CONNECT 4

Luke Mammen and Thomas Gibbons

ECE 3220

FINAL PROJECT

Dr. Rivera

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.

Introduction

The goal of this project was to create a fully functional connect 4 game. The game has several features. It can play either one player vs a CPU or one player vs another player. It can also save the game at any point and then load that game save back. Our main reason for making a connect 4 game was because we both enjoyed connect 4 as kids and wanted to bring it to a computer game.

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.

Proposed method / System description / Implementation

We started the design by looking at the game and identifying 3 main objects that could be symbolized with a class, the game pieces, the columns and the game board. We created a main game class called Connect4 that we used to manage the whole game. In total, we have four classes: gamePiece, gameColumn, gameBoard, Connect4. As you can see in the class diagram, gameColumn includes gamePiece and gameboard include gameColumn and Connect4 includes gameBoard.

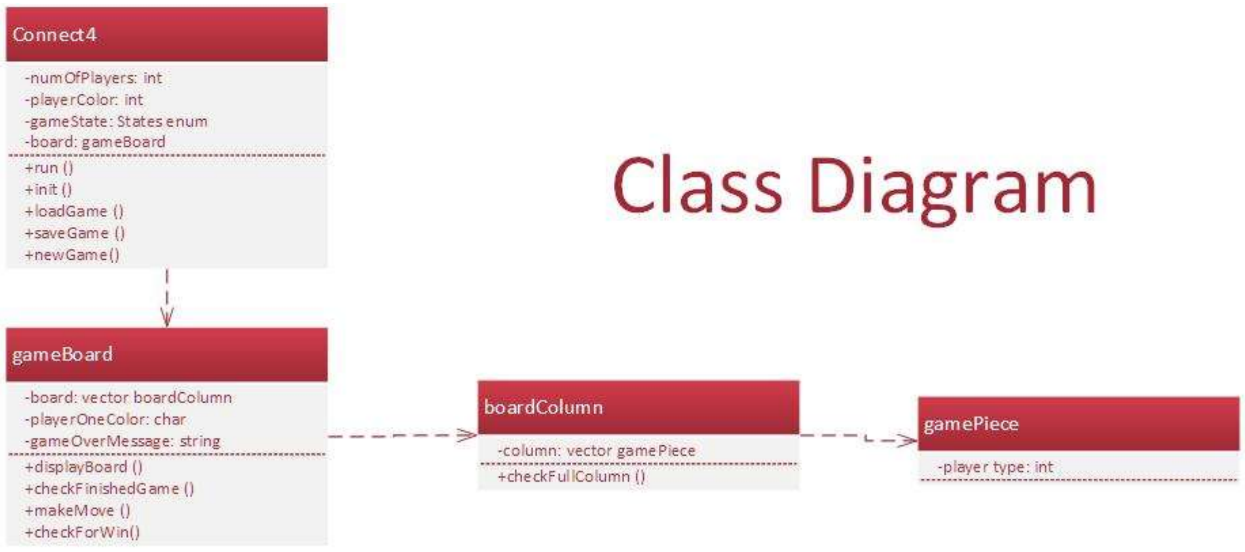
We designed the classes to stack on top of the ones before it. The gameColumn class creates a vector of gamePiece objects. The gameBoard class creates a vector of gameColumn objects. The Connect4 class manages all the logic and displays the UI. The gameBoard class handles checking if there are four piece in a row on the board. It also has a method to display the board on the screen. It also has a method to make a move. This method has a lot of logic to check if the column chosen is full or if the entire board is full. It also calls other methods to check if a connect four has occurred.

