colored checkers in both directions. For example, you cannot create a diagonal with four squares that moves Northwest starting from the square at the second row and second column; you can, however, make a diagonal that moves towards the Southeast.

Because all these concepts are applicable to the board, I created a Board class to address these concepts in the member functions; the actual 6x7 grid is represented with a two-dimensional vector.

2.2 Version Control

This section provides a description of all versions of the project.

2.2.1 Version 1

The program is a functional Connect 4 game, but it doesn't have header files and corresponding cpp files for the Player and Board class. In addition, it's well below the line limit (329 lines) and it doesn't include all the STL concepts required.

2.2.2 Version 2

The program is functional and includes headers files and corresponding cpp files for all classes. The program has an additional header file and cpp file named "InputValidation" which holds functions for validating input, which any class and function can use. Still, the program is below the minimum line requirement and no new STL concepts are introduced. I try to clean the main even more by creating functions within the main. While the main looked cleaner, it wasn't ideal.

2.2.3 Version 3

This program includes the code for a computer mode. Besides that, not many changes.

2.2.4 Version 4

This version adds a header file and cpp for the Game class. The purpose of this class is to help clean up the main. In addition, the Player class is modified to allow the user to initialize the data attributes when a Player object is created. On top of this, more STL concepts are included, concepts such as swap(), bit_vector, and find(). The main is much cleaner. The program is closer to the minimum line requirement, but still not there.

2.2.5 Version 5

This version includes additional member functions for the Board class. One of them is for utilizing the additional STL concept: Maps.

2.2.6 Version 6

This version includes additional STL concepts: queue and priority queue. Line minimum reached: 860. The final product.

2.2.7 Version 7

Version 7 is fully functional, includes all the required STL requirements, and is organized through the use of header files and cpp files.

2.3 Game Rules

The rules are simple: be the first to form a horizontal, vertical, or diagonal line of your own four checkers.

2.4 Description of Code

This section provides a description of the code, more specifically the organization and use of classes.

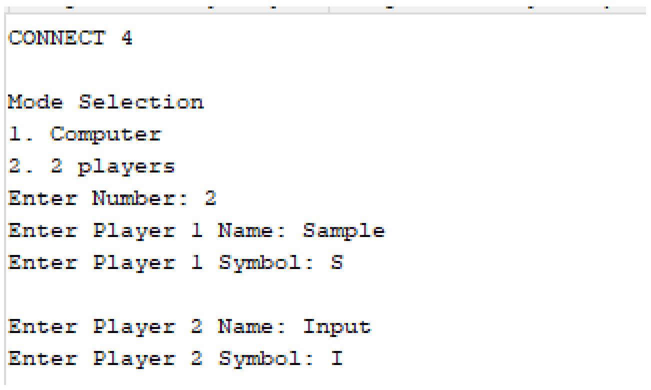
2.4.1 Organization

The program relies heavily on object-oriented programming; it's organized through three separate classes: Player, Board, and Game. Due to this, the main.cpp has a minimalistic look and only has one function for the main menu.

2.4.2 Classes

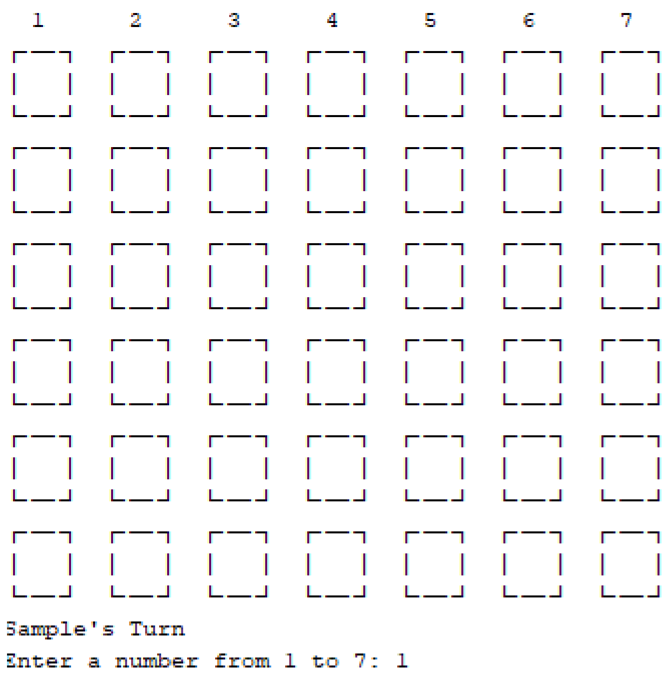
The Board class has four data attributes, one constructor, and eleven member functions.
The Player class has two data attributes, three constructors, and two member functions.
The Game class has four data attributes, one constructor, and three member functions.

