Connect Four Game - C++ Project

C++ Programming Project

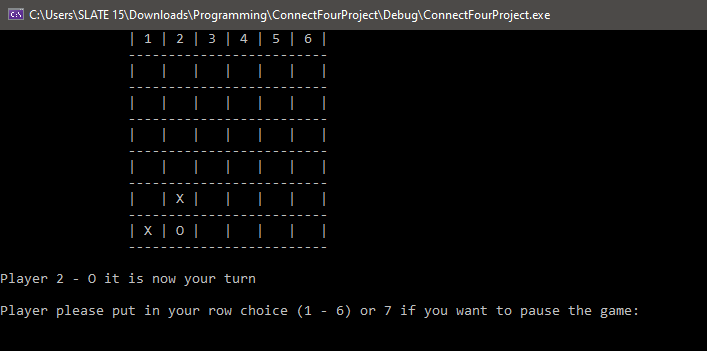
Hannah Ashna Jacob

N0865554

**Specification:**

**Project Aim:** To create a digital version of the Connect Four game

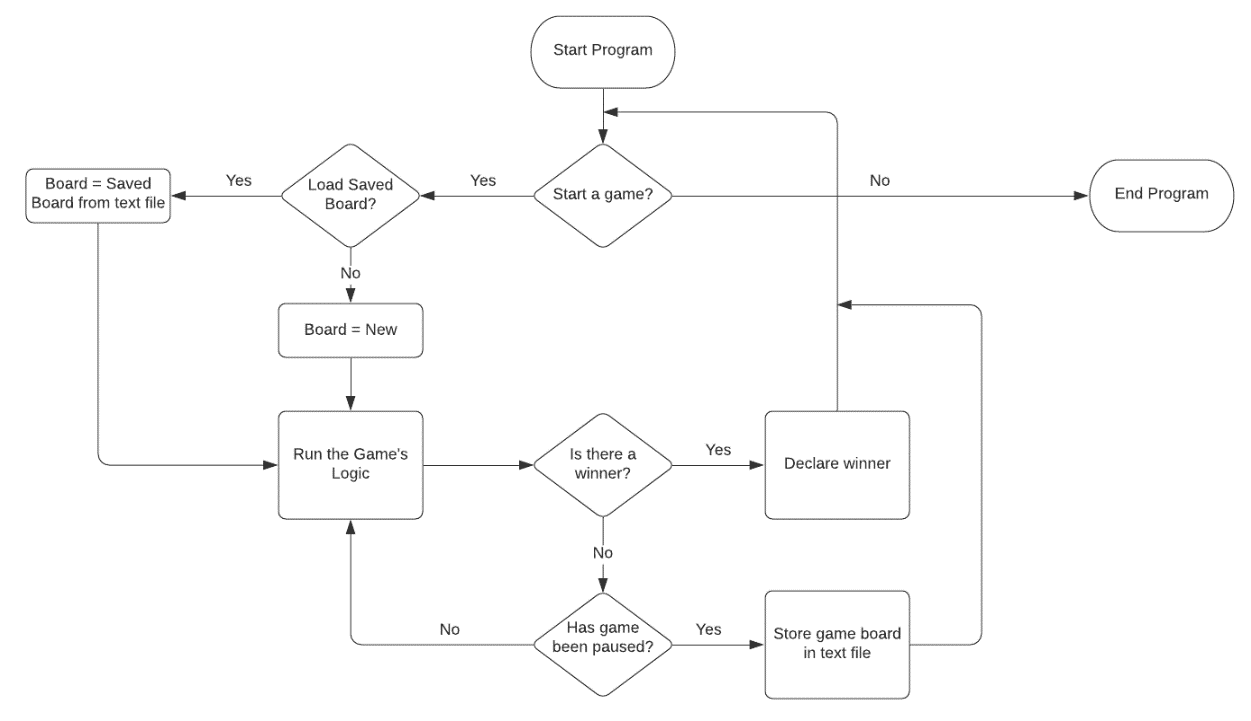
**Description:** The game emulates the basic logic of a standard connect four game. It also includes an additional feature that allows the user to pause a game midway and have the board saved into a text file. This file can then later be retrieved and loaded in as a new board object within the program.