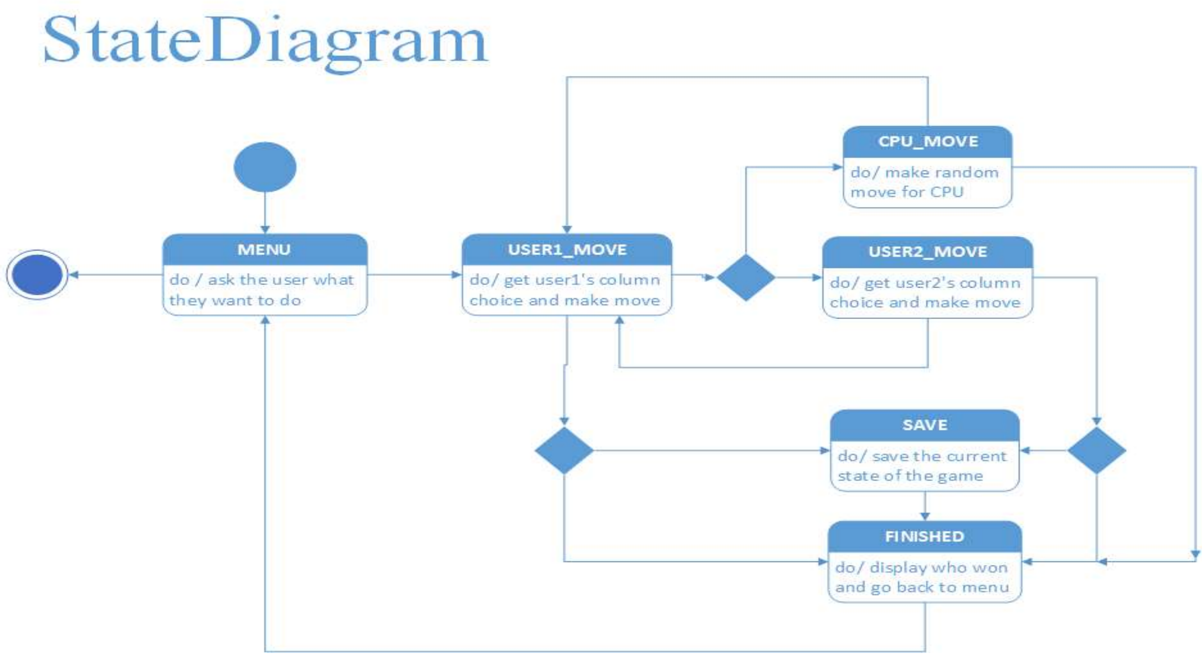
In the Connect4 class is the main game loop in a method called Connect4::run(). We designed the looped on what is called a state machine. The loop consists of a single infinite while loop with a single switch statement inside of it. The cases of the switch statement are defined by and enum containing all the states of our main loop. The states are as follows: INIT, USER1_TURN, CPU_TURN, USER2_TURN, SAVE, LOAD, FINISHED. Inside each state in the switch statement we perform the required action for the given state. For example, in the USER1_TURN state the program asks the user1 for the column they'd like to play their game piece in and then does the proper error checking and makes the move. At the end of each state there is code to decide which state should be executed next. It then changes the state variable accordingly and the while loop loops again. See the state diagram attached to this report for further information.

Most of the input and output is done with the standard cin and cout. The problem we ran into was while the user is making a move we couldn't use cin and at the same time get input from the Arduino because cin is a blocking function. We settled on a function from the Windows API that gets the current state of any key on the keyboard. This allows us to check both the keyboard and the Arduino buttons at the same time.

Our program is very reliable. Together, we've spent tens of hours trying to break the system. Any bugs we did find we patched. We are certain there are no remaining bugs in our program. One thing we had to make sure wouldn't break the game was unplugging the Arduino in the middle of the game. We've got it designed so that if it does get unplugged then the program will just forget there ever was an Arduino and only get input from the keyboard.

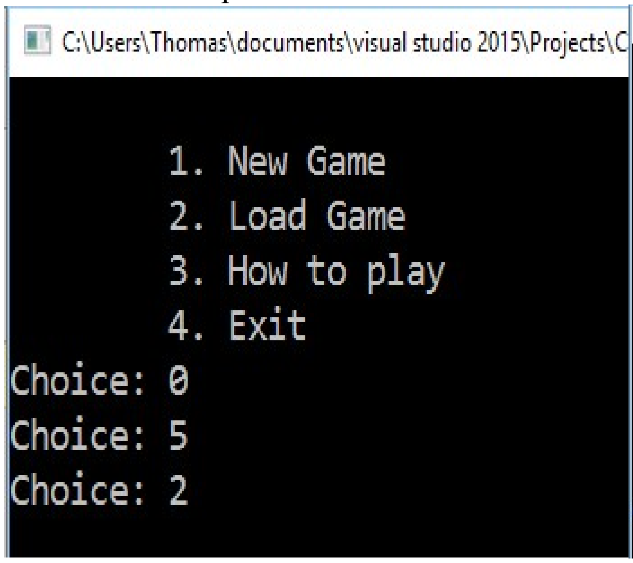
We went through several iterations. The first was the most unpolished. We through together some code to see if our ideas would work. This first iteration only had player vs CPU, it didn't have any save features or even a main menu. During the second iteration, we refactored quite a bit of code to make it easier to read. We added a main menu with options to start a new game, load a saved game, and display the instructions. The main game loop was also rewritten. In iteration one the main loop was very convoluted and hard to follow. In iteration two we redesigned the main loop using a state machine model. This made the code easier to read and the flow of the program easier to follow.




