**Abstract**

Our project is a Connect 4 application made with C++. From the standpoint of the user, the program starts and you are prompted to a main menu. You have options to play a new game, load game, learn how to play Connect 4, and exit. When selecting new game, you are given options on how to initialize the game. You can choose between black and red or 1 and 2 players. Next as done in load game, you are brought to a screen with the game board. You are given the option to select a column or save. After a game, you remain on the board screen until you press enter. This will bring you back to our main menu where you can choose to exit if you are done playing.

**Background**

The majority of our code was made from scratch when it comes to how much we used from other sources. Sometimes we did not necessarily know a good way to implement something, so we would find a better solution out there, but still managed to make it our own in our code.

We did find a solution to getting passed the cin input for our columns. We read up on how to read the key states of the keyboard. This was our solution to taking in Arduino and keyboard input simultaneously. Our process for this is described in the implementation section.

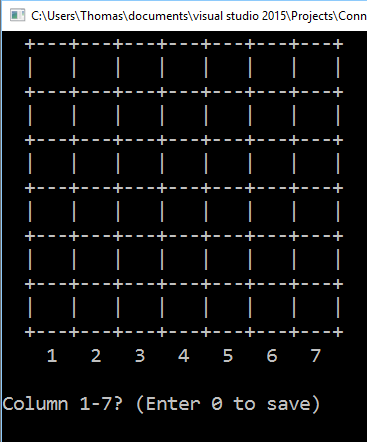
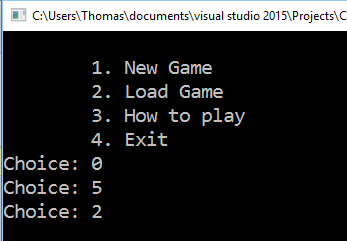
We also used a variety of header files. The majority of them were common headers that we used in previous labs to implement vectors, strings, files and input/output. Although there were a couple uncommon ones, for example we used the stdlib.h header from c to incorporate the rand() function into our computer’s move. We also used the Windows.h header to incorporate our key state input and the ofArduino.h header to incorporate our hardware aspect to the project.

With the Arduino, there are many methods to receiving the input. We took the common approach of saving the current and previous state of the buttons to be pressed. We made them static variables to keep for every use of the function. If at any point they were pressed, the current state would change. The current state would be compared to the previous state to determine if they were different which would be interpreted that the button was pressed. We would then update previous button pressed when current changed. We also could not just return the value of the button pressed if current and previous were different. We had to make sure that the previous state was not on before returning the value. If the previous state was on and current was off we would not want it to simulate a button press. This is just a common approach we came across while trying to take in inputs that was best suited for how we wanted to take in computer input as well.

Our game can be used in an assortment of ways. The most common place that we can implement the game is as a simple application on a computer. Computer games are very common and would be a great selling point, but we also brought hardware into the game to show that we could make a sort of a digital game that you could pull out of a box in your game closet. We could have a digital screen with buttons under the columns for easy use in a fun interactive game anywhere.

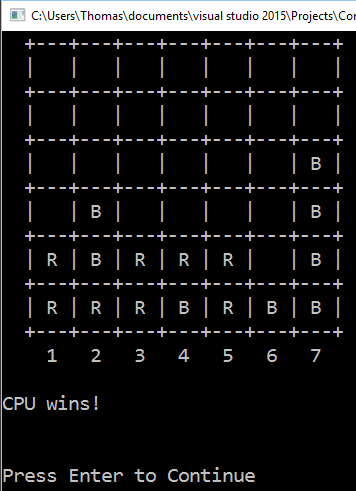
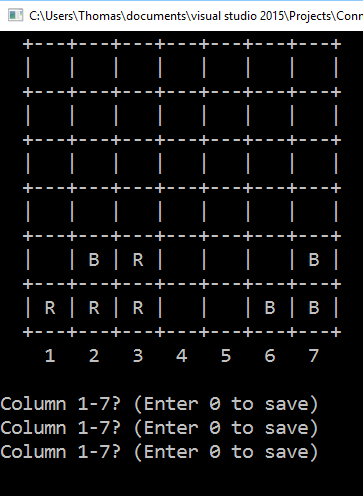
**Experiments and Results**

There were tons of tests we conducted throughout this project. We attempted to break it in every way possible. We believe we have finally programmed it to work under all conditions. The first test we conducted was on the main menu where you are prompted for four choices so anything outside of 1-4 would not be valid input. The way we fixed this problem was by simply asking for their choice of input until it’s within 1-4 as shown below on the left.