3 SAMPLE INPUT/OUT



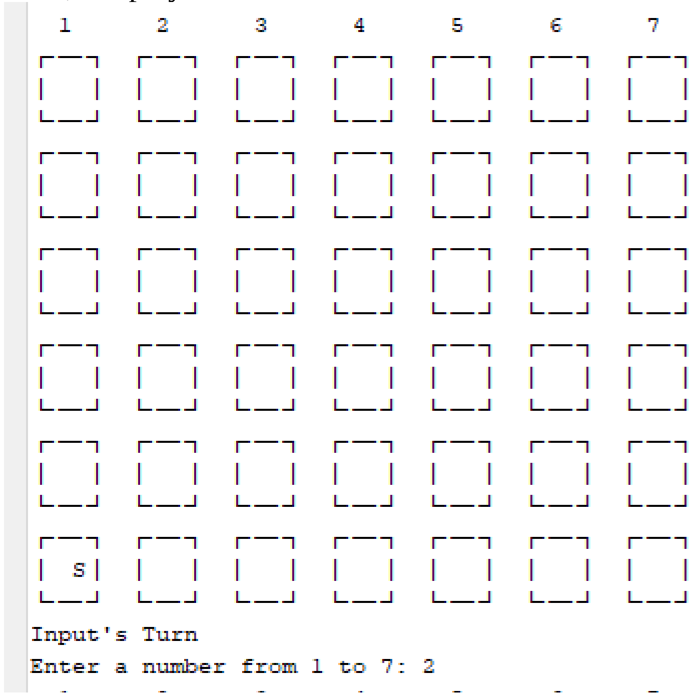
```
CONNECT 4

Mode Selection
1. Computer
2. 2 players
Enter Number: 2
Enter Player 1 Name: Sample
Enter Player 1 Symbol: S

Enter Player 2 Name: Input
Enter Player 2 Symbol: I
```

Main Menu
If the user chooses 2, it will allow both players to enter their name and symbol.



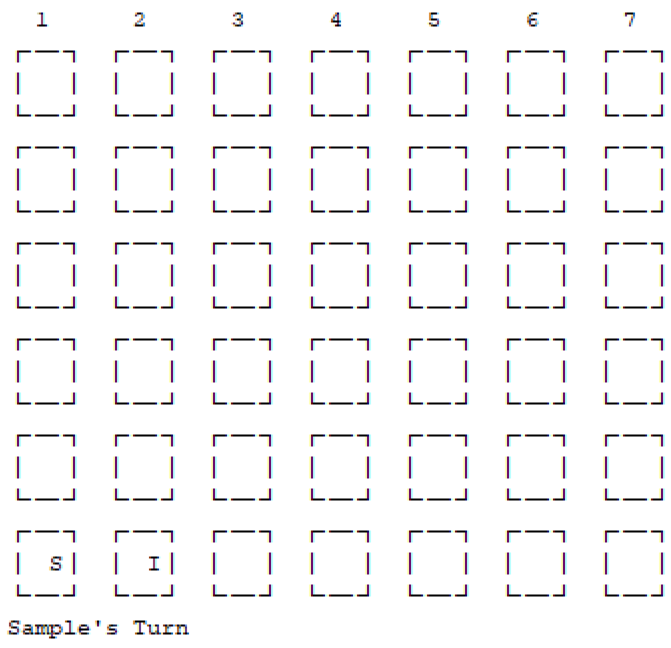
```
  1      2      3      4      5      6      7
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
Sample's Turn
Enter a number from 1 to 7: 1
```

Let’s Player 1 take their turn.
Here, the player selects the first column.