StateDiagram
MENU
do / ask the user what they want to do
USER1_MOVE
do/ get user1's column choice and make move
CPU_MOVE
do/ make random move for CPU
USER2_MOVE
do/ get user2's column choice and make move
SAVE
do/ save the current state of the game
FINISHED
do/ display who won and go back to menu

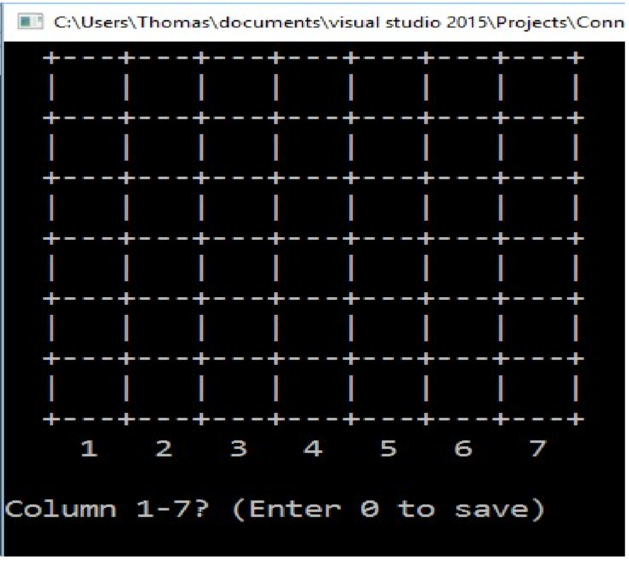
Experiments and Results

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.



C:\Users\Thomas\documents\visual studio 2015\Projects\C

```
        1. New Game
        2. Load Game
        3. How to play
        4. Exit
Choice: 0
Choice: 5
Choice: 2
```

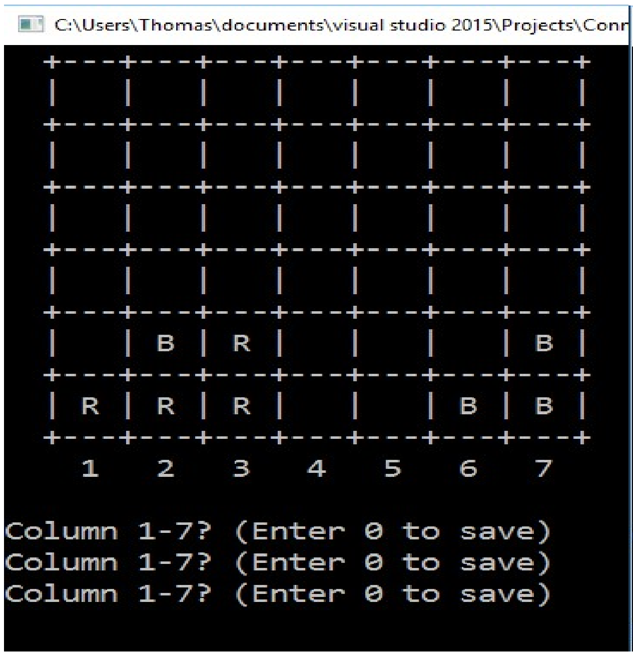


C:\Users\Thomas\documents\visual studio 2015\Projects\Conn

```
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
   1   2   3   4   5   6   7

Column 1-7? (Enter 0 to save)
```

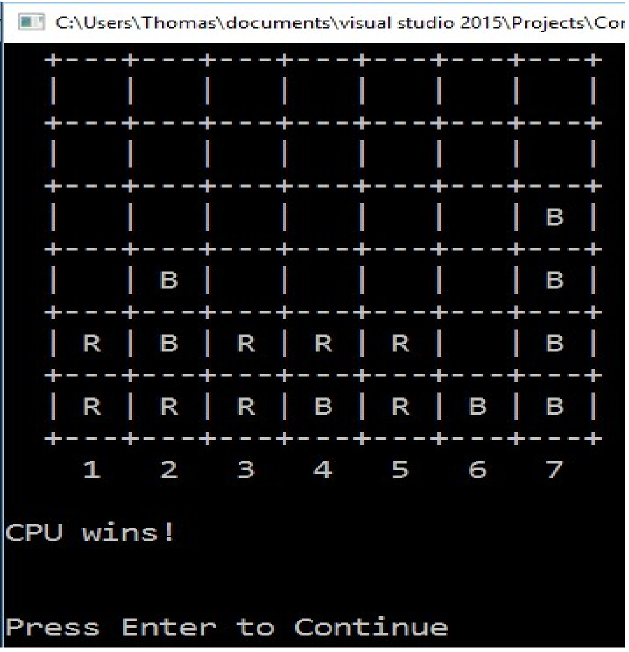
The next possible source of error would be if we tried to load a game without any previous save. The simplest solution was to just have a default save.txt file that was a new, single player game, where player 1 is red. The new game is displayed above on the right that results from loading to start the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