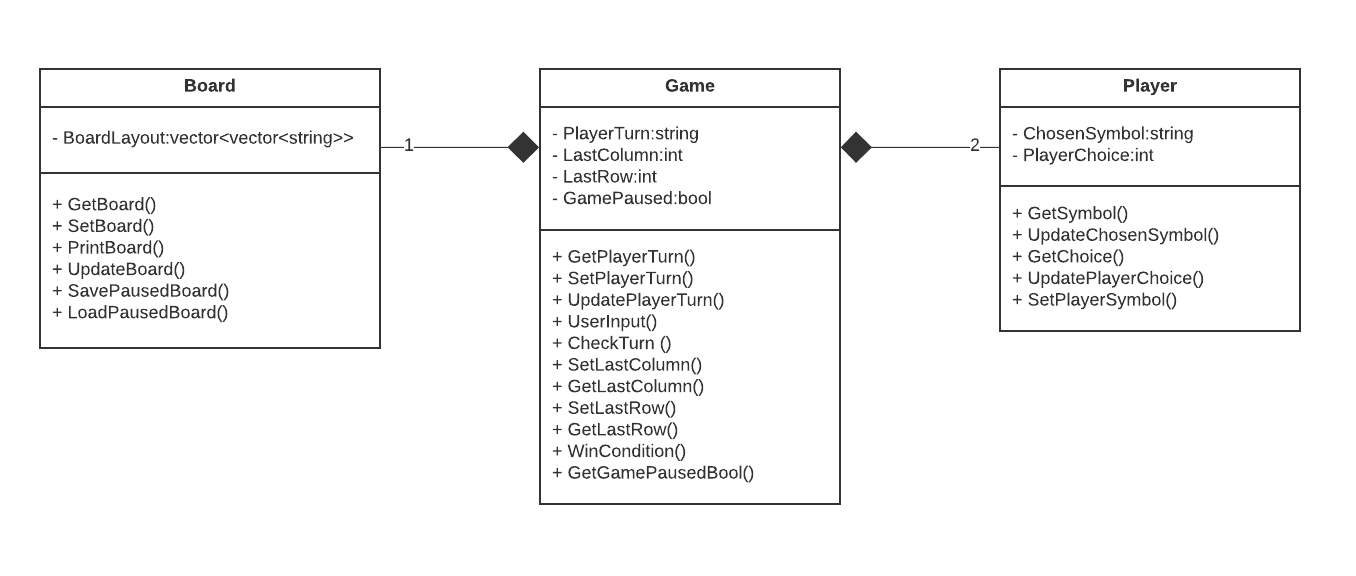
**Requirements:**

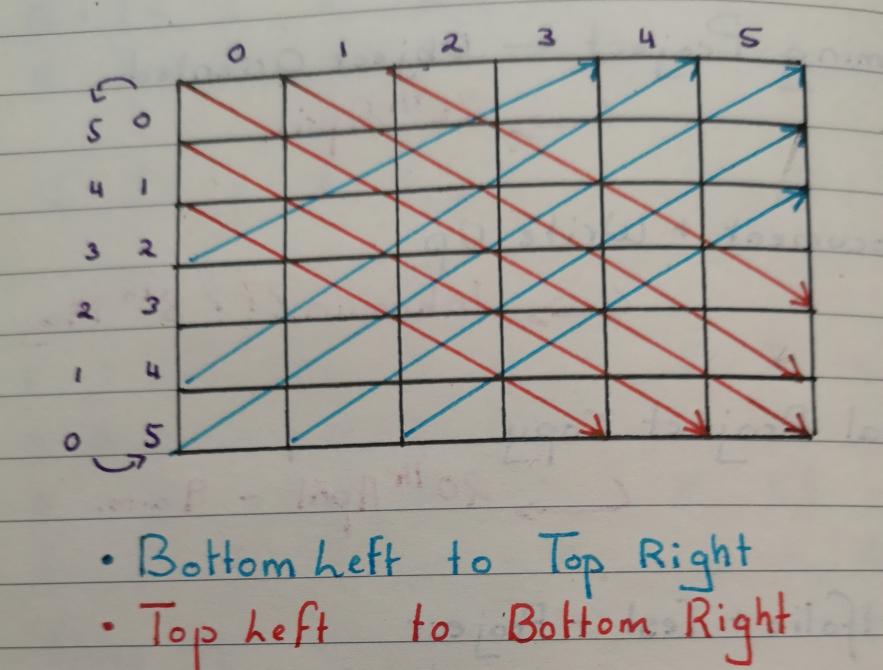
* The program requires 3 separate classes:
  + The game class which will create a new object whenever a new game is created.
  + The player class which will have 2 primary objects; this is to account for the two players for each game.
  + The board class which will create a new board whenever a game is started.
* The program will also require an appropriate file handling system/function to store and retrieve a game’s board whenever a user either chooses to pause the game or load in a previously paused game.
  + This should be classified as a method under the board class
* The program will need include guards within all header files to prevent include recursion/double inclusion errors whenever one class tries to reference another in its method parameters.
  + This will also allow the program to work on other compilers because include guards are compliant and supported across all compilers. Hence, are a much better option compared to just using “#pragma once”.
* The program will also need to use the vector header file as vectors will be used to store the game’s board.
* Develop and format the board appropriately using ASCII-based art so that users can see the board on the console and be able to watch it get updated after each turn.
  + This promotes user-interaction with the program as it makes the game more visual rather than primarily just using text-based interactions.