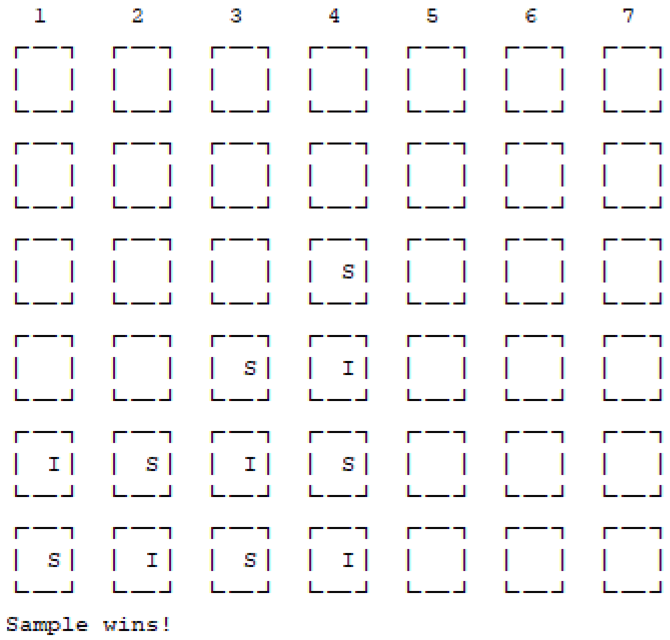
```
  1      2      3      4      5      6      7
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│  │  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐  ┌──┐
│ S│  │  │  │  │  │  │  │  │  │  │  │  │
└──┘  └──┘  └──┘  └──┘  └──┘  └──┘  └──┘
Input's Turn
Enter a number from 1 to 7: 2
```

Player one’s move is shown on the grid
Player two is allowed to take their turn
Here, player two chooses column two



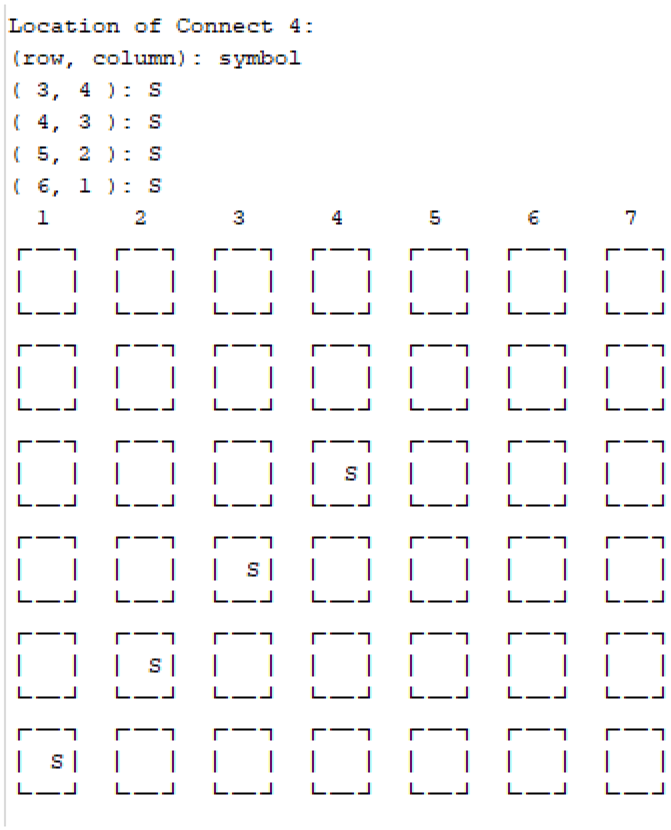
```
 1      2      3      4      5      6      7
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│ S│   │ I│   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
Sample's Turn
Enter a number from 1 to 7: 2
```

Player 2’s move is shown on the grid
From here, repeat until there’s a Connect Four



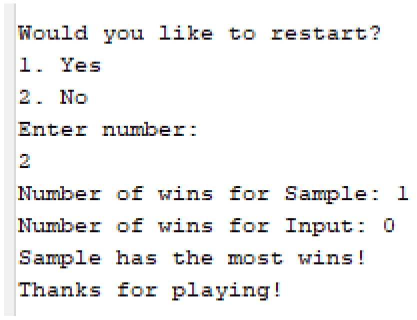
```
 1      2      3      4      5      6      7
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │  │   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │  │   │ S│   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│  │   │  │   │ S│   │ I│   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│ I│   │ S│   │ I│   │ S│   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐   ┌──┐
│ S│   │ I│   │ S│   │ I│   │  │   │  │   │  │
└──┘   └──┘   └──┘   └──┘   └──┘   └──┘   └──┘
Sample wins!
```