C:\Users\Thomas\documents\visual studio 2015\Projects\Conn

```
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   | B | R |   |   |   | B |
 +---+---+---+---+---+---+---+
 | R | R | R |   |   | B | B |
 +---+---+---+---+---+---+---+
   1   2   3   4   5   6   7

Column 1-7? (Enter 0 to save)
Column 1-7? (Enter 0 to save)
Column 1-7? (Enter 0 to save)
```

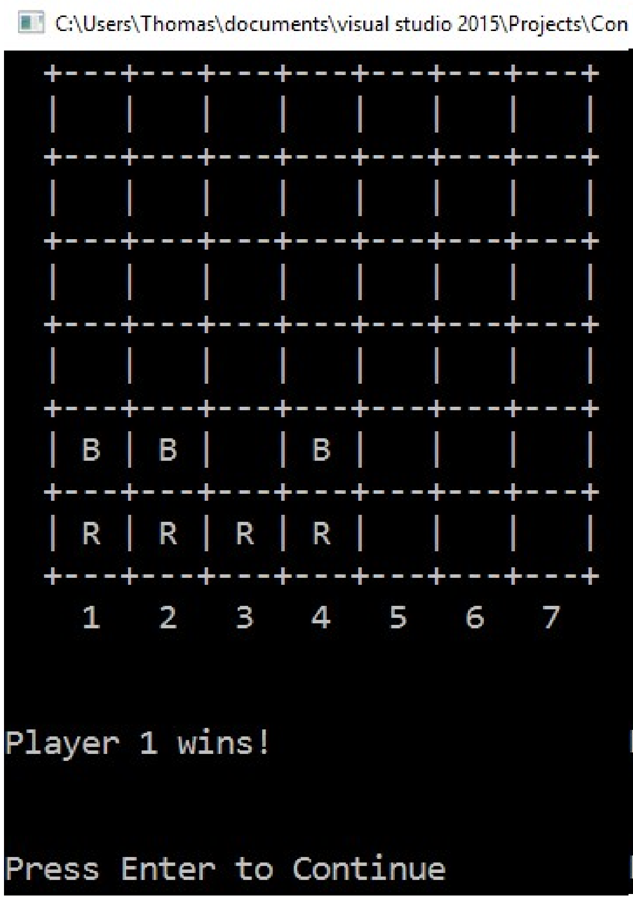


C:\Users\Thomas\documents\visual studio 2015\Projects\Cor

```
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+
 |   |   |   |   |   |   | B |
 +---+---+---+---+---+---+---+
 |   | B |   |   |   |   | B |
 +---+---+---+---+---+---+---+
 | R | B | R | R | R |   | B |
 +---+---+---+---+---+---+---+
 | R | R | R | B | R | B | B |
 +---+---+---+---+---+---+---+
   1   2   3   4   5   6   7

CPU wins!


Press Enter to Continue
```

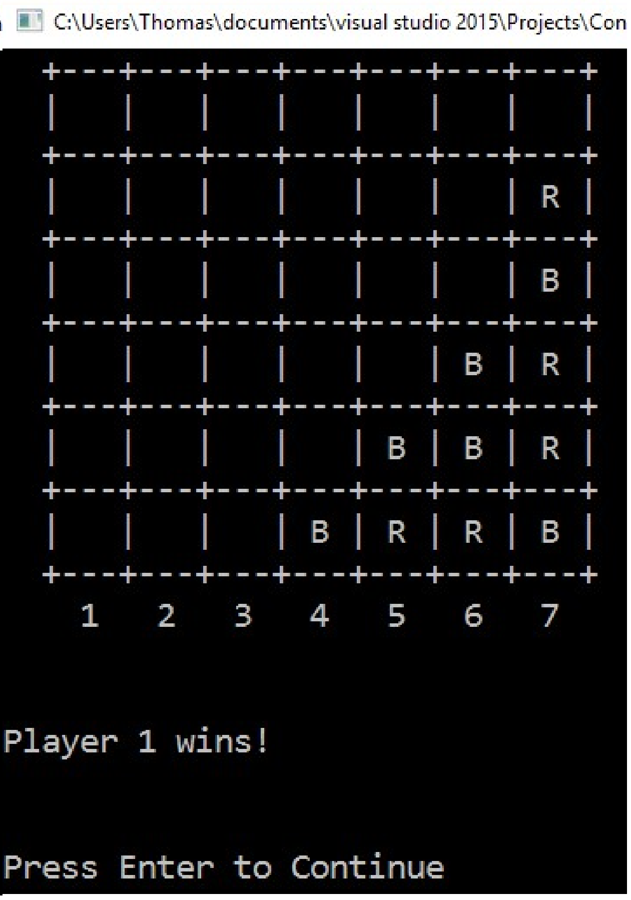
Then we also have tests for all the possible ways on how someone can win and who the person that wins is. For the first example of this we have the computer winning in a vertical way. When the game is over it displays the correct person and the board as shown to the right above.