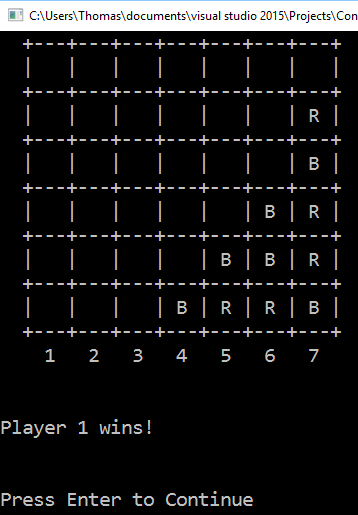
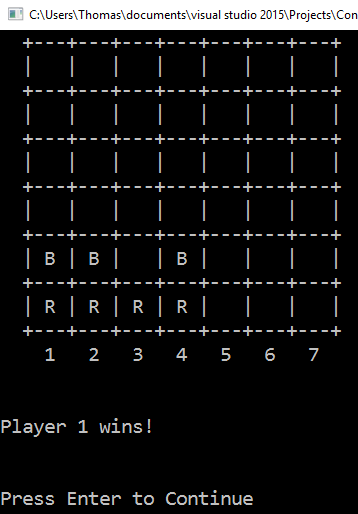
The next possible source of error would be if we tried to load a game without any previous save. The simplest solution was to just have a default save.txt file that was a new, single player game, where player 1 is red. The new game is displayed above on the right that results from loading to start the game.

The next test we would conduct was to make sure we had valid column input. The way we take in column input is by waiting for a button 0-9 to be pressed. So if 8 or 9 is pressed it will display the column/save request again until receiving proper input as shown in the example to the left below. This also allows for other characters to be pressed since it is waiting on the buttons to be pressed from 0-9. The inputs for 8 and 9 are the only ones that will display any sort of error, but could be utilized later if we decided to add other sort of features.



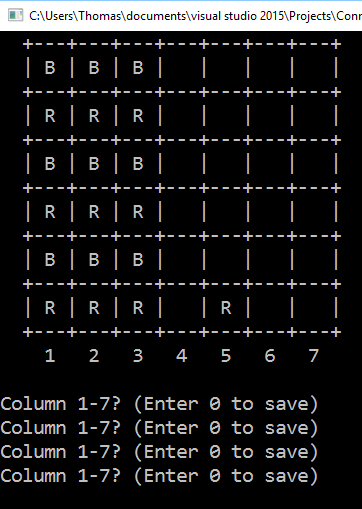
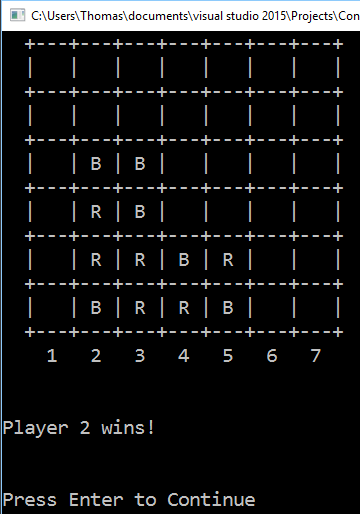
Then we also have tests for all the possible ways on how someone can win and who the person that wins is. For the first example of this we have the computer winning in a vertical way. When the game is over it displays the correct person and the board as shown to the right above.

The next test we did was to show horizontally winning as player 1. This was done in a 2 player game to simplify setting up situations as most tests will be done. In this scenario, player 1 also won by placing the piece in the middle so it would show how we count on both sides of the piece. This is represented in the situation to the left below.

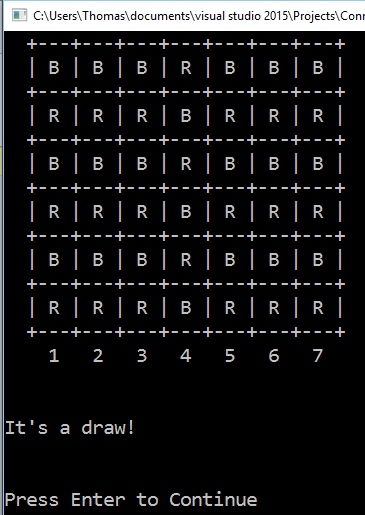


The next test was to show a positive slope connect 4. In this scenario, player 1 is black instead of red and they win by placing a piece in column 6. This represents a different color winning, but still the same player and in a different direction then previously tested.

The next scenario is to show player 2 winning as black. This situation shows winning in the negative slope direction. It also displays correctly that player 2 has won the game. The screenshot of the situation is shown below and to the left.