A connect four happens!



```
Location of Connect 4:
(row, column): symbol
( 3, 4 ): S
( 4, 3 ): S
( 5, 2 ): S
( 6, 1 ): S
  1      2      3      4      5      6      7
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│   │  │   │  │   │  │   │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│   │  │   │  │   │  │   │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│   │  │   │  │   │  │ S │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│   │  │   │  │ S │  │   │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│   │  │ S │  │   │  │   │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐  ┌───┐
│ S │  │   │  │   │  │   │  │   │  │   │  │   │
└───┘  └───┘  └───┘  └───┘  └───┘  └───┘  └───┘
```

The program then shows the location of where the connect four happened and displays it on screen.



```
Would you like to restart?
1. Yes
2. No
Enter number:
2
Number of wins for Sample: 1
Number of wins for Input: 0
Sample has the most wins!
Thanks for playing!
```

The programs ask the user if they would like to restart.
The user selects no, and the program displays how many wins each player had and which player had the most wins.

4 CHECK OFF SHEET CONTENTS

This section shows what concepts of the STL library were used.

4.1 Containers

4.1.1 Sequence Containers

4.1.1.1 Bit_Vector

Bit_Vector was used in winner() function inside the Board.cpp. It was used to store the results from the functions that determine if there was a Connect Four in the diagonals, horizontals, or verticals. Then, it was used to determine if any of the functions returned true.

4.1.1.2 Vector

Vector was used to represent the Connect Four game board, which is the data attribute of the Board class.

4.1.2 Associative Containers

4.1.2.1 Map

Map was used to display where the winning combinations occurred. It is a data attribute named winningSpots for the Board class.

4.1.2.2 Set

Set is used in the checkNum function of the InputValidation header file. The set is named playerOptions and is used for input validation by storing the number of options a player can choose from and checking if the user input matches with one of those options.

4.1.3 Container Adapters

4.1.3.1 Queue

Queue is used as a data attribute for the Game class. This can be seen in the Game.h. The queue is named trackWinner and is used to store the wins of each player and display the number of wins each player had at the end of the game.

4.1.3.2 Priority Queue

Priority Queue is used in the finalResult() function of the Game class. It is named overallWinner and is used for storing the number of wins each player had and determining which player had the most wins.

4.2 Iterators

4.2.1 Random-access Iterator

When it was applicable, the only iterator I used was the Random-access iterator. The random-access iterator functions like a pointer and can be used the access the element of a container. This was used for all my vector, set, maps, and bit_vectors.

4.3 Algorithms

4.3.1 Non-Mutating Algorithms

4.3.1.1 Find()

Find() was mainly used for finding a certain element in a container. It was used with bit_vector for finding if any of the elements were true. It was used with my playerOptions set to find the user's choice in the set.

4.3.2 Mutating Algorithms

4.3.2.1 Swap()

Swap() was used in the computerDrop() function in the board class. It was used to switch a temporary variable with the original variable.

4.3.3 Organization Algorithms

4.3.3.1 Sort()

Sort() was used with bit_vector to speed up the searching process with find().

5 DOCUMENTATION OF CODE

5.1 Pseudocode

```
//Location of function: Board.cpp/checkDiag()
//Purpose: Search for Connect Four along Diagonals

//For diagonals moving from Southeast to Northwest
Start with a 6x7 board
PRECONDITIONS: Start from the 6th row. IF: you're not at the 3rd row, move up a row;
otherwise, STOP.
FOR LOOP: For each row in the board
PRECONDITIONS: Start from the 7th row; IF: you're not at the 3rd column, move a          column
to the left; Otherwise: STOP
FOR LOOP: For each column in the board
IF: Starting from the current row and column, Is there a Connect Four in the
diagonal moving towards the Northwest?
                              IF SO: RETURN TRUE

//For diagonals moving from Northwest to Southeast
PRECONDITIONS: Start from the 6th row. IF: you're not at the 3rd row, move up a row;
Otherwise, STOP.
FOR LOOP: For each row in the board
PRECONDITIONS: Start from the 1st row; If you're not at the 3rd column, move a          column
to the right; Otherwise, STOP.
FOR LOOP: For each column in the board
```