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.



C:\Users\Thomas\documents\visual studio 2015\Projects\Con

```
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+
  | B | B |   | B |   |   |   |
  +---+---+---+---+---+---+---+
  | R | R | R | R |   |   |   |
  +---+---+---+---+---+---+---+
    1   2   3   4   5   6   7


Player 1 wins!


Press Enter to Continue
```

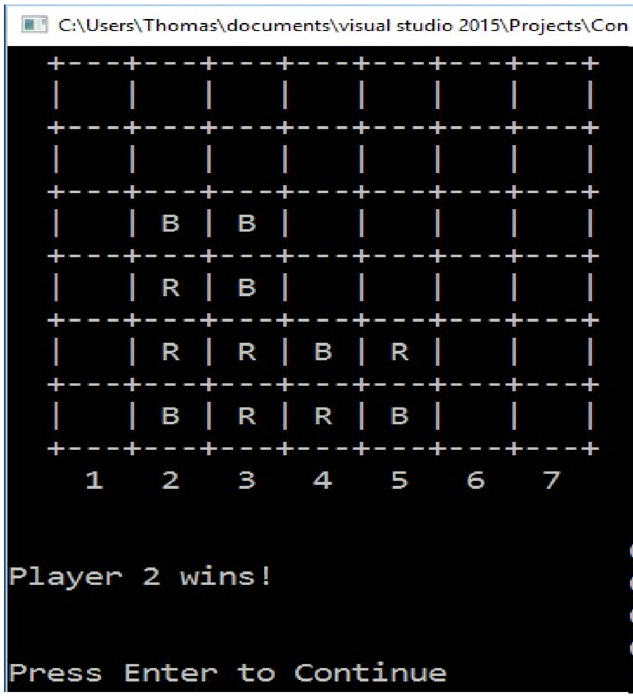


C:\Users\Thomas\documents\visual studio 2015\Projects\Con

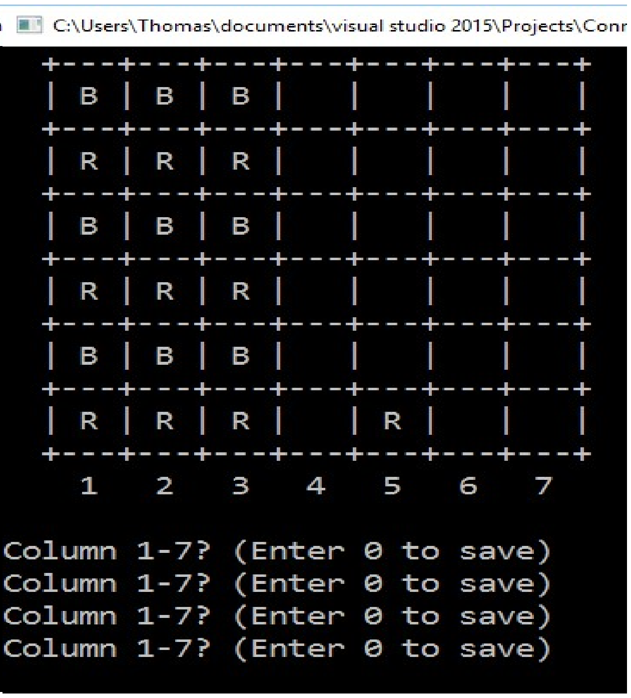
```
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   | R |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   |   | B |
  +---+---+---+---+---+---+---+
  |   |   |   |   |   | B | R |
  +---+---+---+---+---+---+---+
  |   |   |   |   | B | B | R |
  +---+---+---+---+---+---+---+
  |   |   |   | B | R | R | B |
  +---+---+---+---+---+---+---+
    1   2   3   4   5   6   7


Player 1 wins!


Press Enter to Continue
```

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.