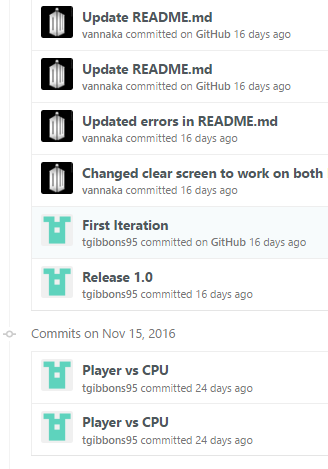
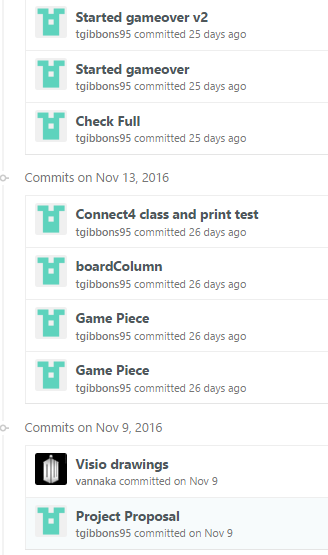
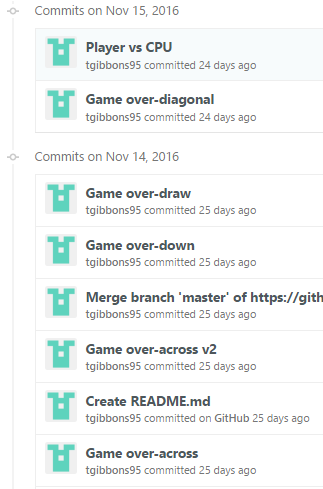
The only other way a game can be finished is in a draw. A draw results from every single column being full. So in setting up this situation, we show a problem we fixed above to the right where a column that is full is requested. It will continue to ask for a different column input. When all of the columns are filled, the game ends in a draw. It will not ask user for input and it will display the result as shown below to the left.

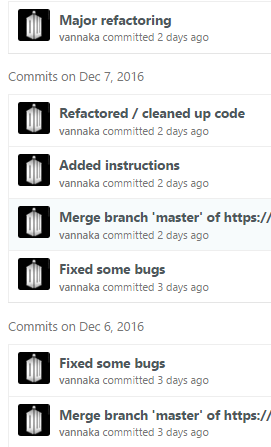
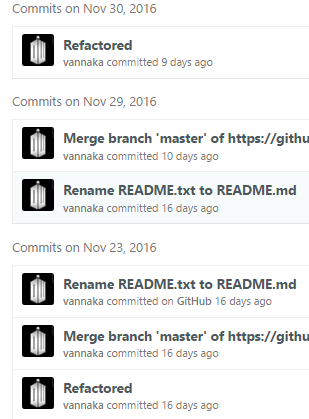


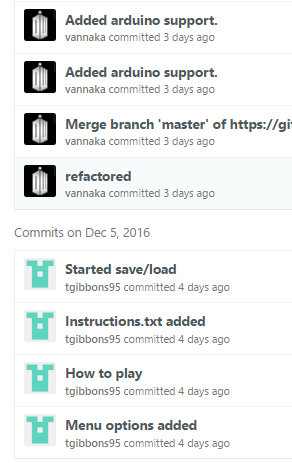
There were also some other tests that cannot easily be shown in a simple screenshot of it working. We had to make sure that the save and load brought the game to the same point in the game. For example, if it was a 2 player game we had to make sure that it started at the right players turn when it returned. We fixed this by saving whose turn it was and then it would switch to the correct player when loaded.

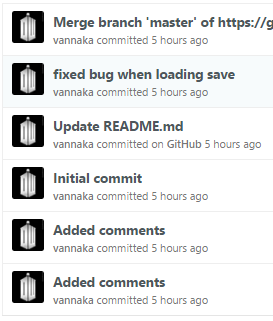
Another situation we ran into was not keeping track of how many pieces were correctly in the columns. At first when we loaded the games we did not correctly update the column lengths so our make move algorithm was placing the new pieces at the bottom and replacing what was there. We fixed it by simply counting how many pieces were not empty in a column. The next issue we had with length was continuing afterwards and playing a new game. We had to reset the lengths and all other appropriate variables like the board and its attributes.

We divided the work very evenly. We got few opportunities to work on it together in person, but we updated read me files and texted on what needed to get done. We both had a very good idea of how the game was going to come together, but it was just a matter of getting all the methods finished and error checking. A lot of our solutions came from bouncing ideas off each other and brainstorming how we want to implement something. The debugging, error checking and testing of the system was some of our biggest time consumption and we both tried to break it for a very long time. As a result we have committed our work to GitHub a lot over the duration of this project.









The game is very robust and has accounted for every error we could throw at the program. There are a limited amount of things that can be thrown and we believe we have accounted for all of them.