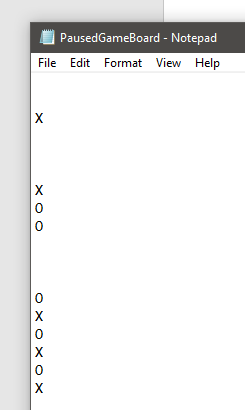
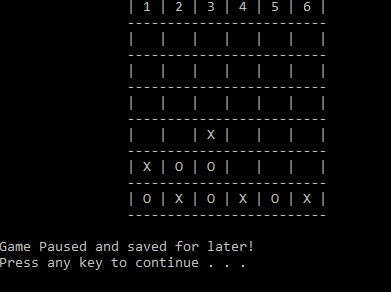
**Design and Implementation:**

**Main Program:** Before working on my program I decided to create a simple flowchart detailing the set of events that would occur when a user started up the program and played the game. This also includes the scenario in which users decided to pause the game.

**Classes:** While creating my program, I decided that it was best to breakdown my three main classes in a way that allowed me to visualize how I intended for my classes to interact with each other. Hence, I created a class diagram for it.

****Through the diagram, I was able to list off a few methods and variables that I intended to implement. I first started with everything being under one class called game, however, this was rather inefficient which resulted in me splitting them into these three classes.

**Win Condition Logic:** This portion of the program was relatively easy when it came to the implementation of the horizontal and vertical win-conditions. However, I still had to consider the two sets of diagonals (top left to bottom right and bottom left to top right). I first attempted to use counter logic for these two remaining conditions just like I had done with the previous two. Unfortunately, it created a whole new set of errors. With some help from stack overflow and a little bit of research, I learn that it was best to first draw up the table and map out a pattern for the diagonals. I noticed that if I check between a certain range of column values and used a “zig-zag” pattern for how my program was checking for symbols then this would allow for me to efficiently create a win-condition for the diagonals.  
  
Subsequently, I decided against using the counter for ‘how many in a row’ were present in a diagonal and chose to just use a series of AND statements instead. Hence, I referred to the stack overflow discussion’s code sample to develop a different style of win condition check for the diagonals. This helped prevent an error which would result in the program over-looking certain symbols and therefore not properly fulfilling the win-condition check’s purpose.

**File Structure:**

The board is stored in a text file wherein each symbol is stored on a single line followed by a break so that the next symbol is stored on the following line below it.

This allowed for the file reading/writing process to be far more efficient as it prevented from symbols being accidentally duplicated elsewhere while the file was being read/written to in a for loop.