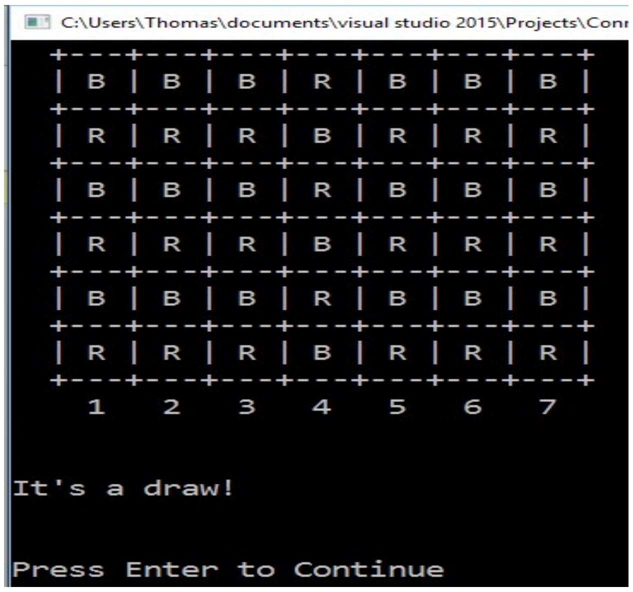


```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   | B | B |   |   |   |   |
+---+---+---+---+---+---+---+
|   | R | B |   |   |   |   |
+---+---+---+---+---+---+---+
|   | R | R | B | R |   |   |
+---+---+---+---+---+---+---+
|   | B | R | R | B |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7


Player 2 wins!


Press Enter to Continue
```



```
+---+---+---+---+---+---+---+
| B | B | B |   |   |   |   |
+---+---+---+---+---+---+---+
| R | R | R |   |   |   |   |
+---+---+---+---+---+---+---+
| B | B | B |   |   |   |   |
+---+---+---+---+---+---+---+
| R | R | R |   |   |   |   |
+---+---+---+---+---+---+---+
| B | B | B |   |   |   |   |
+---+---+---+---+---+---+---+
| R | R | R |   | R |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Column 1-7? (Enter 0 to save)
Column 1-7? (Enter 0 to save)
Column 1-7? (Enter 0 to save)
Column 1-7? (Enter 0 to save)
```

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.

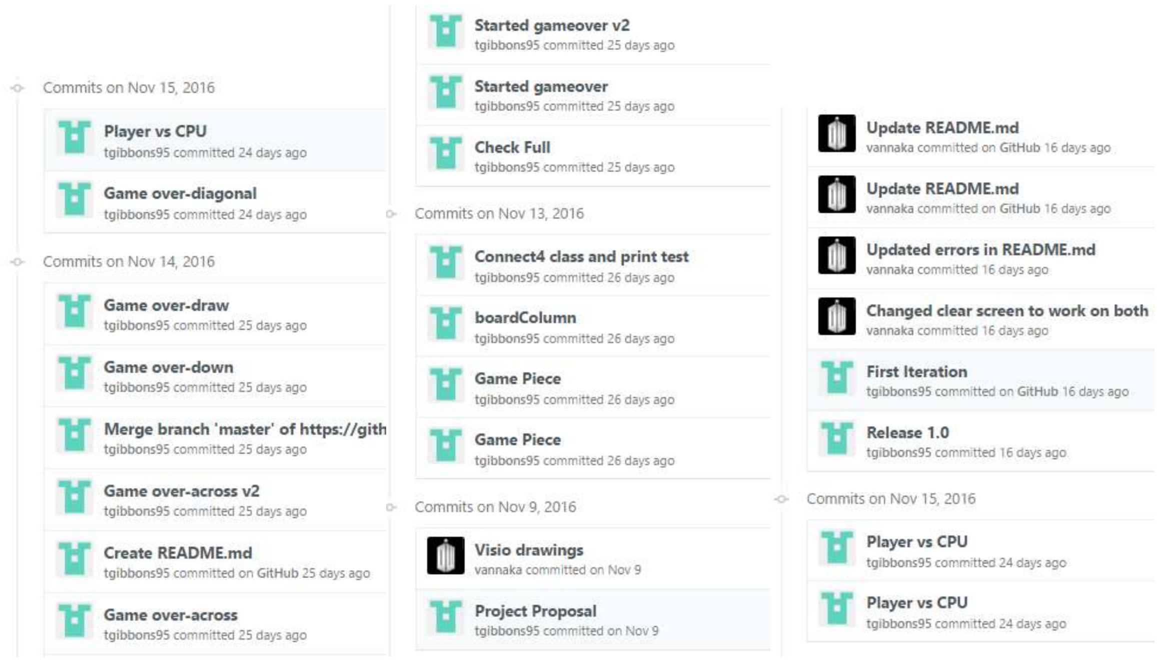


```
+---+---+---+---+---+---+---+
| B | B | B | R | B | B | B |
+---+---+---+---+---+---+---+
| R | R | R | B | R | R | R |
+---+---+---+---+---+---+---+
| B | B | B | R | B | B | B |
+---+---+---+---+---+---+---+
| R | R | R | B | R | R | R |
+---+---+---+---+---+---+---+
| B | B | B | R | B | B | B |
+---+---+---+---+---+---+---+
| R | R | R | B | R | R | R |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7


It's a draw!


Press Enter to Continue
```