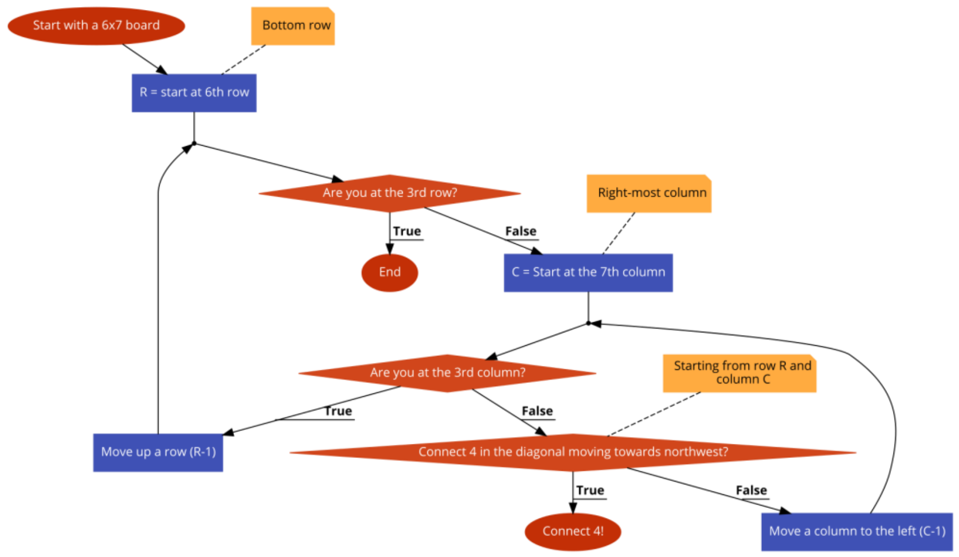
IF: Starting from the current row and column, Is there a Connect Four along the diagonal moving towards the Southeast?

IF SO: RETURN TRUE

RETURN FALSE if there's no RETURN TRUE

End

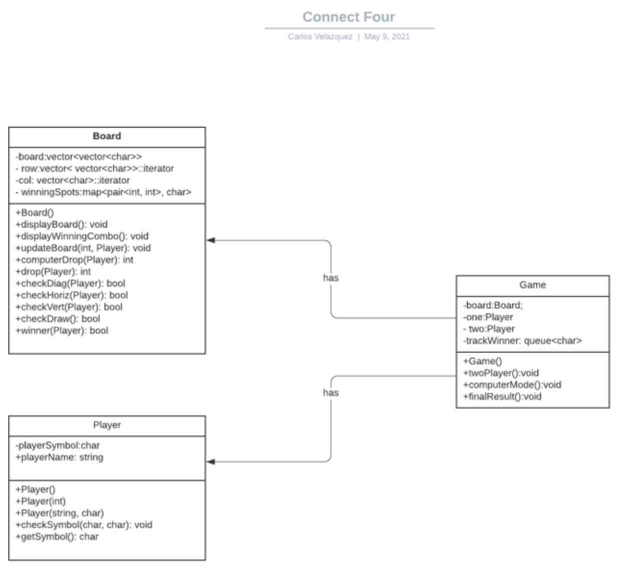
5.2 Flowchart


Start with a 6x7 board
Bottom row
R = start at 6th row
Are you at the 3rd row?
Right-most column
True
False
End
C = Start at the 7th column
Are you at the 3rd column?
Starting from row R and column C
True
False
Move up a row (R-1)
Connect 4 in the diagonal moving towards northwest?
True
False
Connect 4!
Move a column to the left (C-1)

Location in code: Board.cpp/ checkDiag()/ Lines 160 – 174

This piece of code searches for a Connect 4 along diagonals moving Southeast to Northwest.

5.3 UML


Connect Four
Carlos Velazquez | May 9, 2021
Board
-board:vector<vector<char>>
- row:vector< vector<char>>::iterator
-col: vector<char>::iterator
- winningSpots:map<pair<int, int>, char>
+Board()
+displayBoard(): void
+displayWinningCombo(): void
+updateBoard(int, Player): void
+computerDrop(Player): int
+drop(Player): int
+checkDiag(Player): bool
+checkHoriz(Player): bool
+checkVert(Player): bool
+checkDraw(): bool
+winner(Player): bool
has
Game
-board:Board;
-one:Player
- two:Player
-trackWinner: queue<char>
+Game()
+twoPlayer():void
+computerMode():void
+finalResult():void
has
Player
-playerSymbol:char
+playerName: string
+Player()
+Player(int)
+Player(string, char)
+checkSymbol(char, char): void
+getSymbol(): char