**Testing:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Test** | **Success Outcome** |
| User input throughout the entire program | 1. When the program is started up the user will have the choice of starting a game or ending it. They can respond with either (Y or y) to say yes to starting a game or (N or n) to say no to starting a game. First chose either (Y or y) or (N or n) then try an alternative value.  2. When prompted if the user intends on loading in a previously paused game board the user has 2 options, either yes (y) or no (n). Hence, attempt to input these 2 values first in both upper and lower-case and then try other values.  3. When player 1 is asked to choose a symbol, they are given an option of between X and O. Hence like the test above, try using upper-case and lower-case versions of ‘X’ and ‘O’ before then inputting alternative values instead.  4. When users are asked to choose a column value, they have the option to pick between 1-6 and then 7 if they want to pause the game. Try choosing values within the 1-6 range then the value 7 to see if it pauses the game. Finally, attempt to select a value outside the range of 1-7. | 1. When the user responds with ‘Y’ or ‘y’ the game will start, whereas if the user responds with ‘N’ or ‘n’ the program will end. When an alternative value is entered the program will prompt the user for their input again until they respond with an appropriate input value.  2.When the response is either a ‘Y’ or a ‘y’ the program will load up the saved board. However, if the response is ‘N’ or ‘n’ then it will start with an empty board. Finally, if the response is an alternative value it will prompt the user for their input again until they respond appropriately.  3. When the response is O, player 1 is assigned the ‘O’ as their symbol while player 2 is given ‘X’. Subsequently, if player 1 chooses ‘X’ then player 2 gets ‘O’. Lastly, if an alternative value is chosen then the program will prompt the user to keep trying until they select one of the two options.  4. When the user chooses a value between 1-6, the ASCII board art will update the board with their symbol being placed in a column of their choice. However, if the user chooses 7 the game is paused/ended, and the board is saved to a text file. In the case the user picks a value outside of this 1-7 range they will continue to be prompted to pick an appropriate value that fits the range of choice. |
| The ability to pause the game at any given point during a game if there is no winner | While a game of connect four is running, the user will have the option of responding with the number 7 instead of a column value choice between 1-6. Hence, input the value 7 and then check the text file to see if the board has been saved. Alternatively, load up the saved board in a new game. | The board is immediately saved in the text file and is ready to be accessed whenever the user chooses to access a paused game board instead of starting on an empty one. |
| The ability to load in a saved game board instead of starting on a new one | At the beginning of the game, the user is prompted on whether they’d like to use a saved board instead of a new one. By inputting ‘Y’ or ‘y’ the program will then load in the saved board and run the game using it. | The board is immediately loaded in and printed into the console using ASCII art so that the user can see it. |
| Column overflow prevention in case the user keeps trying to pick a column that is already full. | Run a new game of connect four and then repeatedly select one particularly column for both players until it is full. Then try to place a user’s symbol in that column to test whether the program simply allows them or prevents them from doing so. | When the column is full, and a user selects it the program will prompt the user to pick a column value again as their choice is considered an inappropriate value. This prevents the user from causing any vector indexing problems in the case that they continue to overfill a column outside of the board’s range. |
| The game’s win conditions | 1. Horizontal Win Condition – to test this simply play the game and attempt to win it by getting 4 in a row of a player’s symbol in a horizontal pattern.  2. Vertical Win Condition – to test this simply play the game and attempt to win it by getting 4 in a row of a player’s symbol in a vertical pattern.  3. Diagonal Win Condition – to test this simply play the game and attempt to win it by getting 4 in a row of a player’s symbol in a diagonal pattern. Since there are two diagonals the user will have to attempt both a test for both directions, (top-left to bottom-right and bottom-left to top-right). | The program will recognise that there are 4 consecutive placements of the player’s symbol in a row and a particular direction and will declare that that player is the winner and will also output the particular win condition that they used to win (Horizontal, Vertical or Diagonal). |
| Save/Pause an empty game board | Once a game is started, instead of putting in any symbols simply pick option ‘7’ to save the board. Then in a new game, ask to load up a paused game board and attempt to play using it. | The board’s ASCII art should come up with a blank board with no symbols in it. |

**Evaluation:**

The program contains all the base logic to complete the function of it being a connect-four game. The user can win the game using the 4 different win conditions in addition to being able to pause the game at any point if the game has not yet been won.

Users also have the option to load in their saved game instead of starting up a new one. Hence, with regards to this, the project proposal has been fulfilled. I also attempted to implement a timer to the program. However, upon further research into it, I found that it was relatively time-consuming to do and an exceptionally long block of code. I decided against it and chose to focus on improving the pause game feature instead. The timer would be one primary way I would improve this program as it provides an additional level of competitiveness to the game as players compete for the fastest win time.

Subsequently, players can currently only pick an “X” or “O” symbol. This was primarily done so that when it came to loading in a saved game, the program could decide whose turn it should be based on a count of how often each symbol occurred. Adding more symbol options meant that the code would have to be a little bit more complicated and this would’ve taken up more time to implement. Therefore, this would be another way that I would improve the program in the future as it gives the user a little bit more freedom to chose what kind of symbols they want to use for the game.