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.



Commits on Nov 15, 2016

Player vs CPU
tgibbons95 committed 24 days ago

Game over-diagonal
tgibbons95 committed 24 days ago

Commits on Nov 14, 2016

Game over-draw
tgibbons95 committed 25 days ago

Game over-down
tgibbons95 committed 25 days ago

Merge branch 'master' of https://gith
tgibbons95 committed 25 days ago

Game over-across v2
tgibbons95 committed 25 days ago

Create README.md
tgibbons95 committed on GitHub 25 days ago

Game over-across
tgibbons95 committed 25 days ago

Started gameover v2
tgibbons95 committed 25 days ago

Started gameover
tgibbons95 committed 25 days ago

Check Full
tgibbons95 committed 25 days ago

Commits on Nov 13, 2016

Connect4 class and print test
tgibbons95 committed 26 days ago

boardColumn
tgibbons95 committed 26 days ago

Game Piece
tgibbons95 committed 26 days ago

Game Piece
tgibbons95 committed 26 days ago

Commits on Nov 9, 2016

Visio drawings
vannaka committed on Nov 9

Project Proposal
tgibbons95 committed on Nov 9

Update README.md
vannaka committed on GitHub 16 days ago

Update README.md
vannaka committed on GitHub 16 days ago

Updated errors in README.md
vannaka committed 16 days ago

Changed clear screen to work on both
vannaka committed 16 days ago

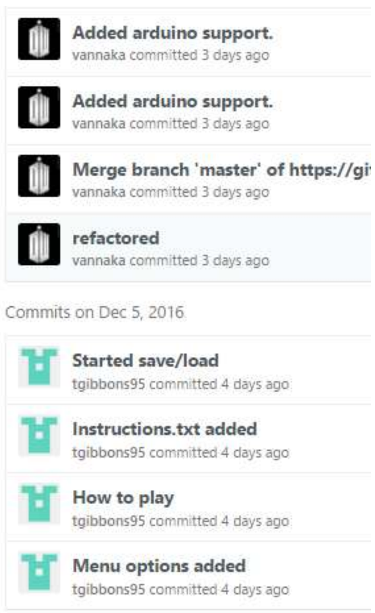
First Iteration
tgibbons95 committed on GitHub 16 days ago

Release 1.0
tgibbons95 committed 16 days ago

Commits on Nov 15, 2016

Player vs CPU
tgibbons95 committed 24 days ago

Player vs CPU
tgibbons95 committed 24 days ago



Added arduino support.
vannaka committed 3 days ago

Added arduino support.
vannaka committed 3 days ago

Merge branch 'master' of https://gi
vannaka committed 3 days ago

refactored
vannaka committed 3 days ago

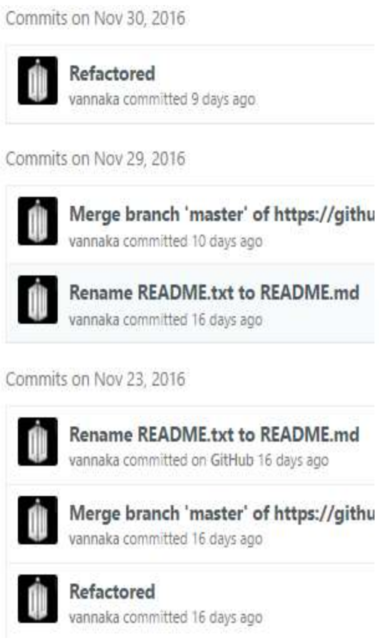
Commits on Dec 5, 2016

Started save/load
tgibbons95 committed 4 days ago

Instructions.txt added
tgibbons95 committed 4 days ago

How to play
tgibbons95 committed 4 days ago

Menu options added
tgibbons95 committed 4 days ago



Commits on Nov 30, 2016

Refactored
vannaka committed 9 days ago

Commits on Nov 29, 2016

Merge branch 'master' of https://githu
vannaka committed 10 days ago

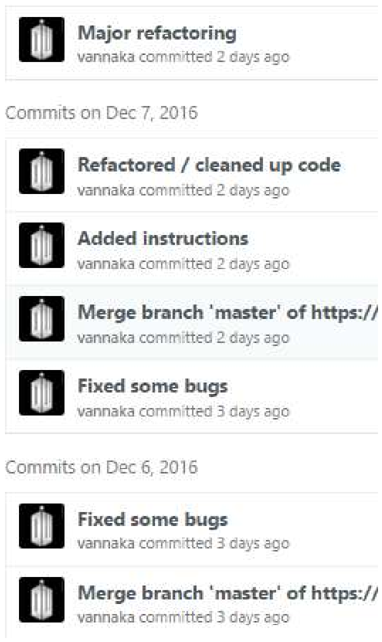
Rename README.txt to README.md
vannaka committed 16 days ago

Commits on Nov 23, 2016

Rename README.txt to README.md
vannaka committed on GitHub 16 days ago

Merge branch 'master' of https://githu
vannaka committed 16 days ago

Refactored
vannaka committed 16 days ago



Major refactoring
vannaka committed 2 days ago

Commits on Dec 7, 2016

Refactored / cleaned up code
vannaka committed 2 days ago

Added instructions
vannaka committed 2 days ago

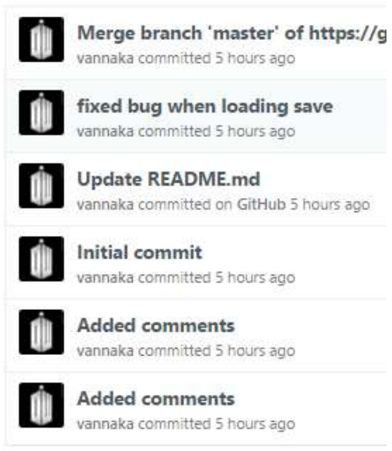
Merge branch 'master' of https://
vannaka committed 2 days ago

Fixed some bugs
vannaka committed 3 days ago

Commits on Dec 6, 2016

Fixed some bugs
vannaka committed 3 days ago

Merge branch 'master' of https://
vannaka committed 3 days ago



Merge branch 'master' of https://g
vannaka committed 5 hours ago

fixed bug when loading save
vannaka committed 5 hours ago

Update README.md
vannaka committed on GitHub 5 hours ago

Initial commit
vannaka committed 5 hours ago

Added comments
vannaka committed 5 hours ago

Added comments
vannaka committed 5 hours ago

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 caught all of them. We have both ran this game many times trying to catch something that will pop out.

Discussion and Conclusion

Our final product ended up just as we had planned it. One problem we encountered was trying to get the keyboard input and the Arduino input at the same time. This was a problem because the standard cin is a blocking function that doesn't let any other code run until the user has entered something. This wouldn't work for us because the Arduino could be used to select the input instead of the keyboard. We ended up using a function from the windows api to get the states of the keyboard. Then we wrote some code that said when the state of the key changes from pushed down to released then return said key. Similar code was written for the Arduino buttons.

One weird problem with our program is that during the game since we are getting input with the win api function the input buffer is still filling up with key presses so then at the end of the game when we ask the user to hit enter to go to the main menu and use cin to get that input all the character that were typed throughout the game are then displayed on the screen. The user then must backspace through these characters before they can enter anything useful. We couldn't find any way of clearing this. There were several suggestions on the internet, but none worked.

As far as the previously expressed problem the program works flawlessly as expected. In the future, all the input could be taken from the Arduino with more buttons. Overall, we had few problems and could implement the code the way we had originally planned. This was a fun project that was at first challenging until we made the class diagram to help aid in the design process.

Appemdix

A repository of our code can be accessed at https://github.com/tgibbons95/FinalProject/tree/master/code/code%203.0. It includes our source files, .txt files, a folder for our headers, and a readMe file for help compiling.